Máster Universitario en Ingeniería Informática

Universidad Politécnica de Madrid

Escuela Técnica Superior de
Ingenieros Informáticos

TRABAJO FIN DE MÁSTER

Application of blockchain technologies to the decentralization of data marketplaces

Autor: María Salgado Iturrino

Director: Francisco Javier Soriano Camino

MADRID, JUNE 16, 2018

Acknowledgment

To my family: because I owe it all to you. Many Thanks!

Thanks to all the project team: Alberto, Fran, Miguel and, of course my tutor, Javier Soriano.

I would like to thank Gonzalo Gomez Lardies for his valuable comments and suggestions, which have been very helpful during the review of this master's thesis document.

Also to all my other mates in the CoNWeT Lab, it was great sharing laboratory with all of you during this year. With a special mention to: Victor, Sandra, Miguel and Alejandro.

I am grateful to all the people around, from teachers like Geno, to University staff like Agustín and María.

And finally, last but by no means least, I must express my very profound gratitude to Juanan. Thanks for always supporting me.

Resumen

Resumen — La comunidad de software libre europea FIWARE ofrece distintas herramientas para la gestión de datos a través de su plataforma tecnológica del mismo nombre. Trabaja con datos en tiempo real basados en el estándar NGSI utilizando la plataforma Orion Context Broker y, gracias al Business API Ecosystem, comercializa y controla su acceso. Esta combinación genera un ecosistema y, en particular, un mercado de datos (en adelante *data marketplace* o simplemente *marketplace*). A través de esta plataforma cualquier participante puede hacer disponibles sus datos (de forma gratuita o con un modelo de pagos), así como acceder a los datos ofrecidos por otros participantes para hacer uso de ellos o para generar valor añadido combinándolos, procesándolos o generando aplicaciones/servicios en base a los mismos. El *marketplace* de la plataforma FIWARE está concebido como un elemento central que integra el catálogo de productos y ofertas, establece los contratos, recibe los pagos y se encarga de la distribución de los beneficios. Por lo tanto, todos los participantes tienen que confiar en él.

Por otra parte, la tecnología de cadenas de bloques (en adelante *blockchain*) es un mecanismo que, basándose en cálculos criptográficos, ofrece un libro maestro contable distribuido (denominado *ledger*) en el que se apuntan transacciones entre participantes que no tienen una relación de confianza entre ellos. La principal característica de este libro maestro, que se distribuye entre múltiples nodos participantes de la red, es que es inmutable, no repudiable y no depende de relaciones de confianza entre participantes o en una entidad central.

El objetivo de este trabajo es doble. Por un lado se estudia la aplicación de la tecnología *blockchain* en lo relativo al catálogo, las compras y los pagos, elementos todos ellos clave del *marketplace*, para establecer una arquitectura descentralizada capaz de gestionar el almacenamiento y acceso a las ofertas, así como la creación de acuerdos de forma distribuída. Por otro lado, se pretende convertir la distribución de datos a un modelo P2P, que no requiera una relación de confianza con el *marketplace*, ni entre ellos.

Este proyecto analiza las tecnologías *blockchain* existentes y plantea distintas arquitecturas, combinándolas para satisfacer distintos requisitos, como modelos de precio complejos que requieren *accounting.* Todas las propuestas están validadas mediante pruebas de concepto cuyas especificaciones técnicas también se detallan. Al final, todos los resultados se recogen en las conclusiones que contienen una guía de decisión estratégica para diseño de arquitecturas de *marketplaces* de datos descentralizados.

Palabras clave — Mercado de datos, descentralización, Blockchain, Distributed Ledger Technologies, Tangle, Hyperledger

Abstract

Abstract — The European open source community FIWARE offers different tools for data management through its technological platform of the same name. It works with real-time data based on the NGSI standard using the Orion Context Broker platform and, thanks to the Business API Ecosystem, markets and controls its access. This combination generates an ecosystem and, particularly, a *data marketplace*. Through this platform, any participant can make their data available (free of charge or with a payment model), as well as access data offered by other participants to use it or to generate added value by combining, processing or generating applications/services based on it. The FIWARE's marketplace is conceived as a central element that integrates the catalog of products and offers, establishes contracts between suppliers and consumers, receives payments and is responsible for the distribution of benefits. Therefore, all participants in these types of exchanges have to trust it.

On the other side, the blockchain technology is a mechanism that, based on cryptographic calculations, offers a distributed ledger to register transactions between participants who do not trust each other. The main characteristic of this ledger, which is distributed among multiple participating nodes of the network, is that it is immutable, non-repudiable and does not depend on trust relationships between participants or in a central entity.

The goal of this paper is twofold. On the one hand, the application of blockchain is studied in terms of catalog, purchases and payments to establish a decentralized architecture capable of managing storage and access to offerings, as well as the creation of agreements in a distributed way.. On the other hand, we intend to convert the data distribution to a P2P model, which does not require a trust relationship with the marketplace, nor between them.

This project analyses the existing *Distributed Ledger Technologies* (DLTs) and proposes different architectures, combining them to satisfy different requirements, such as complex price models that require accounting. All the proposals are validated through proofs of concept whose technical specifications are also detailed. In the end, all the results are included in the conclusions that contain a strategic decision guide for the design of decentralized data marketplaces architectures.

Key words — Data Marketplace, decentralization, Blockchain, Distributed Ledger Technologies, Tangle, Hyperledger

Contents

1 Introduction 1

1.1 Context . . . 2

1.1.1 The FIWARE project and the CoNWeT laboratory . . . 3

1.1.2 The technical context: blockchain . . . 5

1.2 Motivation . . . 6

1.3 Objectives . . . 6

1.4 Document Structure . . . 8

2 State of the Art 9

2.1 Blockchain . . . 10

2.1.1 Blockchains classification . . . 11

2.1.2 Smart contracts . . . 13

2.1.3 Consensus . . . 16

2.1.4 Security on the blockchain . . . 21

2.1.5 Distributed Ledger Technologies . . . 22

2.2 Decentralized Data Marketplaces . . . 33

2.2.1 Blockchain-based data distribution marketplaces . . . 33

2.2.2 Cloud Services . . . 34

2.2.3 Sharding-based . . . 35

2.2.4 Peer to Peer . . . 35

2.3 FIWARE access data technologies . . . 36

2.3.1 CKAN . . . 36

2.3.2 NGSI . . . 37

2.3.3 Orion Context Broker . . . 38

2.3.4 Business API Ecosystem . . . 38

3 Marketplace decentralization scenarios 41

3.1 Scenario 1. Decentralized catalogue and P2P data distribution . . . 44

3.2 Scenario 2. Accounting: PEP . . . 47

3.3 Scenario 3. Decentralized accounting: ledger 48
3.4 Scenario 4. Blockchain data distribution 50
3.5 Scenario 5. Two ledgers 52

4 Proofs of concept 57
4.1 Hyperledger Composer 57
4.1.1 Hyperledger Composer Playground deployment on Bluemix . 59
4.1.2 Hyperledger Composer Playground deployment over Hyperledger Fabric in two different physical machines 62
4.1.3 Angular web application running with a Rest Hyperledger Server 65
4.2 Hyperledger Sawtooth 66
4.3 Tangle 68
4.3.1 Full-node connected to the testnet 70
4.3.2 Two full-nodes connected to the mainnet 71
4.3.3 Private Tangle network 72
4.3.4 Some peers connected to the mainnet using Masked Authentication Messages (MAM) 74

5 Conclusions 79
5.1 Strategic architecture analysis for decentralized data marketplaces . 80
5.2 Blockchain and the current business models 84

6 Future Work 87

Bibliography 91

A Scripts 95
A.1 Sawtooth scripts 95
A.2 Tangle deployment script 105
A.3 MAM testing scripts 108

List of Figures

1.1 Global view of FIWARE components. 4
1.2 Blockchain decision path. 7

2.1 Oraclize, an oracle service 14
2.2 Vitalik Buterin's proposal scheme for decision and operation using humans or robots according to the business model 17
2.3 Bitcoin Block Generation Time vs Difficulty. 19
2.4 Bitcoin defines an electronic coin as a chain of digital signatures to verify ownership. 23
2.5 Bitcoin Merkle Tree 24
2.6 IOTA is a Tangle-based cryptocurrency 31
2.7 Offline transactions (partitioning) 32

3.1 Scenario 1. A decentralized catalogue and Peer-to-Peer (P2P) data distribution with a PEP 45
3.2 Scenario 2. A decentralized catalogue, P2P data distribution with accounting through a monitoring PEP 47
3.3 Scenario 3. A decentralized catalogue and accounting with P2P data distribution 49
3.4 Scenario 4. A decentralized catalog and data distribution in one ledger 50
3.5 Do IoT and blockchain relationship make sense? 52
3.6 Scenario 5. One ledger for catalog and the other ledger for data distribution 54

4.1 Hyperledger Composer Playground deployment on Bluemix 60
4.2 BAE Hyperledger modelization. 61
4.3 Smart Plastic use case sequence diagram. 62
4.4 Proof of Concept (PoC) of two different physical machines Hyperledger deployment 63

4.5 Screenshot of the first machine Composer deployment. 64
4.6 Screenshot of the second machine Composer deployment. 64
4.7 Screenshot of the REST API explorer interface. 66
4.8 Sawtooth PoC. Submitting and subscribing to events from console. 67
4.9 Sawtooth events system architecture. 68
4.10 A transaction is not confirmed until a milestone verifies it (directly or indirectly) 69
4.11 A full-node synchronized with the Tangle mainnet (milestone 426550) connected with 7 neighbours 73
4.12 Data publisher in the Tangle. MAM Demo 76
4.13 Data receiver in restricted mode in the Tangle. MAM Demo 77
4.14 Data receiver in private mode in the Tangle. MAM Demo 77
4.15 Data receiver in public mode in the Tangle. MAM Demo 78

5.1 Decentralized data distribution Decision Flowchart Diagram (DFD) 83

6.1 Spark data process in real time. 89

List of Tables

2.1 Blockchains classification table 11

3.1 Scenarios comparative table 45

4.1 Glossary mapping between Hyperledger and FIWARE. 58

Acronyms

ACL Access Control List. 61, 81

BAE Business API Ecosystem. 43, 44, 46, 53, 84

BFT Byzantine's Fault Tolerance. 19, 81

CA Certification Authority. 63

DFD Decision Flowchart Diagram. 79, 82, 83

DLT Distributed Ledger Technology. 10, 29, 84, 85, 88

EVM Ethereum Virtual Machine. 27, 29

GE Generic Enablers. 3, 43

IoT Internet of Things. 52, 53, 68, 75, 82, 84, 85

IRI IOTA Reference Implementation. 71

MAM Masked Authentication Messages. 53, 55, 70, 72, 74–78, 83

MSP Membership Service Provider. 46, 58, 81

P2P Peer-to-Peer. 45, 49, 81–83

PEP Police Enforcement Point. 46

PoC Proof of Concept. 63, 67

PoET Proof of Elapsed Time. 20, 29

PoS Proof of Stake. 21

PoW Proof of Work. 72

RWMC Random Walk Monte Carlo. 72

SGX Software Guard Extensions. 66

VM Virtual Machine. 63, 71, 72, 74

ZMQ Zero Message Queue. 67

1 Introduction

- So, what is Blockchain?
+ Probably the most hyped term in the history of technology and business. It's going to save the world. It's going to cure diseases. It's going to be amazing.

Karim R. Lakhani,
professor at Harvard Business School

The FI-NEXT European project, framed in the FIWARE European open-source software community supported by the FIWARE Foundation[1] under the auspices of the European Commission, aims at offering different tools for the management of open data (CKAN with extensions). The project manages either static and real-time data based on NGSI through the Orion Context Broker platform, as well as tools (Business API Ecosystem and APInf) for the data commercialization and administration in order to support the creation of a global data economy ecosystem.

[1] https://www.fiware.org/foundation

This combination generates a data market (hereinafter *marketplace*), in which any participant of the platform can make their data available to the community (free of charge or with a payment model), while also being able to use the data which others provide. Users are allowed to generate value from data by combining, processing or creating applications and services that consume it.

This platform is conceived as a central element that integrates the catalogue of products and offers, and that interacts with suppliers and consumers. As a consequence, it establishes agreements with suppliers and consumers, receives payments and is responsible for the distribution of benefits. Therefore, all participants in this type of exchanges have to trust on this central element that is the marketplace.

Blockchain is a mechanism that, based on cryptographic calculations, offers a distributed ledger in which transactions are recorded between participants who do not have a trust relationship between them. These transactions do not have to be monetary, they refer to real or virtual products that pass, completely or partially, from one hands to another. The main characteristic of this ledger, which is distributed among multiple participating nodes of the network based on the mentioned blockchain, is that it is immutable, not repudiable, and does not depend on trust relationships between participants or between participants and a central entity.

The objective of the present project is twofold. On the one hand, the blockchain technology is intended to be applied to the idea of a global data marketplace mentioned above, to establish a ledger in commercial transactions and access permissions, as well as in the monitoring and management of access to data by the participants who have acquired that right. On the other hand, it is intended to convert the existing marketplace into a decentralized element, making only the catalogue function. Transactions between participants can be carried out in a P2P model, in which a trust relationship with the marketplace is not required, nor among them.

1.1 Context

This section defines the project context. It starts exposing the global project of FIWARE where the Business API Ecosystem and Orion Context Broker are placed. It also introduces CoNWeT which is the laboratory where this project is developed. Afterwords it describes the current technical context of blockchain which is the main technology this project is about.

1.1.1 The FIWARE project and the CoNWeT laboratory

FIWARE is an open sustainable platform built around public, royalty-free and implementation-driven software standards. It makes and shares open source technology for smart solutions of different domains.

A smart application in FIWARE terminology collects relevant information from different sources about what is happening at any given time and exploits it for the benefit of its user. This is known as *context information*. To get an intelligent behaviour the current and historical context information is processed, visualized and analysed on a large scale.

FIWARE wants to promote a standard that describes how to collect, manage and publish context information and additionally provides elements that allow to exploit this information once it is collected, such as some APIs to connect to the Internet of Things. That standard is key to build a single digital market for smart applications where apps and solutions can be ported from one client to another without major changes. It also solves in a simple way how to capture information from sensor networks although they communicate using different protocols and IoT languages. For this reason, it is able to solve the complexity of treating the information collected by the sensors and translating them into a common language.

These standards in FIWARE are called Generic Enablers (GE), each one support interoperability with the others. All GE specifications are public and royalty free. A FIWARE Generic Enabler Implementation(GEi) is a reference implementation of one or more GE provided by FIWARE as a fully functional example. The different GEs developed within the FICORE project are organized in the following technical chapters acording to [11]: *Cloud Hosting*, *Data/Media Context Management*, *IoT Services Enablement*, *Advanced Web-Based UI*, *Applications/Services and Data Delivery*, *Security* and *Advanced Middleware, Interfaces to Networks and Robotics.*

In the *Applications/Services and Data Delivery* chapter, one of the major architectural blocks is the Business Framework which is responsible for the commercialization of FI products (Apps, services and data). This includes:

- Publicizing, purchasing and monetizing of products.
- Search and discovery from different providers of different digital assets, including apps, datasets, APIs, services, etc., that best fit the customer requirements.
- Management of different business aspects related with the selling of digital

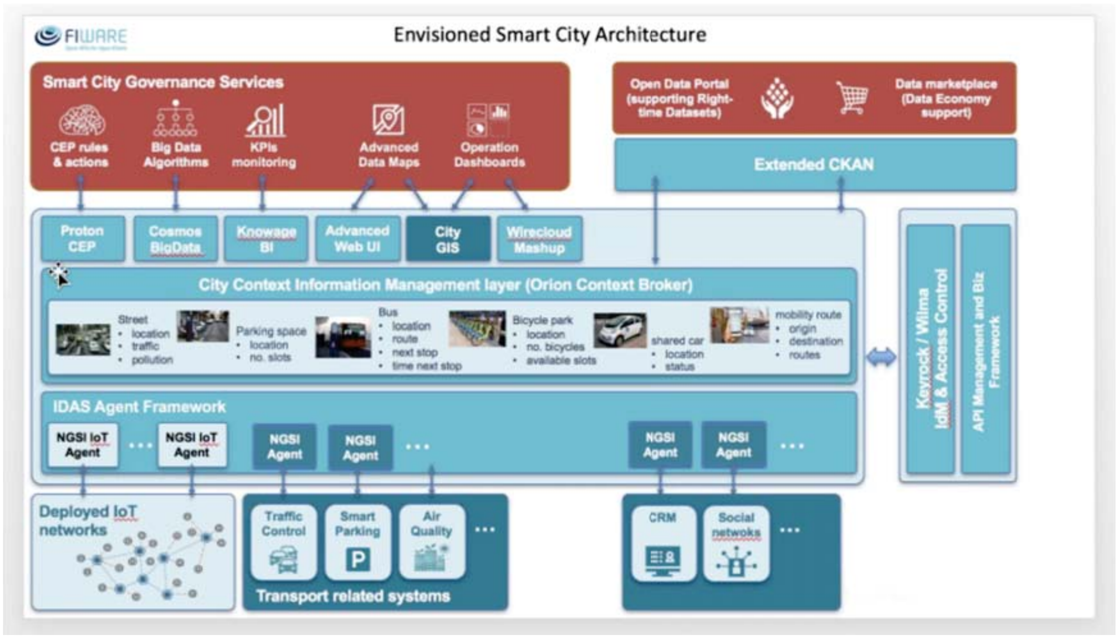


Figure 1.1: Global view of FIWARE components.

Source: Juanjo Hierro' slides[25]

assets, including the management of pricing and revenue models, charging and revenue sharing.

- Accounting of the usage of those products to enable pay-per-use pricing models.

The current state of the Business Framework is detailed on section 2.3.

There is an active FIWARE community that supports developers, understanding and using the technology. It funds and guides start-ups and SMEs who wants to use it to power up their business by promoting FIWARE-based projects. FIWARE is proving to be useful in other areas of IoT such as Smart Agrifood or Smart Industry (Industry 4.0), where standardization is playing a crucial role. The European Commission in April included FIWARE among its recommendations on digitization as a platform based on its strategy.

The CoNWeT Lab (Computer Networks and Web Technologies Laboratory) is part of the CETTICO research group from the Escuela Técnica Superior de Ingenieros Informáticos de la Universidad Politécnica de Madrid. CoNWeT is dedicated to research around Internet of Things (IoT) and web technologies, leading the development of the FIWARE Business API Ecosystem (BAE). UPM participates in the FIWARE project through CETTICO.

1.1.2 The technical context: blockchain

Blockchain is "probably the most hyped term in the history of technology and business" according to Karim R. Lakhani, professor of Harvard Business School. Since the creation of Bitcoin in 2008 its popularity rapidly increased because the technology is disrupting for the current business models. It proposes a new philosophy: decentralization. Blockchain also introduces a new paradigm which challenges our current system: the trustless trust.

The first field where blockchain strongly irrupted is the financial sector. There are 1759 cryptocurrencies according to `investing.com`'s current list. Not all of them are significant, or even legitimate, because it is surprisingly easy to create your own one. The total market capitalization is around $369 billion however the top 20 cryptocurrencies account for 89% of the total quantity. It evidences that blockchain technology is spread and anyone can access it.

Developers and the IT sector are anxious to explore the new possibilities and to face the new problems that blockchain is causing. There are many investigation teams in Universities and companies all over the world working hard on exploiting it before than the others. New blockchain projects with different targets are appearing more and more frequently: Ethereum, Hyperledger, Rootstock (RSK), Corda, Kadena, etc. This is the reason why many companies are creating frameworks to easily deploy a blockchain network: Amazon, Microsoft Azure, IBM and many other giants are providing enterprise solutions, as well as lots of start ups and small companies. New ideas which are based on blockchain concepts are emerging like Tangle which is a distributed ledger implemented on a Directed Acyclic Graph instead of a chain.

There are already many working commercial solutions based on blockchain, however proof-of-concepts continue to constitute the largest type of enterprise blockchain activity. Industries which are investing in blockchain the most are financial services, auto-mobiles, aviation, shipping, retail, telecommunications and IoT

Blockchain has many advantages but is not a mature technology, it is on development yet. It means that there are many theoretical approaches which are not feasible to implement and dead ends that hinder research.

1.2 Motivation

Using blockchain makes the difference and revalues the data marketplace. The main points to highlight when describing a decentralized marketplace are the following ones:

- It is a **trustless based** system. The accounting of information access is made by the ledger so neither providers or consumers may suspect the other or the central organization. Blockchain records transactions eliminating human error and protecting data from possible tampering. It does not require trusting other participants. The integrity of any entity is garanteed.
- It is a **reliable** system. If data distribution is blockchain based, consumers have all the grantees that they receive what they are paying for supported by all the ledger if not. This is because every node validates all the transactions [47].
- It is a **decentralized** system. Data providers do not need to trust in a central entity to store its data. It enables the fast, secure and cheap transfer of assets across the globe between organizations.
- It is an **automatic** system. Time-consuming contractual transactions can bottleneck any trade but with smart contracts agreements can be automatically validated, signed and enforced.

All the exposed advantages of blockchain could be tempting anyone to implement it in any use case just because it is trendy. In figure 1.2 there is a decision path to determine if blockchain is required or not. In a data marketplace there is an important need to shared data, there are multiple parts involved with conflicting incentives and not trusted between themselves, governing rules are not the same for all the participants, accounting is needed for some complex pricing models and registries are immutable.

1.3 Objectives

The present project has two main objectives: (1) the application of Blockchain technology to establish one or several ledgers for the management of purchases and access to data, and (2) the decentralization of the marketplace based on

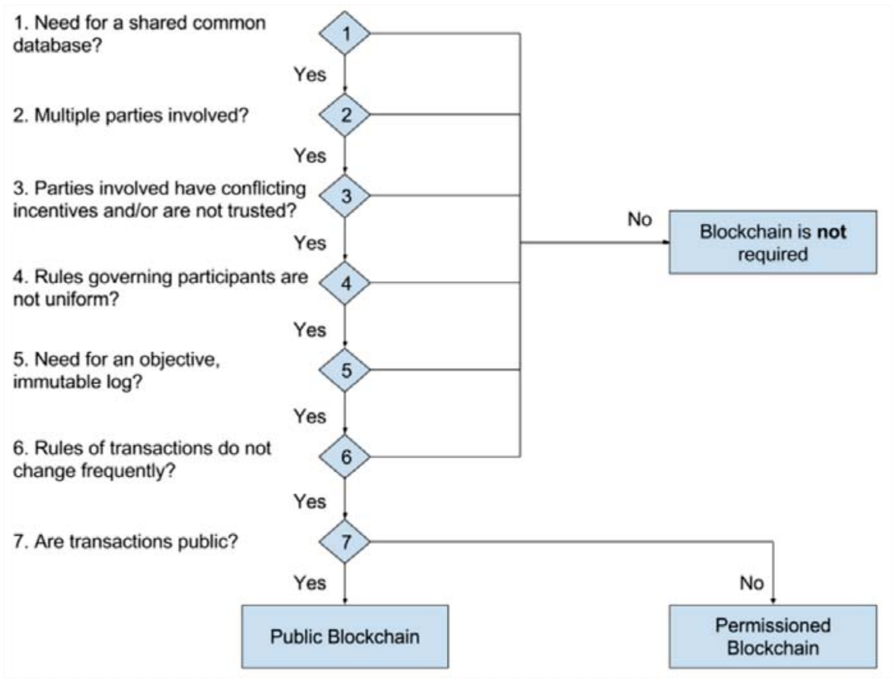

1. Need for a shared common database?
Yes
2. Multiple parties involved?
Yes
3. Parties involved have conflicting incentives and/or are not trusted?
Yes
4. Rules governing participants are not uniform?
Yes
5. Need for an objective, immutable log?
Yes
6. Rules of transactions do not change frequently?
Yes
7. Are transactions public?
Yes
No
Blockchain is **not** required
No
Public Blockchain
Permissioned Blockchain

Figure 1.2: Blockchain decision path.

Source: Blockchain for Business course by the Linux Foundation.

the realization of the transactions (and the consequent right to access data) in trusted distributed mode. These general objectives are specified in a list of specific objectives:

- Analysis of the existing blockchain technologies and selection of the one that best suits the two general objectives of the project.
- Storing transactions of agreements (acquisition of the right to access data and services) in a ledger. These transactions are sent between customers/sellers and the marketplace, but the objective is to send them directly between individuals.
- Support different price models automatically.
- Store your own access rights to the data in a ledger, and manage them automatically based on different supported price models.

1.4 Document Structure

The current document is divided into the following chapters dealing with the different objectives identified in section 1.3.

The second chapter *State of the Art*, contains an introduction to blockchain. It presents the main concepts to understand the technology and it proposes different criteria for classifying blockchain solutions. Afterwards, it details how smart contracts work, the different consensus algorithms, the security problems associated to blockchain and the most important existing blockchain projects such as *Bitcoin*, *Ethereum*, *Hyperledger* and *Tangle*. The chapter also includes an overview of the existing decentralized data marketplaces classified by how data is distributed: *blockchain-based*, *cloud services*, *sharding-based* and *peer-to-peer*. Moreover, there is a review of the technologies that have been useful for the development of the current project.

The next chapter, *Scenarios of marketplace decentralization*, exposes five different architectures that have been designed in order to achieve an adequate decentralization level according to the requirements of marketplaces. Advantages and disadvantages of each of them are analysed in detail.

Then, chapter 4, *Proofs of concept*, presents the prototypes that supports the theoretical approach of the previous scenarios. It details the modelization process and the final implementation of each of them.

Chapter 5, *Conclusions*, collects the results of the tests and compares the different scenarios together to raise the conclusions of the work carried out. It justifies how the developed project satisfies the existing objectives.

Finally, chapter 6, *Future Work* proposes open research lines that can be applied in a future evolution of the FIWARE Business Framework and the Orion Context Broker. This section is specially relevant because, since blockchain is such a recent technology, every day new projects and technologies are emerging and changing bases. To work with blockchain is necessary to be more than ever on the crest of the wave.

2
State of the Art

If I did not reflect so much, study and plan the ascension carefully, I would have been dead for a long time.

Tomo Cesen,
first man who climbed the Alps in winter

In the first section of this chapter, the blockchain concept and its main features are introduced as well as different criteria to classify blockchain technologies and their security challenges. Then, it explains how smart contracts and consensus algorithms work. Afterwards there is an overview of the most important existing blockchain projects: *Bitcoin*, *Ethereum*, *Hyperledger* and *Tangle*. The second section evaluates the *State of the Art* regarding commercial solutions that implement decentralized data marketplaces grouped by their data distribution method: *blockchain-based*, *cloud services*, *sharding-based* and *peer-to-peer*. Finally, the last section describes the most relevant technologies that are used in the FIWARE Business Application Ecosystem and in this project.

2.1 Blockchain

Blockchain is a new technology which is disrupting for the current business models. This section describes the general principles of blockchain and its most important working networks and technologies. At the beginning, it explains *Bitcoin* [35] because it appeared in October 2008 as the first cryptocurrency ergo it is the first application using blockchain technology. Secondly it describes *Ethereum*, which was launched in July 2014, because it is by far the most important blockchain implementation after Bitcoin, its core idea is using a blockchain to track economic transactions can be generalized to other kind of transactions. Afterwords, we introduce *Hyperledger* which is an open source collaborative framework created to advance cross-industry blockchain technologies hosted by The Linux Foundation. This chapter also includes *Tangle* which was specifically created for IOTA, a cryptocurrency focused on Internet of Things, although it does not fit exactly the paradigm of blockchain.

What is blockchain?

Blockchain is a peculiar kind of database. It provides an immutable shared distributed digital ledger to store every transaction chronologically and publicly. Data is provided cooperatively by a pool of participants instead of a central administrated database. The Blockchain term is also substitute by Distributed Ledger Technology (DLT). Trust between participants appear when they reach consensus about the existance, state and evolution of some transactions. There are different ways to achieve consensus that will be analyzedd in following chapters. This are the five basic principles underlying the technology according to Harvard University [29].

1. **Distributed database**. Each participant on a blockchain has access to the entire database and its complete history. No single peer controls the data or the information. Every party can verify the records of its transaction partners directly, without an intermediary.

2. **Peer-to-Peer transmission**. Communication occurs directly between peers instead of through a central node. Each node stores and forwards information to the rest of participants. This is the great value of blockchain: there is no a central trusted party.

3. **Transparency with pseudo-anonymity**. Every transaction and its associated value are visible to anyone with access to the system. Each node, or user, on a blockchain has a unique alphanumeric address that identifies

it. Users can choose to remain anonymous or provide proof of their identity to others. Transactions occur between blockchain addresses generated using asymmetric cryptography.

4. **Consensus to achieve irreversibility and integrity**. Once a transaction is entered in the ledger, the records cannot be altered, because they are linked to previous transaction record that (hence the term "chain"). Various computational algorithms and approaches are deployed to ensure that the recording on the database is permanent, chronologically ordered, and available to all the peers. Participants trust a blockchain system because there is a common consensus protocol which verifies and confirms each transaction before it is written into the ledger. This consensus provides every user an unalterable and updated copy of the ledger.

5. **Software agents**. The digital nature of the ledger allows that blockchain transactions can be tied to computational logic. Users can set up algorithms and rules that automatically trigger transactions between nodes. The popular name for this programs is *smart contract*.

6. **Security**. Blockchain is a very reliable technology since there is no a central failure point. A malicious attacker has to not only hack the chain and reverse the transactions, but then he would have to actually impact every single peer that has a copy of that ledger as well. It does not mean those things won't happen, but it would require extraordinary effort to control the network.

2.1.1 Blockchains classification

It is easy to divide existing blockchains into two clearly differentiated groups: *permissionless* and *permissioned*. There are also combinations of both that can be considered as *hybrid* blockchains. The following table is a classification overview based on [41][40], which is explained in detail in the following sections.

Features	**Permissionless**	**Permissioned**	**Hybrid**
Access	Public	Private	Public
Participation	Open	Closed	Closed
Decentralization	Decentralized	Distributed	Decentralized or distributed
Anonymity	Pseudoanonymous	Anonymous	Pseudoanonymous

Table 2.1: Blockchains classification table

Hybrid blockchains take the combination that better suits them. Usually hybrid blockchains require an invitation to join the network, but transactions are public so every node receives them. One example is Sovrin. It is a software ecosystem for private and secure identity management which uses open-source, distributed ledger technology.

2.1.1.1 Permissionless blockchains

The most popular permissionless blockchains are the first ones, Bitcoin and Ethereum. They were designed to be:

- **Public**. Anyone, even a non-participant, can access and consult any transaction information. Since nodes do not know the others, the blockchain strength is based on the quantity of nodes that are protecting it and the incentives they receive for fulfilling that role. The more nodes, the more secure the network is.
- **Open**. Anyone can join and take part in the network with a minimum technical knowledge.
- **Decentralized**. Every node is equally important. There is no a central entity that controls everything.
- **Pseudoanonymous**. Participants are not personally identifiable but their addresses are tracked due to its public nature.

A *token* is the exchange unit used in blockchains. It is an alphanumeric string.

2.1.1.2 Permissioned blockchains

There are many legal and confidential issues associated with the technology and this is the reason of the emergence of permissioned blockchains. Not everyone can take part but the code is still public. Permissioned blockchains are:

- **Private**. Not all data registered on the ledger has public diffusion. Just some participants can access specific data. Each node knows the others. In private blockchains participants have a compromise with the network so the more trusted and well known nodes are, the more secure the network is. It does not depend on the quantity of nodes.

- **Closed**. Just invited participants can enrol and take part in the network.
- **Distributed**. The number of nodes is limited and regulated by someone.
- **Anonymous**. Anonymity level can be established according to the purpose of the network to protect some info. Users can be (or not) perfectly identified.

2.1.2 Smart contracts

A smart contract is a piece of code that acts as a binding agreement between two or more participants without an intermediary. Its clauses are programmed previously and are self-executable, autonomous, independent and automatic. This assures the fulfilment of every clause without a third party to supervise it.

Another good definition, according to Vitalik Buterin [8], is that *"a smart contract is the simplest form of decentralized automation, and is most easily and accurately defined as follows: a smart contract is a mechanism involving digital assets and two or more parties, where some or all of the parties put assets in and assets are automatically redistributed among those parties according to a formula based on certain data that is not known at the time the contract is initiated."*

Smart contracts can be applied on public and private blockchains. There are many platforms in the private and enterprise environment such as Monax [34] or Corda [6] [24]. In the public sector Ethereum, explained in section 2.1.5.2, is specifically designed to create applications on blockchain in the broadest sense. Bitcoin, detailed in section 2.1.5.1, is incorporating new layers to implement smart contracts. Another public blockchain is NXT [36] which gives some templates that are not customizable but easy to invoke.

The life cycle of a smart contract consists on three simple steps:

- *Code.* Programming the conditions and the consequences.
- *Deploy.* Publishing it on the blockchain and storing it.
- *Call.* Executing the code. Each node has to execute it and they have to achieve a consensus about results. According to the program, the blockchain has to be updated or not.

Smart contracts can need some extra information which is not included into the blockchain.*Oracles* provides external data to the blockchain. Oraclize[1] is a service

[1]`http://www.oraclize.it`

which offers an HTTP API to get data from the outside. It serves thousands of requests every day on Ethereum, Bitcoin and Rootstock. Internally replicates an "If This Then That" logical model. This means that it will execute a given set of instructions if some other given conditions are met. This flexibility enables the engine to be leveraged in many different ways and contexts, even outside of the blockchain context. A valid request for data to Oraclize, done via the native blockchain integration or via the HTTP API, should specify the following arguments: a data source type, a query and, optionally, an authenticity proof type. The common classification of oracles distinguishes two types:

- *Inbound oracles* are reliable instructions which injects external data to be included into the blockchain. An example is an oracle in charge of accessing flights arrival information.
- *Outbound oracles* execute actions outside the blockchain. One example is an oracle in charge of emitting a refund order to the insurance company.

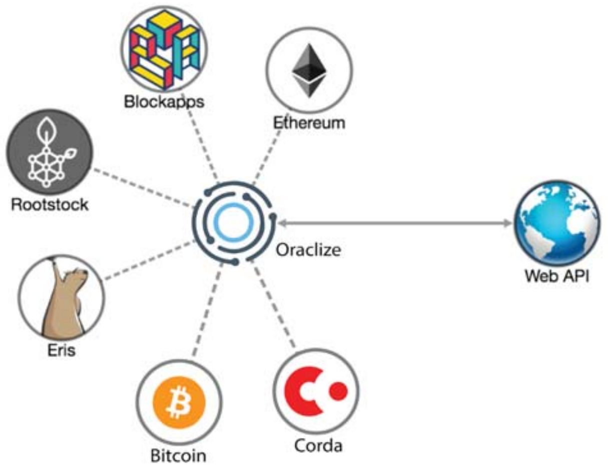


Figure 2.1: Oraclize, an oracle service

Source: `http://www.oraclize.it`

Sometimes is better to use external oracles (*partial on-chain*) than incorporating all the logic in the blockchain (*total on-chain*) because they require minor computational cost than executing a smart contract. The more complex the code is, the more expensive computationally. There is an important problem because for very complex smart contracts the CPU investment is very high. The solution for reducing costs is using different consensus models than *Proof-of-Work* like *Proof-of-Stake* that are explained in detail in section 2.1.3.

Smart contracts are not only limited by technology, which is improving fast, but by legal issues. *Internet of Things* (IoT) is another new technology which is fighting with blockchain against the traditional privacy law.

Based on this software capacity, a new type of application has emerged: *Decentralized applications* (Dapps). According to David Johnston [31] they match some properties:

- The code is open-source and any update is decided by consensus of its users. It must operate autonomously and with no entity controlling the majority of its tokens.
- Data and registers are encrypted and stored on a public decentralized blockchain.
- A cryptographic token is needed to access to the application and any contribution should be rewarded in the application's token.
- Tokens are generated when the cryptocurreny launches. Cryptographic algorithm are executed by the nodes which are rewarded with the tokens for their contribution. For example, *Proof-of-Work* in Bitcoin.

The development process of a Dapp often entails a white paper, a working prototype, an initial coin offering (ICO), its implementation and launch. The architecture of a smart contract in a Dapp has three layers to consider: the blockchain, oracles and manual intervention.

Beyond the concept, the actual importance of Dapps is the impact on the current models of applications and web. The main applications that we use daily are based on centralized business models. It is not surprising that first Dapps are alternatives to digital giants such as OpenBazaar [2] for eBay, Storj[3] for Dropbox, La'Zooz [4] for Uber or FireChat [5] for WhatsApp. Independently from who pays for the service (the final user or companies), new business models which gives value to the user generated content will be enabled. There will be new options where users can accept to share all their data, just some or even receive income as a reward for contributing with the Dapp.

The majority of Dapps are built on Ethereum. From its origin, one of its targets was to create an environment to develop blochchain-based applications. *State of*

[2]`https://www.openbazaar.org`
[3]`https://storj.io`
[4]`http://lazooz.org`
[5]`https://www.opengarden.com/about.html`

the Dapps [14] is a good repository to explore use cases and meet the developer community.

There are big expectations on smart contracts. There are infinite use cases to apply them. The most basic one is verifying an economic balance of a wallet, this is what the Ethereum trading smart contract does. In the future, the concept can be related with artificial intelligence that can make decisions according to some context information.

Surronding smart contracts, some new interesting terms are appearing [8]. In this section just three are introduced. The first one is *Decentralized Organizations* (DO). DO is an organizational model where some people interact according to a protocol defined and run on a blockchain. The second concept is *Automated or Autonomous Agents* (AA). They are machines which emulate the human brain and decision making process, they are supposed to adapt to circumstances and evolve. Currently there are simple approaches to AA. The last, which appears as a combination of the others is *Decentralized Autonomous Organization (DAO).* They are DO where decisions are made by Vitalik Buterin, Ethereum co-founder, explains a DAO as "*an entity that lives on the internet and exists autonomously, but also heavily relies on hiring individuals to perform certain tasks that the automaton itself cannot do*". His proposal scheme is shown in figure 2.2.

2.1.3 Consensus

One of the hardest problems in blockchain development is achieving effective consensus. We start always from the premise that not all the participants on the network are honest. This section exposes the main consensus algorithms to vote to verify every change and achieve the approval of the majority, in consequence, this majority must be defined.

Proof of Work

A *proof-of-work* (PoW) is achieving a piece of data which requires a selectable amount of work to compute from the service requester to produce but must be easy for others to verify. The cost is the processing time that a computer spends in solving a kind of mathematical problem so the algorithm rewards those nodes who solve the cryptographic puzzle in order to validate transactions and create new blocks(i.e. mining). The concept of PoW as an economic measure to avoid denial of service attacks or other abuses such as spam on a network was invented

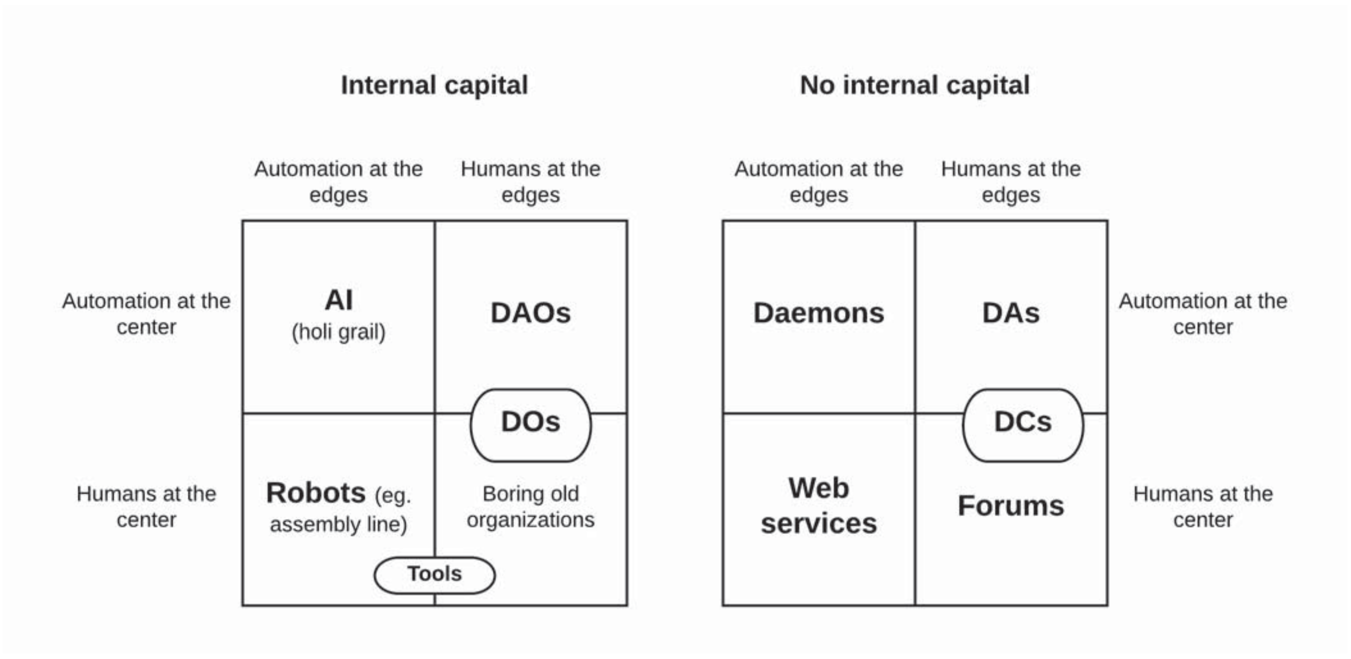


Figure 2.2: Vitalik Buterin's proposal scheme for decision and operation using humans or robots according to the business model

Source: Vitalik Buterin's article [14]

in 1993 by Cynthia Dwork and Moni Naor in their paper *Pricing via Processing or Combatting Junk Mail* [13].

In Bitcoin and Ethereum coins are mined so nodes are minners. Mining a coin is investing computing time in solving the problem. The most widely used PoW scheme is based on SHA-256 and was introduced as part of Bitcoin, *Hashcash* [2] [4]. It was invented in 1997 by Adam Back and used by SpamAssassin (Microsoft). In the case of Hashcash the problem that miners have to solve consists of hashing the message again and again, varying each time a small data called *nonce*[6], until finding a hash that matches certain conditions, in Hashcash it is asked to start with a determined number of zeros. This idea builds on a security property of cryptographic hashes: they are designed to be hard to invert (pre-image resistant property). This means that computing y from x is cheap, $y=H(x)$, but it is very hard to find x given only y. Remember that it is impossible to predict what a

[6] *Nonce* in cryptography is called an arbitrary, random or sequential number that is only used once when performing some type of operation. The only goal of the nonce in our context is to introduce a change in the content of the hash code. In this way we can make all the necessary attempts and each time we will obtain a different hash.

hash will be, so the only way to find a concrete hash is through brute-force. This pre-image resistant principle perfectly covers the need of avoiding brute-forcing creation of new hashes because it is computationally infeasible but it is very easy to verify with just a single hash invocation.

Nowadays Bitcoin works with two hash iterations (denoted SHA256 function squared) to improve security due to a partial attack that SHA1 received in 2005. This attack was based on finding an easy way to find *birthday collisions*. A birthday collision means that given one pre-image of x of hash y where y=$H(x)$, the task is to find another pre-image x' of hash y so that y=$H(x')$, as this implies $H(x)$=$H(x')$[4].

The way to regulate the quantity of new coins is to increase difficuly (costly, time-consuming). To increase the difficulty we change the objective to start the hash code with more zeros. The time needed to compute such a hash collision is exponential with the number of zero bits. So zero bits can be added (doubling the amount of time needed to compute a hash with each additional zero bit) until it is too expensive for malicious nodes to generate valid hash codes. The Bitcoin network periodically resets the difficulty level to keep the average rate of block creation at 6 per hour. In figure 2.3 [5] the blue line represents the average *block generation time* of 2016 blocks. Block generation time is also known as *confirmation time*. The grey line represents the average block generation time of 1008 blocks. If grey line is less than blue line, the generation time is decreasing. The more grey line is lower than blue line, the faster that the generation time is decreasing. After generating 2016 blocks, Bitcoin adjusts difficulty to estimated difficulty in order to keep the block generation time at 600 seconds.

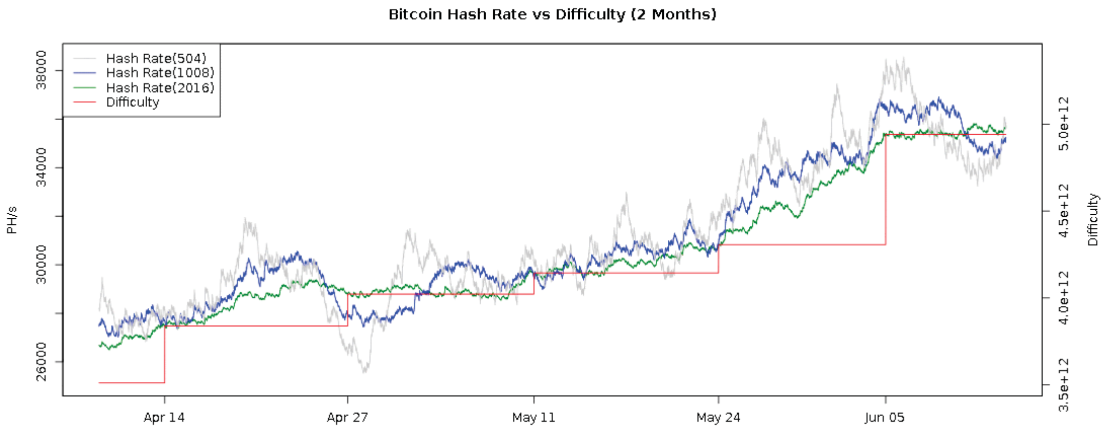


Figure 2.3: Bitcoin Block Generation Time vs Difficulty.

Source: Bitcoin Wisdom [5].

As a summary, in a traditional PoW blockchain miners *vote* on which transactions came at what time with their quantity of computational power. The more CPU power you have, the proportionately bigger your influence is.

Proof of Stake

Proof-of-Stake (PoS) is another type of algorithm to achieve distributed consensus. The more percentage of the issued tokens you have, the more weight you have on the votation. So the algorithm depends on a validator's economic stake in the network. In a PoS blockchain based a set of validators take turns proposing and voting on the next block, and the weight of each validator's vote depends on the size of its deposit.

In general, a PoS algorithm works as follows [9]: The blockchain keeps track of a set of validators, and anyone who holds the blockchain's base cryptocurrency (for example in Ethereum's case, ether) can become a validator by sending a special type of transaction that locks up their ether into a deposit. The process of creating and agreeing to new blocks is then done through a consensus algorithm that all current validators can participate in.

There are mainly two major types from the algorithmic perspective: *chain-based* and *Bizantine Fault Tolerance style* proof of stake (Byzantine's Fault Tolerance (BFT)).

- In **chain-based** proof of stake, the algorithm via various random combination selects a validator during each time slot (eg. every period of 10 seconds), and assigns that validator the right to create a single block, and this block must point to some previous block (normally the block at the end of the previously longest chain), and so over time most blocks converge into a single constantly growing chain. One of the most known algorithm of this category is **Proof-of-deposit**.

- In **BFT-style** proof of stake, validators are randomly assigned the right to *propose blocks*, but *agreeing on which block is canonical* is done through a multi-round process where every validator sends a "vote" for some specific block during each round, and at the end of the process all (honest and online) validators permanently agree on whether or not any given block is part of the chain. Note that blocks may still be *chained together*; the key difference is that consensus on a block can come within one block, and does not depend on the length or size of the chain after it. Some of the most popular algorithms of this type are **Simple Bizantine Fault Tolerance** (SBFT), **Redundant Byzantine Fault Tolerance** (RBFT) and **Sumeragi** [26] [10].

In a PoS cryptocurrency the probability of finding a block of transactions - and receiving the corresponding reward - is proportional to the quantity of accumulated coins. Its strategy is based on the supposition that nodes who have more coins are specially interested in the network survival and good performance. This is the reason to make them responsible to protect it from possible attacks. The protocol rewards them reducing the difficulty to find blocks (inversely proportional to the number of coins that they demonstrate to own).

Proof of Burn

The idea is that miners should show proof that they burned some coins. This means sending them to a verifiably unspendable address. This is expensive from their individual point of view, just like proof of work. The advantage is that it consumes no resources other than the burned underlying asset. To date, all proof of burn cryptocurrencies work by burning PoW-mined cryptocurrencies. The most extended algorithm of PoB is the Iain Stewards'implementation in Slimcoin [44] which is another cryptocurrency.

Proof of Elapsed Time

It is a lottery-like algorithm used by Hyperledger Sawtooth (More detail in section 2.1.5.3) that leverages Intel's Software Guard Extensions (SGX) to implement a leader-election lottery system. Instead of using computational effort to solve cryptographic puzzles (like Bitcoin), the system uses a trusted execution environment to generate random wait times—potentially a much more energy-efficient approach. [42]

Its strategy is that each participant in the blockchain waits a random amount of time and the first participant who finishes the waiting time, becomes the leader of the new found block. Proof of Elapsed Time (PoET) comes from Intel, and it relies on SGX, a special CPU instruction set. It allows applications to run trusted code in a protected environment. In this case, for PoET the trusted execution ensures that the lottery winner waited a random amount of time.

The details of the protocol are complex but here is a simplified outline [32] :

Joining the network:

- A new participant downloads the trusted code for the blockchain.
- On initialization, the trusted code creates a new key par.
- Participant sends a SGX attestation to the rest of the network as a part of a join request.

Participating in the lottery:

- Participant obtains a signed timer object from the trusted code.
- Participant waits for the time specified by the timer object.
- Participant obtains a certificate (signed by the trusted code's private key) that the timer has completed. Participant sends this certificate to the rest of the network along with the new block for the blockchain.

2.1.4 Security on the blockchain

As blockchain is still not a mature technology, it is an active investigation area. There are some conceptual problems that have different solutions according to the consensus algorithm and the network features. The main weakspot of a blockahin is the **double spending** attack. It means spending the same money twice. As tokens are digital and its identifiers are just hash codes, trying to reuse them more than once. Bitcoin's blockchain keeps a chronologically-ordered, time-stamped transaction ledger from the very start to avoid this problem.

Consensus is a fundamental problem of distributed computing. The distributed set of participants guarantee its consistency despite potentially malicious participants that behave arbitrarily, also called ***Byzantine*** **failures** [21].There are some possible attacks that can still happen:

- **51% attack**. If a miner achieves more than a half of the hashing power in a Proof of Stake (PoS) blockchain, he has the total control of the network forever. Even if a new participant would appear, mining rewards and transactions, this malicious miner can get it and still would continue to grow. It does not happen on a PoW blochcain because a new participant with more computational power can suddenly appear and recover the control.

- **Race Attack**. When an attacker sends the same coin in rapid succession to two different addresses, the obvious outcome is that only one of them will get included. If the merchant don't wait for confirmations of payment, then in a case like this, there's a 50% chance he got the double spent coin and won't receive that money. If the attacker gets the confirmations first, then the merchant won't receive his funds. That's why it is suggested to wait for a minimum of 6 posterior confirmed transactions to check that the participant didn't receive falsified tokens.

2.1.5 Distributed Ledger Technologies

2.1.5.1 Bitcoin

Bitcoin is a cryptocurrency, a digital asset designed to work as a medium of exchange that uses cryptography to control its creation and management, rather than relying on central authorities. It uses peer-to-peer technology to operate avoiding any central authority or banks. This protocol allows full nodes (*peers*) to collaboratively maintain a peer-to-peer network to exchange Bitcoins and assets.

Bitcoin was introduced by Satoshi Nakamoto [7] in October 2008 in the paper *Bitcoin:A Peer-to-Peer Electronic Cash System* [35]. This paper detailed methods of using a peer-to-peer network to generate what was described as "a system for electronic transactions without relying on trust". On January 2009 the genesis block of Bitcoin - block number 0 - was mined by Satoshi Nakamoto and had a reward of 50 bitcoins. There is a Bitcoin Foundation which works as its official public face where developers can collaborate.

Money can be exchanged without being linked to a real identity. Unless someone chooses to link their name to a bitcoin address, it is hard to tell who owns the address. A bitcoin address is an identifier of 26-35 letters and numbers. They are generated with no fees by any user of Bitcoin. There are currently two address formats in common use: *Common P2PKH* (which begins with number 1) and newer *P2SH* type (which starts with number 3). Bitcoin does not keep track of users; it keeps track of addresses where the money is. Each address has two important pieces of cryptographic informations: a public and a private key. The public key, which is the Bitcoin wallet's address, can be looked up by everyone and

[7]Satoshi Nakamoto is presumed to be a pseudonym for the person or people who designed the original bitcoin protocol in 2008 and launched the network in 2009. Nakamoto was responsible for creating the majority of the official bitcoin software and was active in making modifications and posting technical information on the Bitcoin Forum.

everybody can send bitcoins to it. The private address, or private key, is known just by the owner to send bitcoins from it. Because of this, it is very important that this private key is kept in secret. To send bitcoins from an address, you prove to the network that you own the private key that corresponds to the address, without revealing the private key.

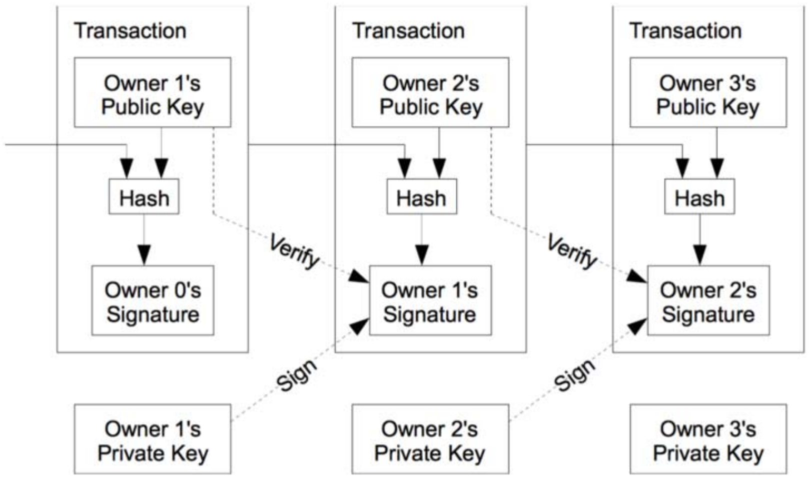


Figure 2.4: Bitcoin defines an electronic coin as a chain of digital signatures to verify ownership.

Source:
`https://medium.com/crypto-currently/what-exactly-is-bitcoin-3d5417bff390`

There are three basic components on the network: *wallets*, *nodes* and *miners*.

- **Wallets**. They generate transactions, sign and send them to the nodes. Transactions are validated and added to a waiting queue (*mempool*) available for miners.
- **Nodes**. They validate blocks and transactions and update the ledger. There are different kind of nodes and some can be miners. **Full nodes** download and verify every block and transaction prior to relaying them to other nodes. **Archival nodes** are full nodes which store the entire blockchain and can serve historical blocks to other nodes. **Pruned nodes** are full nodes which do not store the entire blockchain. There is a method, called **Simplified Payment Verification** (SPV) for verifying if particular transactions are included in a block without downloading the entire block. The method is used by these lightweight Bitcoin clients. At first, peers need to discover their neighbours to connect themselves.

- **Miners**. They generate the blocks of the chain. Firstly, they choose some transactions from those that are already on the waiting list. Then, they obtain the *Merkle tree* of the chosen ones. The *root hash* of that Merkle tree is used as part of the message to hash with the Hashcash algorithm. There is a miner who manages to find a valid hash faster than the others. It communicates it to the other nodes, who verify and update the blockchain. Only then, miners restart their processes and start generating the next block encouraged by the new bitcoins reward that they will obtain if they find the hash required by the difficulty level.

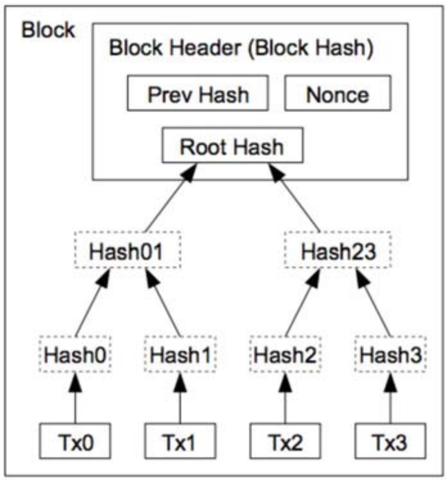


Figure 2.5: Bitcoin Merkle Tree

Source: `https://bitcoin.stackexchange.com/questions/30329/what-shape-of-merkle-tree-does-the-bitcoin-client-build`

An important point is what happens when a miner discovers a new block. It broadcasts the new block to its peers using one of the following methods [3]:

- **Unsolicited Block Push**. The miner sends a block message to each of its full node peers with the new block. The miner can reasonably bypass the standard relay method in this way because it knows none of its peers already have the just-discovered block.
- **Standard Block Relay**. The miner, acting as a standard relay node, sends an inv message to each of its peers (both full node and SPV) with an inventory referring to the new block. The most common responses are:
  - Each *blocks-first* (BF) peer that wants the block replies with a getdata message requesting the full block.

  - Each *headers-first* (HF) peer that wants the block replies with a getheaders message containing the header hash of the highest-height header on its best header chain, and likely also some headers further back on the best header chain to allow fork detection. That message is immediately followed by a getdata message requesting the full block. By requesting headers first, a headers-first peer can refuse orphan blocks as described in the subsection below.
  - Each *Simplified Payment Verification* (SPV) client that wants the block replies with a getdata message typically requesting a merkle block.

  The miner replies to each request accordingly by sending the block in a block message, one or more headers in a headers message, or the merkle block and transactions relative to the SPV client's bloom filter in a `merkleblock` message followed by zero or more transactions messages.

- **Direct Headers Announcement**. A relay node may skip the round trip overhead of an inv message followed by `getheaders` by instead immediately sending a headers message containing the full header of the new block. A HF peer receiving this message will partially validate the block header as it would during headers-first IBD, then request the full block contents with a `getdata` message if the header is valid. The relay node then responds to the `getdata` request with the full or filtered block data in a block or `merkleblock` message, respectively. A HF node may signal that it prefers to receive headers instead of `inv` announcements by sending a special `sendheaders` message during the connection handshake.

If two different miners find a valid hash at the same time, there will be two concurrency chains. When a node receive two chains, the "real" will be the longest because it is backed or voted by the majority as in the case of the popular *Byzantine generals* in the dilemma solved by Satoshi.

Tokens value is not determined by miners, nodes or wallets, it depends on the market movements. Investors can sell and buy them depending on external factors. This means that market rules can change. An update on the protocol is decisive for any participant so it can be done just with the majority approval. At this point developers would publish the new software which will not work until it achieves an important percentage acceptance. When a developer has a proposal, he presents it to the community through a document called *Bitcoin Improvement Proposal* (BIP). BIPs are enumerate secuencially and are discussed in public online forums. Depending on the impact on the networt *forks* (which are software bifurcations that are incompatible and produce a new chain) can be classified into *softforks* and *hardforks*[41].

- **Softfork**. It is a temporary split of the chain that is produced by the activation of more restrictive rules. New software validates part of the blocks created with the old software and ignores the rest. Old nodes will continue to validate all the blocks. Miners who work with outdated nodes will be building a weaker chain. In this case what matters is that the updated miners are the majority. This type of fork is not problematic. As we have seen, the changes are activated when it is already accepted by the majority of miners and it incentives the rapid adoption by the rest. But if it were the case of a premature activation without the majority, the fork would behave like a hardfork and two conflicting chains would be created.

- **Hardfork**. It is a definitive bifurcation of the chain that is produced by any relaxation in the rules that makes valid some blocks that in the previous version would be rejected. The updated nodes will validate all blocks, but those who have not accepted the changes will not be able to validate the new blocks. Therefore, unlike the softfork, it is required that all the nodes, and not only the miners, get updated. This type of fork should be avoided whenever possible, since it would create two valid chains that exclude each other. To successfully perform a hardfork, all users, nodes and miners must have been updated before.

2.1.5.2 Ethereum

Ethereum [7] is an alternative protocol for building decentralized applications. It particularly emphasizes on situations where rapid development time, security for small and rarely used applications, and efficiently interaction, are important. Ethereum can do this because its ultimate abstract foundational layer is a blockchain built-in Turing-complete programming language. This means that is fully customizable, everyone can create smart contracts, its own rules and transaction formats and functions.

It was created in 2013 by Vitalik Buterin, a Russian young man who was disappointed with having central authorities when playing a computer game, *World of Warcraft*. He designed Ethereum, which is still improving, and it is an active investigator. He has written many publications many detailed publications about its advances.

The philosophy behind Ethereum is based on the following principles: simplicity, universality, modularity, agility, non-discrimination and non-censorship.

In Ethereum, the state is made up of objects called *accounts*. Each account has a 20-byte address and state transitions, which are direct transfers of value and information between accounts. An Ethereum account contains four fields: a nonce, the current ether balance, the contract code (if present) and the account's storage (empty by default). There are two types of accounts: *externally owned accounts*, controlled by private keys, and *contract accounts*, controlled by their contract code. In a contract account, every time the contract account receives a transaction, its code executes. A contract, as it was explained in 2.1.2 and figure 2.2, is an "autonomous agent" that lives inside of the Ethereum execution environment, always executing a specific piece of code when triggered by a message or transaction, and having direct control over its own ether balance and its own key/value store to keep track of persistent variables.

Transactions are signed data packets that store a message to be sent from an externally owned account. Each transaction contains: the recipient of the message, a signature identifying the sender, the amount of ether to transfer from the sender to the recipient, an optional data field, a STARTGAS value - representing the maximum number of computational steps the transaction execution is allowed to - and a GASPRICE value - representing the fee the sender pays per computational step -. Contracts have the ability to send "messages" to other contracts. *Messages* are virtual objects that are never serialized and exist only in the Ethereum execution environment. Essentially, a message is like a transaction, except it is produced by a contract and not an external actor.

The code in Ethereum contracts is written in Solidity, which is compiled to a stack-based bytecode language, referred to as Ethereum Virtual Machine (EVM) code. The EVM is made of three types of memory space:

- The stack, a last-in-first-out container to which values can be pushed and popped.
- Memory, an infinitely expandable byte array.
- The contract's long-term storage, a key/value storage. Unlike stack and memory, which reset after computation ends, storage persists for the long term.

Ethereum blocks contain a copy of both the transaction list and the most recent state. Aside from that, two other values, the block number and the difficulty, are stored in the block. This is the main difference between Ethereum and Bitcoin.

A block validation consists on checking: the previous block reference, the timestamp, the block number, its difficulty, its transaction root (Merkle Tree), its uncle root, the gas limit and the PoW. Afterwords it adds the block and rewards the miner.

In general, there are three types of applications on top of Ethereum. The first category is financial applications, providing users with more powerful ways of managing and entering into contracts using their money. This includes sub-currencies, financial derivatives, hedging contracts, savings wallets, wills, and ultimately even some classes of full-scale employment contracts. The second category is semi-financial applications, where money is involved but there is also a heavy non-monetary side to what is being done. Finally, there are applications such as online voting and decentralized governance that are not financial at all.

One common concern about Ethereum is the issue of scalability. Like Bitcoin, Ethereum suffers from the flaw that every transaction needs to be processed by every node in the network.

2.1.5.3 Hyperledger

Hyperledger [8] is a collaborative open source project created to advance cross-industry blockchain technologies. It is a global collaboration, hosted by The Linux Foundation, including leaders in finance, banking, Internet of Things, supply chains, manufacturing and Technology. Apache Software Foundation co-founder Brian Behlendorf describes the vision for Hyperledger as "*a business blockchain umbrella*".

It was launched in 2016 with a technical and organizational governance structure and 30 founding corporate members. Initially, the Hyperledger Technical Steering Committee started two business blockchain framework codebases into development: Hyperledger Fabric, a codebase combining work by Digital Asset, libconsensus from Blockstream and OpenBlockchain from IBM; and Hyperledger Sawtooth, developed at Intel's incubation group. Since 2016 the project has added seven more business blockchain codebases.

The characteristic elements of a Hyperledger blockchain project are:

- **Channels**. They are data partitioning mechanisms that let transaction visibility for stakeholders only.
- **Chaincode**. It encapsulates both the asset definitions and the business logic (Smart Contracts).

[8] `www.hyperledger.org`

- **Ledger**. It contains the current world state of the network and a chain of transaction invocations.
- **Network**. Collection of data processing peers which consistently replicate these data.
- **Ordering service**. Collection of nodes that orders transactions into a block.
- **World state**. It reflects the current data about all the assets in the network (LevelDB and CouchDB)
- **Membership service provider** (MSP). It manages identity and access for clients and peers.

Hyperledger incubates and promotes a range of business blockchain technologies which are briefly exposed in this section. It includes distributed ledger frameworks, smart contract engines, client libraries, graphical interfaces, utility libraries and sample applications. The Hyperledger umbrella strategy encourages the re-use of common building blocks and enables rapid innovation of DLT components.

Hyperledger Frameworks

- **Sawtooth**. It is a modular platform for building, deploying, and running distributed ledgers. Hyperledger Sawtooth includes a novel consensus algorithm, PoET, which targets large distributed validator populations with minimal resource consumption. There are many supply chain use cases built on Sawtooth because it can provide physical traceability in a trustless world through its chances with IoT sensors.
- **Iroha**. It is a business blockchain framework designed to be simple and easy to incorporate into infrastructural projects requiring distributed ledger technology. It is focused on mobile applications development and other user-facing apps. It provides iOS, Android, Scala, Python and Javascript libraries. Its consensus algorithm is Sumeragi (YAC).
- **Fabric**. Intended as a foundation for developing applications or solutions with a modular architecture, Hyperledger Fabric lets components, such as consensus and membership services, to be plug-and-play.
- **Burrow**. It is a permissionable smart contract machine. The first of its kind when released in December, 2014, Burrow provides a modular blockchain client with a permissioned smart contract interpreter built in part to the specification of the EVM.

- **Indy**. It is a distributed ledger, purpose-built for decentralized identity. It provides tools, libraries, and reusable components for creating and using independent digital identities rooted on blockchains or other distributed ledgers for interoperability.

Hyperledger Tools

- **Caliper**. It is a blockchain benchmark tool, which lets users to measure the performance of a specific blockchain implementation with a set of predefined use cases.
- **Cello**. It aims to bring the on-demand "as-a-service" (Blockchain-as-a-Service, BaaS) deployment model to the blockchain ecosystem to reduce the effort required for creating, managing and terminating blockchains.
- **Composer**. It is a collaboration tool for building blockchain business networks, accelerating the development of smart contracts and their deployment across a distributed ledger. Developers can program smart contracts in Javascript and use its REST API.
- **Explorer**. It can view, invoke, deploy or query blocks, transactions and associated data, network information, chain codes and transaction families, as well as any other relevant information stored in the ledger.
- **Quilt**. It offers interoperability between ledger systems by implementing ILP, which is primarily a payments protocol and is designed to transfer value across distributed ledgers and non-distributed ledgers.

2.1.5.4 Tangle

The Tangle is a novel new distributed ledger architecture that is based on a Directed Acyclic Graph (DAG). One might refer to it as a "blockchain without blocks and the chain" (semantically, it is not really a blockchain) [12]. It follows some important principles of blockchain because it is still a distributed database, a P2P network and relies on a consensus and validation mechanism.

The two most apparent differences between Tangle and blockchain are how the Tangle is structured (a DAG), and how it achieves consensus. In IOTA there are no "blocks" in the classical sense. Instead, a single transaction references two past transactions. This referencing of transactions is seen as an attestation: with your transaction you attest directly that two transactions, and indirectly that a subsection of the Tangle are valid and conform to the protocols rules.

Tangle transactions are processed in parallel. This feature makes IOTA to achieve high transaction throughput. As the Tangle grows with more transactions, IOTA becomes faster and more secure as long as there is an optimum number of nodes. A transaction confirmation lets for asynchronous settlement, it means that there are not temporal restraints on transaction finalization. The reason is because the Tangle miners and users are no longer decoupled entities. It does not use a conventional consensus algorithm which needs a certain arbitrary time interval for a block creation. Another reason is that Tangle is programmed in ternary with is an alternative from the traditional binary code and it is much more efficient than binary.

David Sønstebø, Sergey Ivancheglo, Dominik Schiener, and Dr. Serguei Popov created IOTA in 2015. The IOTA Foundation is a non-profit foundation located in Germany. IOTA is a new public cryptocurrency that utilizes Tangle as its core. It has a very clear and focused vision of enabling the paradigm shift of the Internet of Things and Industry 4.0.

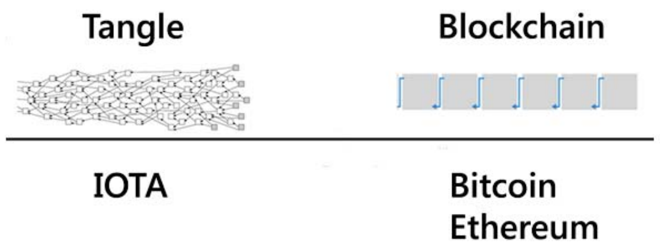


Figure 2.6: IOTA is a Tangle-based cryptocurrency

Source:
`https://www.allcrypto.com/analysis/iota-hoping-become-backbone-iot/`

To make a transaction in IOTA there is a simple process before broadcasting it to the network.

1. **Signing**. The transaction inputs are signed with private keys.

2. **Tip selection**. Two tips (i.e. unconfirmed transactions) are selected using Markov chain Monte Carlo (MCMC). These transactions will be referenced by your transaction (branchTransaction and trunkTransaction)

3. **Proof of Work**. The network accept a transaction after the sender node does some PoW.

IOTA has no transaction fees, scales and allows partitioning which means that anyone can fluidly branch off and back into the network as figure 2.7 shows.

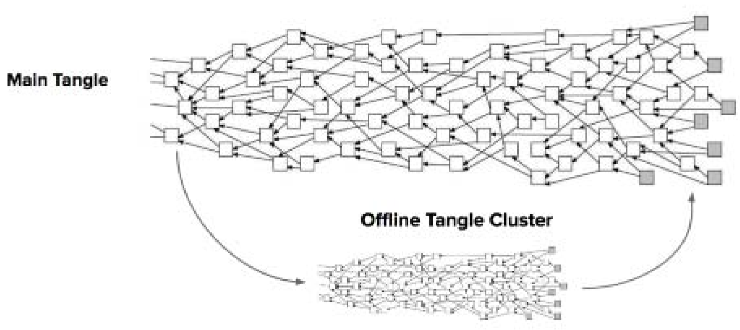


Figure 2.7: Offline transactions (partitioning)

Source:
`https://blog.iota.org/a-primer-on-iota-with-presentation-e0a6eb2cc621`

It has many components developed in different languages that enable developers to choose their most preferred client suited for their specific deployments. Currently they are working on:

- Java. The IOTA Reference Implementation (IRI) which is the first official IOTA implementation is written in Java.
- C++. Unlike Java, it does not require a Virtual Machine installed on the devices in order to run code and it compiles to machine code in real time, making it a lot more suitable for these resource restrained IoT devices.
- Rust is considered to be one of the most promising and agile languages, particularly for IoT due to its safety, fast execution and massive concurrency support.
- Go is very similar to Rust in terms of concurrency. Various Blockchain projects are using it.

2.2 Decentralized Data Marketplaces

In this section it is evaluated the State of the Art regarding commercial solutions that implement decentralized data marketplaces. Concretely, existing commercial solutions have been divided into four main categories based on their data distribution model: *Blockchain*, *Cloud Services*, *Sharding*, and *Peer to Peer*. The majority of decentralized marketplaces have their own token. Many of them just support simple pricing models so they distribute data using blockchain because timing to get paid does not matter. In this section there is a brief explanation of the most representative solutions of each case.

2.2.1 Blockchain-based data distribution marketplaces

OpenBazaar

OpenBazaar[9] is a store for digital goods, not just data. It's a peer to peer application that doesn't require middlemen to do online commerce. Data is distributed across the network instead of storing it in a central database. It is built in Bitcoin Blockchain. Nobody has control over OpenBazaar, each user contributes to the network equally and is in control of their own store and private data. It integrates for payments. It is permissionless trade.

OpenBazaar offers a unique feature of Bitcoin that helps preventing fraud: Multi-signature escrow. In this e-commerce marketplace buyers and sellers must agree to trust a third party before they start the trade, and then the buyer sends the payment in Bitcoins to an escrow account. Those bitcoins can only be released when two out of three parties agree the destination wallet. Normally the buyer and seller are the two parties in agreement, but if there is a dispute the third party will settle the dispute. These third parties which offer dispute resolution are selected in an open marketplace.

It is important to choose a good moderator that is trustworthy for both parties on a decentralized network. Its way to prevent identity theft and fraud is using Bitcoin and multi-signing.

[9]`https://www.openbazaar.org`

Datum

It is a decentralized marketplace for social and IoT data[10]. It is powered by Ethereum, BigchainDB and IPFS. It is a decentralized and distributed high performance NoSQL database backed by a blockchain ledger. This technology lets anyone to securely and anonymously backup structured data from social networks, wearables, smart homes, and other IoT devices. Datum provides a marketplace where users can share or sell data on their own terms. The DAT token facilitates transactions in the Datum network, providing value to the data on the network.

IOTA Data Marketplace

This data marketplace uses the IOTA mainnet to securely store, sell and access data streams[11]. It lets the user to choose a sensor from a map view. For each of them, the user can purchase directly thanks to its micropayments feature. From this moment his applications and data analysis tools can access and use the data of the sensor. Advantages and disadvantages of involved technologies are discussed in chapter 5.

2.2.2 Cloud Services

Datareum

The Datareum[12] platform is made up of a few key components: *requesters* ask for data, and they may commission a survey directly or purchase survey results that have already been resold on the data marketplace. *Providers* are those who give away data for free or sell it, though they may previously be requesters who are providing data through resale.

The *Data Marketplace* is where smart contracts determine the legitimacy of data entered into the platform, and it also confirms whether or not the data is delivered of rejected and if so it completes the exchange or payment between the two parties. The *DataVault* is a secure location in which data providers can enter all their basic information and therefore can be monetized according to the user's

[10]`https://datum.org`
[11]`https://data.iota.org`
[12]`https://www.datareum.net/`

preference.

2.2.3 Sharding-based

Enigma

It is a decentralized computation platform with guaranteed privacy. It is private because uses secure multi-party computation (sMPC or MPC) [22] , data queries are computed in a distributed way, without a trusted third party. Data is split between different nodes, and they compute functions together without leaking information to other nodes. Specifically, no single party ever has access to data in its entirety; instead, every party has a meaningless piece of it.Enigma is also scalable. Unlike blockchains, computations and data storage are not replicated by every node in the network. Only a small subset perform each computation over different parts of the data. The decreased redundancy in storage and computations enables more demanding computations.

The key new utility Enigma brings to the table is the ability to run computations on data, without having access to the raw data itself. For example, a group of people can provide access to their salary, and together compute the average wage of the group. Each participant learns their relative position in the group, but learns nothing about other members' salaries. In practice, any program can be securely evaluated while maintaining the inputs as a secret.

Today, sharing data is an irreversible process; once it is sent, there is no way to take it back or limit how it is used. Allowing access to data for secure computations is reversible and controllable, since no one but the original data owner ever sees the raw data. This presents a fundamental change in current approaches to data analysis.

2.2.4 Peer to Peer

Wibson

It is a blockchain-based, decentralized data marketplace that provides the infrastructure for individuals to securely and anonymously sell private information that is validated for accuracy [46]. Wibson is built up on a set of core principles: transparency, anonymity, fairness, censorship resistant, and the individual's ultimate control over the use of their personal information.

It proposes a new market-based approach that leverages the latest developments in blockchain, cryptography and market design to level the playing field between data consumers (companies, organizations) and data owners (essentially, individuals).

In the Wibson marketplace, citizens, for the first time, will be able to participate in an efficiently functioning, decentralized data marketplace that provides both financial incentives and control over their personal information, all without sacrificing privacy.

2.3 FIWARE access data technologies

This section contains a brief introduction about FIWARE and describes the main technologies that are involved in the management of data access in the FIWARE Business Framework.

2.3.1 CKAN

Comprehensive Knowledge Archive Network (CKAN) is a web based, open-source, free software management system used for the storage and distribution of open data aimed at data publishers wanting to make their data public. CKAN is mainly used by public institutions to share their data with the general public.It is one of the most extended Open Data publication platforms and is becoming the de facto standard for data publication in Europe.

CKAN is designed for publishing and managing data either through the web user interface or through the API, and it has a modular architecture, allowing plugins to add custom features. CKAN can be extended with the use of a WireCloud extension that can use WireCloud to visualize its data, by creating a dashboard and embedding it on CKAN. In this regard, the provided extensions integrate CKAN with the FIWARE platform, enabling the right-time context information served by a context broker to be published as dataset resources, making it easier to be discovered and consumed. In addition, the integration of the FIWARE Security and Business Frameworks provide an enriched access control and enable explicit acceptance of data terms and conditions, usage accounting, or data monetization.

There is a GE called FIWARE CKAN Extensions [18]. In particular, the following extensions are being provided:

- **OAuth2**. This extension enables the integration of CKAN with an external OAuth2 Identity Manager (e.g The FIWARE IdM)
- **Private Datasets**. This extension improves the default dataset permissions features provided by CKAN by enabling to create protected datasets which can be accessed by a set of users included in an access list.
- **NGSI View**. This extension enables the publication of right-time context data as dataset resources by allowing to define them as NGSI v2 queries to a Context Broker.
- **BAE Publisher**. This extension simplifies the monetization of private datasets by enabling the creation of products and offerings in the Business API Ecosystem GE directly using the dataset information.
- **WireCloud View**. This extension lets embedding WireCloud (Application Mashup GEri) dashboards as dataset resources views, enabling the creation of rich and highly customizable visualizations for the published data.
- **Data Requests**. This extension includes a new section in CKAN intended to let users of the platform to ask for data which is not yet published.

2.3.2 NGSI

The FIWARE NGSI (Next Generation Services Interface) is a REST API intended to manage the entire lifecycle of context information, such as updates, queries and subscriptions.

The NGSI API defines a data model for context information, a context data interface used for exchanging information using queries, subscriptions and update operations, and a context availability interface, used to obtain information about the context data interface.

The main elements of the NGSI data model are entities, attributes and metadata. Each entity has an ID field and a type field (used to describe the type of thing represented by the entity); the combination of both fields defines the entity. Each entity has a set of attributes, that have a name, a type and a value. Attributes can have meta-data, which provides additional information about the attribute.

The three main interaction types with the NGSI API are one-time queries for context information, subscriptions to receive context information updates and unsolicited updates that get invoked by the context providers.

2.3.3 Orion Context Broker

The Orion Context Broker [38] is an implementation of the Publish/Subscribe Context Broker GE, providing NGSI interfaces. This means that it is the implementation of the FIWARE NGSI API that offers the creation of context elements and to manage context elements through updates, queries and subscriptions. Using these interfaces, clients can do several operations:

- Register context producer applications, e.g. a temperature sensor within a room
- Update context information, e.g. send updates of temperature
- Being notified when changes on context information take place (e.g. the temperature has changed) or with a given frequency (e.g. get the temperature each minute)
- Query context information. The Orion Context Broker stores context information updated from applications, so queries are resolved based on that information.

2.3.4 Business API Ecosystem

The Business API Ecosystem GE [16] is the result of the collaboration between FIWARE and the TMForum. In this regard, the Business API Ecosystem GE is a joint component made up by integrating the FIWARE Business Framework with a set of standard APIs (and its reference implementations) provided by the TMForum in its TMF API ecosystem.

This component allows the monetization of different kind of assets (both digital and physical) during the whole service life cycle, from offering creation to its charging, accounting and revenue settlement and sharing. In this way, the Business API Ecosystem provides sellers the means for managing, publishing, and generating revenue from their products, apps, data, and services.

By using the Business API Ecosystem you have access to the following key features:

- Support for the management of catalogues, products, and offering.
- Support for rich pricing models, including recurring payments, pay-per-use, etc.

- Support for accounting callbacks.
- Support for billing and charging.
- Integrated support for PayPal, including customer charges and seller payments.
- Support for revenue sharing, including models with multiple stakeholders involved.

The Business API Ecosystem exposes its complete functionality through TMForum standard APIs; concretely, it includes the catalog management, ordering management, inventory management, usage management, billing, customer, and party APIs.

There is an **API Umbrella PEP** [1](Police Enforcement Point) which provides some functionalities like API keys, rate limiting, analytics and caching.

3 Marketplace decentralization scenarios

The difference between a mountain
and a molehill is your perspective.

Al Neuharth

To introduce this chapter, it is important to highlight that the architecture of a marketplace fully depends on the pricing models it supports. There is no unique solution which covers all the needs for every marketplace. This section propounds some theoretical intermediate scenarios that have been elaborated as part of this Master's Thesis, starting from the current centralized marketplace proposed by FIWARE to a fully decentralized one that supports the more complex pricing models which requires accounting (See table 3.1).

The components that usually take part in every data marketplace are the following ones:

- **Dataset**. It is the digital asset which is bought and sold. It is a collection of information which lists values of some variables for each member of the dataset. Data is becoming very important with Internet of Things because datasets are not just historic data any more, nowadays sensors are collecting information and we are interested in processing it in streaming. FIWARE

Data Models enable data portability for different applications including Smart Cities. They are writed to be used together with FIWARE NGSI version 2 [17].

```
[
  {
    "id": "poi",
    "type": "PointOfInterest",
    "category": {
      "type": "List",
      "value": [
        "439"
      ],
      "metadata": {}
    },
    "description": {
      "type": "Text",
      "value": "Palazzo Massimo alle Terme located
                             in Largo di Villa Peretti street.",
      "metadata": {}
    },
    "location": {
      "type": "geo:json",
      "value": {
        "type": "Point",
        "coordinates": [
          12.498236,
          41.901367
        ]
      },
      "metadata": {}
    },
    "name": {
      "type": "Text",
      "value": "Palazzo Massimo alle Terme",
      "metadata": {}
    }
  },
}
```

- **Customer**. It is the entity (person or organization) which is interested in the information. The customer pays for accessing the information published in a dataset. He can search and filter the available offerings and create an order for acquiring them. Each customer can manage his inventory where the acquired products and the requested orders are stored.

- **Seller**. It is the owner of the dataset. The seller publishes its datasets with different pricing models in the catalogue of the marketplace. He manages the catalog, the product specification and offering. There are different actions that can be carried out by a seller for managing its received product orders.
- **Admin**. This role is intended for administrators of the Business API Ecosystem (BAE). They install the software and the plug-ins. An admin can list, create and update product categories. He also manages asset plugins.

Currently the marketplace is centralized. It means that sellers offer their dataset and contractual clauses through the BAE. Customers manifest their interest using the centralized platform. The marketplace has a *Police Enforcement Point* (PEP) - in our case it is deployed with an API Umbrella - which manages access permissions and monitors queries. Our marketplace is built on the BAE GE which is explained in section 2.3.4. The BAE is a marketplace of apps, data and services. A marketplace is one among so many use cases where blockchain can disrupt the current industry. If we consider that CKAN datasets, WireCloud components, accountable services, etc. as assets, then we can talk about a platform that enables users to create and exchange digital assets on a blockchain.

There are two FIWARE components that seem likely to improve or be complemented if they use a distributed ledger instead of a centralized database. They are the BAE and the Orion Context Broker. The main reasons for this proposal are:

1. It needs a shared common database.
2. It has multiple parties involved.
3. These parties have conflicting interests and are not trusted.
4. It has different rules to govern participants.
5. It needs an objective and immutable log.
6. Rules and transactions do not change frequently.
7. Transactions can be public or not.

The Business API Ecosystem supports different pricing plans. The available types are: *one time* for payments that are made once when purchasing the offering, *recurring* for charges that are made periodically (e.g a monthly payment), and *usage* for charges that are calculated applying the pricing model to the actual usage made of the acquired service. Tracking all the queries from each node to a dataset in an objective and immutable log is very interesting in order to support usage payments. The concept of smart contract is also important because different actions can take place depending on the asset type and the conditions of the participants.

The FIWARE marketplace also provides *revenue sharing* (RS) models which specify how the revenues generated by an offering or set of offerings must be distributed between, the owner of the Business API Ecosystem instance, the provider of the offering and the related stakeholders involved [19]. These models are trust-based and blockchain precisely ensures data integrity.

The principal advantages provided by the blockchain marketplace transaction tracking are:

- Consistent data record across different sellers.
- Immutable transaction record enables security and trust between parties.
- Smart contracts let asset creators to set terms for how secondary transactions occur.

Once the starting scenario is described, the following sections deepen in the intermediate steps until the fully decentralization of the BAE and the distribution data channel which are shown in table 3.1. For each of them advantages and disadvantages are discussed. As can be seen not all the combinations are covered because it makes no sense a blockchain which is not auditable. All the proofs of concept are detailed in chapter 4.

3.1 Scenario 1. Decentralized catalogue and P2P data distribution

This scenario decentralizes the BAE but not the Orion Context Broker. This first step to decentralize the marketplace consists on publishing the product offerings catalog in a ledger instead of deploying it on the cloud. Agreements between sellers

		Scenarios				
		1	2	3	4	5
Catalogue	Ledger	x	x	x	x	x
Data distribution	P2P	x	x	x		
	Blockchain				x	x
Accounting	PEP		x			
	Ledger			x	x	x
Scalability	Streaming	x	x	x		x
Auditability	Accounting			x	x	x
	Data distribution				x	x

Table 3.1: Scenarios comparative table

and customers about a data offering with a determined pricing model are made and tracked on the blockchain.

As it is shown in figure 3.1 data is distributed in the same way than in the initial scenario. It uses a peer-to-peer system regulated by an API Umbrella PEP which is a proxy that sits in front of the assets and regulates access control, rate limiting and analytics. Each participant has its own blockchain node. Depending on the participant role, it is allowed to make different kinds of transactions. A prototype was developed as a first approach in Hyperledger Composer because it provides a playground environment with a visual interface to test the network.

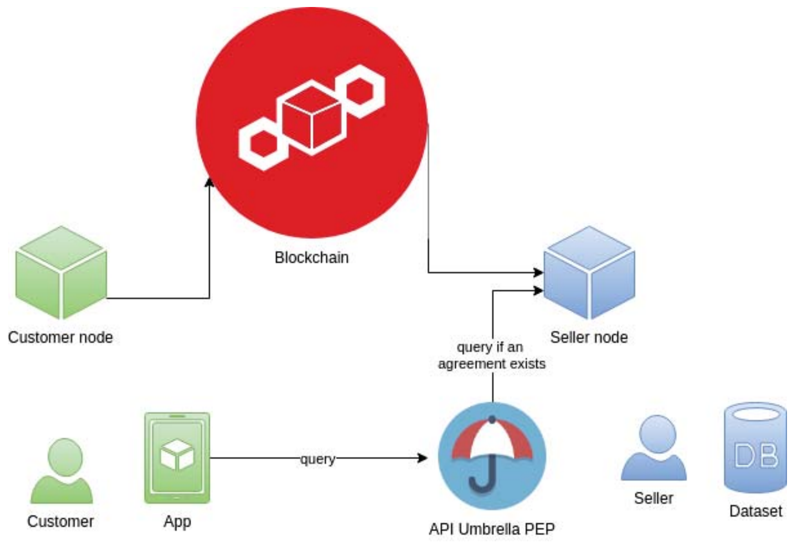


Figure 3.1: Scenario 1. A decentralized catalogue and P2P data distribution with a PEP

The architecture keeps the same flow as the current one that uses the BAE. For this case, on the ledger, sellers publish offerings on the catalog by making a transaction *createOffering* where customers can search and choose any of them with another transaction *makeAgreement*. These transactions are publicly tracked forever on the ledger so when a customer tries to access some data the Police Enforcement Point (PEP) verifies whether there is a valid agreement and grants access if so.

For the management of participants: authentication, login and access. Hyperledger has a Membership Service Provider (MSP) and a REST server. Although figure 3.1 is simplified, multiple users on the same node are supported because Hyperledger Composer supplies an individually treatment for the final user.

Advantages

Apart from what is already explained about getting an immutable, distributed and secure log in a marketplace, it is very interesting to highlight that when a customer chooses an offering and orders it, no response is required from the seller so it works asynchronously. There is a high interest on the system speed to take advantage of the continuous flow of generated data on streaming.

Disadvantages

Although this architecture is enough for the simplest marketplaces there are some important weaknesses to consider:

- The main problem is that accounting is necessary for usage payment models.
- It is paradoxical that every participant must trust the PEP because it manages the data access.
- It is not possible to audit that the customer is actually getting what is established on the agreement.

3.2 Scenario 2. Accounting: PEP

The second scenario solves the first problem raised above by entrusting the PEP with carrying out the accounting. This architecture proposes exactly the same flow that in the previous section but with the difference that the PEP is in charge of registering every access, query and response of a dataset on a local database as it is shown in figure 3.2.

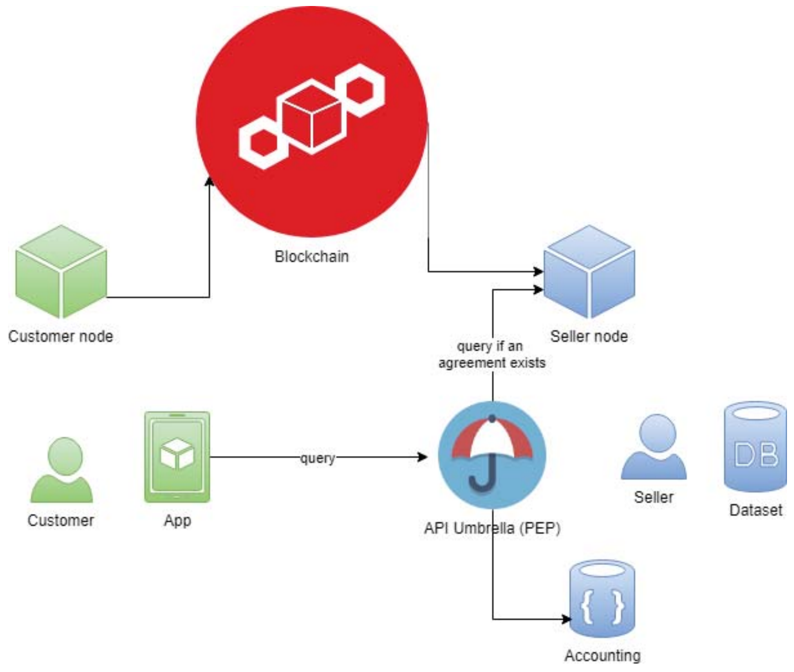


Figure 3.2: Scenario 2. A decentralized catalogue, P2P data distribution with accounting through a monitoring PEP

Advantages

This simple change provides a fast solution to support complex pricing models like usage payment or revenue sharing. These models make the difference with other marketplaces with classic methods. Datasets owners will be much more interested in commercializing their information on systems which support more combinations and produce major profit.

Disadvantages

- Apart than access, accounting is also managed by the PEP. It serves as the gatekeeper and "front door" to a digital resource. This means that each seller owns at least one PEP, managed by himself. The seller is in charge of registering accounting details that are used for billing the service. This is the reason why this scenario requires even greater confidence in the PEP than the previous one. Nobody can assures that its behaviour is completely honest.
- It is still not possible to prove that the customer is receiving the data he is paying for.

3.3 Scenario 3. Decentralized accounting: ledger

The third scenario goes one step further as it eliminates the PEP from the architecture. It takes advantage of the fact that each participant is already connected to the blockchain because it is necessary to publish and show interest for an offering. It is taken for granted that everybody, seller or customer, has access to the ledger through a node. Starting from this premise, the participants themselves are able to introduce the necessary information on the ledger to account queries, access and data exchanges.

The process is now longer than previous scenarios but we can divide it into the following steps:

1. The customer app queries directly the seller app for some data.
2. The seller app through its blockchain node ask to the ledger if there is an existing active agreement.
3. It receives a response to revoke the customer's query if it is negative.
4. If it is a positive response the app achieves the requested data.
5. The seller app sends the data to the customer app at the same time that both nodes, seller and customer, enter a transaction in the ledger. This transaction contains meta-information about the sent data such as a hash code to assure integrity or the volume size. It must be multi-signed by both parts before being accepted in the ledger.

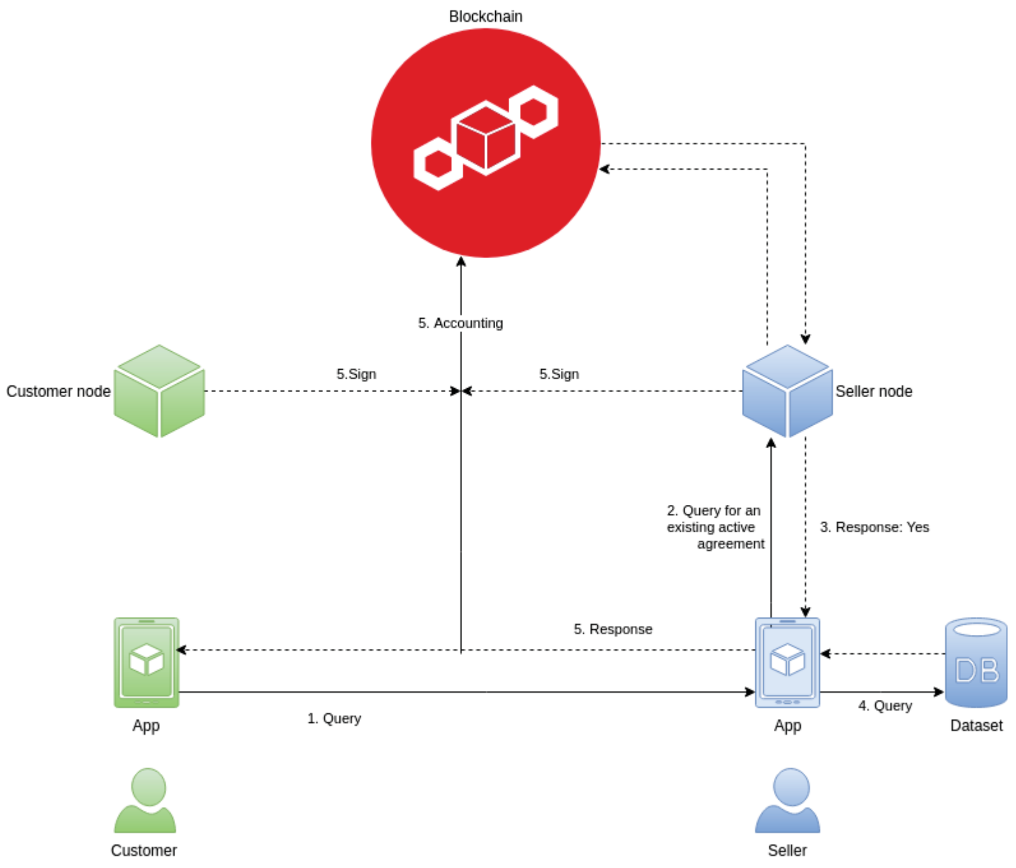


Figure 3.3: Scenario 3. A decentralized catalogue and accounting with P2P data distribution

Advantages

The main new gain is the disappearance of the PEP as a figure that requires relying on the other party. All the participants trust the ledger so making the accounting on it is auditable because it grants an immutable, secure and persistent log of transactions. This scenario also retains all the advantages of the previous ones.

Disadvantages

It does not provide auditability for data distribution. If the customer wants to cheat so he does not recognize that data was properly received and he does not sign the accounting transaction, the seller has no way to reclaim the payment because data was distributed peer-to-peer. There is no notary or witness who belies the scam.

A good possible way to overcome this obstacle is encrypting data. It allows us to establish a communication protocol that forces the customer to sign the accounting transaction in order to receive the keys to decipher it. This solution would imply a jump back for streaming data because it loses speed as it adds more request and response sequential messages.If the priority of the marketplace is to audit the data distribution and void this kind of problems, it could be valid to sacrifice the scalability of the system.

3.4 Scenario 4. Blockchain data distribution

The fourth scenario makes the most of the ledger and exploits it to the fullest by also entrusting it the data distribution in addition to the catalog functions. Everything is done with blockchain transactions. As it is shown in figure 3.4 the customer app queries the ledger generating and event. The seller node receives a notification because it is subscribed to this event and it sends the requested data to the ledger. It delivers the data to the customer as a response of the first query.

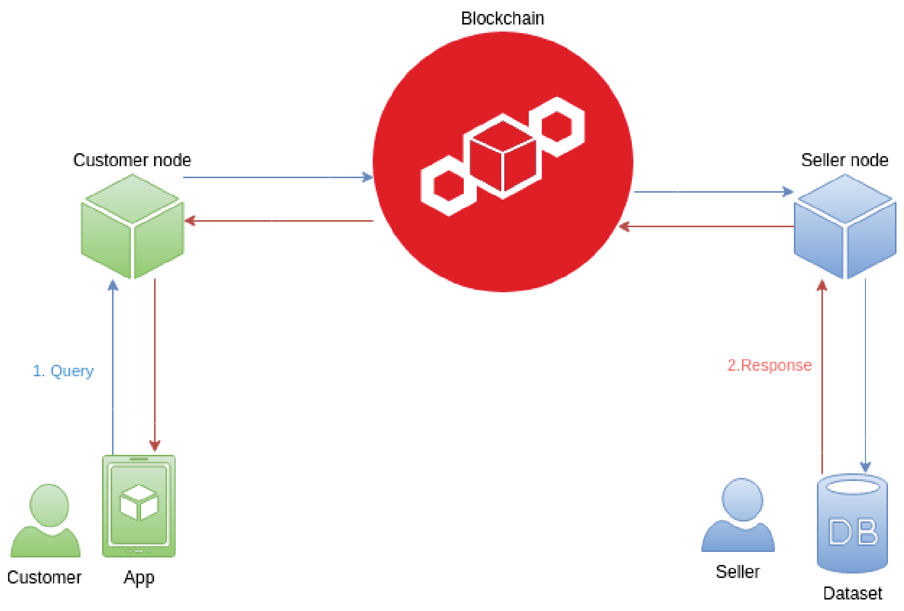


Figure 3.4: Scenario 4. A decentralized catalog and data distribution in one ledger

Advantages

Distributing the data using blockchain connotes that everything is auditable because it is immutability tracked in the ledger. Trust problems described in the previous scenario do not exist any more.

Disadvantages

The more transactions that the network needs to process, the longer each transaction takes. In Bitcoin the average time for one confirmation is from 30 minutes to over 16 hours in extreme cases [43]. It means that if the marketplace has a lot of traffic and the ledger is being overloaded with many transactions, the system is inefficient. The customer would receive data which was updated on the time the provider sent the information to the ledger and this implies this scheme does not support streaming data.

Note that this scenario decentralized the channel to distribute the data but not the storage of it. At first sight, the Orion Context Broker seems likely to improve if it uses a distributed ledger instead of a centralized database but our decision was keeping it because decentralizing would be not efficient at all. I justify this design decision just below,

Many marketplaces store the information directly on the ledger. Data, in our case, comes from IoT devices like sensors which are collecting a lot of information and sending it continuously. Any time that a measure changes a new transaction is sent to the ledger. The growing of the ledger is completely uncontrolled because each peer could be sending data as soon as a sensor measures anything new, almost every second. The ledger keeps an historical log of all the changes in exchange for efficiency. This fact rises a big question: Do IoT and blockchain relationship make sense?

In our use case, which is a data marketplace focused on IoT streaming information, sacrifying efficiency in return of keeping a historic log does not make sense but there are other applications which can be very interested in doing that. The ledger just receive the specific data that is queried for. This fact reduces the number of transactions and control an indiscriminated growing of the ledger.

A good example is an analytic application which calculates predictions and statistics from the historical changes of some data. Smart contracts would run automatically the calculus associated with the correspondent type of transaction which is included on the ledger.

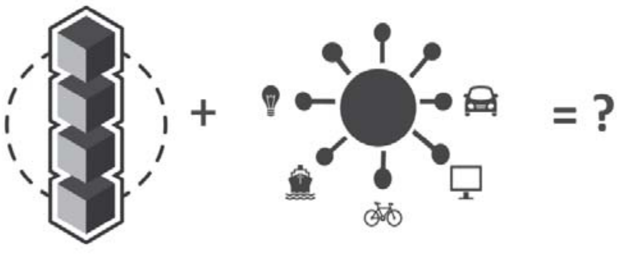


Figure 3.5: Do IoT and blockchain relationship make sense?

Source:
`https://blog.iota.org/a-primer-on-iota-with-presentation-e0a6eb2cc621`

Keeping the historical is priority when data is critical, another example case would be the control of a plain that crashes. Analysing each input from all the sensors would allow a posterior investigation to identify the cause of the accident.

This project is a clear case of a design decision which consists on storing data locally in order to achieve efficiency providing data in streaming with the minimum delay.

This scheme has a weakness if the system provides a subscription model to the customers. This model allows customers to be subscribed to determined changes on the data. In this case the customer is in charged of sending the new information to the ledger every time the conditions are met (i.e.: the temperature is less than 0^{o}). If it happens very often and there are many subscribers, this scenario can converge to the one that we are trying to avoid because actually it would be working as a decentralized context broker and the system would be very inefficient.

3.5 Scenario 5. Two ledgers

There are many blockchain technologies which are focused on different specific applications. Using each of them for its own field grants efficiency. They were named on section 2.1.5.3 but the most interesting ones for us are listed below with their objectives:

- Sawtooth: Physical traceability of any supply chain. It provides scalability for managing Internet of Things (IoT) sensors information.

- Tangle: IOTA, a cryptocurrency for the IoT industry. It offers features to establish a machine-to-machine micropayment system.
- Hyperledger Composer: Offering an abstraction layer to build applications and business networks on blockchain.
- Hyperledger Fabric: Developing applications with a modular architecture maximum customizable.
- Hyperledger Indy: Built for decentralized identity.

Given the previous proposed schemas it is proven that blockchain systems present a paradigm to solve: efficiency versus auditability. For our concrete case this fifth scenario describes a solution which takes the best of different blockchains technologies. We propose to use two ledgers with a different technology to use each of them for a different purpose. We have chosen Hyperledger Composer for the decentralization of the offerings catalogue and Tangle (IOTA) for the data distribution. The election of these technologies is based on the proofs of concept detailed on chapter 4.

As it is shown in figure 3.6, the customer application queries for the data to the Tangle which notifies the seller's app. The seller node assures that there is an existing active agreement by querying the Hyperledger Composer ledger. If everything is in order, it sends to the Tangle the requested information that is delivered as a response to the customer. At the same time, it sends a transaction to Hyperledger Composer including meta information for the accounting. All the exchanged information in Tangle is published in an independent channel to assures privacy. There will be one pair of Tangle channels for each agreement: the first one for sending queries by the customer and the other one for sending the responses by the seller. The decision of having two channels per agreement instead of two channels per customer-seller is to minimize damages in case of a transaction failure. As data messages are chained, if there is an error, the stream breaks down.

Although it is not represented in the figure, in order to avoid an uncontrolled growing of the Tangle, it is consider to create a snapshot and to include it into the Hyperledger periodically. It is not necessary to keep a historic of all the information which is distributed using the ledger and after some time it will loose efficiency.

Since data distribution is realized using MAM in restricted mode, some changes are necessary in the catalogue decentralization. The catalog is where offerings are published and agreements are made. This scenario is the final solution for our case, so it is worthy to detail the new working flow of the decentralized BAE. To simplify the model, from this point we will talk about a new concept, `TangleStream`, to

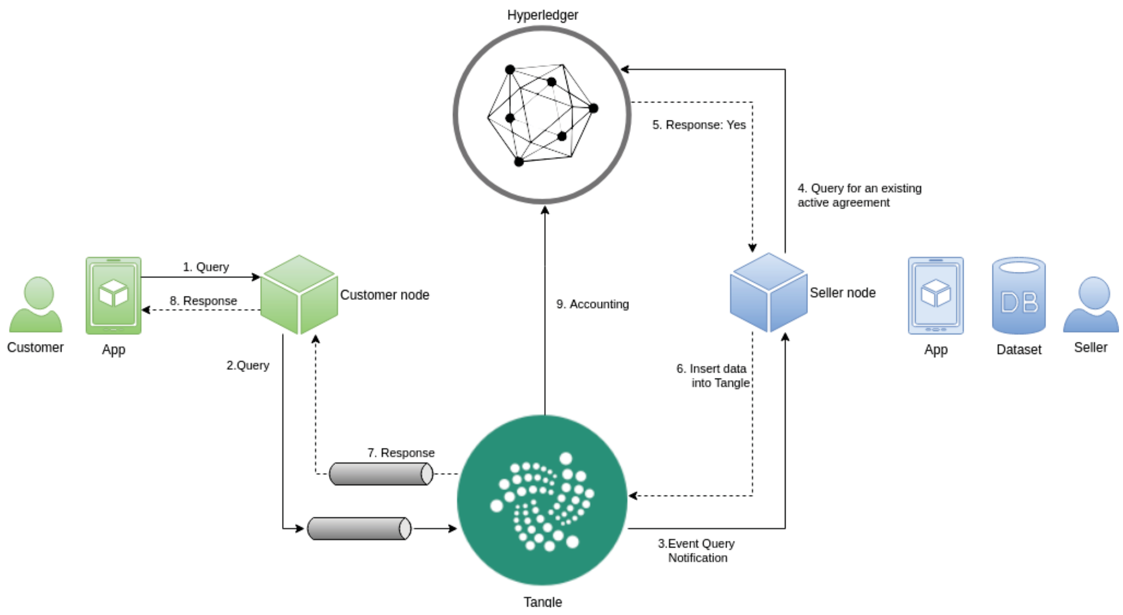


Figure 3.6: Scenario 5. One ledger for catalog and the other ledger for data distribution

abstract from implementation details. It is a Tangle channel in restricted mode only accessible by users with the right `side_key` and `root` where all the messages are chained. The whole process to exchange information related to the Tangle is as follows:

1. Everyone can publish a dataset by submitting a transaction `CreateDataset` including some information to describe it.
2. Sellers publish offerings of a determined dataset on the catalog by making a transaction `CreateOffering` on the ledger where customers can search.
3. A customer can choose any offering with another transaction `MakeAgreement` which contains the `side_key` and the `root` of its `TangleStream` where it will send queries. The `Agreement` asset will be created with the status `PAYMENT_PENDING` and the offering owner is notified.
4. The customer sends a `PaymentCompleted` transaction with a proof of the payment to change the status of the agreement to `ACTIVE`.
5. Once the seller receives a notification of the payment, it sends another transaction `AcceptAgreement` with the `side_key` and the `root` of the `TangleStream` where data is being delivered.

6. There is another type of transaction called `SettleAgreement` which renews the agreement once it is over.

Figure 4.3 in section 4.1.1 shows the UML sequence diagram of an specific example.

Advantages

Scalability is achieved at the same time as auditability when using these two separate ledgers. On the one hand, Tangle provides the possibility of storing huge amounts of data and velocity to access it. This is possible thanks to the Tangle's feature MAM. It brings the concept of data stream breaking the existing limitation of working just with isolated information or independent assets. The added value is that it also controls data access and preserves the privacy of the transactions with its different modes. On the other hand, Hyperledger Composer brings the immutability when storing the transactions registry assuring auditability to the system. Storing Tangle snapshots into the second ledger combinates both features together.

Disadvantages

Tangle is not a fully tested technology. It is still being developed so there are many features to come that probably will change this architecture. Smart contracts are promised for June 2018. Interactions between ledgers from their own smart contracts are also to come and this would be a nice improvement respect to this scenario which deposits some responsability on the seller node to check for an existing agreement and to deliver the data.

4
Proofs of concept

Champions know there are not shortcuts to the top. They climb the mountain one step at a time.

Judi Adler

Each scenario elaborated in the previous chapter has its foundation on a practical proof of concept that has been developed as part of this Master's Thesis. Since blockchain is a very innovative concept, the intention of this Master's Thesis is to contribute to the knowledge and development of blockchain, not only in a theoretical way, but with evidences that endorse our proposals. All the prototypes have been developed with different blockchain open-source technologies. This chapter presents the technical road map of the three ones that have been researched: *Hyperledger Composer*, *Hyperledger Sawtooth* and *Tangle*.

4.1 Hyperledger Composer

Hyperledger, briefly exposed in section 2.1.5.3, has a big project which is **Hyperledger Composer**. It is a set of collaboration tools for building blockchain

FIWARE	**HYPERLEDGER**
Product	Asset
Bundle	Account
Product Offering	Offering
Category	-
Catalogue	-
Pricing plan	-
Revenue Sharing model	-
Order	-
Asset Type / Plugin	-
Transaction	Transaction
User (Seller/Customer)	Participant
-	Holdings
-	Liability
Product Offerings	Exchange Offer

Table 4.1: Glossary mapping between Hyperledger and FIWARE.

business networks. It works with modern tools like JavaScript and Node.js. The key point is that it offers an interesting abstraction level over **Hyperledger Fabric** which is the basic blockchain engine.

Fabric provides the starting architecture of a blockchain solution. Like other blockchain technologies, it has a distributed ledger, it uses smart contracts, and it is a system whose participants manage their own transactions. It is private and permissioned. It means that, instead of doing Proof of Work to validate transactions because peers are unknown identities, all the members trust in the MSP because all transactions can be detected and traced by authorized regulators and auditors. It offers many pluggable options so the ledger can be stored in multiple formats, consensus mechanisms can be chosen and swapped in and out, and different MSP are supported. The ledger is a combination of the **world state** and the **transaction log** which are respectively the state of the ledger at a given time and the record of all transactions up to that point. The smart contracts are written in the **chaincode** which is invoked by the runtime [27].

Since there are lots of Fabric's own terms, the glossary mapping shown in 4.1 with the terminology of Hyperledger (specially Fabric) and FIWARE, was realized to design an adequate blockchain modelization for the data marketplace of this project.

The two main tools which have been used for learning and testing are **Hyperledger Composer Playground** and **Hyperledger Composer REST**

Server. The first one provides a web interface which is very intuitive for developing and testing purposes. This application can be deployed on an IBM Bluemix [30] instance or over a local Fabric installation. The second one is for installing and running a complete REST API to be used from external apps.

These are the different phases carried out in this proof of concept which are exposed in detail just below:

1. Hyperledger Composer Playground deployment on Bluemix.
2. Hyperledger Composer Playground deployment over Hyperledger Fabric in two different physical machines.
3. Angular web application running with a Rest Hyperledger Server.

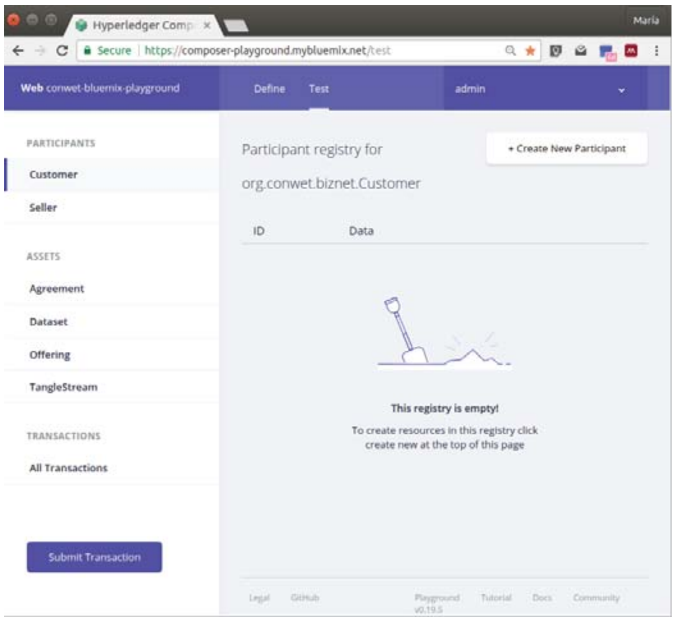
4.1.1 Hyperledger Composer Playground deployment on Bluemix

Using the Hyperledger Composer Playground on Bluemix[1] has many advantages. It makes very fast to implement a **Business Network Definition** because it provides a web interface with one tab to edit all the configuration files and other tab for testing as figure 4.1 shows.

The Business Network Definition consists on three main components:

- **Model**. It defines the business domain. It is written on the specific Hyperledger modelization language (.cto) and defines the structure and relationships between model elements.
- **Logic**. The transaction processor functions implement all the transaction types defined in the Business Network Definition's model files. Each of them has an associated function which is automatically invoked by the runtime when transactions are submitted using the Business Network Connection API. These are the smart contracts. Business requirements are implemented in JavaScript.
- **Access Control**. This file has the collection of rules to manage the access to data and the rights of the different participants.

[1]Available online on `http://composer-playground.mybluemix.net`



Participant registry for org.conwet.biznet.Customer

This registry is empty!

To create resources in this registry click create new at the top of this page

Figure 4.1: Hyperledger Composer Playground deployment on Bluemix

The implementation of our model was done taking advantage of the fact that the changes made on every file are immediately updated in the network testing scenario. The model defines transactions, participants, assets and events. To understand the implemented transaction types is important to contextualize that this ledger is used to decentralize the catalogue of our data marketplace, not the data distribution. It is explained in section 3.5.

As it is shown in figure 4.2, there are different types of assets (in green): `Dataset`, `Offering`, `Agreement` and `TangleStream`. There is just one type of participant (in blue) defined for this business app which is `User`. It can play with both seller and customer roles. There are also enums (in orange). When an user submits a transaction to the ledger, the smart contract is in charge of creating or updating the corresponding asset as required and to launch the events which are explained in section 4.1.3.

A **Business Network Card** provides all the information needed for connecting to a blockchain business network. A participant can access the network only with a valid one. It contains the identity for a single participant. Cards are grouped under a **Connection Profile**. Hyperledger Composer Playground allows to issue the identity for a new network participant to connect to the runtime. This means that is possible to login into the playground site with his profile to test. All of this is done from a high abstraction level because it does not allow to customize

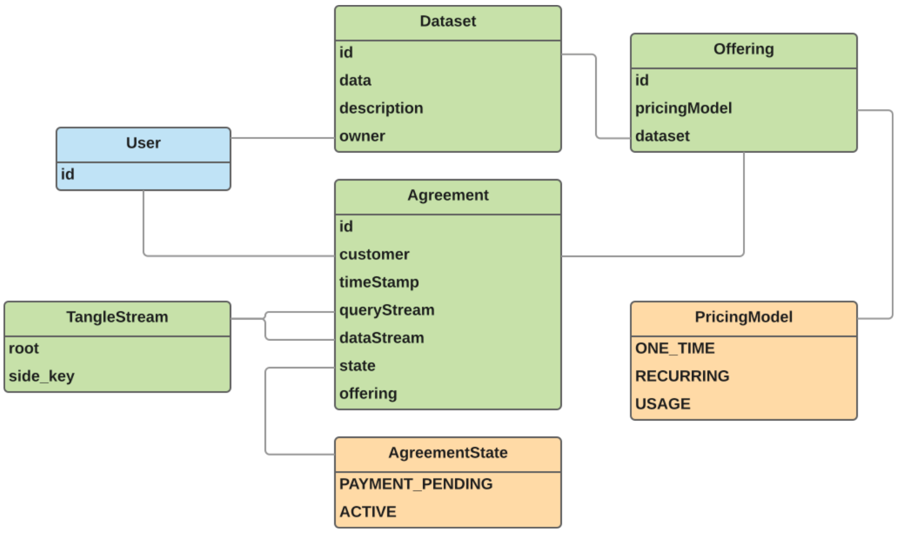


Figure 4.2: BAE Hyperledger modelization.

connections profiles.

The access control implementation was tested using this feature to allow participant to submit only pertinent transactions. Permissions have been configured in a very restrictive way using Access Control List (ACL)s. Controlling the reading access to the agreements ensures privacy for sellers and customers. All `TangleStream` instances are ciphered with the other's public key to prevent any filtration.

There is an option to export the entired network in a specific format which is a Business Network Archive (.bna). Since the Composer chaincode is written in Go, it uses a JavaScript interpreter to execute the code within a Go procces and this is why JavaScript can be used. From an archive with this extension, the Composer is capable to deploy the correspondant business network with its own Fabric network taking the following steps:

1. Mapping the public JavaScript API to the underlying Fabric Go API calls and starting the Fabric instance.

2. Creating the world-state extracting information from the file so it is available to the interpreter.

Many use cases were tested using this deployment to proof the Business Network Definition. One representative proven example case within the frame of industry 4.0 is *Smart Plastic*. One participant which acts as a seller is a plastic figures production factory whose machines have sensors and produce information such as the remaining level of plastic. Other organizations could be interested in this information, for example, plastic providers. The customer in this case is called "plastic provider 1". Figure 4.3 shows a sequence UML diagram to illustrate the full working flow.

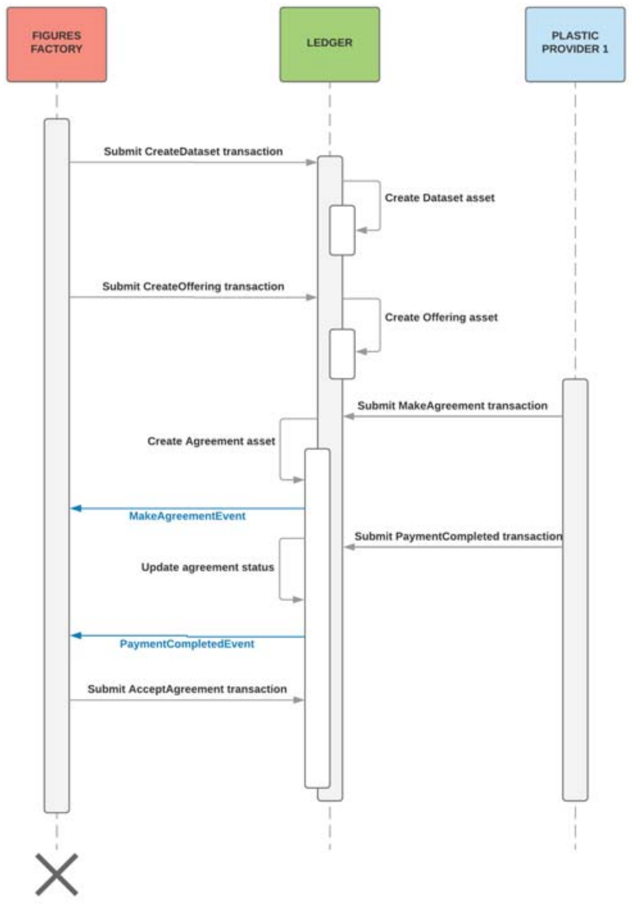

FIGURES FACTORY
LEDGER
PLASTIC PROVIDER 1
Submit CreateDataset transaction
Create Dataset asset
Submit CreateOffering transaction
Create Offering asset
Submit MakeAgreement transaction
Create Agreement asset
MakeAgreementEvent
Submit PaymentCompleted transaction
Update agreement status
PaymentCompletedEvent
Submit AcceptAgreement transaction

Figure 4.3: Smart Plastic use case sequence diagram.

4.1.2 Hyperledger Composer Playground deployment over Hyperledger Fabric in two different physical machines

From the previous section is deducted that for a full customization, it is necessary to deploy a Fabric network instead of using the Bluemix tool. This proof of concept is very important because the identity management of the network is quite confusing since Fabric's users have different functions than Composer's users.

A Business Network Card is represented by `.card` files which contain a `medatada.JSON` file, a connection profile and certificates. The connection profile is specified in another JSON file with some parameters such as version, number of peers and IP addresses.

Deploying a business network in a Hyperledger Fabric instance for the first time requires the Hyperledger Composer chaincode to be installed on the Hyperledger Fabric peers. This process requires special Hyperledger Fabric privileges possessed by a peer administrator. So a peer admin business network card was created. This is the unique user that works in both Fabric and Composer level.

An example was deployed in localhost using Docker to simulate two different organizations with a pair of peers in each of them. Every organization needs a network manager who acts as its Certification Authority (CA)[2]. For settling concepts and testing the privacy level between peers, another deployment was done in two different physical machines. The architecture is shown in figure 4.4.

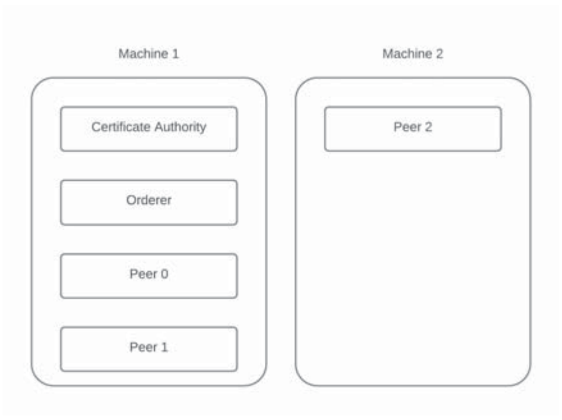
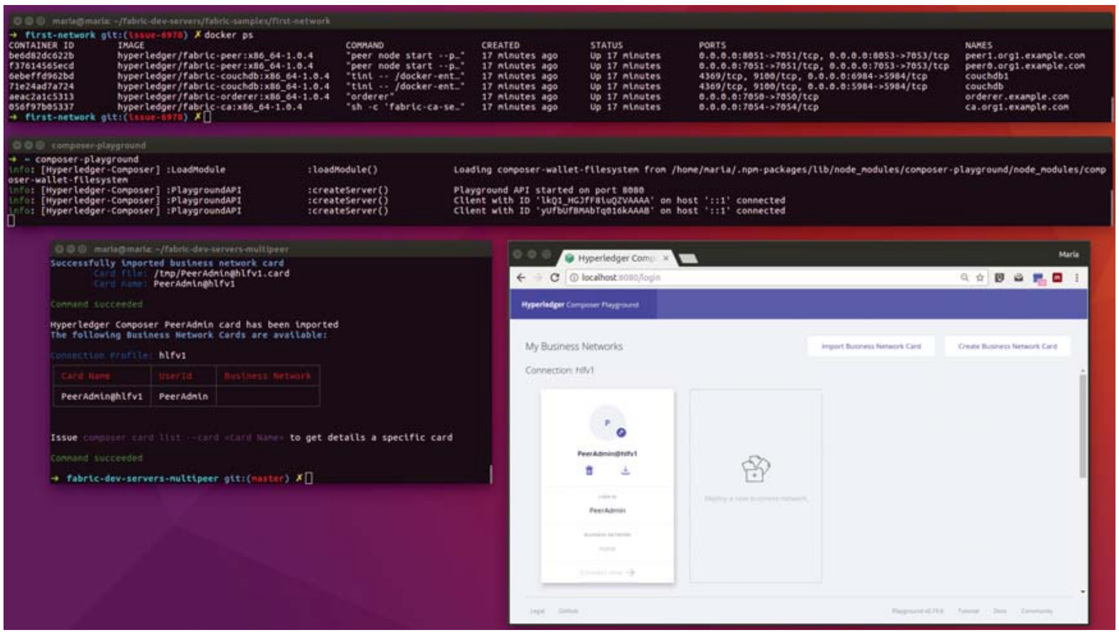


Figure 4.4: PoC of two different physical machines Hyperledger deployment

As it can be seen in figure 4.5, the first machine has the PeerAdmin business network card which deploys the Fabric instance and can be shown in the Hyperledger Composer Playground web interface. It has six running dockers because there are: two peers, the orderer and the CA of the organization. Each peer has its own CouchDB which runs in a different Virtual Machine (VM).

The second machine has just one peer. This is the reason why there are two

[2]An installation guide and files for our business network are available on `https://github.com/msalitu/hyperledger-bae`.



```
→ first-network git:(issue-6970) ✗ docker ps
CONTAINER ID        IMAGE                                     COMMAND                  CREATED             STATUS              PORTS                                            NAMES
be6d82dc622b        hyperledger/fabric-peer:x86_64-1.0.4      "peer node start --p…"   17 minutes ago      Up 17 minutes       0.0.0.0:8051->7051/tcp, 0.0.0.0:8053->7053/tcp   peer1.org1.example.com
f37614565ecd        hyperledger/fabric-peer:x86_64-1.0.4      "peer node start --p…"   17 minutes ago      Up 17 minutes       0.0.0.0:7051->7051/tcp, 0.0.0.0:7053->7053/tcp   peer0.org1.example.com
6ebeffd962bd        hyperledger/fabric-couchdb:x86_64-1.0.4   "tini -- /docker-ent…"   17 minutes ago      Up 17 minutes       4369/tcp, 9100/tcp, 0.0.0.0:6984->5984/tcp       couchdb1
71e24ad7a724        hyperledger/fabric-couchdb:x86_64-1.0.4   "tini -- /docker-ent…"   17 minutes ago      Up 17 minutes       4369/tcp, 9100/tcp, 0.0.0.0:5984->5984/tcp       couchdb
aeac2a1c5313        hyperledger/fabric-orderer:x86_64-1.0.4   "orderer"                17 minutes ago      Up 17 minutes       0.0.0.0:7050->7050/tcp                           orderer.example.com
056f97b05337        hyperledger/fabric-ca:x86_64-1.0.4        "sh -c 'fabric-ca-se…"   17 minutes ago      Up 17 minutes       0.0.0.0:7054->7054/tcp                           ca.org1.example.com
→ first-network git:(issue-6970) ✗
```

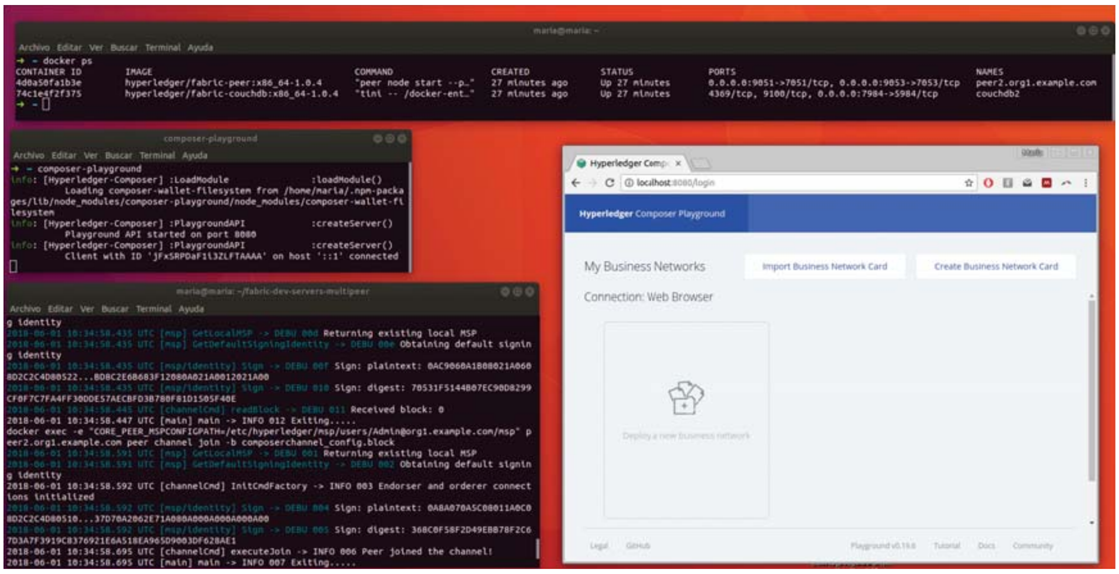
```
→ ~ composer-playground
info: [Hyperledger-Composer] :LoadModule            :loadModule()         Loading composer-wallet-filesystem from /home/maria/.npm-packages/lib/node_modules/composer-playground/node_modules/comp
oser-wallet-filesystem
info: [Hyperledger-Composer] :PlaygroundAPI         :createServer()       Playground API started on port 8080
info: [Hyperledger-Composer] :PlaygroundAPI         :createServer()       Client with ID 'lkQ1_HGJfF8luQZVAAAA' on host '::1' connected
info: [Hyperledger-Composer] :PlaygroundAPI         :createServer()       Client with ID 'yUfbUfBMAbTq016kAAAB' on host '::1' connected
```

```
Successfully imported business network card
        Card file: /tmp/PeerAdmin@hlfv1.card
        Card name: PeerAdmin@hlfv1

Command succeeded

Hyperledger Composer PeerAdmin card has been imported
The following Business Network Cards are available:

Connection Profile: hlfv1
┌─────────────────┬───────────┬──────────────────┐
│ Card Name       │ UserId    │ Business Network │
├─────────────────┼───────────┼──────────────────┤
│ PeerAdmin@hlfv1 │ PeerAdmin │                  │
└─────────────────┴───────────┴──────────────────┘


Issue composer card list --card <Card Name> to get details a specific card

Command succeeded

→ fabric-dev-servers-multipeer git:(master) ✗
```

Hyperledger Composer Playground
localhost:8080/login
My Business Networks
Import Business Network Card
Create Business Network Card
Connection: hlfv1
PeerAdmin@hlfv1
PeerAdmin
Deploy a new business network

Figure 4.5: Screenshot of the first machine Composer deployment.

Dockers running in the screenshot shown in figure 4.6. The Hyperledger Composer Playground web interface does not show the PeerAdmin card as well as any other information which is public. Only assets and transactions specifically allowed in our permissions file are accessible from this peer.



```
→ ~ docker ps
CONTAINER ID        IMAGE                                     COMMAND                  CREATED             STATUS              PORTS                                            NAMES
4d0a50fa1b3e        hyperledger/fabric-peer:x86_64-1.0.4      "peer node start --p…"   27 minutes ago      Up 27 minutes       0.0.0.0:9051->7051/tcp, 0.0.0.0:9053->7053/tcp   peer2.org1.example.com
74c1e4f2f375        hyperledger/fabric-couchdb:x86_64-1.0.4   "tini -- /docker-ent…"   27 minutes ago      Up 27 minutes       4369/tcp, 9100/tcp, 0.0.0.0:7984->5984/tcp       couchdb2
→ ~
```

```
→ ~ composer-playground
info: [Hyperledger-Composer] :LoadModule                  :loadModule()
        Loading composer-wallet-filesystem from /home/maria/.npm-packa
ges/lib/node_modules/composer-playground/node_modules/composer-wallet-fi
lesystem
info: [Hyperledger-Composer] :PlaygroundAPI               :createServer()
        Playground API started on port 8080
info: [Hyperledger-Composer] :PlaygroundAPI               :createServer()
        Client with ID 'jFxSRPDaF1i3ZLFTAAAA' on host '::1' connected
```

```
g identity
2018-06-01 10:34:58.435 UTC [msp] GetLocalMSP -> DEBU 00d Returning existing local MSP
2018-06-01 10:34:58.435 UTC [msp] GetDefaultSigningIdentity -> DEBU 00e Obtaining default signin
g identity
2018-06-01 10:34:58.435 UTC [msp/identity] Sign -> DEBU 00f Sign: plaintext: 0AC9060A1B08021A060
8D2C2C4D80522...8D8C2E68683F12080A021A0012021A00
2018-06-01 10:34:58.435 UTC [msp/identity] Sign -> DEBU 010 Sign: digest: 70531F5144B07EC90D8299
CF0F7C7FA4FF30DDE57AECBFD3B780F81D1505F40E
2018-06-01 10:34:58.445 UTC [channelCmd] readBlock -> DEBU 011 Received block: 0
2018-06-01 10:34:58.447 UTC [main] main -> INFO 012 Exiting.....
docker exec -e "CORE_PEER_MSPCONFIGPATH=/etc/hyperledger/msp/users/Admin@org1.example.com/msp" p
eer2.org1.example.com peer channel join -b composerchannel_config.block
2018-06-01 10:34:58.591 UTC [msp] GetLocalMSP -> DEBU 001 Returning existing local MSP
2018-06-01 10:34:58.591 UTC [msp] GetDefaultSigningIdentity -> DEBU 002 Obtaining default signin
g identity
2018-06-01 10:34:58.592 UTC [channelCmd] InitCmdFactory -> INFO 003 Endorser and orderer connect
ions initialized
2018-06-01 10:34:58.592 UTC [msp/identity] Sign -> DEBU 004 Sign: plaintext: 0A8A070A5C08011A0C0
8D2C2C4D80510...37D70A2062E71A000A000A000A000A00
2018-06-01 10:34:58.592 UTC [msp/identity] Sign -> DEBU 005 Sign: digest: 368C0F58F2D49EB878F2C6
7D3A7F3919C8376921E6A518EA965D9003DF628AE1
2018-06-01 10:34:58.695 UTC [channelCmd] executeJoin -> INFO 006 Peer joined the channel!
2018-06-01 10:34:58.695 UTC [main] main -> INFO 007 Exiting.....
```

Archivo Editar Ver Buscar Terminal Ayuda
Hyperledger Composer Playground
localhost:8080/login
My Business Networks
Import Business Network Card
Create Business Network Card
Connection: Web Browser
Deploy a new business network

Figure 4.6: Screenshot of the second machine Composer deployment.

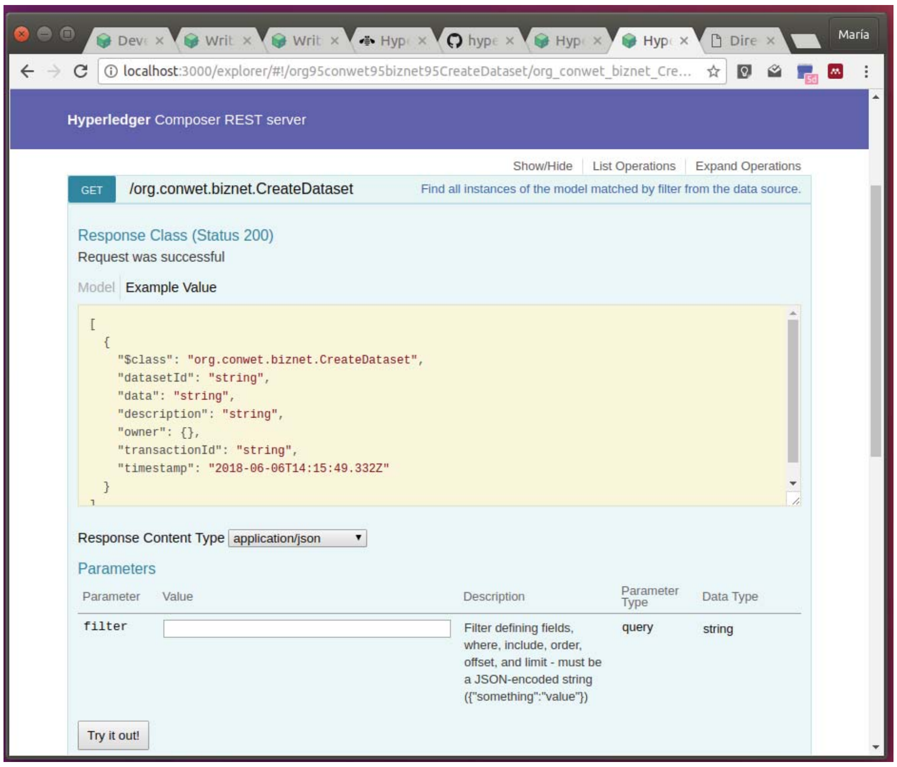
This result is very important for granting privacy between peers. Hyperledger Fabric provides an efficient method which is using **channels** for one-to-one private communications but Hyperledger Composer does not support more than one channel. Actually, Fabric's channels are implemented with a different ledger for each one. This implementation is not interesting for this project because different ledgers cannot interact between them and data crossing is necessary for the correct working of the marketplace.

4.1.3 Angular web application running with a Rest Hyperledger Server

Once a Hyperledger Composer deployment is done, the way to see the content and to interact with it is by using a tool called `composer-rest-server`. It generates a Loopback-REST interface to access the network. It needs to communicate with Hyperledger Fabric through a business network card. Once it is set up [39], it provides an intuitive interface which is shown in figure 4.7 to explore the API REST. This REST server supports OAuth authentication and can be secured using HTTPS and TLS.

The REST server can subscribe to events emmited from a deployed business network and publish business events for consumption by client applications over WebSockets.

Web applications that need to interact with deployed business networks should make calls to the REST API. In this proof of concept, an initial version of an Angular application called BizApp was implemented and deployed on localhost. This app provides an user interface to interact with the ledger, its assets and the other participants. It does not add any aditional functionality but it allows the user to receive notifications when other participant submit any transaction which affects him. In this prototype, for example, if Alice creates a dataset and publishes an offering and afterwords Bob makes an agreement for this offering, Alice receives a notification. This application is the first version of which would replace the current website of the marketplace.


Devı
Writ
Writ
Hyp
hype
Hyp
Hyp
Dire
María
localhost:3000/explorer/#!/org95conwet95biznet95CreateDataset/org_conwet_biznet_Cre...
Hyperledger Composer REST server
Show/Hide
List Operations
Expand Operations
GET
Model
Try it out!

/org.conwet.biznet.CreateDataset

Find all instances of the model matched by filter from the data source.

Response Class (Status 200)

Request was successful

Example Value

```
[
  {
    "$class": "org.conwet.biznet.CreateDataset",
    "datasetId": "string",
    "data": "string",
    "description": "string",
    "owner": {},
    "transactionId": "string",
    "timestamp": "2018-06-06T14:15:49.332Z"
  }
]
```

Response Content Type application/json

Parameters

Parameter	Value	Description	Parameter Type	Data Type
filter		Filter defining fields, where, include, order, offset, and limit - must be a JSON-encoded string ({"something":"value"})	query	string

Figure 4.7: Screenshot of the REST API explorer interface.

4.2 Hyperledger Sawtooth

A private Sawtooth [3] network was deployed to test this technology [37] [28]. Some of the distinctive features which motivated us to prove it are:

- Scalability. The framework is able to process parallel transactions.
- Enhanced security and reliability to proof honesty in the peers network because the chaincode runs in a secure environment inside certain Intel architectures with Software Guard Extensions (SGX) capabilities. It is not mandatory to implement the system using the provided secure environments, since we don't have the resources that support this technology; we have tested the technology using by default security configuration.
- Ethereum Contract Compatibility with Seth. Smart Contracts written in Solidity for Ethereum networks can be deployed in the Sawtooth network

[3]`https://sawtooth.hyperledger.org`

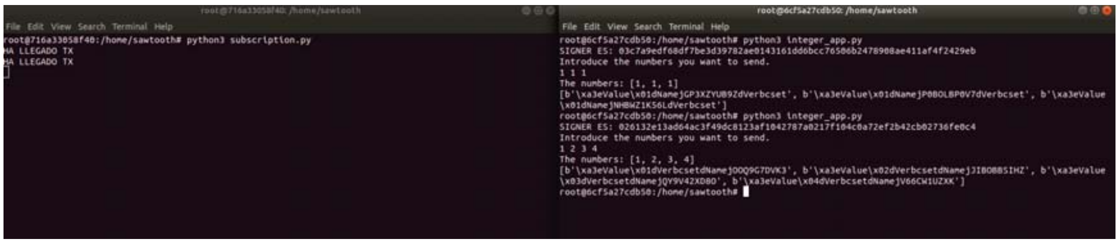
too.

- Pluggable consensus algorithms. It is chosen by the developers and the business network administrator. They specify the configuration of the consensus algorithm by submitting a transaction.

In Sawtooth, the data model and transaction language are implemented in a transaction family. Tests were done using the Intkey Transaction's family although users are supposed to build custom transactions families that reflect our own requirements. Sawtooth implements the Smart Contract Framework by specifying the operations that the defined transaction family supports.

The Sawtooth API uses Protobuf [20] to specify the transaction and batches format. Generating the Python implementations of .proto files was the first challenge before developing two PoCs using this framework:

1. To test scalability, it was necessary to automatize how to submit transactions and storing information on the ledger. A Python script was developed. It generates as many batches as the user specifies in the first argument, containing the number of transactions given in the second parameter. The generated numbers are random. This script works as it is shown in the right console of the figure 4.8.



root@716a33058f40: /home/sawtooth
File Edit View Search Terminal Help
root@6cf5a27cdb50: /home/sawtooth
File Edit View Search Terminal Help

```
root@716a33058f40:/home/sawtooth# python3 subscription.py
HA LLEGADO TX
HA LLEGADO TX
```

```
root@6cf5a27cdb50:/home/sawtooth# python3 integer_app.py
SIGNER ES: 03c7a9edf68df7be3d39782ae0143161dd6bcc76506b2478908ae411af4f2429eb
Introduce the numbers you want to send.
1 1 1
The numbers: [1, 1, 1]
[b'\xa3eValue\x01dNamejGP3XZYUB9ZdVerbcset', b'\xa3eValue\x01dNamejP0BOLBP0V7dVerbcset', b'\xa3eValue
\x01dNamejNHBWZ1K56LdVerbcset']
root@6cf5a27cdb50:/home/sawtooth# python3 integer_app.py
SIGNER ES: 026132e13ad64ac3f49dc8123af1042787a0217f104c0a72ef2b42cb02736fe0c4
Introduce the numbers you want to send.
1 2 3 4
The numbers: [1, 2, 3, 4]
[b'\xa3eValue\x01dVerbcsetdNamejOOQ9G7DVK3', b'\xa3eValue\x02dVerbcsetdNamejJIBOBBSIHZ', b'\xa3eValue
\x03dVerbcsetdNamejQY9V42XD8O', b'\xa3eValue\x04dVerbcsetdNamejV66CW1UZXK']
root@6cf5a27cdb50:/home/sawtooth#
```

Figure 4.8: Sawtooth PoC. Submitting and subscribing to events from console.

2. To test Sawtooth integration with working apps, an application was developed in Python (the script to send transactions to the network is available on annex A.1. The app(A.1) lets the user to send the numbers entered in the console. The app uses Zero Message Queue (ZMQ) and the Sawtooth ZMQ integration interface to notify any other program when a transaction is attached to the Blockchain. This PoC can be also used to benchmark the capabilities of the blockchain, for example the time to process every transaction because there are different types of events to which the developer can subscribe which eases the debugging and testing process. The whole events system architecture is shown in figure 4.9.

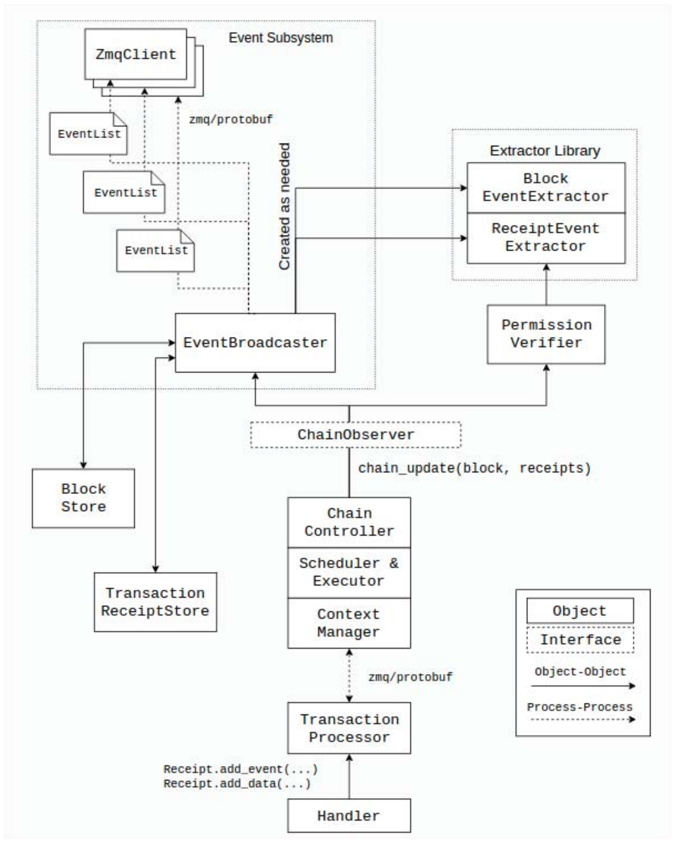


Figure 4.9: Sawtooth events system architecture.

Source: `https://sawtooth.hyperledger.org`

4.3 Tangle

This project works with a marketplace which is within the IoT frame. Sensors generate a huge amount of data that must be stored and processed. Tangle is a technology capable to work with this amount of information because it is designed to improve performance proportionally to the number of transactions. These IoT devices can broadcast data using different types of wireless technologies such as WiFi, Bluetooth, LoRa, etc..

Although Tangle does not provide an accessible and detailed documentation for developers, it has two public networks to work with: the **mainnet** and the **testnet**. The first one supports IOTA, a cryptocurrency designed for micropayments in IoT systems. The second one is target for developers, investigators and beginners.

Tangle is briefly introduced in section 2.1.5.4 but to understand this proof of concept there are some technical details that are described just below.

The network has a **coordinator** to ensure that malicious attacks cannot occur. Every minute the coordinator makes a normal transaction with its signature on it, and we call this **milestone**. Every transaction (including milestones) verifies two other transactions. A **tip** is an unconfirmed transaction, it becomes a transaction as soon as another tip verifies it but it is not a verified transaction until a milestone verifies it as it is shown in figure 4.10. When someone wants to know if a transaction is verified, he should find the newest milestone and check if it indirectly verifies the transaction. The reason that the milestones exist is because if that user just picked any random transaction, there is the possibility that the node he is connected to is malicious and is trying to trick him into verifying its transactions. The people who operate nodes cannot fake the signatures on milestones, so everybody can trust the milestones to be legit [15]. If the coordinator starts acting maliciously, nodes will just reject its milestones. There is a provisional way to keep working the network without a coordinator which is beyond the scope of this project.

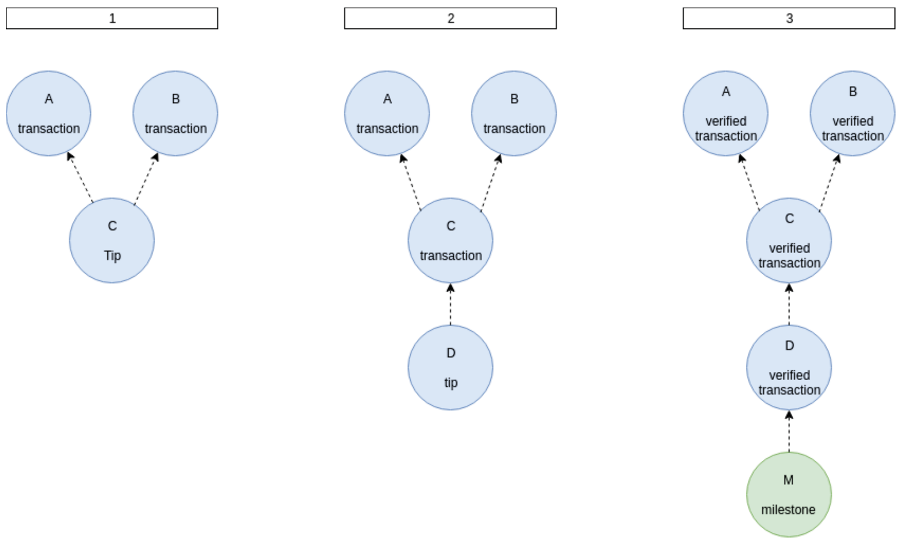


Figure 4.10: A transaction is not confirmed until a milestone verifies it (directly or indirectly)

The Tangle uses the **gossip protocol** to propagate transactions through the network. It is a communication protocol based on the way social networks disseminate information or how epidemics spread.

There is a separate technical feature of IOTA that captures a **snapshot** of all balances every determined time and prunes the history and data of all the transactions to start fresh in order to prevent the ledger from expanding too much in size. These addresses with balances act like a new genesis address, but no previous history or data will be attached.

An experimental module, **Masked Authenticated Messaging (MAM)**, is a second layer data communication protocol which adds functionality to emit and access an encrypted data stream over the Tangle regardless of the size or cost of device. IOTA's consensus protocol adds integrity to these message streams. Given these properties, MAM promises that it fulfills an important need in industries where integrity and privacy meet. If nodes are listening the channel ID (= address) in real time, the message (gossipped through the network) will be received by a subscriber when it reaches the subscriber's node. Being able to secure data's integrity and to control its access management is a prerequisite for data marketplaces [23]. MAM protects messages with symmetric key encryption. Only authorized parties will be able to read and reconstruct the entire data stream. Currently around 1.5 KB is the size of data which could be carried through the Tangle by one package and this is enough if we consider raw text information from sensors.

IOTA provides an API which can be used from an IOTA Java client that makes possible to interact with the local node and the network.

To become familiar with the Tangle a Python library for the IOTA core, Pyota, has been used[4]. It implements both the official API, as well as newly-proposed functionality such as signing, bundles, utilities and conversion.

This proof of concept is divided in four phases which are detailed just below:

1. Full-node connected to the testnet.
2. Two full-nodes connected to the mainnet.
3. Private Tangle network.
4. Some peers connected to the mainnet using MAM.

4.3.1 Full-node connected to the testnet

To fully understand how the Tangle works not just from a theoretical point of view, a full-node was set up and connected to the IOTA testnet. It was launched[5]

[4]Source code is available on `https://github.com/iotaledger/iota.lib.py`.
[5]Following the official tutorial available on `http://iota.partners/`.

in the localhost.

The most important step is the IOTA Reference Implementation (IRI) configuration. IRI is the software that an IOTA node runs. The easiest and fastest way to get IRI is by getting the already compiled version[6]. Some parameters were tuned to connect to the testnet, instead of connecting to the mainnet because our learning purposes.

In next steps, some extra services were deployed for a complete configuration:

- Nelson, which is a service that takes care of neighbours automatically. It dynamically adds nodes and synchronizes with them.
- Prometheus is another service which is used to monitor and get statistics.
- Graphana which is a dashboard builder for visualization and metrics.

The testnet ended up only providing knowledge about the tangle concepts and not for sending transactions and querying. The reason is that, since there are not many active nodes, the node gets desynchronized over and over interrupting all the processes. Therefore, we decided to test on the mainnet in the next phase.

4.3.2 Two full-nodes connected to the mainnet

This proof of concept was developed following the steps of the tutorial to set up a full IOTA node using Docker[7]. Using Docker was decided because launching the node in a VM isolates it from the localhost. It also allows to run many nodes on the same physical machine.

The first step to deploy a node, was finding neighbours to synchronize the node with the Tangle by exchanging my IP address with theirs. There are some channels in Slack and Discord to chat and find other node owners which are also finding neighbours. An Italian developer with two nodes, Nardella Antonio, was contacted and we added each other. It took 6 hours to fully synchronize with the Tangle. Being synchronized means that the last milestone identifier of the Tangle is the same as the last milestone identifier of the node. This implies that the node does not work properly and results got from queries are not guaranteed to be true.

[6]Available in `https://github.com/iotaledger/iri/releases`.

[7]Available on `https://github.com/ioiobzit/iota-nelson-node/`.

Once the node was synchronized we sent some transactions (with no fees and value 0 IOTAs) to the Tangle with the message: "Hello from CoNWeT!". The process to attach any transaction to the Tangle is:

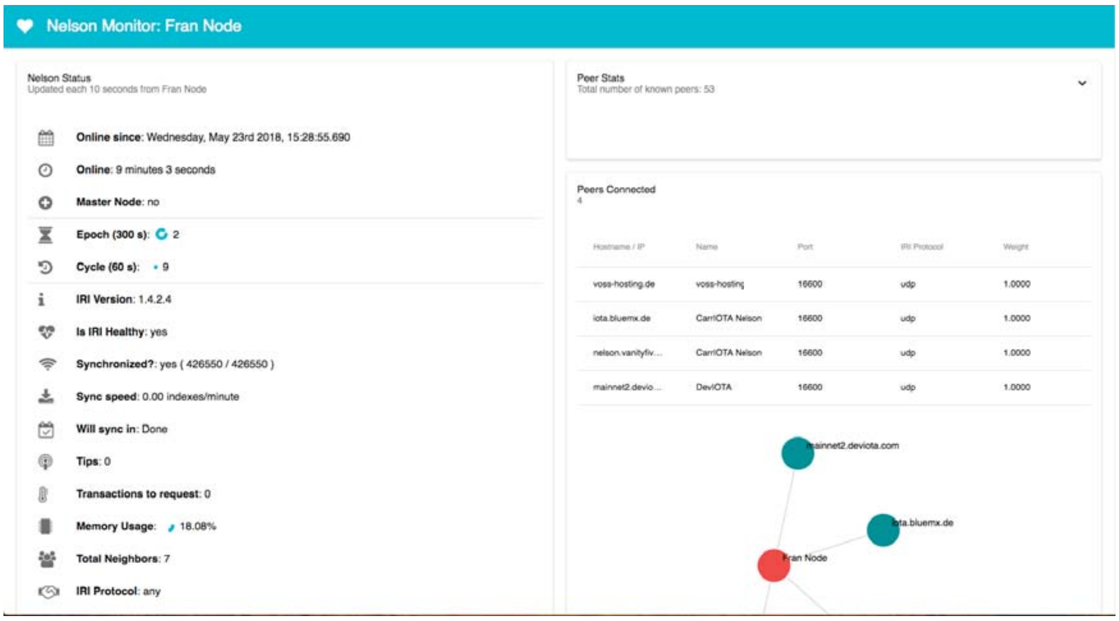
1. The node creates the transaction and sign it with its private key.
2. Find two tips using `getTransactionsToApprove` which returns the a `trunkTransaction` and `branchTransaction`. These are the transactions that the node is going to validate and reference with this transaction. It uses the Random Walk Monte Carlo (RWMC) algorithm. After selecting them, the node checks if the two transactions are not conflicting and does some Proof of Work (PoW) by solving a cryptographic puzzle (curl).
3. Send the transaction to the Tangle using `attachToTangle` API call. It returns a value formed by `trunkTransaction` + `branchTransaction` + `nonce` which is the valid trytes accepted.
4. Broadcast the transactions to all neighbours using the `broadcastTransactions` API call with the value obtained by the previous step.

After having one node fully working in the Tangle mainnet monitored as it is shown in figure 4.11, a second node was set up to test communication between them. The most difficult part was querying the ledger to find a specific transaction because this network just supports cryptocurrency transactions which are simple economic balances of IOTA tokens. This raw configuration of both nodes does not implement any way to control data access or a transaction privacy. As we are developing a data marketplace, there are two ways to send more complex data and to regulate its access: through a customized private Tangle network or using MAM. These are exactly the following proofs of concept.

4.3.3 Private Tangle network

The network was deployed using VMs where Docker Containers were launched. This test simulates a real scenario in which nodes may be working in a cluster of computers. We used Docker Swarm to achieve orchestration between different worker VMs leaded by the manager they were associated with. This adds an abstraction layer because the Manager hides the worker nodes that work for it. The manager in fact represents the node itself and it works as a load balancer.

We deployed the network manually at first but it was automated by developing a bash script (A.2) that does the whole process. This automatization lets us



Nelson Monitor: Fran Node

Nelson Status
Updated each 10 seconds from Fran Node

Online since: Wednesday, May 23rd 2018, 15:28:55.690
Online: 9 minutes 3 seconds
Master Node: no
Epoch (300 s): 2
Cycle (60 s): 9
IRI Version: 1.4.2.4
Is IRI Healthy: yes
Synchronized?: yes (426550 / 426550)
Sync speed: 0.00 indexes/minute
Will sync in: Done
Tips: 0
Transactions to request: 0
Memory Usage: 18.08%
Total Neighbors: 7
IRI Protocol: any

Peer Stats
Total number of known peers: 53

Peers Connected
4

Hostname / IP	Name	Port	IRI Protocol	Weight
voss-hosting.de	voss-hosting	16600	udp	1.0000
iota.bluemx.de	CarrIOTA Nelson	16600	udp	1.0000
nelson.vanityfiv...	CarrIOTA Nelson	16600	udp	1.0000
mainnet2.devio...	DevIOTA	16600	udp	1.0000

Figure 4.11: A full-node synchronized with the Tangle mainnet (milestone 426550) connected with 7 neighbours

to deploy many nodes faster and without errors. Before running the script, the `iri.jar` executable is needed. It is available online but the recommendation is to compile it from the source code following these steps:

1. Cloning the source code from the official repository[8].
2. The maximum number of allowed peers must be updated because it is 0 by default. There is a file in the following path `/src/main/java/com/iota/iri/conf` called `Configuration.java` in which the variable `MAX_PEERS` allows the user to decide the amount.
3. Comment out the check snapshot signature part in `src/main/java/com/iota/iri/snapshot.java` (lines 50-82).
4. Build the jar using Maven. In the root folder execute these commands:

```
mvn clean compile
mvn package
```

Afterwards, running the script requires three parameters:

[8]`https://github.com/iotaledger/iri`

- The number of managers, which is equal to the quantity of nodes in the network.
- The number of workers per node, which is the amount of VMs given to each node to process the transactions.
- A path to a folder in the localhost which contains: the `iri.jar`, the `milestone_generator.jar`, the container `Dockerfile` and the `docker-compose.yaml`.

The script creates each VM, assigns them an IP, copies the necessary files, adds the nodes as neighbours and deploys services as a Docker Swarm in an overlay network. It also designates the first manager as the coordinator of the network. This node is in charge of launching milestones every minute.

The efficiency of this network was tested with different number of nodes, bundles and transactions. The conclusion from these tests is that for a private little network, is very difficult to keep a good balance between the number of nodes and the volume of data stored in the ledger. This implies that Tangle cannot assure that 51% attack or double spending will not happen. Additionally, the difficulty to model complex assets instead of just cryptocurrency tokens supports the decision to discard Tangle as a feasible technology to implement the data distribution functionality. The next prototype was proved using MAM which theoretically solves these problems.

4.3.4 Some peers connected to the mainnet using MAM

This last proof of concept with Tangle was developed thanks to Robert Lie's slides and video tutorials [33]. This section briefly describes some technical details about MAM which are important for its understanding.

A Masked Authentication Messaging is a work in progress module build on top of IOTA that allows to send encrypted messages from authenticated parties. It securely stores messages in the Tangle, each on a separate address so only authorised parties will be able to read and reconstruct the entire message stream. It has three connotations:

- The message is encrypted (**Masked**).
- The message is confirmed to be coming from the device (**Authenticated**).

- A continuous message stream is created on the Tangle until the device stops publishing the data (**Messaging**).

In MAM every message has a signature and references to the next one so there is a encrypted message stream. The channel ID is called *root* so any message is attached to the Tangle using the root.

We used the Masked Authentication Messaging Demo[9] which uses the JavaScript MAM client for web applications[10]. As it was said before, it is under development and may have some changes in the future. It works on WebAssembly which is a new binary format for executing code on the web and is programmed in Rust. This API works with the concept of *MAM object* which has the necessary fields to chain messages into the Tangle. The most relevant field for this proof is the mode. Although to view the payload (readable message, not trytes) the root is needed in every mode, there are three different modes: *public*, *private* and *restricted*.

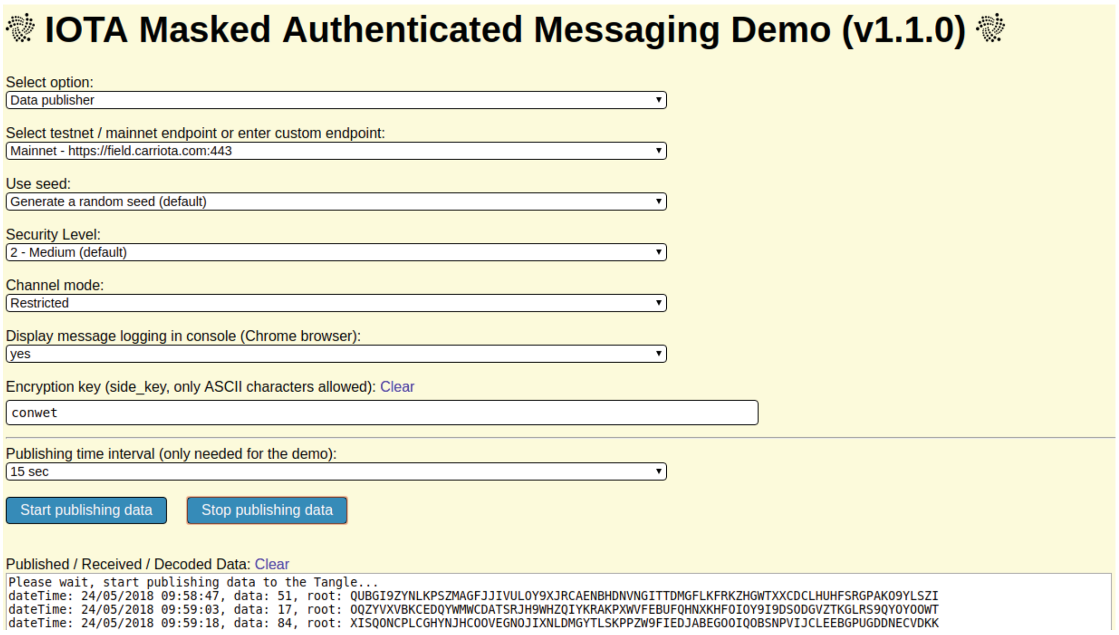
- **Public**. Messages can be accessed by anybody using the address because the address is the root.
- **Private**. Messages can only be unencrypted by users who know the right root and the root cannot be deducted from the address due to the hash is a one way function. The address is the hash of the root.
- **Restricted**. Messages can only be unwrapped by users who have the `side_key` which is used to encrypt the message. The address is the hash of the root.

In this proof of concept, four peers are connected to the Tangle. For this proof all of them use an available node of the mainnet for testing educational purposes from Carriota [11]. Each of them plays a different role to test how MAM works:

1. The data publisher (see figure 4.12) sends information to the Tangle simulating that it receives the data every 15 seconds from an IoT device. It uses the restricted channel mode with an encryption key (`side_key`) which must be shared with the pertinent peers to be allowed to access the data. They also need the `root` which is a field that identifies the transaction. It hashes some information related to the transaction itself and also to the

[9]Available on `https://www.mobilefish.com/services/cryptocurrency/mam.html`
[10]`https://github.com/iotaledger/mam.client.js`
[11]`http://carriota.com`

next one. Including this information makes the difference between sending an isolated data transaction from sending a data stream which is treated as set of chained transactions. An interested peer just need the root of the first transaction of the data stream to be subscribed to receive all the information.



IOTA Masked Authenticated Messaging Demo (v1.1.0)

Select option: Data publisher

Select testnet / mainnet endpoint or enter custom endpoint: Mainnet - https://field.carriota.com:443

Use seed: Generate a random seed (default)

Security Level: 2 - Medium (default)

Channel mode: Restricted

Display message logging in console (Chrome browser): yes

Encryption key (side_key, only ASCII characters allowed): Clear

conwet

Publishing time interval (only needed for the demo): 15 sec

Published / Received / Decoded Data: Clear

```
Please wait, start publishing data to the Tangle...
dateTime: 24/05/2018 09:58:47, data: 51, root: QUBGI9ZYNLKPSZMAGFJJIVULOY9XJRCAENBHDNVNGITTDMGFLKFRKZHGWTXXCDCLHUHFSRGPAKO9YLSZI
dateTime: 24/05/2018 09:59:03, data: 17, root: OQZYVXVBKCEDQYWMWCDATSRJH9WHZQIYKRAKPXWVFEBUFQHNXKHFOIOY9I9DSODGVZTKGLRS9QYOYOOWT
dateTime: 24/05/2018 09:59:18, data: 84, root: XISQONCPLCGHYNJHCOOVEGNOJIXNLDMGYTLSKPPZW9FIEDJABEGOOIQOBSNPVIJCLEEBGPUGDDNECVDKK
```

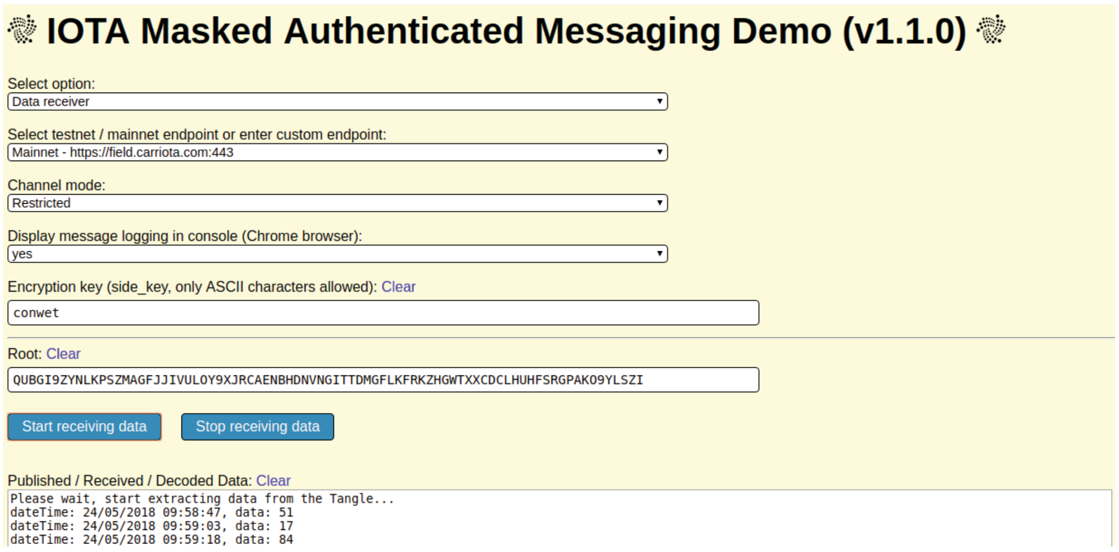
Figure 4.12: Data publisher in the Tangle. MAM Demo

2. The data receiver in restricted mode mode knows the encryption key and the root of the first message. It can decypher the message and follow the stream properly as it is shown in figure 4.13.

3. The data receiver in private mode only knows the root but not the `side_key`. This works like in a radio communication where both peers are not in the same frequency, messages are invisible for the receiver as figure 4.14 demonstrates.

4. The data receiver in public mode only discovers data which is not encrypted. As the previous peer, the figure 4.15 shows that no data is extracted from the Tangle even knowing the root.

Afterwards, some scripts were implemented to create a customized client which can be found in annex A.3. Of these tests it follows the satisfactory conclusion that Tangle with MAM is a technology which fits the requisites of a decentralized data distribution for a complex data marketplace.



Select option:
Data receiver

Select testnet / mainnet endpoint or enter custom endpoint:
Mainnet - https://field.carriota.com:443

Channel mode:
Restricted

Display message logging in console (Chrome browser):
yes

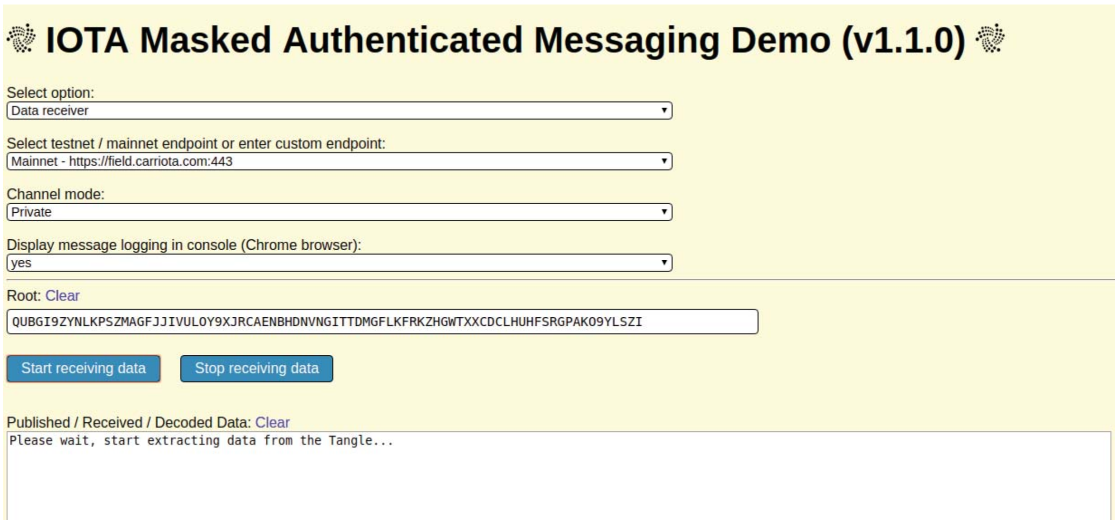
Encryption key (side_key, only ASCII characters allowed): Clear
conwet

Root: Clear
QUBGI9ZYNLKPSZMAGFJJIVULOY9XJRCAENBHDNVNGITTDMGFLKFRKZHGWTXXCDCLHUHFSRGPAKO9YLSZI

Published / Received / Decoded Data: Clear

```
Please wait, start extracting data from the Tangle...
dateTime: 24/05/2018 09:58:47, data: 51
dateTime: 24/05/2018 09:59:03, data: 17
dateTime: 24/05/2018 09:59:18, data: 84
```

Figure 4.13: Data receiver in restricted mode in the Tangle. MAM Demo



Select option:
Data receiver

Select testnet / mainnet endpoint or enter custom endpoint:
Mainnet - https://field.carriota.com:443

Channel mode:
Private

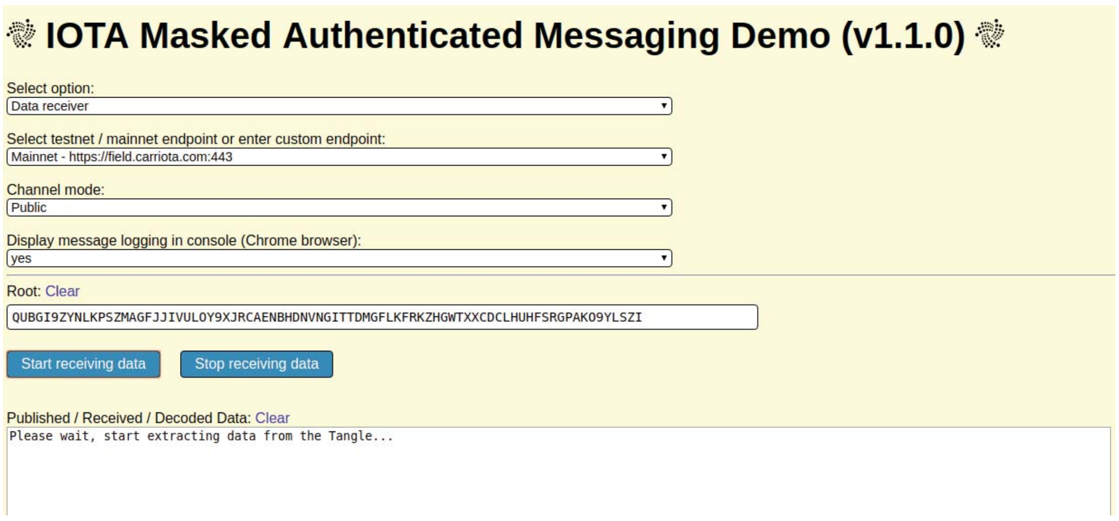
Display message logging in console (Chrome browser):
yes

Root: Clear
QUBGI9ZYNLKPSZMAGFJJIVULOY9XJRCAENBHDNVNGITTDMGFLKFRKZHGWTXXCDCLHUHFSRGPAKO9YLSZI

Published / Received / Decoded Data: Clear

```
Please wait, start extracting data from the Tangle...
```

Figure 4.14: Data receiver in private mode in the Tangle. MAM Demo



IOTA Masked Authenticated Messaging Demo (v1.1.0)

Select option:
Data receiver

Select testnet / mainnet endpoint or enter custom endpoint:
Mainnet - https://field.carriota.com:443

Channel mode:
Public

Display message logging in console (Chrome browser):
yes

Root: Clear

```
QUBGI9ZYNLKPSZMAGFJJIVULOY9XJRCAENBHDNVNGITTDMGFLKFRKZHGWTXXCDCLHUHFSRGPAKO9YLSZI
```

Published / Received / Decoded Data: Clear

```
Please wait, start extracting data from the Tangle...
```

Figure 4.15: Data receiver in public mode in the Tangle. MAM Demo

5 Conclusions

Once at the top, I felt that my life was going to turn around.

Maurice Herzog,
first man in climbing an 8K high mountain

In this chapter the outcome of this Master's Thesis is presented. It describes the results and conclusions derived from the work that have been carried out in the context of this project. In this regard, in the first section, a DFD has been designed to present the different criteria involved in the architecture design when decentralizing an existing data marketplace or creating a new decentralized one from scratch. Additionally, this section discuses the advantages and disadvantages of the proposed solutions and the chosen technologies. The second section reflects, from a theoretic point of view, on the actual implantation of blockchain in the FIWARE project and in general in the current business models.

Before starting to evaluate the results and conclusions, I want to remark that blockchain is a new technology. It means that it is still in development so it is changing continuously. Conceptually, there are very big differences between technologies. Nothing is well documented because there is a great lack of knowledge and experts in the field. Moreover, blockchain is a very hyped term because cryptocurrencies are moving a lot of money, so it is important to filter the existing information by checking sources and testing by oneself. Furthermore, there are lots of theoretical ideas which are actually impossible to deploy. All these difficulties add value to the development of this project, which has been a great challenge.

5.1 Strategic architecture analysis for decentralized data marketplaces

As it is clear from the work done, a data marketplace can be divided - from an architectural point of view - in three independent modules: *payment*, *catalog* and *data distribution*. Depending to the requirements and objectives of each marketplace, it could be interesting to decentralize one of the modules, two or the three of them. Even if the decision is to decentralize the marketplace completely, it is not necessary to use the same technology for every module. Moreover, it is best to use a specific technology for each one. This section analyses module by module its possible decisions and discusses their advantages and disadvantages.

The first module is the **payment** one. The decision taken in this module acts an inflection point for the architecture design. There are two different essential options which completely condition all the other decisions: integrating payment with cryptocurrency or not.

- If not, this module is easily outsourced to any popular economic platform such as Paypal or WePay. It is important to remark that in this case the catalog technology should be a permissioned blockchain which identifies every user for authorizing and auditing the payment.
- If using cryptocurrency is chosen, there are two well difference ways:
  - Choosing an existing cryptocurrency such as Ethereum, Bitcoin or any other. It simplifies the payment and adds value to the marketplace because it is decentralized to the maximum. There are some ones, like IOTA, which are specifically thought for mycropayments. There are still reluctance on the part of many users regarding the use of cryptocurrencies so an exhaustive market study should be done before choosing any of them. In some cases, like Ethereum, it is allowed to deploy the marketplace on the proper cryptocurrency network and this means that perhaps is possible to attract new users who are already in the network.

– Creating a new token and a designing how it works. Surprisingly, this is the most popular option between emerging startups and new business which are trying to monetize data. It presents many advantages in terms of customization but it is important to highlight the difficulty of carrying it out successfully. There are some mature technologies which are worthy to choose, even with their restrictions and limit scope, because their enhanced security.

The **catalog** module can be seen as a particular case from a general digital marketplace where there are sellers and customers who exchange digital assets. This is a common scenario and there are very customizable commercial blockchain solutions.

The most important feature to figure out is privacy. In most cases information relative to the trades should be only accessible for the participants who take part in. There are some ways to cypher the information with the public key of the other to preserve confidentiality. The most disrupting and secure method is using communication channels, as Hyperledger Fabric proposes. This means that there is one specific ledger for each seller-customer pair. The inconvenience is that ledgers cannot interact between them. Sometimes crossing information is essential for the marketplace. It is important to measure up which is the priority in each case. Other proposal, performed by Hyperledger Composer, to grant confidentiality is having programmatic access control to all the records in the ledger by using ACLs which define the permissions and some checks in the smart contracts.

The last module is **data distribution**. Since we live in the information era where data is the new oil, it is more important than ever to keep the control of the access preserving privacy, to store it properly ensuring integrity and to grant its availability.

Currently, there are many P2P data distribution systems because a third of trust bothers especially. How can someone assures that the information you are giving to the central party is not being distributed to anyone else? How can someone audit that the central party is not using it for lucrative purposes in a dishonest way? There are questions that go beyond: how can someone be sure that when he has sold information, the buyer will not resell it?

Current traditional systems do not have any reassuring response for these concerns, neither P2P which avoids the central figure. This is why precisely the use of blockchain is justified to distribute data. Blockchain can provide *security* (BFT), *confidentiality* (anonymity and MSP) and *integrity* (immutability of the ledger). If we add to these parameters *availability*, the balance is unbalanced. Consensus algorithms and synchronization are not efficient and fast. This is not a problem when the kind of data which is being commercialized is static or historical. The big inconvenience is working with streaming data because a high velocity response is required to the system, such as in an emergency medical context. For this cases, where availability is a priority, there are some technologies which sacrifices security to achieve a high performance. The conclusion of this project is that combining some of them can reach the balance we are looking for.

The DFD shown in figure 5.1 has been elaborated as a resume of all the achieved results. It sketches a guide to help in the decision making process to choose the most adequate architecture for the data distribution decentralization. We start from the premises that the catalog module is already decentralized using Hyperledger Composer on an Hyperledger Fabric instance and that the payment module is outsourced.

Considering this last premise, if payment is implemented using a cryptocurrency such as Ethereum, data can be distributed through the same ledger. On the one hand, this option could be very interesting if it is an own-designed ledger with its own cryptocurrency because it is supposed to optimize performance according to the pricing models, the participants needs and the type of data (yellow in the figure 5.1. On the other hand, limiting the marketplace to only one cryptocurrency is a disadvantage.

If using cryptocurrencies is not the chosen option because of the market environment or other reasons, the next decision to take depends on the kind of data, its volume and specially, the required availability.

- Historical static data which does not need to be send and accessed on streaming can be distributed P2P (orange) or in the same ledger as the catalogue, Hyperledger Composer, if accounting is necessary (blue). It is remarkable that P2P data distribution is not trusty and auditable. There are some formulas to partially solve this lack of security like registering meta-information about the data shipments in the catalogue ledger.
- Streaming data that are continuously coming from IoT devices can generate large volumes of information. Distributing it on the same catalogue ledger would not be efficient because such a big amount of transactions would slow down the system. Low velocity is not acceptable for emergency systems or real time systems.

  If we are in this situation, the next step is to look at the privacy and confidentiality requirements. Sometimes, cyphering messages so just implicated participants have the keys, is enough. In these situations a

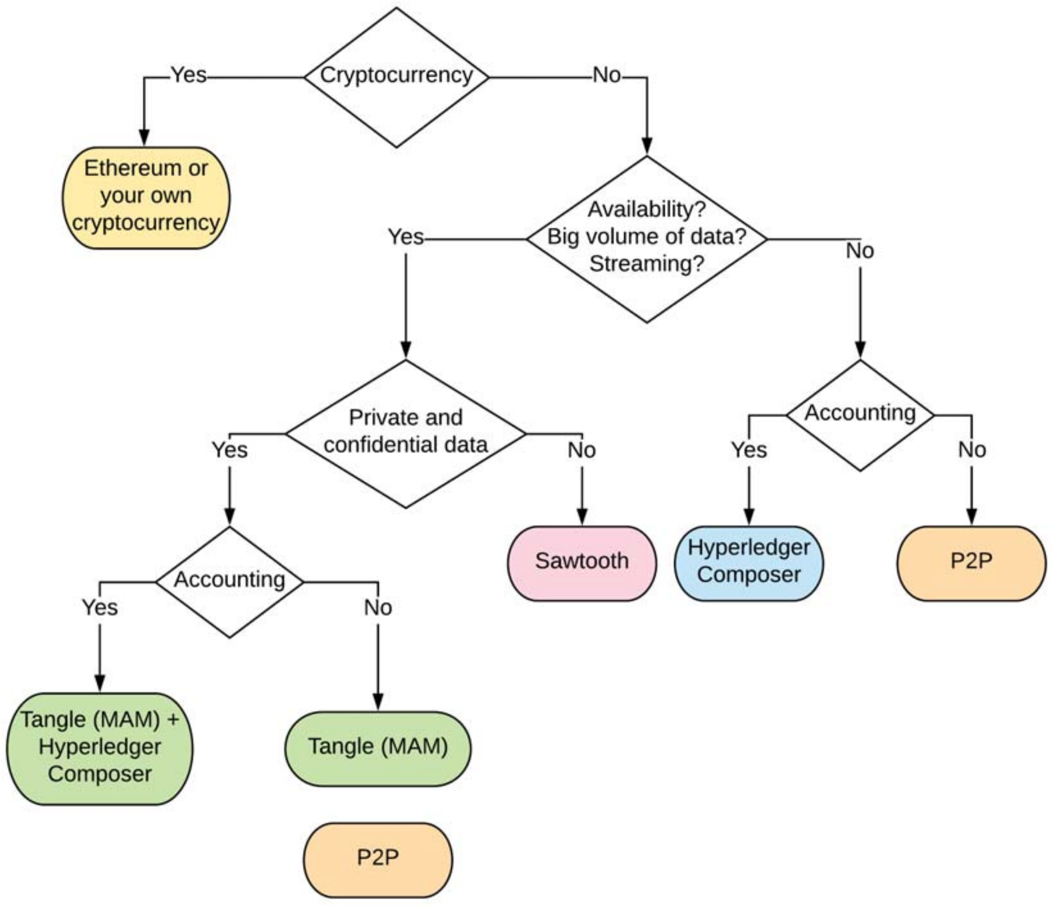

Cryptocurrency
Yes
No
Ethereum or your own cryptocurrency
Availability? Big volume of data? Streaming?
Yes
No
Private and confidential data
Yes
No
Accounting
Yes
No
Sawtooth
Hyperledger Composer
P2P
Accounting
Yes
No
Tangle (MAM) + Hyperledger Composer
Tangle (MAM)
P2P

Figure 5.1: Decentralized data distribution DFD

technology like Sawtooth (pink) fits perfectly as a solution. In other circumstances, transactions must be invisible, not just cyphered, for the other participants. In these cases, data can be distributed using Tangle (green) which precisely improves performance when more transactions are submitted into the ledger. Other option, with the disadvantages mentioned above, is P2P (orange).

There are some pricing models which requires accounting (like usage payment). Tangle works doing periodically snapshots so it looses previous information. To provide a trust accounting avoiding a central database (either local or on cloud), the solution is to submit transactions before each snapshot into the ledger where the catalog is. The combination of Tangle (MAM) and Hyperledger Composer provides a efficient data distribution and a legit accounting solution.

5.2 Blockchain and the current business models

From the perspective of the study carried out up to this moment, it is interesting to propose a theoretical reflection. It is the moment to make some comments about our use case, the actual implantation of blockchain in the FIWARE project.

FIWARE is a big European project. It is complex and many diverse participants are involved, with the consequent difficulty to reach an agreement in such a radical change. Despite this, one of its most outstanding features is innovation. FIWARE is committed to the most recent technologies such as IoT and is able to stay on the crest of the wave. The project devotes a large part of its benefits to research, in fact, several universities are involved.

Decentralizing the BAE according to the architecture proposed by the fifth scenario would add great value to the marketplace. The most notable benefits are:

- Being the first ones in completely decentralize such a complex data marketplace. By having it working in a real scenario, new ways of exploiting this technology will arise and that could be the origin of new data monetization models.

- Being able to promise users security, confidentiality, availability and trust without asking for confidence in the project itself. It implies caption of new community members attracted by the innovation of the technology and its garantees. The direct consequence is an increase in data trading volume.

- Elimination of problems associated with fraudulent actions or attempts thereof. For sure, it will increase its reputation and there will be more investors.

Of course, there are some problems associated with the integration of blockchain in the project. First, the resistance to change of existing users in the network. This common effect will be exaggerated by the dubious reputation that precedes DLT-based systems. Secondly, we cannot ignore the conflict that the immutability of this technology presents with the new reform of the data protection law. Users who take part on any DLT-based system must expressly waive their right to delete personal information.

Linking with the issue of the importance of personal information and as a consequence of the research around the data markets carried out throughout this project, it should be noted that they are a quarry to be exploited. The purchase of information on secure platforms dedicated to this is a business which is beginning to gain importance.

Big data, machine learning, prediction and recommendation systems, artificial intelligence and IoT are here. All of these new technologies work with data. Smart cities and Industry 4.0 are now a reality. The devices that work with these technologies generate large amounts of information whose processing and analysis allows to obtain really useful results.

Actually, blockchain commercial solutions are still immature. Currently, the applications that are making money with blockchain are cryptocurrencies exchange houses. But blockchain is not just the technology behind a cryptocurrency. Precisely this project illustrates a way to exploit this technology in a completely separate field, the commercialization of data. There are multiple business use cases where blockchain can be a revolution: insurances and financial companies taking advantage of smart contracts, production supplying chains to prove the origin and the treatment of natural products, electoral systems which be fully auditable, etc.

Blockchain will break the business model schemes that are working today. All of them have in common a central regulatory entity that will tend to disappear. Or adapt and reinvent or become obsolete. This is the reason why many big companies are developing DLT-based projects which will be launched soon. Also, lots of startups are emerging with ideas related with blockchain because it is a mine of new business ideas. We are living in the dawn of the establishment of decentralized systems.

The developers community is working hard in researching and creating open-source projects related to blockchain and other DLTs. The Linux Foundation with Hyperledger provides all the necessary tools to learn and test. Universities are also collaborating by sharing knowledge in web-seminars, thesis and papers.

In general, there is a simultaneous fear and fascination with this new technology. It has a lot of potential. There are many business cases in which it could be applied, providing a lot of value. Definitely, it is worth investing time and money in blockchain research.

6
Future Work

We shouldn't delay forever until every possible feature is done. There's always going to be one more thing to do.

Satoshi Nakamoto

Throughout this investigation, new collateral ideas have been continuously arising. They have been discarded, or at least postponed, to maintain a single line of research that in itself was sufficiently complex. In this chapter, the most interesting ideas are collected. They are possible future work to expand this thesis and add value to the data marketplace.

Final implementation of the proposed architecture

The immediate next step is to actually implement the decentralized Bizapp which interacts with the two ledgers. Prototypes and proofs of concept from this thesis should be put together and developed in greater detail to make a commercial solution.

In addition, quality and acceptance tests should be carried out. It would also be necessary to design a plan for the launching process with the necessary steps so that all the information and the current participants of the marketplace could be migrated without any unforeseen events.

The implementation of the payment module of the platform is beyond the scope of this project but even to outsource the payment process, it is necessary to look for compatible alternatives that follow, as far as possible, the blockchain's decentralization philosophy.

Regarding to the catalogue and the data distribution modules, the modelization is already designed and tested. It would be promising to conduct a market analysis with research on new pricing models. Subscriptions to streams of data should be implemented, as well as notifications when some conditions are matched. For example, to receive an alarm when the temperature sensor in a freezing camera exceeds 5 degrees.

Technical improvements

The DLTs included in this research are still under development so they are increasing their functionalities while improving the existing ones. As of today, Tangle does not have smart contracts but the appearance of Qubic[1] has already been announced. It is a new module which integrates oracles, smart contracts and outsourced computations. Obviously it is an immediate line to explore.

Hyperledger encompasses a multitude of projects, many of them are currently being incubated and may soon be more suitable for this marketplace. For this reason the designed architecture is modular. This allows to change the technologies of any module without affecting the rest of the project.

It is very important to be aware of the news that are appearing. Specially advances in the communication and integration between different blockchain technologies could contribute more value to our project. The interaction between both ledgers, both to check if there are active contracts and to perform the accounting, could be automatic without the intervention of our application. This would improve performance and of course, confidence in the system. If this feature does not appear soon, developing it would be an awesome challenge.

[1]`https://qubic.iota.org`

Knowledge marketplace

Taking advantage of smart contracts and the availability of the information stored in the ledger, statistical operations and predictive analysis could be performed directly as data from sensors arrive. It means that, in addition to purchase data, the marketplace could sell the knowledge acquired from data science. This would allow customers to make complex queries and take advantage of cross-information between different datasets.

Big data stream processing is trend for achieving analytics in real time. From data streams it is possible to get instant analytics results using platforms like Spark Streaming [45]. It is useful for tasks like fraud detection because it permits to detect anomalies and to stop possible fraudulent transactions before they are committed. The following figure gives a detailed explanation how Spark process data in real time because it analyzed the data before it hits disk.

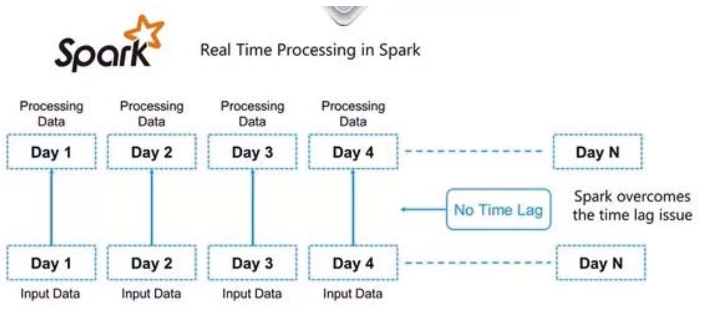


Figure 6.1: Spark data process in real time.

Source: Vaseekaran's article[45]

As it can be imagined, this would involve a higher computational cost, which is why new approaches should be made to deploy nodes with greater capabilities. It would also be necessary to design a model in which the owners of the datasets decide whether they make the data available to the common processing engine or keep them in private mode. For this, a system of incentives should be proposed for those who share the information.

Bibliography

[1] API Umbrella (2015). API Umbrella - Open Source API Management.

[2] Back, A. (2002). Hashcash -A Denial of Service Counter-Measure.

[3] Bitcoin Developer Guide (2016). https://bitcoin.org/en/developer-guide#p2p-network. Date Aceessed: 25/04/2018.

[4] Bitcoin Hashcash (2017). https://en.bitcoin.it/wiki/Hashcash. Date accessed: 15/04/2018.

[5] Bitcoin Wisdom (2013). https://bitcoinwisdom.com/bitcoin/difficulty. Date accessed: 15/04/2018.

[6] Brown, R. G., Carlyle, J., Grigg, I., and Hearn, M. (2016). Corda: An Introduction.

[7] Buterin, V. (2013). Ethereum White Paper.

[8] Buterin, V. (2014). https://blog.ethereum.org/2014/05/06/daos-dacs-das-and-more-an-incomplete-terminology-guide. Date accessed: 20/04/2018.

[9] Buterin, V. (2016). https://github.com/ethereum/wiki/wiki/Proof-of-Stake-FAQ. Date Accessed: 25/04/2018.

[10] Cachin, C. and Vukoli, M. (2017). Blockchain Consensus Protocols in the Wild.

[11] de la Vega García, F. (2015). A proposal for a Business Framework targeted at emerging business services and applications.

[12] Dominik Schiener (2017). https://blog.iota.org/a-primer-on-iota-with-presentation-e0a6eb2cc621. Date Accessed: 27/04/2018.

[13] Dwor, C. and Naor, M. (1993). Pricing via Processing or Combating Junk Mail.

[14] Ethereum (2018). https://www.stateofthedapps.com. Date accessed: 20/04/2018.

[15] Eukaryote (2017). https://www.reddit.com/r/Iota/comments/7c3qu8 Date Accessed: 18/05/2018.

[16] FIWARE Academy (2016). Course: Business API Ecosystem.

[17] FIWARE Academy (2017). FIWARE Data Models.

[18] FIWARE CKAN Extensions (2017). https://catalogue.fiware.org/enablers/fiware-ckan-extensions. Date Accessed: 27/04/2018.

[19] FIWARE Lab (2017). List Revenue Sharing Models. Technical report.

[20] Google Developers (2017). https://developers.google.com/protocol-buffers/ Date Accessed: 05/05/2018.

[21] Gramoli, V. (2017). From blockchain consensus back to Byzantine consensus. *Future Generation Computer Systems*.

[22] Guy Zyskind, Alex Sandy Pentland, and Nathan Oz (2015). Enigma: Decentralized Computation Platform with Guaranteed Privacy.

[23] Handy, P. (2017). https://blog.iota.org/introducing-masked-authenticated-messaging-e55c1822d50e Date Accessed: 18/05/2018.

[24] Hearn, M. (2016). Corda: A distributed ledger.

[25] Hierro, J. (2017). FIWARE for smart cities - transforming cities into engines of growth.

[26] Hyperledger Architecture Working and (WG), G. (2017). Introduction to Hyperledger Business Blockchain Design Philosophy and Consensus.

[27] Hyperledger Fabric (2018). https://hyperledger-fabric.readthedocs.io/en/release-1.1/blockchain.html Date Accessed: 29/05/2018.

[28] Hyperledger Sawtooth (2017). Sawtooth Lake 0.7 documentation.

[29] Iansiti, M. and R. Lakhani, K. (2017). The Truth About Blockchain. *Harvard Business Review*, page 9.

[30] IBM Cloud Bluemix (2017). https://www.ibm.com/cloud-computing/bluemix/es Date Accesed: 04/05/2018.

[31] Johnston, D. (2015). https://github.com/DavidJohnstonCEO Date accessed: 18/04/2018.

[32] Kynan Rilee (2018). https://medium.com/kokster/understanding-hyperledger-sawtooth-proof-of-elapsed-time-e0c303577ec1. Date Accessed: 29/04/2018.

[33] Lie, R. (2018). Masked Authentication Message - IOTA. Technical report.

[34] Monax Community (2014). Monax.

[35] Nakamoto, S. (2008). Bitcoin: A Peer-to-Peer Electronic Cash System.

[36] Nxt Community (2014). NxtWhitepaper_v122_rev4.pdf.

[37] Olson, K., Bowman, M., Mitchell, J., Amundson, S., Middleton, D., and Montgomery, C. (2018). Sawtooth: An Introduction.

[38] Orion Context Broker (2016). https://catalogue.fiware.org/enablers/publishsubscribe-context-broker-orion-context-broker Date Accessed: 26/04/2018.

[39] Perry, J. S. (2018). Hyperledger Composer basics, Part 3: Deploy locally, interact with, and extend your blockchain network Exploring more Hyperledger Composer tools, UIs, and security.

[40] Preukschat, A. (2017). Los tipos de Blockchain: públicas, privadas e híbridas.

[41] Preukschat, A., Kuchkovsky, C., Gómez Lardies, G., Díez García, D., and Molero, I. (2017). *La revolución industrial de Internet.*

[42] Sawtooth (2017). PoET 1.0 Specification https://sawtooth.hyperledger.org/docs/core/releases/latest/architecture/poet.html Date Accessed: 03/05/2018.

[43] Steven Buchko (2017). https://coincentral.com/how-long-do-bitcoin-transfers-take/ Date Accesed: 10/05/2018.

[44] Stewards, I. (2014). Slimcoin: A Peer-to-Peer Crypto-Currency with Proof-of-Burn.

[45] Vaseekaran, G. (2017). https://medium.com/@gowthamy/big-data-battle-batch-processing-vs-stream-processing-5d94600d8103 Date Accessed:05/05/2018.

[46] Wibson (2017). https://medium.com/wibson/introducing-wibson-a-decentralized-data-marketplace-empowering-individuals-to-monetize-their-own-7747b59bfef1 Date Accessed: 27/04/2018.

[47] Zyskind, G., Nathan, O., and Pentland, A. S. (2015). Decentralizing privacy: Using blockchain to protect personal data. *Proceedings - 2015 IEEE Security and Privacy Workshops, SPW 2015*, pages 180–184.

A
Scripts

A.1 Sawtooth scripts

Sending transactions script

```
import sys
import cbor
import string
import random
from urllib.error import HTTPError
from random import randint
import urllib.request

from hashlib import sha512

from sawtooth_sdk.protobuf.transaction_pb2 import TransactionHeader
from sawtooth_sdk.protobuf.transaction_pb2 import Transaction
import sawtooth_sdk
from sawtooth_signing import create_context
from sawtooth_signing import CryptoFactory
from sawtooth_sdk.protobuf.batch_pb2 import BatchHeader
from sawtooth_sdk.protobuf.batch_pb2 import Batch
from sawtooth_sdk.protobuf.batch_pb2 import BatchList

#Creates a signer who will sign the tx and validate
#its identity in front of the validator
context = create_context('secp256k1')
private_key = context.new_random_private_key()

signer = CryptoFactory(context).new_signer(private_key)
print("SIGNER ES: {}".format(signer.get_public_key().as_hex()))

#Checks for correct amount of parameters
if len(sys.argv) != 3:
        print("Use: ./send_tx NumOfBatches NumOfTxPerBatch")
        sys.exit(1)

NUM_BATCHES = int(sys.argv[1])
NUM_TX_PER_BATCH = int(sys.argv[2])

#Generate the payload of the tx
#Every family has a structure to follow
#Uses the cbor format

N = 10

ran_addr = [
        [
        ''.join(random.choice(string.ascii_uppercase + string.digits)
        for _ in range(N))
        for j in range(NUM_TX_PER_BATCH)
        ] for i in range(NUM_BATCHES)
]

payload_arr = [
        [{
                'Verb': 'set',
                'Name': ran_addr[i][j],
                'Value': randint(0, 30000)
         } for j in range(NUM_TX_PER_BATCH)
        ] for i in range(NUM_BATCHES)
]

#Generate the bytes of the payload
payload_bytes_arr = [
        [
        cbor.dumps(payload_arr[i][j])
        for j in range(NUM_TX_PER_BATCH)
        ] for i in range(NUM_BATCHES)
]

#Generate an integerkey address holding 'example'
tx_addr = [
        [
        sha512('intkey'.encode('utf-8')).hexdigest()[0:6]
        + sha512(ran_addr[i][j].encode('utf-8')).hexdigest()[-64:]
        for j in range(NUM_TX_PER_BATCH)
        ] for i in range(NUM_BATCHES)
]
print("LA DIRECCION ES: {}".format(tx_addr))

#Generate the tx header
tx_header_arr = [
        [
        TransactionHeader(
                family_name='intkey',
                family_version='1.0',
                inputs=[tx_addr[i][j]],
                outputs=[tx_addr[i][j]],
                signer_public_key = signer.get_public_key().as_hex(),
                batcher_public_key = signer.get_public_key().as_hex(),
                dependencies=[],
                payload_sha512 =
                        sha512(payload_bytes_arr[i][j]).hexdigest()
                ).SerializeToString()
        for j in range(NUM_TX_PER_BATCH)
        ] for i in range(NUM_BATCHES)
]

#Signer signs the header:
```

```
signature_arr = [
        [
                signer.sign(tx_header_arr[i][j])
        for j in range(NUM_TX_PER_BATCH)
        ] for i in range(NUM_BATCHES)
]

#Generate an array with all the transactions
tx_arr= [
                [Transaction(
                        header=tx_header_arr[i][j],
                        header_signature=signature_arr[i][j],
                        payload = payload_bytes_arr[i][j]
                        )
                for j in range(NUM_TX_PER_BATCH)
                ]
        for i in range(NUM_BATCHES)
        ]

#Create the BatchHeader
batch_header_arr = [
        BatchHeader(
                signer_public_key = signer.get_public_key().as_hex(),
                transaction_ids = [
                        tx_arr[i][j].header_signature
                for j in range(NUM_TX_PER_BATCH)
                ]
        ).SerializeToString()
        for i in range(NUM_BATCHES)
]

#Create the batch with the tx
signature_batch_arr = [signer.sign(batch_header_arr[i])
for i in range(NUM_BATCHES)]

batch_arr = [
        Batch(
                header = batch_header_arr[i],
                header_signature = signature_batch_arr[i],
                transactions = tx_arr[i]
        ) for i in range(NUM_BATCHES)]

#Collect all Batches in a BatchList:
#can contain batches from different clients
batch_list_arr = [
        BatchList(
                batches = [batch_arr[i]]
        ).SerializeToString()
        for i in range(NUM_BATCHES)
]

#Send Batches to Validator
try:
        request_arr = [
                urllib.request.Request(
                        'http://rest-api:8008/batches',
                        batch_list_arr[i],
                        method = 'POST',
                        headers = {'Content-Type': 'application/octet-stream'}
                )
        for i in range(NUM_BATCHES)]

        response_arr = [urllib.request.urlopen(request_arr[i])
        for i in range(NUM_BATCHES)]

except HTTPError as e:
        response = e.file

print(payload_arr);
```

IntegerChain app

```
import sys
import cbor
import string
import random
from urllib.error import HTTPError
from random import randint
import urllib.request
```

```
from hashlib import sha512

from sawtooth_sdk.protobuf.transaction_pb2 import TransactionHeader
from sawtooth_sdk.protobuf.transaction_pb2 import Transaction
import sawtooth_sdk
from sawtooth_signing import create_context
from sawtooth_signing import CryptoFactory
from sawtooth_sdk.protobuf.batch_pb2 import BatchHeader
from sawtooth_sdk.protobuf.batch_pb2 import Batch
from sawtooth_sdk.protobuf.batch_pb2 import BatchList

#Creates a signer who will sign the tx and validate
#its identity in front of the validator
context = create_context('secp256k1')
private_key = context.new_random_private_key()

signer = CryptoFactory(context).new_signer(private_key)
print("SIGNER ES: {}".format(signer.get_public_key().as_hex()))

#Generates a transaction per number. All transactions go
#in the same batch, which is sent inside a batchlist.
print('Introduce the numbers you want to send.')
nums = [int(x) for x in input().split()]
print('The numbers: {}'.format(nums));

lenNums = range(len(nums))

#Generate the payload of the tx
#Every family has a structure to follow
#Uses the cbor format
N = 10
ran_addr = [''.join(random.choice(string.ascii_uppercase + string.digits)
for _ in range(N))
            for i in lenNums
]

#Generate the bytes of the payload
payload = [cbor.dumps(
        {
```

```
        'Verb': 'set',
        'Name': ran_addr[i],
        'Value': num
        }
) for i, num in enumerate(nums, 0)]

#Generate the integerkey addresses
tx_addr = [
        sha512('intkey'.encode('utf-8')).hexdigest()[0:6]
        + sha512(ran_addr[i].encode('utf-8')).hexdigest()[-64:]
        for i in lenNums
]

#Generate the tx header
tx_header_arr = [
        TransactionHeader(
                family_name='intkey',
                family_version='1.0',
                inputs=[tx_addr[i]],
                outputs=[tx_addr[i]],
                signer_public_key = signer.get_public_key().as_hex(),
                batcher_public_key = signer.get_public_key().as_hex(),
                dependencies=[],
                payload_sha512 = sha512(payload[i]).hexdigest()
        ).SerializeToString()
        for i in lenNums
]

#Signer signs the header:
signature_arr = [
        signer.sign(tx_header_arr[i])
        for i in lenNums
]

#Generate an array with all the transactions
tx_arr= [
        Transaction(
                header=tx_header_arr[i],
                header_signature=signature_arr[i],
                payload = payload[i]
        )
               for i in lenNums
]

#Create the BatchHeader
batch_header = BatchHeader(
                signer_public_key = signer.get_public_key().as_hex(),
                transaction_ids = [tx_arr[i].header_signature
                for i in lenNums]
        ).SerializeToString()

#Create the batch with the tx
signature_batch = signer.sign(batch_header)

batch = Batch(
                header = batch_header,
                header_signature = signature_batch,
                transactions = tx_arr
        )

#Collect all Batches in a BatchList:
#can contain batches from different clients
batch_list = BatchList(
                batches = [batch]
        ).SerializeToString()

#Send Batches to Validator
try:
        request = urllib.request.Request(
                 'http://rest-api:8008/batches',
                  batch_list,
                  method = 'POST',
                  headers = {'Content-Type': 'application/octet-stream'}
                )
        response = urllib.request.urlopen(request)

except HTTPError as e:
        response = e.file

print(payload);
```

Subscription script

```python
import subprocess

from zmq import *
import zmq

from sawtooth_sdk.processor.context import Context
from sawtooth_sdk.messaging.future import Future
from sawtooth_sdk.messaging.future import FutureResult

from sawtooth_sdk.protobuf.validator_pb2 import Message

from sawtooth_sdk.protobuf.state_context_pb2 import TpStateEntry
from sawtooth_sdk.protobuf.state_context_pb2 import TpStateGetRequest
from sawtooth_sdk.protobuf.state_context_pb2 import TpStateGetResponse
from sawtooth_sdk.protobuf.state_context_pb2 import TpStateSetRequest
from sawtooth_sdk.protobuf.state_context_pb2 import TpStateSetResponse
from sawtooth_sdk.protobuf.state_context_pb2 import TpStateDeleteRequest
from sawtooth_sdk.protobuf.state_context_pb2 import TpStateDeleteResponse
from sawtooth_sdk.protobuf.state_context_pb2 import TpReceiptAddDataRequest
from sawtooth_sdk.protobuf.state_context_pb2 import TpReceiptAddDataResponse
from sawtooth_sdk.protobuf.state_context_pb2 import TpEventAddRequest
from sawtooth_sdk.protobuf.state_context_pb2 import TpEventAddResponse
from sawtooth_sdk.protobuf.events_pb2 import *
from sawtooth_sdk.protobuf.client_event_pb2 import *

ip_cmd = subprocess.Popen(["/sbin/ifconfig eth0 | grep 'inet addr:'
| cut -d: -f2 | awk '{ print '$1}'"], stdout=subprocess.PIPE, shell=True)
(out, err) = ip_cmd.communicate()

#url contains the validator url
url = "tcp://"+out.decode("utf-8").rstrip('\n')+":4004"

print("URL:", url)

#Generate Event Subscription
subscription = EventSubscription(
        event_type="sawtooth/block-commit",
        filters = [
                EventFilter(
                        key = "addr",
                        match_string = "",
                        filter_type = EventFilter.REGEX_ANY
                )
        ]
)

#Send Event Subscription

#Connect to validator
ctx = zmq.Context()
socket = ctx.socket(zmq.DEALER)
socket.connect(url)

#Construct the request
request = ClientEventsSubscribeRequest(
        subscriptions = [subscription]
).SerializeToString()

#Construct the message wrapper
correlation_id = "123"
msg = Message(
        correlation_id = correlation_id,
        message_type = Message.CLIENT_EVENTS_SUBSCRIBE_REQUEST,
        content = request
)

#Send the request
socket.send_multipart([msg.SerializeToString()])

#####################################################################

#Receive the response
resp = socket.recv_multipart()[-1]

#Parse the msg wrapper
msg = Message()
msg.ParseFromString(resp)
```

```
#Validate the response type
if msg.message_type != Message.CLIENT_EVENTS_SUBSCRIBE_RESPONSE:
        print("Unexpected Msg Type received")

response = ClientEventsSubscribeResponse()
response.ParseFromString(msg.content)

#Validate the response status
if response.status != ClientEventsSubscribeResponse.OK:
        print("Subscription failed: {}".format(response.response_message))

####################################################################

#Receive the events
while True:
        resp = socket.recv_multipart()[-1]

        #Parse the msg Wrapper
        msg = Message()
        msg.ParseFromString(resp)

        #Validate response type:
        if msg.message_type != Message.CLIENT_EVENTS:
                print("Unexpected Message Type")

        #Parse the response
        #events = EventList()
        #events.ParseFromString(msg.content)

        #Print the events in the list
        #for event in events:
        #        print(event)

        print("HA LLEGADO TX")
```

A.2 Tangle deployment script

```
#!/bin/bash
```

```
#The manager doesnt need to add the workers to the env because
#the localhost deploys the docker-compose.yaml
#in fact its never needed cause the docker-compose.yaml is copied
#to every node in the folder specified.
#if we want the manager to add to the env the workers, we need to
#install docker-machine in the managers with the following command:
#   docker-machine ssh
#        manager$i "base=
#                https://github.com/docker/machine/releases/download/v0.14.0 &&
#   curl -L $base/docker-machine-$(uname -s)-$(uname -m) >/tmp/docker-machine &
#   sudo install /tmp/docker-machine /usr/local/bin/docker-machine"

if [ $# -ne 3 ]
then
        echo "Use: ./iotanet_gen.sh #NumOfStacks #NumOfWorkersPerManager
                DirFolderWithContents"
        exit 1
fi
for i in $(seq "$(($1))")
do
 #Managers creation
 docker-machine create --driver virtualbox manager$i

 #Get the last created manager's IP
 lastManagerIP=$(docker-machine ls | grep "manager$i" | awk '{print $5}')
 lastManagerIP=(`echo $lastManagerIP | sed -e 's/tcp:\/\//'/g`)
 lastManagerIP=(`echo $lastManagerIP | sed -e 's/:[0-9]*$/'/g`)

 #Managers Initialization
 token=$(docker-machine ssh manager$i
 "docker swarm init --advertise-addr $lastManagerIP" | grep -m1 docker)

 #Managers receive the Folder with the node's Software
 docker-machine scp -r $3 manager$i:/home/docker

 #Managers generate the image using the Dockerfile they received
 docker-machine ssh manager$i "docker build /home/docker/IOTA_node
        -t iotanode:lastest"

 #Localhost adds the Manager to the environment
```

```
        docker-machine env manager$i

 #Managers initialize the service
 docker-machine ssh manager$i "docker run --name managercont$i
 -p 14265:14265 -p 14600:14600/udp -p 15600:15600 iotanode:lastest
         > log_fich" &

 for j in $(seq "$(($2))")
         do
         #Workers creation
         docker-machine create --driver virtualbox worker$i-$j

         #Workers join the manager
         docker-machine ssh worker$i-$j "eval $token"

         #Workers receive the Folder with the node's Software
         docker-machine scp -r $3 worker"$i-$j":/home/docker

         #Workers generate the image using the Dockerfile they received
         docker-machine ssh worker$i-$j
                 "docker build /home/docker/IOTA_node -t iotanode:lastest"

         #Localhost adds the workers to the environment
         #We will manage every node from the localhost.
         docker-machine env worker$i-$j

 done

 #Docker Stack deployment in the managers
 docker-machine ssh manager$i \
         "docker stack deploy --with-registry-auth
                 -c /home/docker/IOTA_node/docker-compose.yml stack$i"
done
#Managers add other managers as neigbors.
managerIPs=(`docker-machine ls | awk '$1 ~ "manager" {print $5}'`)
for i in "${managerIPs[@]}"
do
   for j in "${managerIPs[@]}"
      do
         if [ $i != $j ]
```

```
            then
                x=(`echo $i | sed -e 's/:[0-9]*$/:14265'/g`)
                x=(`echo $x | sed -e 's/tcp/http'/g`)
                y=(`echo $j | sed -e 's/:[0-9]*$/:15600'/g`)

                cmd="curl $x
                    -X POST
                    -H 'X-IOTA-API-Version: 1'
                    -d '{\"command\": \"addNeighbors\", \"uris\": [\"$y\"]}'"

                eval $cmd

                echo "$x adds $y"
            fi
      done
done
#The first manager (coordinator) generates a milestone every 2 minutes.
while :
do
    docker-machine ssh manager1 "docker exec managercont1 java
         -jar iota-testnet-tools-0.1-SNAPSHOT-jar-with-dependencies.jar
                 Coordinator localhost 14265"
    sleep 120
done
```

A.3 MAM testing scripts

Publisher script

```
//Get the numbers from the publisher
const readline = require('readline');

const rl = readline.createInterface({
  input: process.stdin,
  output: process.stdout
});

const Mam = require('../lib/mam.client.js')
const IOTA = require('iota.lib.js')
const iota = new IOTA({ provider: `http://NODE'S_IP:14265` })

// Initialise MAM State - PUBLIC
let mamState = Mam.init(iota)

// Publish to tangle
const publish = async packet => {
    // Create MAM Payload - STRING OF TRYTES
    const message = Mam.create(mamState, packet)

    // Save new mamState
    mamState = message.state

    // Attach the payload.
    console.log('Root: ', message.root)
    console.log('Address: ', message.address)

    await Mam.attach(message.payload, message.address)
}

rl.question('Introduce the numbers you want to send.', (nums) => {
  var arrElems = nums.split(" ");
  for(i = 0; i < arrElems.length; i++){
        publish(iota.utils.toTrytes(arrElems[i]+""));
  }
  rl.close();
});
```

Subscriber script

```
const Mam = require('../lib/mam.client.js')
const IOTA = require('iota.lib.js')
const iota = new IOTA({ provider: `http://NODE'S_IP:14265` })

// Init State
let root = 'ROOT_OF_THE_FIRST_MSG_OF_THE_STREAM'

// Initialise MAM State
```

```
let mamState = Mam.init(iota)

const execute = async () => {
        // Fetch the messages syncronously
        const resp = await Mam.fetch(root, 'public')
        /*while(true){
                const resp = await Mam.fetch(root, 'public')
                root = resp.nextRoot;
                conso
        }*/
        //var bytes = iota.utils.transactionObject(resp.message)
        for msg in resp.messages:
                console.log(iota.utils.fromTrytes(msg))
        //console.log(resp.nextRoot)
}

execute()
```