João Pedro Guita de Almeida

Student number 42009 (MIEI)

Blockchain Enabled Platforms for the Internet of Things

Dissertation submitted in partial fulfillment
of the requirements for the degree of

Master of Science in
Computer Science and Informatics Engineering

Adviser: Henrique João Lopes Domingos,
Assistant Professor, DI/FCT/UNL,
Nova Lincs Research Center

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

September, 2018

Blockchain Enabled Platforms for the Internet of Things

This document was created using the (pdf)LaTeX processor, based in the "novathesis" template[1], developed at the Dep. Informática of FCT-NOVA [2].
[1] https://github.com/joaomlourenco/novathesis [2] http://www.di.fct.unl.pt

For my parents and for my petite amie...

Acknowledgements

Firstly, I would like to thank my advisor, Professor Henrique Domingos, for welcoming my idea for this project and for his enthusiastic guidance throughout this work. I would like to thank my family for the endless support and encouragement throughout my studies. Lastly, I thank all my friends who have turned the last few years into an unforgettable journey, in particular to Dharita Queshil for always keeping my head up.

Abstract

The Blockchain and the Internet of Things (IoT) have gained a lot of attention in the last few years, since both technologies enable the possibility of creating a more connected and independent world. This combination enables the design of computing systems and cyber-physical environments without the need of centralized trusted entities, giving users the freedom and control of their operations, in a decentralized ledger model. By using storing and logging mechanisms supported by the Blockchain, data is immutable and independently audited, guaranteeing that it is neither modified nor deleted. At the same time, applications can benefit from the reliability and fault-tolerance assumptions provided by the Blockchain in supporting transactions between users and involved devices.

In this thesis, it was studied and proposed a generic solution for a Blockchain-enabled IoT software architecture. The proposed solution enables the advantages of using decentralized logging and ledgering, without the interference of central authorities, inherently supported by the base Blockchain reliability, availability and security foundations. These capabilities are envisaged as key-benefits for a new generation of clean-slate approaches for IoT applications with the required scalability criteria.

The research conducted in the dissertation work, studied the base software foundations, relevant components and implementation options that enable the identified advantages of using Blockchain components and services, to leverage more scalable and trustable IoT platforms. Our proposed solution aims to provide an architecture that contributes to a more appropriate design for secure and reliable IoT systems. In this trend we propose a better use of edge-based support for local-enabled processing environments supporting IoT devices and users' interactions, with operations intermediated by proximity hubs acting as gateways to the Blockchain, where the operations are regulated and controlled by verifiable smart-contracts involving data and transactions.

Keywords: Blockchain, IoT, Decentralized Ledgering, Reliability, Fault Tolerance, Intrusion Tolerance, Smart-Contracts

Resumo

A *Blockchain* e a Internet das Coisas (IdC) têm ganho muita atenção nos últimos anos, dado que as duas tecnologias permitem a criação de um mundo mais interligado e independente. A combinação das duas tecnologias permite o desenho de sistemas computacionais e ambientes ciber-físicos que não necessitam de entidades centrais confiáveis, permitindo aos utilizadores a liberdade e o controlo com autonomia sobre as operações, escrutinadas em registos descentralizados. Sendo utilizado um mecanismo de armazenamento como a *Blockchain*, os dados ficam imutavelmente guardados, garantindo-se que não são alterado nem eliminados. Ao mesmo tempo, tendo o armazenamento garantias implicitas de fiabilidade e tolerância a falhas, as aplicações da IdC poderão benificiar dessas garantias no suporte de transações com processamento de contratos inteligentes (ou *smart-contacts*) que suportam interações entre utilizadores e dispositivos involvidos.

Nesta dissertação, foi estudada e proposta uma solução genérica para uma arquitetura que combina a *Blockchain* de forma a integrar aplicações e dispositivos para a IdC. A solução permite integrar o registo e notarização descentralizados dos dados e operações, sem interferência de autoridades centrais e com características acrescidas de confiabilidade. Estas características podem ser usadas como fundamento de uma nova geração de arquiteturas para a IdC, respeitando os devidos requisitos de fiabilidade e escalabilidade. A solução proposta tenta proporcionar uma arquitetura que contribui para um melhor desenho da IdC, com mais garantias de confiabilidade e escalabilidade e com redução de custos operacionais e riscos de segurança das aplicações. Neste sentido, a dissertação propõe, implementa e avalia uma arquitetura que permite suportar aplicações de IdC suportadas em ambiente periféricos ou de proximidade (*Edge*), onde podem ter lugar interações e processamento de dados locais, sendo essas operações intermediadas por hubs que atuam como *gateways* para a blockchain, sendo então reguladas e controladas por contratos inteligentes (ou *smart-contracts*) que envolvem os respetivos dados e transações.

Palavras-chave: *Blockchain*, IdC, Registo descentralizado, Confiabilidade, Tolerância a Falhas, Tolerância de intrusões, Contrato Inteligente

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Context and Motivation

Information is currently the most valuable resource [72], people are sharing their personal data in trade for comfort and social reputation and big companies like Google or Facebook are selling user's profiles to advertisers [29]. This happens in a context in which those companies are often regarded as reliable and trusted entities by users, where such trust is centralized by nature. This discussion has emerged security and privacy concerns, because people are becoming more and more aware of the dangers this topic can bring.

At the same time, many systems and applications, including those managing sensitive data, are using cloud-enabled outsourced services, such as, data-repository or data-processing services. In this case, cloud-service providers also act as centralized trusted computing bases, without scrutiny and auditing functions evaluating the reliability, security and trustability guarantees.

This way, it is important to study possibilities on how to avoid the use of intermediaries for services, for example, the presence of a bank to complete a transaction between two accounts or peer-to-peer data transfers without trustability assumptions from centralized authorities.

IoT Platforms. The network of Internet of Things (IoT) is rapidly growing. According to many observers, today, the number of connected devices has already surpassed the world population in size [73] and continues to grow. Some estimations claim that we can expect 50 to 60 billion devices to be connected to the Internet globally in the first years of the decade 2020 [31]. These devices are widely used in the context of different applications, in smart homes, offices, factories, farms, and cities.

While having different application domains involving more or less specific functional

requirements, current IoT applications and systems share the same architectural approach: the use of cloud-enabled services in order to process and store sensor data they collect, as well as provide the desired service for users, possibly using smartphones, to interact with the back-end cloud-services. In this architectural model, adopted by many well-known IoT development and operational platforms prompted by different IoT service providers (e.g., [5], [60], [1], [41]).

Clouds are used to offload the tasks, to unify the operation of an increasing variety of IoT devices, otherwise not feasible to perform on the resource-limited hardware of such devices. At the same time, collected data and operations' loggings are managed, through single interfaces offered by the Cloud-enabled platforms managed by service providers, namely, Cloud-IoT Service Providers, to analyze user interactions with devices or possible interactions between devices, to make expected decisions for the better user experience.

In order to access cloud services, smart devices from specific manufacturers, can use Internet connections, as individual devices or often through specialized local gateways supporting specific and sometimes proprietary communication protocols, interconnecting sets of devices and the central cloud management platform. Once more, in such architectural models, IoT applications are really managed under central trusted authorities, with a lack of scrutiny and auditable functions allowing the independent verification of data-integrity, privacy conditions or reliability and trustablity assumptions, as verifiable guarantees for users or partners involved in specific IoT applications. For example, in the domain of smart-homes, an Internet survey reported that a large majority of interviewed people would not want to install devices, such as smart thermostats, that can monitor their movements around the home, even if it would allow them to save some money in the monthly energy bill [23]. Considering trustability, security and privacy concerns, the users of current platforms are more and more concerned about what data service providers can collect exactly, how much authority they have over it, and who they share it with, with the complete lack of knowledge about possible operations or transactions involved their IoT devices and other entities that can have access to data and operations fired by the devices.

Users are forced to trust the service provider with data or operations' logging stored in the cloud, completely outside of their control, with a lack of knowledge on what data is effectively sent/received and how data is operated. In this situation, users have no way to ensure how their personal information, evidences of operations or operation states are processed by the IoT cloud providers, nor can they control the granularity, the state and logs of related operations on the exposed data. For example, in an IoT consortium, where different partners can be involved, those operations can be done from different entities or partners, without any scrutiny of the end users (as consumers).

Blockchains and IoT. As the Blockchain and the Internet of Things (IoT), as two representative dimensions in the recent research of Distributed Systems, have been gaining a lot of attention in the last few years, with different Blockchain platforms available in the

research and market areas, beyond the cryptocurrency and other ecosystems (e.g. [34], [10], [56], [32], [13], [15], among others). IoT and Blockchains have become the topic of many studies in terms of possibilities to build a more connected, independently verified and trustable world. Both technologies enable the possibility of creating a more interconnected world involving the interaction of users with a variety of applications, devices and smart-things that will surround certainly, more and more, our daily lives in the near future.

The use or integration of blockchains is regarded at the same time as an inherent distributed storage and a decentralized ledgering approach, that can leverage a new generation of more trustable and scalable IoT platforms. Such platforms can provide a better control in the operation of interconnected things that operate and are processed in the edge, complementarily to other possible cloud-enabled services. In recent research publications, the use of Blockchains for IoT is usually regarded as a way to improve security and privacy, or to address trust decentralization concerns [9], [68], [58], [55], [62]. However, the impact of having blockchained enabled software running in constrained IoT devices has been referred as a relevant issue. Furthermore, the requirements imposed by scalability conditions and reliability or intrusion tolerance (including byzantine fault-tolerance models beyond the replication and consistency models related with the blockchain consensus-layer) [64], [30], [50] and, particularly, the impact of such issues in the way how blockchains can be used for the requirements of IoT platforms, is still an open issue.

Blockchained IoT Platforms. Blockchain-enabled IoT platforms are interesting as a work hypothesis in addressing new design options of innovative autonomous systems for decentralized IoT operations, without the need of centralized entities acting as central points of authority and not entirely controlled trust computing bases in proving the required trustability, reliability, scalability, security and operation management assumptions, under independent auditing of users and partners involved.

We believe that Blockchain enabled IoT platforms and their software architectures can bring important advantages, as clean-slate approaches for a new generation of IoT software systems operating under scrutinizing decentralized logging and ledgering functions, with independent control in the notarization of transactions, under verifiable non-repudiation guarantees. Those advantages can also help in providing a better use of edge-processing and edge-based operation control, as well as, in promoting better scalability criteria, when compared with the current IoT technology and available platforms. Transactions and data can be immutably stored, managed and verified by all the involved entities, anytime, anywhere, with full authenticity, integrity and persistency guarantees. In the hypothesis of the design of generic solutions for Blockchain-enabled IoT software architectures the idea is to enable the advantages of using decentralized logging and ledgering, without the interference of central authorities. At the same time, IoT applications supported in such platforms can inherently offer better characteristics for reliability,

availability and security foundations, as key-benefits for a new generation of clean-slate approaches for the IoT applications with the required scalability criteria.

1.2 Problem

In this section it will be defined the problem statement of this thesis, as well as the objectives and the contributions.

1.2.1 Problem Statement

From the context and initial considerations in Section 1.1, the statement of this thesis is as follows:

It is possible to improve the reliability and trustability guarantees of IoT platforms and software architectures, by introducing Blockchain-enabled mechanisms for decentralized logging and ledgering, information flow control of operations involving IoT applications, as well as, to provide an independent auditing environment in supporting data sent or received by user's smartphones and IoT devices in such applications.

In the problem context we are primarily focused on trustability, reliability and scalability issues in the interoperability model between IoT devices and user's interactions and Blockchain-enabled services and their components. Nevertheless, there are other important concerns, such as other security and privacy properties, which are also related to the dissertation problem. However, these other properties are not in the scope of the dissertation approach.

Concerning the components and services, as provided by different blockchain platforms, it is particularly relevant to study how smart-contracts in Blockchain-Enabled IoT platforms can be used, as general-purpose computation processing control elements regulating operations and data interchanged between users and devices. Also related to the problem, it is necessary to support operations and smart-contracts in an infrastructure implemented by replicated asset registries, as materialized by the current blockchains, preferentially addressing the use of such supports as leveraging mechanisms in extensible solutions. In this research stream, the use of state-machine replication models with enhanced Byzantine Fault Tolerant (BFT) solutions in blockchain consensus planes is particularly interesting, in order to support dependable and verifiable execution of operations of IoT applications, under reliability, intrusion tolerance and availability arguments.

1.2.2 Problem Addressing

Following the stated problem, it is relevant to address in the thesis how to improve verifiable trustability properties for operations executed in smart devices, by introducing mechanisms to perform information flow control. This can be a problem, considering

that many IoT devices are resource-constrained devices, disallowing the possibility of introducing complex processing for this purpose, at the device level. To circumvent this problem, the control of network traces (and payload data) generated by IoT devices can be inspected, controlled, possibly aggregated and forwarded by local smart-hubs, with less processing limitations, as gateways to blockchain enabled services. These smart-hubs can work in the intermediation between end-users or IoT devices (as edging-devices supporting local or proximity operations) and the backend blockchain services (that can be supported as core-services, for example, cloud-enabled blockchain execution nodes, supporting a permissioned model of consensus that must include all the entities involved, including the end-users). At the same time, smart-hubs provide a way to decouple the operations in the edge (e.g., supporting invocations from IoT or users' devices and providing local-service endpoints that can be able to support message-based communication, HTTP or HTTPS based REST (or Representational State Transfer) invocations.

In approaching the problem as discussed above, the blockchain minimizes the role of centralized authorities in the verification of data, transactions consistency and promotion of decentralized notarization services allowing different users and entities a scrutiny of consistency guarantees, anytime and anywhere.

Thus, to approach the problem statement, the dissertation introduces the notion of blockchained IoT platforms (or Blockchain-Enabled IoT platforms), considering (i) the use of the blockchain as a foundation component in a clean-slate approach of future IoT platforms; (ii) the combination of edged-based IoT processing with locally provided Smart Hubs, acting as intermediation nodes, between the edged devices and the blockchain (and complementarily with other specific cloud-based services); and (iii) the extension of smart-contract regulated transactions for a better verification, validation and execution of transactions involving humans and things, or things-and-things, logged in the blockchain.

These three facets can provide new foundations for a new generation of more reliable, trust and scalable IoT approach, with a better processing control for all the intervenient entities.

1.3 Objectives and Contributions

1.3.1 Objective

Considering the problem statement and related considerations, the objective of the thesis is to study and propose a generic architecture for a Blockchain-enabled IoT software platform, with new design foundations and components for a clean-slate approach to improve reliability, trustability and scalability.

In the envisaged approach, the idea is to explore advantages leveraged from Blockchain-based services, operating under permissioned conditions. Under the permissioned model, blockchains supporting IoT operations restrict the actors involved that can contribute to the consensus of the replicated system state, which also helps in scalability conditions

in the trade-offs between efficiency, consistency and reliability, a well known problem widely discussed in the blockchain research agenda. In a permissioned blockchain, only a restricted set of entities and their related nodes (in a possible large number of nodes) have the rights to validate the block transactions, under the verification of smart-contracts. This permissioned model may also restrict access to approved actors who can create or modify those smart contracts.

Our designed solution aims to provide an architecture that must be able to support IoT ecosystems by services in a permissioned-oriented blockchain, addressing reliability effectiveness and scalability criteria. In this trend we also promote a better use of edge-based support for local-enabled processing environments of things (IoT devices and user-interaction devices, such as smartphones), intermediated operations supported by smart-hubs as gateways to control operations regulated and controlled by verifiable smart-contracts, data and transactions in a blockchain.

1.3.2 Contribuitions

The main contributions related to the above objective, presented and discussed in the present thesis report are the following:

1. Design, of a blockchained IoT platform proposal and related software architecture;

2. Study of selected blockchain platforms and their characteristics, as possible leveraging solutions to support the mechanisms for reliable ledgering services for the proposed IoT platform, to support verifiable IoT-based transactions regulated by smart-contracts, and the selection of the better identified platform for the purpose of the dissertation;

3. Development and validation of the proposed platform with a prototype of the proposed software architecture and components, used as a reference prototype and proof-of-concept implementation for future IoT applications:

    - As a better solution to keep performance and to provide better effectiveness, scalability and decentralized trustability criteria;
    - As a strategy to promote a better use of edge-based support of local-enabled processing environments regarded as islands of IoT devices, with operations and users' interactions controlled by verifiable smart-contracts, with the intermediation of edge-enabled smart-hubs.

In the described contributions, Smart Hubs are regarded as possible extensible pluggable processing entities that can be enhanced with other functions beyond the primary communication gateway feature (interconnecting IoT devices, users and Blockchain-Enabled Services), the feature primarily explored in the dissertation. Examples of extension functions are: pre-processing or data aggregation components, filtering and local

dataflow-control for security and privacy requirements, as well as, specific-application gateway services between users or IoT devices and other application-specific cloud-enabled services. Examples of such services are: payment gateways, customer relationship management systems, authentication services, public-key infrastructures, etc.

1.4 Document Structure

The remaining of the report is divided in five chapters. In Chapter 2, there is an approach to the related work, organized in four main sections: a generic background for the thesis objectives, blockchain in large-scale environments, analysis of different blockchain platforms and approaches for blockchain-enabled IoT applications. Chapter 3 presents a system model and architecture for our proposed blockchain-enabled IoT platform. Chapter 4, presents the implementation details in prototyping our proposal. In Chapter 5 it is shown the evaluation conducted to validate our proposal and related prototype. Lastly, in Chapter 6, we summarize the main conclusions and possible future work directions from the achieved results in the dissertation.

Chapter 2

Related Work

In this chapter we discuss the foundations of the blockchain, as well as its characteristics and properties. In this study we conduct a thorough analysis of reliability, scalability issues, as well as, components and service layers in the design and operation of current platforms. Then, we outline the support for blockchain-supported transactions, consistency mechanisms and decentralization issues, also summarizing relevant trade-offs and the role of access control mechanisms for permissioned versus permissionless blockchains (Section 2.1). Following, we describe the implications and scalability concerns of blockchains in large-scale environments (Section 2.2). From the above notions, terminology and considerations, we survey how different characteristics and their nuances are addressed in different blockchain platforms (Section 2.3), discussing specific support mechanisms, permission-models, control of transaction flows, consensus mechanisms and guarantees and models for replication and ledgering. Then, we address the notion of Blockchain-Enabled IoT (Section 2.4), discussing relevant characteristics for such platforms, limitations that must be considered and initial design issues to be addressed. Finally we conclude the chapter (Section 2.5) by presenting a summary on the covered topics, as a baseline discussion to address the proposal of a Blockchain-Enabled IoT platform and software architecture, as we approach later, in Chapter 3.

2.1 Generic Background

2.1.1 Blockchain Foundations

The concept of Blockchain was first introduced in 2008 by Satoshi Nakamoto, with the Bitcoin cryptocurrency ecosystem as the application environment [54] and Blockchain as the core technology. Although the name of the author was used by the unknown person or people who developed the Bitcoin system, the Blockchain notion emerged, essentially, as

a replicated database supporting peer-to-peer transactions, with particular characteristics primary targeted in supporting relevant characteristics, such as, the avoidance of double spending for digital currency and the related consistency guarantees for the scalability of P2P operations and inter-networked participants, or blockchain nodes.

Even considering that in many approaches, a blockchain is intrinsically related to the support of cryptocurrency environments, given the explosion of such cryptocurrencies, it is particularly interesting in the context of the present dissertation to demystify the blockchain definition: a replicated database to support lists of records organized in blocks, under state-machine consistency guarantees, which are linked by integrity properties using cryptographic primitives. Each block [28], according to its number of records, contains a cryptographic hash of the previous block, a timestamp and transaction data, usually represented as sequences in a Merkle Tree root hash.

By definition, the blockchain is resistant to the modification or removal of maintained information, and it was currently implemented as open and distributed ledger environment, storing transactions between any two parties, in a verifiable and independent way.

Thus, a blockchain solution consists in an electronic peer-to-peer system that allows assets to be transferred from one party to another without an intermediary, but also, is a more generalized or abstract vision, to support peer-to-peer transactions and data-transfers between the network peers. Generically, the blockchain relies on a network of nodes that perform tasks of validating and relaying transactions in their blocks, according to certain validation guarantees. Each of these nodes has a copy of the blockchain state, from the moment it joins the network until the moment a new transaction must be validated and recorded, under the regulation of different processing criteria. This vision is particularly interesting when we address a blockchain services as open, reusable and extensible services or components for software architectures targeted for different purposes and applications, not being strictly related to cryptocurrency environments and their functional requirements.

The base communication infrastructure for a blockchain is, therefore, a peer-to-peer (P2P) network, where the inter-node communication environment support writes and reads, as well as, the implicit independent validation of new blocks. Given the non-tampering characteristic, once validated and recorded, data in any given block of the blockchain cannot be affected by forgery, modification or deletion, retroactively, without the required alteration of all the subsequent blocks, which requires the consensus of the network majority of the members. For this reason, blockchains are considered as secure and replicated databases by design, where fault-tolerance properties, that include fail-stop and byzantine fault-tolerance guarantees, a decentralized consensus model and consistency guarantees are base ingredients of the resulting dependable distributed computing system.

2.1.2 Blockchain characteristic and their nuances

The Blockchain functions support a distributed ledger environment in which transactions have to be verified, independently, by every member of the network. The identification of the network members is secured by some form of verifiable authenticated global names. In practice, this identification is established by the use of public keys, using public-key cryptographic methods, which translates to each node address or node name in the blockchain. The correspondent private-key is then used to digitally sign or authenticate digital assets for each node, establishing proofs of authentication.

Regarding node identifiers, a blockchain can use the base assumption of public keys as identifiers, or external global names, that must be securely mapped to the respective public keys or other related signed identity attributes. The former authentication is the usual method in open and flat blockchains, as addressed in cryptocurrency ecosystems, or represents a layer in namespace bindings in other possible models that have been addressing more recently in the blockchain research agenda, such as hierarchic names, or models for hierarchies of blockchains, designed to address a different approach for horizontal scaling and to optimize trade-offs between byzantine fault-tolerant consensus protocols for consistency guarantees, and the throughput of transactions that can be achieved. We will discuss other scalability issues related to blockchain models later.

For now, we introduce other common characteristics usually present in a blockchain environment, as well as, their nuances as supported by different blockchain implementations.

Openness and permission model. There are two types of Blockchains: permissioned and permissionless blockchains. Permissionless blockchains, such as Bitcoin blockchain, enable the open participation of any entity in the network and in the consensus process, as characterized in [51]. In permissionless blockchains any participant node can create an address and begin interacting with the blockchain network. This means that in this model any node can join the network, participate in the process of block verification and consensus, and in a more broad vision, can also create specific rules for P2P transactions and block processing. However, in a permissioned blockchain model, the actors who can contribute to the consensus of the system state are managed to be restricted. In this case, only a restricted set of nodes have the rights to validate block transactions. A permissioned blockchain may also restrict access to approved actors that can be involved in the setup of rules that regulate data and transaction processing.

Permissionless blockchains and characteristics. Related to the permissionless model supported in the blockchain design model, different characteristics can also have nuances, if a permissioned model is addressed, in the way those characteristics are achieved and described [51]:

- *Decentralization* - represents a key feature when the system is not dependent on a central authority;

- *Transparency* - meaning that data records are transparent to each node;
- *Open Source* - if the support technology is open source so it can be used for free or to create any application;
- *Autonomy* - related to the system support operating independently by implementing different consensus algorithms, implying on possible different consistency guarantees, by configuration or by dynamic adaptive mechanisms;
- *Immutability* - the guarantee that records will be stored forever and cannot be changed, unless someone controls more than the majority of the nodes;
- *Anonymity* - when participant public names or addresses cannot match real life identities, of persons or represented entities, mapped in this address through the system.

Permissioned blockchains. As introduced before, in permissioned blockchains, participants are well defined, but the model can be addressed in different ways. Permissioned blockchains are commonly classified as consortium or fully private. In consortium blockchains transactions can be read publicly or restricted to certain participants and the consensus process is restricted to predefined nodes in the network. The Hyperledger Fabric [34] is an example of a consortium blockchain. Fully private blockchains have the same permissions for reading transactions, however writing new transactions is under the responsibility of a unique entity. This way, are basically hybrid blockchains for reading with centralized, and possibly access-control enforced, writers.

Permissioned blockchains are faster, easier to implement, and consume less resources [61]. With this type of blockchains, it is possible to use consensus algorithms that consume less resources and show higher performances, namely better throughput under low latency conditions, unlike Proof of Work or Proof of Stake mechanisms, as we will present below and usually adopted in open and permissionless blockchain models.

2.1.3 Blockchain structure and operation

As stated before, blockchain represents a sequence of blocks and each block will store a certain number of transactions. For practical reasons, a block has a maximum number of transactions it can hold that depends on the block size and the size of the transaction, and indirectly, limited by the performance to be supported.

A block consists of a header and a body. The block header includes [28]: A block version indicating which set of block validation rules to follow, a Merkle tree with the hash value of the transactions in the block, a timestamp representing the current time in universal time in seconds, nBits for the target threshold of a valid block hash, a Nonce designating the 4-byte field which usually starts with 0 and increases for every hash calculation and the Parent's Block 256-bit hash pointing to the previous block. Finally,

the block body is composed by a transaction counter and transactions. This structure is very common to permissionless blockchains and also supported with minor changes in different permissioned blockchains.

The general operation of a blockchain can be explained in three main steps [51]:

1. When a node wants to share information, it broadcasts it to all the other nodes in the P2P network.

2. Nodes create a new block when they receive valid information and it is enough to create a new block.

3. The consensus process is executed and, in case of success, it is appended to the chain.

The services offered by a blockchain ecosystem can be decomposable into different layers that can be addressed in different planes [17]:

- *Network Plane* responsible for the propagation of transaction messages. For example, in Bitcoin it is used to propagate all valid transactions to all nodes in the network.

- *Consensus Plane* used to designate a set of transactions that is accepted by all nodes, as well as a total or partial order of these transactions.

- *Storage Plane* functioning as a global memory that stores and provides availability for authenticated data produced by the Consensus Plane. Storage applications could be databases or Key-Value storage such as Dynamo, MongoDB, MySQL or BigTable.

- *View Plane* represents a state that is derived from the application of all transactions.

- *Side Plane* allowing consensus to be achieved outside the main chain.

2.1.4 Blockchain structure and other characteristics

There are other characteristics that can be also interesting in the study of blockchain implementations, regarding the different structural planes, as well as, the provided services and mechanisms. For the scope of the dissertation, we consider the following characteristics:

- *Openness* – related to the possibility of support and integration of any application, through externalization interfaces, in such a way that the blockchain components and services can be reusable for different applications, at different levels of integration. The characteristic can be regarded in different levels of granularity, for example, total openness of any blockchain plane as a reusable plane, or limited openness, when restricted to a specific plane or to a common and limited externalized interface hiding internally the logics to access to the other planes.

- *Extensibility* - if the mechanisms and software components implementing different functions, or blockchain planes, are open, designed and implemented to be extended, in order to address specific application requirements in the blockchain processing functions;

- *Ordering Flexibility* – if there is flexibility to support multiple ordering services in the consensus plane, possibly extended as pluggable consensus components as mechanisms managing different information flows. In this, we include the possibility to have support for different consensus protocols with different reliability semantics and byzantine fault-tolerance guarantees, as pluggable components, implementing different levels of approach for scalability, throughput and consistency trade-offs.

2.1.5 Blockchain Transactions

Transactions are important in blockchain systems. Each transaction is signed by the user's private key [28]. The transaction is authenticated using digital signature and then broadcast to all the other members of the network for verification.

The signature process is composed by 2 phases. A signing phase in which the sender encrypts the data using his private key and a verification phase in which the data is validated using the sender's public key. Usually, the signature algorithm used in many cases, such as Hyperledger Fabric [2], is the elliptic curve digital signature algorithm (ECDSA). The set of private and public keys are stored in a data structure named wallet. This digital signature, allows to guarantee non-repudiation of data. In the case of Hyperledger Fabric, keys are a part of the user.

The essential block of a Bitcoin transaction is a transaction output [3]. Transaction outputs are indivisible chunks of bitcoin currency, stored in the blockchain and recognized as valid by all the network members. Every transaction that is made will represent a state transition in the Unspent Transaction Output (UTXO) set, the set of all unspent transaction outputs. In summary, when a user is said to have received a bitcoin it means that his wallet has found a UTXO that can be spent with one of the keys that are controlled by that wallet.

In Hyperledger Fabric there is not a crypto-currency associated. A transactions in Hyperledger Fabric is composed by a header, signature, proposal and a response.

2.1.6 Consistency Mechanisms

When using the blockchain, one of the most important matters is finding a way to be able to agree on values without the need of trusting anyone. To solve this problem, it is necessary to incorporate decentralized consensus algorithms.

In [54], is presented a mechanism for emergent consensus. Emergent consensus is an algorithm that does not have a specific moment where consensus occurs, but as [3] states,

that *"consensus is an artifact of the asynchronous interaction of thousands of independent nodes, all following simple rules"*. There are various ways of solving this problem, the main ones are:

- *Proof of Work (PoW)* - first presented in 1992 as way to battle against spam emails [24], is based in a piece of data that is difficult or time-consuming to produce but is easily verifiable by other nodes [51]. It is a process in which it is changed the Nonce in the block header is such way that the hash of the header is less than a pre-specified target. This target's difficulty is adjusted according to the amount of blocks that can be generated.

- *Proof of Stake (PoS)* - this mechanism doesn't rely on a cryptographic puzzle to be solved since it will generate a lot of wasted power. This method is translated in a form of proof of ownership of the currency [46] and coin age. For example, if Bob sends Alice 100 coins and Alice does not spend it for 20 days, Alice will have 2000 coin days of coin age.

- *Proof of Elapsed Time (PoET)* - capable of supporting a vast amount of nodes. It uses safe instructions, provided by the CPU (enclaves) in which, each node requests a wait time from the enclave and the node with the shortest amount of time is elected as the leader. This leader election technique guarantees that leadership election is distributed among all nodes in the network in a balanced way. This leader is then capable of proposing a block for acceptance, that will then be added to the chain [33].

- *Ripple* - consensus algorithm that relies on collectively-trusted subnetworks within the larger network. In the network, nodes are divided in server, for taking part in the consensus protocol, and client for fund transferring [75]. It is divided in rounds and at least 80% of the votes are necessary for a value to be accepted.

- *Practical Byzantine Fault Tolerance (PBFT)* - A Byzantine fault tolerant state machine replication algorithm providing both liveness and safety properties for (n-1)/3 faulty replicas. This algorithm is to be applied to an asynchronous system rather than an impractical synchronous system [12].

- *Tendermint* - byzantine fault tolerant replication application that supports up to 1/3 faulty replicas. It consists in a voting consensus engine called Tendermint Core that the same transactions are recorded on every machine in the same order [48]. The algorithm is divided in four steps and in each step 2/3 of the votes are necessary.

Although these are the main algorithms for consensus, there are other approaches, such as Proof of Existence, Proof of Importance, Delegated Proof of Stake and Proof of Activity. These are some common names for consistency mechanisms that can appear as variants or hybrid approaches of PoW and PoS based consensus algorithms [54].

Consensus algorithms such as PoW or PoS are used in permissionless blockchains. In this case PoW as a consensus mechanism, are used under the assumption that peers do not rely on trust. Indeed, a model with stronger assumptions regarding trust can be associated with better performance results.

For example, a Byzantine Fault Tolerant replication protocol with a number of selected trusted entities may overcome some of these problems regarding performance and scalability. As illustrated in [17], using PBFT greatly outperforms Bitcoin in both transaction latency and throughput. In the use of PBFT state machine replication and consensus for blockchains, it is assumed a closed and small consensus group, usually parameterized for four participants, and never more than fifteen. However, for certain requirements and for the enlargement of consensus groups, PBFT approaches can also have problems in throughput, latency conditions and horizontal scalability purposes for permissionless blockchains. Other approaches have been more recently proposed to avoid such possible limitations. For example, in [26] the idea is to break a large collective into smaller subgroups (using a sharding technique), reducing the overhead of transaction processing in a large and flat blockchain. Those subgroups are organized in shards, where membership size is regulated for security vs. performance trade-offs, and shards are re-formed periodically, as the network evolves. This means, that the consensus tasks are split among different nodes, so each node has a reduced processing and there is an improve in throughput because the consensus group is small. Performance does not improve linearly with shard count.

In the ByzCoin blockchain model [25] the idea is to de-conflate membership and consensus by electing temporary leaders by mining key-blocks and the leader signs microblocks with limited number of transactions to accelerate the throughput achievement. The approach is a model in which a PoW mechanism is used, but mining yields temporary membership share, in a gradually-rotated consensus group. The approach uses an efficient tree-structure model of communication to support a collective signing model in which only one participant signs on compressed messages to optimize the *PREPARE* round of the PBFT consensus.

2.1.7 Decentralized Trust and Shared Ledgering

In many current systems, trust is mediated by a central authority. Transactions between two parties are dependent on a central authority that is responsible for data validations and for authenticating both parties.

The evaluation of trust in descentralized environments involves three main aspects [67]. First, there needs to be an incentive for good behavior, this way participants will act more responsibly in the future. Second, trust evaluation should provide the means for good participants to avoid working with malicious participants. Third, the results of trust evaluation allows the detection of malicious participants.

A ledger is an old definition that defines the current state of a business as a register

of transactions [39]. In a blockchain, these registers are implicitly validated in their trustability conditions in an autonomous and independent way, by means of transaction processing and consensus guarantees. For example, in Hyperledger Fabric the ledger is a sequenced, tamper-resistant record of all state transitions and peers will maintain a copy of the ledger to every network it belongs. We must notice that are two parts for a blockchain ledger: (1) the world state, which is a database that holds the current values for the ledger states and (2) the transaction log, which will store the update history for the world state [36].

2.1.8 Access Control Enforcement and Anonymization

In general, access control enforcement techniques are used to limit resources or services in a system according to the user's rights. These rights are usually expressed through enforcement policies. One way to include rights and enforcement policies in the blockchain would be through Smart Contracts, which is detailed in the next section. The Smart Contract can be queried directly and transparently, so when a user requests an operation, it could be verified at access request time. In a primary overview, a smart contract is nothing more than a computer protocol, as an executable program, or simply executable code, intended to digitally facilitate, verify, or enforce the negotiation parameters or performance requirements of a contract, regarded as verifiable conditions in transactions involving data or other assets. The code behind a smart contract contains specific terms, rules or conditions, as well as the expression of values and invariants, that are executed when triggered by specific agreed events.

We must notice that in the blockchains' world, smart contracts are designed and supported to inherit some of the blockchains' properties:

- Immutability, which means a smart contract can never be changed and no one can tamper with or break a contract, as programmable rules for the verification of transaction and block processing.

- Distribution, which means that the outcome of the contract is independently validated by every node in the blockchain network, just like any transaction, under the same consistency scrutiny.

In open or permissionless blockchains, the blockchain is a public ledger that represents all transactions made. This way, anyone can consult transactions that occurred. However, since the user's accessible information is only its public address, or more specifically, a public key, this key is not linked to an actual name or a home address, for example, anonymizing the interventions.

According to [18] there are two main properties to describe a system for a full anonymous cash model, but this model can be applied for the privacy of any blockchain model:

- *Untraceability* - for each incoming transaction, all possible senders are equiprobable.

- *Unlikability* - for any two outgoing transactions, it is impossible to prove they were sent to the same person.

2.2 Blockchain in Large-Scale Environments

2.2.1 Scalability Issues and Implications

We addressed before how the design model and implementation options of the different planes in a blockchain structure can affect the performance under scalability criteria.

For a system to be used in a larger scale, its performance, in a network with a large number of nodes, is measured in order that it does not cause a burden to the users to use the system, compared with a small number of nodes. For example, in the current scale of the Bitcoin, it takes at least 10 minutes to confirm a transaction and has a maximum throughput of 7 transactions/sec. As a way of comparison, Visa is capable of confirming a transaction in seconds and has a peak throughput of 56,000 transactions/sec [17].

These values pose the question of whether a decentralized blockchain is capable of matching the performance of Visa, for digital currency transactions. For this purpose, in [17], the authors propose some ways to improve the blockchain's performance, for example, increasing the maximum block size or even remove the limit as a way of improving effective throughput or minimizing latency by reducing the block interval which would require also a reduction in the block size.

There are two main problems in the Bitcoin's network protocol that degrade its performance namely:

1. On one hand, a node must fully receive and validate a transaction before propagating it as a way of avoiding denial-of-service by dissemination of invalid transactions;

2. All transactions must be transmitted twice, one to disseminate the block and another one to disseminate the block in which the first transaction is contained.

To address this, different solutions have been proposed, as stated in [17]. For example, to solve the first problem, there should be rate limits for nodes that generate invalid transactions. To mitigate the problem, this approach benefits honest nodes. As a solution to solve the second problem, a node will only fetch transactions it does not possess.

2.2.2 Scalability Dimensions and Heterogeneity

In a practical application of the blockchain, there will be, possibly, different types of data and nodes. In different cases there is data that must be available to every nodes, data that should only be available to a few selected nodes and even some data that is of interest that can be extrapolated by the stored content. This way, we can define two types of scalability dimensions: scale-in or scale-out.

Scale-in scalability must be addressed when all nodes share the same blockchain, which means that this chain will grow indefinitely. These issues can be achieved by "scalability tuning"and invariants expressed as system assumptions, together with other vertical and horizontal scaling conditions in internal service planes of the blockchain services, as presented before.

When a blockchain scales-out it means that several blockchains are used to differentiate types of data or nodes, in a certain architecture and topology. For example, we can have blockchains associated with two different domains of implementation and we can have a third blockchain that serves as a metadata storage and ledger, for those two blockchains, in a hierarchic model. This way, it is possible to consult information related to the data or even create hierarchical models for blockchain management under scale-out conditions. In this approach, the scale-in conditions can be regarded as transparent, and we can also have different approaches for different scale-in conditions in different blockchains belonging to the considered hierarchy.

IoT applications rely on devices that can be seriously resource constrained, therefore it is important to reduce data processing on each node. Using a scale-out approach and introducing trusted nodes it is possible to create permissioned blockchains that use different consensus algorithms, such as PBFT, under certain restrictions of the blockchain membership, which outperforms the PoW model as used in the Bitcoin in both transaction latency and throughput [17].

2.3 Blockchain Platforms

2.3.1 Characteristics and Design Criteria

Before, we addressed that there are different types of blockchains: public, consortium or private. Remembering, in a public blockchain every node has the right to join or leave the network as they will, and this is the case of Bitcoin. In a consortium blockchain not every node has the right to validate transactions and this process relies on assigning this role to a few selected nodes. A private blockchain implies on a more centralized structure, since a single entity controls the transaction process and has the power to make decisions such as controlling what consensus protocol is used. In Section 2.1, we also analyzed different characteristics that can be present, with distinct flavors, in various blockchain platforms.

Now, in this section, various platforms will be described and compared, considering such characteristics. For comparison purposes, several characteristics of each platform will be taken into account.

It is also relevant to analyze the type of support provided for smart contracts in the studied blockchain platforms. The analysis of their characteristics and the existent support for smart contracts is particularly relevant for the choice of a blockchain environment, including the related services and components, that is more appropriate to leverage a blockchain-enabled IoT platform, as we want to address in the dissertation objective.

- *Permissions* - differentiating between permissioned and permissionless blockchains.
- *Open Source* - if the platform is open source.
- *Consensus Protocol* - what consensus protocol is used.
- *Transaction Flow* - how platforms process transactions.
- *Scalability* - how scalability impacts performance, which is usually evaluated in terms of throughput and latency.

2.3.2 Smart Contracts and Access Control Enforcement

Smart Contracts are supported in different blockchains, with differences in terms of expressiveness power, programming language and support model. The support model can also have dependencies on the access-control enforcement mechanisms, as well as, the roles that can be executed by the blockchain nodes.

For example, in Hyperledger Fabric [34] , there are endorsement policies, which enable to let a certain peer know whether a transaction is properly endorsed. Hyperledger Fabric also uses an Access Control List (ACL), allowing for the management of resources by the association of an access rule to a set of identities [35]. Hyperledger Fabric provides the Client Identity Chaincode Library (CID) [38] which enables the definition of access control policies based on the identity of a client directly in the chaincode. This library allows the definition of a set of attributes of users in a associated X509 certificate.

2.3.3 Smart Contracts and Programming Support

It was already stated that smart contracts are, in essence, programs. The term "Smart Contract"was introduced a long time ago by Nick Szabo, in 1996 in [70] where the author stated that a decentralized ledger could be used to store contracts in a digital format. This way, contracts would be converted to code, replicated to the network and would then be verified by the computers that compose the network.

Since smart contracts are an important part of a decentralized environment, it is crucial for blockchain platforms to support it. This way, it is essential to acknowledge if and how different blockchain platforms implement smart contracts, the expressiveness allowed and how they support possible extensibility conditions.

For example, Bitcoin supports a similar concept to Smart Contracts, however it is implemented in a restrictive scripting language that is not Turing-complete [10]. On the other hand, Ethereum provides a contract-oriented, high-level language for the development of Turing-complete Smart Contracts, Solidity.

As another example, Hyperledger Fabric supports the concept of smart contracts as chaincodes. The chaincode will implement the application logic and will run during the execution phase [2]. This way, the chaincode will be triggered when a transaction is proposed and it will decide the state changes to be applied to the ledger. In Hyperledger

Chaincodes are, in its essence, programs written in Golang, but it can be also written in Node.js, Java and eventually, is more or less easy to have or to develop those contracts, expressed in other different languages [37].

Chaincodes deal with the initialization and management of ledger states by the received transactions. These chaincodes run in different containers from the peer and the state that is generated by a chaincode is not accessible by another chaincode, this way being able to provide isolation guarantees.

In the remaining sections we will analyze a set of different blockchain platforms.

2.3.4 Ethereum

Similarly to Bitcoin [54], Ethereum [10] is a distributed public blockchain network. It is a programmable blockchain that allows users to create their own operations. The Ethereum Virtual Machine (EVM) is a Turing complete software that runs on every Ethereum node and allows anyone to run any program.

Ethereum has three main concepts for its functioning: accounts, transactions and messages. There are two types of accounts: externally owned, which have no code and are controlled by private keys and contract accounts which are controlled by contract code.

Ethereum messages are comparable to Bitcoin transactions with the additions that can be created by either an external entity or a contract. Messages can contain data and recipients can send a response when receiving a message in case it is a contract account. Transactions are used to refer to signed data packages that will be used to store messages to be sent from an externally owned account.

Ethereum also introduced the notion of gas, which translates the amount of work required to execute a certain operation in a fee that is set in the transaction in order to prevent infinite code loops and exponential blowups. If the gas limit is reached before the transaction finished, the transaction will be considered invalid and the gas will not be refunded to the sender, because computational power was already spent.

For a block to be considered valid the previous block reference is checked for block existence and validity, the new block's timestamp has to be greater than the previous block's timestamp, the proof of work has to be valid and the final state that will be applied is verified for errors.

Next, it will be presented two platforms that were developed based on the Ethereum implementation.

- *Quorum* [56] is a distributed ledger protocol developed by JP Morgan that is based on the Ethereum implementation using the Go-language. It is a private or permissioned blockchain that uses a voting algorithm to achieve consensus and has data privacy with the introduction of private transaction identifiers. Transactions are split in public and private. Public transactions are validated by every node in the network, but for private transactions, only nodes that are related to it will execute the contract code associated, all other nodes will simply skip this step. This way,

the database will be split in a public state database and a private state database. All nodes will have the same public state database, which does not happen for the private state database. As a consensus mechanism, Quorum offers two possibilities a Raft-based mechanism and Istanbul which is inspired by the PBFT algorithm. A new majority voting protocol is currently under development, QuorumChain. In this protocol, only a few selected nodes are able to vote which block should be the head block at a particular height. The creation of a block is a task also attributed to certain nodes according to its role. The main difference between the two algorithms is that PBFT is able to tolerate crash and byzantine faults, while QuorumChain is only able to tolerate crash faults.

- *Hydrachain* [32] is another extension of Ethereum for permissioned distributed ledgers. As a consensus protocol it uses a byzantine fault tolerant protocol which was inspired in the Tendermint [48] Byzantine consensus algorithm, but relies on a registered and accountable set of validators that guarantee the order of transactions. It provides the tools to create smart contracts using the Python language and since it is capable of bypassing the EVM, native contract execution is faster. These contracts are compatible with EVM based contracts, this way it is possible to have both on the same chain.

2.3.5 Hyperledger

Hyperledger [34] or The Hyperledger Project is an open source project founded by the Linux Foundation as a way of fueling the development of decentralized projects. It consists of several platforms and the main ones will be described next:

- *Fabric* [33] is a platform for permissioned ledgers that follows a modular architecture. It supports channels, this is, transactions are only available to nodes that are in the same channel, this way creating multiple Hyperledger Fabric instances, this is, multiple blockchains. Channels are initialized with a configuration block called the genesis block.

  It uses chaincodes, which are comparable to smart contracts, that will enforce a set of rules when reading or modifying values from storage.

  Endorsing peers endorse transactions before they are committed, this way, being in agreement with the specified endorsement policies.

  It also supports a portable notion of membership [2] and each peer is properly authenticated by the Membership Services Provider when joining a channel. Hyperledger Fabric also provides an implementation of a Certificate Authority (CA), the Fabric-CA, but it is also possible to integrate commercial CAs. Fabric-ca is the entity which deals with the registration of entities, issues Enrollment Certificates and certificate renewal and revocation. Kafka [45] and SOLO were the chosen mechanisms

to implement consensus, in order to decide the transaction order, which consist in a voting-based system. Although these mechanisms provide crash fault tolerance, they do not provide Byzantine fault tolerance, but given its modular architecture, the consensus protocol can be exchanged by a protocol that is capable of providing Byzantine Fault Tolerance.

BFT-SMaRt [64] is one example of a BFT protocol that has been implemented for Hyperledger Fabric [8] which will be detailed in a section detailing Hyperledger Fabric with Byzantine Fault Tolerance.

Hyperledger Fabric follows a mechanism of execution of execute-order-validate [2], this way enabling the possibility of a transaction being executed before the ledger update. First, a transaction is executed and its correctness is verified, this way being endorsed. It is in this phase where the execution of the chaincode occurs. Then, the ordering step corresponds to the consensus protocol and finally, the transaction will be validated following trust assumptions defined in the active endorsement policy. Hyperledger Fabric follows a hybrid replication model since both active and passive replication models are used.

The orderer is the entity that guarantees consistency in every peer's ledger [2]. This entity will establish the total orderer of the transactions and will functions as an intermediary between peer and Kafka. This entity does not validate transactions.

- *Iroha* [33] is a modularized distributed ledger still in development. It prioritizes the mobile environment and has created and made available libraries for the main mobile operating systems. It supports Chaincode which is executed in a sandboxed Java VM. It introduced Sumeragi, a Byzantine Fault Tolerant algorithm which is inspired by the B-Chain algorithm [22]. It splits the set of peers in two $2f+1$ groups and needs $2f+1$ confirmations to validate a transaction. This way, only the first group is essential to validate a transaction and the second group is only necessary when faults happen in the first group. It also uses Hijiri, the algorithm for peer reputation and leader election, that is used to determine the order of processing nodes by identifying which servers are most reliable.

- *Sawtooth* [33] is an open source project for a distributed ledger targeting modularity and safe smart contracts. It introduces the term Transaction Family and provide several core transaction families as way of reflecting transaction types. Sawtooth provides two mechanisms to achieve consensus: Dev_mode and PoET. PoET was explained in greater detail in the Consensus and Consistency section. Dev_mode, as defined in the documentation is a "simplified random leader algorithm that is useful for developers and test networks that require only crash fault tolerance".

- *Burrow* [33] is a permissioned blockchain project created by Monax [52] that executes Ethereum smart contracts on a permissioned VM. It aims to provide high

transaction throughput, multi-chain support, security and data privacy. The permissioned EVM makes sure that correct permissions have been given. It also ensures finality which is guaranteed by the EVM that attributes a random amount of gas to an execution, making sure that it ends. As a consensus mechanism it uses the Byzantine Fault Tolerant Tendermint protocol [48] to guarantee transaction ordering and finality.

2.3.6 Hyperledger Fabric with BFT Consensus

Hyperledger Fabric does not officially provide a Byzantine Fault Tolerant (BFT) consensus protocol, but due to its modularity, it is possible to plug an implementation of a BFT protocol.

In [64] it is detailed an implementation of the state machine replication algorithm BFT-SMaRt for Hyperledger Fabric, which not only provides BFT, but also shows results that its ordering service achieves up to 10.000 transactions/s and it takes 0,5s to write a transaction to the chain.

BFT-SMaRt's ordering service is composed by an ordering cluster and a set of frontends. The ordering cluster consists of $3f + 1$ nodes that will collect envelopes (transactions) from a frontend and order them by following the Bft-SMaRt's replication protocol.

Frontends will serve as proxy between the client and the ordering service to deliver the envelopes and will also receive the blocks that are generated from the ordering service and deliver them to the peers so these blocks can be added to the ledger. This way, a node will create a block containing the envelopes and the hash of the previous block, when it gathered enough envelopes. The node then generates a digital signature for the block and disseminates it to all frontends, which will collect $2f + 1$ matching blocks.

The used ordering service implementation [8], relies on the modular architecture of Hyperledger Fabric to implement a state machine replication protocol on top of BFT-SMaRt's consensus algorithm [64]. When clients send their requests, the consensus process is initiated.

The consensus process consists in three phases: propose, write and accept. The propose phase consists on one replica, the leader, proposing the received batch of requests. The other replicas will validate this batch and the leadership of the sender. In the case it is a valid batch, the replicas will register the batch and send a write message to all other replicas. In the case a replica receives $\frac{n+f+1}{2}$ write messages, it will send an accept message. By receiving $\frac{n+f+1}{2}$ accept messages the decision is complete.

In [64] it is also referred an optimization for geo-replicated environments to the original protocol, WHEAT which enables the delivery of client messages after the write phase and executing the accept phase after sending to the client.

2.3.7 Chain Core

Chain Core [13] is a protocol for an asset management shared ledger. It supports multiple and inter-operable networks in which assets can be managed by transaction posting. Assets are attributed a unique ID between all the chains and when the block is created, the network is capable of maintaining a unique and immutable order to prevent double-spending.

The protocol has three programs to maintain system specifications: Issuance, Control and Consensus programs. The issuance program specifies the set of rules to be applied to new units of an asset, the control program has the rules for spending asset's units and the consensus program controls the rules for new block acceptance conditions. As a consensus mechanism it uses the federated consensus in which a block to be approved needs approval from a set of block signers.

2.3.8 Corda

Corda [15] is an open source distributed ledger platform developed with the purpose of recording, managing and synchronizing financial agreements between institutions. It is a different platform because only parties that legitimately have to know about an agreement, will be able to.

In Corda, networks are semi-private, this means that each network will have a doorman that will enforce rules before a node enters the network. The node will have to provide the required information and, in case the doorman is satisfied, the node will receive a TLS certificate. This certificate will confirm the node's identity when it tries to communicate with other nodes in the network.

Corda also introduced the concept of states, which are shared facts that are acknowledged by one or more nodes. States can contain arbitrary information like stocks, bonds or identity information. Since states can not be modified, changes will be converted to a sequence of states that will represent the evolution of a state.

To achieve consensus, Corda requires two types of consensus to be reached: Validity consensus and Uniqueness consensus. Validity consensus consists of checking that every party involved in a transaction not only possesses the transaction itself but also all the past transactions that led to that transaction. Uniqueness consensus is a solution for the double-spending problem, since it will check that the inputs of a transaction have not been consumed by any other transaction.

2.3.9 Comparative summary on studied Blockchain Platforms

Table 2.1 represents a summary of all the platforms evaluated according to the categories analyzed. For clarification, a bitcoin-like consensus mechanism means that transactions are processed in a similar way as in Bitcoin. Transactions are propagated and validated by all members of the network.

Table 2.1: Platforms description summary.

Platform	Type	Transaction Flow	Consensus	Ledger Replication
Ethereum	Permissionless	Bitcoin-like	PoW	Global
Quorum	Permissioned	Private transactions are propagated to a subset of nodes and private transactions are propagated globally	Raft and Istanbul	Partial
Hydrachain	Permissioned	Private transactions are propagated to a subset of nodes and blocks are propagated globally	HC Consensus	Global
Hyperledger Fabric	Permissioned	Transactions are proposed to endorsers, then to the consensus service and block is then sent to other peers	Kafka or Solo	Partial
Hyperledger Iroha	Permissioned	Transactions are sent to the consensus mechanism, blocks are then propagated to the network	Sumeragi	Global
Hyperledger Sawtooth	Permissioned	Bitcoin-like	PoET and Dev_mode	Global
Hyperledger Burrow	Permissioned	Transactions are sent to consensus mechanism and blocks are propagated to the network	Tendermint	Global
Chain Core	Permissioned	Bitcoin-like	Federated Consensus	Global
Corda	Permissioned	Bitcoin-like	Validity and Uniqueness	Global in public ledger, partial in private ledger

Hyperledger Fabric is an open source project of modular nature that supports chaincode, which corresponds to Smart Contracts. The fact that the consensus mechanism can be altered to a more appropriate algorithm, PBFT for example, is a big advantage when dealing with IoT devices because it can transform consensus in a process that consumes less resources. The adoption of a different consensus algorithm can also increase transaction throughput and provide Byzantine Fault Tolerance if necessary.

By supporting channels, the division of peers in groups, it enables the possibility of adapting an architecture that is able to scale and guarantees privacy by only permitting communication between peers from the same channel using the Membership Service Provider authentication.

Hyperledger Fabric is also able to provide a simulation of the effects of a transaction, called Read-write set, which permits that a quick temporary response can be returned to the application.

Security guarantees can be met by the support of Certificate Authorities for certificate generation. For the mentioned reasons *Hyperledger Fabric* is the most appropriate platform for the design of a generic architecture.

2.4 Blockchain for IoT Applications

2.4.1 Scalability and Blockchained IoT

Nowadays, the usual approach when developing IoT applications is using a "cloud-first" methodology, this is, all data that is collected by the devices is sent to the cloud without any type of filtering or control by the owner.

The processing of the collected data is done in the cloud where a higher processing power exists. In this section, it is described approaches used to solve issues related to the ownership of the collected data and scalability of this type of systems.

In [59], the concept of Blockchain as a Service and evaluates two different solutions using Cloud and Fog computing. It refers that the main reasons why hosting a blockchain directly on IoT devices are the lack of computational resources, the lack of sufficient bandwidth and the need to preserve power. This study differentiates the two computing techniques as fog being resource limited but being able to achieve low latency and cloud being able to scale out and overcome resource constraints by raising latency. In the experiments, the authors realized that adding delays to the multichain will reduce network traffic and that the multichain is not the bottleneck and performance is primarily dependent on the network card and traffic. In conclusion, the results indicate that fog computing outperforms cloud computing for this type of technology.

In [21], the author presents a secure and resource friendly solution for blockchain using IoT devices. This solution is based on a three tier architecture which includes a smart home, an overlay and a cloud storage.

- The smart home is composed by IoT devices, a local immutable ledger and a local storage. This ledger is managed centrally by a Smart Home Manager (SHM). The SHM will process all incoming and outgoing transactions and communicates with IoT devices and local storage with a shared key.

- The overlay tier is constituted by the nodes. These nodes are arranged in clusters in order to decrease network overhead and in each cluster, a Cluster Head (CH) exists, which is responsible for block generations. These CHs also have a public blockchain which will store transactions sent by the overlay user and will be used to gain reputation.

- The cloud storage tier will associate the user's data with the block number as a way of SHMs being able to authenticate. The authors present a trust system as a way of decreasing overhead for block verification based on the transaction history since only a portion of the transactions in new blocks will need to be validated.

With this architecture it is possible decrease both traffic and processing overhead when comparing with results obtained in a Bitcoin network.

In [6], several security and privacy preserving techniques for IoT and blockchain systems were discussed. The authors propose an architecture for dataset sharing in

research communities in such way that integrity is preserved. To achieve this, a reference integrity metric was created and is maintained using the blockchain. This architecture uses a central hub that stores references to other repositories in the network where the datasets are stored. It solves the challenge of lifetime in the dataset. That is, an owner of a dataset may not be interested in sharing it permanently. Only the integrity metric is maintained by the blockchain and will remain there, this way if the owner decides to stop sharing it, the dataset will not be available.

2.4.2 Blockchain-Enabled IoT Applications

2.4.2.1 Characteristics

As [20] states, Internet of Things (IoT) are devices that generate, process and exchange volumes of not only security and safety-critical data, but also privacy sensitive information. These devices have to implement security measures on top of the processing of application functions and all this in a resource constrained environment.

Although the IoT has grown exponentially since most devices used are now connected to the Internet, there are still some issues related to privacy and security. One example of these issues is the lack of control that the user has in the data that is sent to the service providers, the cloud for example, when using IoT devices. As explained in its Smart Home example [20], there are some security measures that must be applied, such as devices should be indirectly accessible, different sectors of the system should use different transaction structures and the use of symmetric encryption.

As defined by the article [47], a blockchain solution with IoT can also solve server downtimes and unavailability of services due to the fact that there is not a single point of failure and information is replicated across several devices.

2.4.2.2 Blockchain Limitations for IoT

Using a blockchain in an IoT environment could be very positive, nevertheless there are some disadvantages that have to be overcome. In a system like Bitcoin, the mining process is very demanding in terms of processing power. IoT devices are, usually, very resource constrained, which means that all processing that is done in the node has to be minimized. The process of mining is usually very timely which will translate in higher latency times. One possible solution could be to change the consensus mechanism to an algorithm that requires less processing power from devices. Instead of having all nodes competing to solve the puzzle first, it could be used an algorithm for consensus that does not rely on each node's computing power. For example, using an algorithm such as PBFT can greatly outperform both transaction latency and throughput results obtained in Bitcoin [17].

As observed in a previous section, the blockchain has scalability issues when expanding its number of nodes. When using IoT devices, it is expected that the number of devices is high, this way it is expected that performance is affected. Another issue is related to

traffic, some IoT devices can be bandwidth limited, in these cases, using a blockchain can generate more traffic than desirable.

2.4.2.3 Design Issues

When designing an architecture for a system that uses a blockchain and IoT devices, there are several aspects that have to be taken into consideration.

First, since IoT devices are not very resource powerful, the system can not be too computationally heavy. A solution could be the use of the PBFT algorithm [12] as a consensus algorithm.

Second, in Bitcoin's system, there is a monetary reward for a peer that behaves correctly in the network, but in some cases, a reward could be a speed up in block validation time by only needing to validate a subset of the transactions contained in a block based on a reputation system [21].

Third, if a system is growing exponentially, strategies will be necessary in order to minimize the impact in performance. An example of this can be the introduction of a hub that plays the role of representative of a cluster of nodes in a global blockchain centralizing communications in the cluster.

An example of this is [6] where a hub is introduced. This last solution also solves the privacy issue where a user is able to take control of the data that leaves its smart home for example.

2.5 Summary

This chapter presented an overview on principles and foundations of the Blockchain, presenting the characteristics and properties of several Blockchain platforms and highlighting some ongoing and relevant problems on scalability and reliability issues, as well as, performance bottlenecks. These concerns are particularly relevant for the use of blockchain-enabled services and mechanisms, as leveraging elements for Blockchain-Enabled IoT Platforms and their software architectures.

Several development platforms were studied and compared in this chapter, focusing on some relevant characteristics, as possible approaches for the thesis objective. Among the different characteristics found in the studied platforms, Hyperledger Fabric (HLF) was selected as a viable and appropriate platform to develop a generic architecture for IoT integration and for our expected contributions, as initially addressed in the Chapter 1. Summarizing, considering the objective and contributions of this dissertation, the relevant criteria for this choice are the following:

1. HLF supports a modular architecture, enabling designers to plug in their preferred implementations for specific components, which is a strong advantage. Important components of the architecture can be easily plugged, including consensus or cryptographic functions, for example. Exploring such modularity, we can address a

"pluggable" use of different consensus algorithms, to better achieve optimized balances between fault tolerance and consistency guarantees, security and performance. Particularly, this gives the opportunity to design and support different solutions for byzantine-fault-tolerance requirements, as different modules in the consensus plane support, in addition to the officially provided crash fault tolerant Kafka solution, for distributed consensus and ordering service;

2. Related with the previous point, HLF has a design structure promoting separation concerns between membership management and consensus control. This is an advantage for combining membership management at the level of the order-service and the supported byzantine-fault-tolerant state-machine replication model, a relevant issue for scalability and performance, in the global consistency model of the network;

3. HLF provides the interesting concept of HLF channel, as a way to control the information that can be shared only between nodes belonging to the same channel, with the possible support of multiple chains, which can be used to isolate and structure transactions between different peers in a more straightforward way. We also find that it is possible to execute the HLF's off-chain consensus mechanism in clusters, local to the entire blockchain network, maintaining the support for a distributed ledger environment under scalability control, without depending on a third-party and avoiding the risks of centralized components in the network.

4. In the modular architecture of HLF, a relevant advantage is the clean separation of transaction processing into three phases: distributed logic processing and agreement ("chaincode"), transaction ordering, and transaction verification and commitment. This separation has important advantages: fewer levels of trust and verification are required across node types, and network scalability and performance can be optimized, with possible conditions and invariants expressed for chaincodes.

5. At the same time, HLF provides openness and extensibility of chaincodes, usable as a foundation for smart-contracts, allowing for possible extensions in ruling and expressiveness of programmable transactions' processing control and required invariants and verifications, during the execution of such smart contracts;

6. Despite not providing anonymity, as happens in other Blockchain platforms, this is not an issue for the dissertation purpose, in the sense that we envisage IoT environments in scenarios such as Consortium-Platforms, where devices and participants act in a permissioned model, supporting transactions only available to a subset of the nodes, which is different when compared with open environments supporting a permissionless model and public transactions, as happens for example in many public cryptocurrency ecosystems;

7. Compared with other platforms with some characteristics similar to the requirements we are looking for, HLF is available and extensible as an open-source platform model, with larger and dynamic research and development community conducting academic and industrial work initiatives. Due to the flexibility, openness and design structure of reusable internal services, we find that HLF can be more easily adopted for different deployment environments, namely: controlled blockchained services distributed in private data centers, or use of blockchain backend services running in a cloud-provider computing and storage solutions (a facet that we explored in the dissertation, for implementation and experimental evaluation purposes).

In our study of how to address IoT architectures and their possible scalability requirements, we find solutions where the authors of proposed architectures defined the major flaws and vulnerabilities to be aware when dealing with such systems. These studies gave an insight of techniques and design models that can be applied in the design of large scale blockchain services for IoT, such as the use of a hierarchies or Cluster Heads, and the use of intermediation Hubs, as gateways between IoT devices and blockchain-enabled services.

Considering the analyzed characteristics in Blockchain-enabled services, and selecting the Hyperledger Fabric solution as a reference blockchain implementation for the objective of the dissertation, we can address integration or adaptation capabilities, where the concerns initially discussed for scalable, trustable and reliable IoT Platforms can be merged in a trustability model, under the advantages of a decentralized ledger environment.

In the next chapter, it will be defined a proposal for a generic architecture for addressing a Blockchain-enabled IoT platform and related software architecture.

Chapter 3

System Model

In this chapter it is presented the System Model and reference architecture for a blockchain enabled IoT platform. This way, the chapter discusses the main goals and the related components that were particularly targeted for the proposed platform, starting by introducing the system model assumptions, following by overviewing the system architecture, from which we illustrate the support for a reference IoT-application support scenario. To conclude we discuss some relevant dependability issues, as addressed in our proposed solution.

3.1 System Model Overview

In this section it is given an overview of the system model, given its entities, the system operation and the application-level support.

3.1.1 System Model

We will start the explanation of our system model by summarizing the features of the blockchain services that we will use as "backend"services for our proposed IoT platform. In the related work chapter, there were several Blockchain characteristics defined which were applied when designing the system model.

The main characteristics that were applied are the decentralization features, which are achieved by using the Hyperledger Fabric's distributed ledger, the immutability of auditable operations and autonomy, because the system is able to process transactions representing interactions between users and IoT devices, without requiring a central trust entity. Another important characteristic about the use of Hyperledger Fabric for our proposed architecture is the reliability conditions and consistency guarantees, under dependability criteria, enabling the possibility of changing the consensus algorithm to

address different guarantees, ranging from the "native" fail stop model in HLF version 1.1, to a byzantine-fault-tolerant model, as we analyzed in our proposal, as well as, in its implementation and experimental evaluation.

Following the different planes in a blockchain, as discussed in the related work, we will structure the blockchain services in our solution in the same way as defined for blockchain services, e.g., as a way of separating the different layers of services. Then, we identify in our proposed solution, the corresponding planes, as follow:

- **The Network plane** This plane represents the base foundations of Hyperledger Fabric, which is composed by peers, endorsing peers, orderers, CAs and replicas. Endorsing peers will enable the endorsement of transaction, digital signature, creating the correspondent endorsement signature. Orderers will collect envelopes (with data for transactions' processing) and execute the replication protocol, with the total order guarantees in operations' execution. The bootstrap Certificate Authority created by Hyperledger Fabric, will deal with the registration of identities and the involved replicas distributed in the blockchain, executing P2P transactions and storing consistently the related data.

- **The Consensus plane** This plane supports different consensus guarantees for consistency control. In our approach we consider two flavors for the consensus plane: the native Kafka fail-stop service (initially existent in the HLF version 1.1), and an enhanced order service, supported for Byzantine Fault Tolerant guarantees. This service is supported by Bft-SMaRt – a state machine replication/consensus library [7] with the provided optimizations for wide-area communication [63], leveraged from the modular architecture of HLF and initially implemented to be rehearsed and experimentally evaluated. We must notice that Hyperledger Fabric provides Kafka as an ordering service. Although it has high performance and supports fail-crash tolerance, it is not a Byzantine Fault Tolerant (BFT) consensus solution. This way, we aimed to evaluate experimentally a solution that incorporated such BFT assumptions, targeted by the Bft-SMaRt enabled ordering service for our HLF deployment. This solution is also able to achieve acceptable performance values, even when peers are geographically distant, particularly under controlled membership conditions at the level of the consensus group.

- **The Storage plane** This plane corresponds to the layer responsible for storing the data which will act as the distributed database for the proposed IoT platform, to store data processed under the verification conditions established by chaincodes.

- **The View plane** Represents the visibility state of the blockchain for integrated applications, being the visible state to external clients for user interaction in an IoT application environment (which in our case can be IoT devices or users' devices, interacting through hubs with the blockchain-enabled services, as explained later). This means that the view plane provides the necessary services to externalize the

final state from all the operations executed and logged in the backend blockchain services, when supporting IoT applications.

By taking advantage from the model of HLF Chaincodes, are in its essence regarded as programs that are responsible for implementing the application's logic ruling the executed transactions, we use chaincodes in our proposed architecture seen as the base for the expression and establishment of smart-contracts, holding the definition of the data structure, arguments, and conditions do the necessary verifications. Leveraged by the HLF support, the logic processing and agreement ruled by those smart contracts have data contexts represented in JSON, and they are fully queriable, with the data-model compatible to be stored in key-value stores in each peer. The HLF key/value storage (KVS) used was CouchDB [4], as an alternative to the native solution LevelDB [49]. As reference, the industrially available HLF packaged solution from IBM [40] offers today alternative storage solutions, such as MongoDB [53].

3.1.2 System Architecture

In our system model and related architecture we consider a set of main entities, as represented in Figure 3.1: Blockchain, IoT devices, Mobile Users, IoT Service Provider (SP), Smart Hubs (SH), Channels and Cluster Heads (CH). These entities will be described in this section for a better insight of the proposed system model.

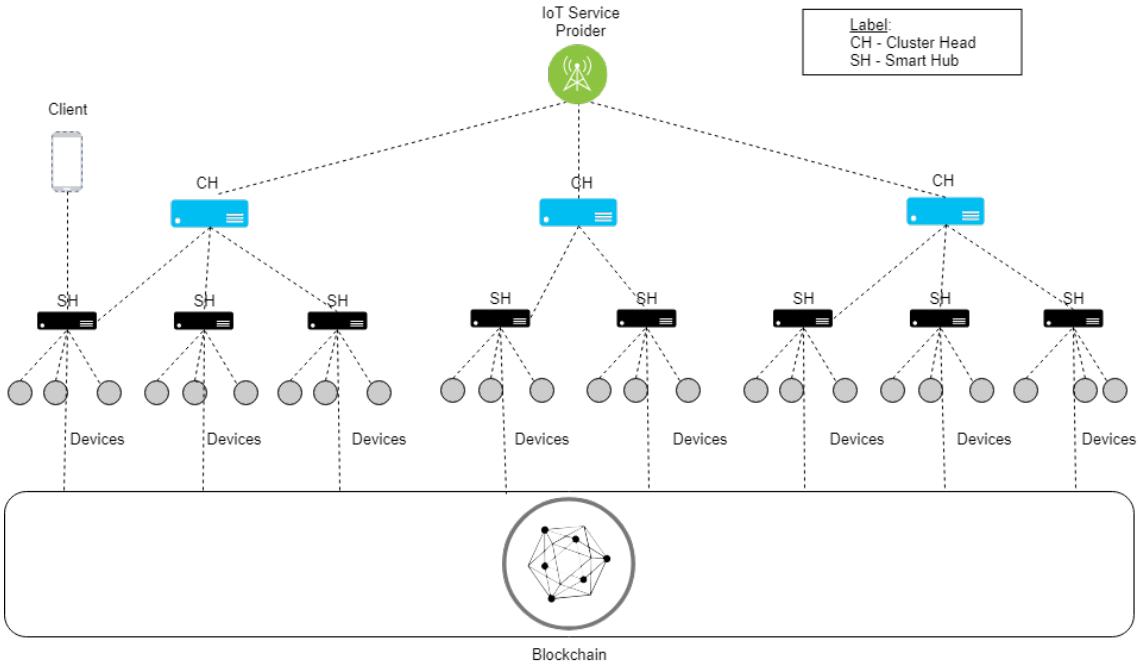

Figure 3.1: Representation of the system's architecture.

The **Blockchain** represents the distributed registry and ledger, with the backend services supported by each planes, and providing the means to store transactions, supporting the interactions of the external entities. In our software architectural model, such

services are leveraged by HLF, which is responsible for the receiving and processing of transactions proposed and primarily pre-processed by Smart Hubs (SH).

Transactions are signed and proposed by the Smart Hubs, and then the endorsing peers (in the HLF based network plane) confirm that the transaction is well formed and that it has not been already submitted. The signatures are validated by the Hyperledger's Membership Service Provider, under permissioned conditions, meaning that it is verified that the SH has the right to perform the proposed operation on the used HLF channel. We must notice that in the HLF it is possible to set up Endorsement Policies for transactions. This enables definition of which entities need to endorse a certain transaction, in order for the peer to consider it as properly endorsed.

The **IoT devices** are specific devices, from different vendors, as in our architecture are considered as application-specific devices. Later, we will instantiate the use of our system model in the context of a locker management system, and in this case, these devices are smart-locks, which will open or close on a signal sent by the hubs with which the devices communicate. In general, IoT devices can be very heterogeneous, in its specific functions or sensing capabilities, software/firmware/hardware packaging, as well as, in supported communication protocols.

Depending on different solutions, the Smart Hubs used for the interconnection of such devices, must provide the base support, according to the specific interoperability requirements, which stands for the nature, characteristics and processing capabilities of those Smart Hubs.

Client is the device used by the user in order to interact in the context of an IoT application. Related to such interaction, the client provides the transaction's details supported in endpoint services offered by the Smart Hub. We envisage Clients as mobile applications accessing IoT services, which will solely contact the IoT platform via a Smart Hub and trough a provided REST service interface, for HTTP or HTTPS based interactions.

The **IoT Service Provider**, is the stakeholder which could benefit from the proposed architecture in order to implement its business, as the provider of IoT applications.

For the purpose of the dissertation, we can see this entity as an external entity, interacting with the proposed platform model through Cluster Head components, supported by REST-based operations, using interfaces that provide the tools for managing the applications, sending the required parameters that will be used as input parameters for execution control, managed in the proposed architecture. We can see such parameters as a variety of parameters, such as application-level parameters: pricing conditions, payment-gateway endpoints for payments and reception of payment proofs, agreement or liability conditions for users, etc.

However, we can also consider other parameters that can influence the way how transactions will be validated and completed in the blockchain. All those parameters can be included as parameters and rules, in the expression of smart contracts, to be considered, at the level of the blockchained services. Then, the parameters and conditions will be invariants for processing control and validation of transactions, scrutinized and audited

under verifiable logging conditions by all the required entities involved in the distributed ledgering environment.

Smart Hub is a device regarded as software/firmware/hardware appliance that operates as a smart gateway between the other entities and the backend blockchain services. In a generic vision, it acts at the middleware level, as an intermediary for the communications between users, IoT devices and the blockchain-enabled services (accessible from the provided services at the View-Plane level). A Smart Hub will have a set of IoT devices, which it will be responsible for the IoT devices and users interacting with such devices, behind the Smart Hub, form local edged IoT ecosystems, interconnected through the Smart Hubs, and using the functionality provided. The Smart Hubs forward locally-provided operations as remote transactions to be executed and controlled in the backend blockchain.

A **Channel** represents a mechanism which allows for private communications between different peers that belong to the same channel. These channels are mapped in the HLF channels serving as as an entity that aggregates Smart Hubs associated to the same Service Provider. This enables the possibility of a Service Provider, which could hold different Smart Hubs from the same business, to consult transactions that were inserted in all of them.

The **Cluster-Head** acts as an intermediary for communications between Smart Hubs and Service Providers. The communications with the Service Providers are done via the REST interfaces for remote operations. This entity follows the same architectural concerns expressed in [21] which enables a reduction in network overhead and delay, since communications will be only done with the targeted Smart Hubs in each case.

Presented the entities of our proposed system model for a blockchain-enabled IoT platform, we must add some information on addressing the notion of Smart Hubs, to clarify how we see the materialization of these components in a more generic way, and the more focused vision for the purpose of prototyping our system model and related software architecture, in the scope of the current dissertation.

3.1.3 Materialization of Smart Hubs

The use of the "Smart" word to express our imagined functionality that must be provided by a Smart-Hub must be regarded in a generic perspective. The Smart-Hub is seen as an intelligent gateway that can be designed to be a pluggable computing appliance, where different service modules can be installed for running, on top of an hardened Operating System. The idea is to look to such service modules, as possible virtualized and isolated pluggable software containers, installed and managed in a flexible way and dedicated to the specific required functions.

We envisage different functions, such as: data aggregation functions, data and traffic filtering functions (possibly using tainted-data analysis and filtering mechanisms to avoid undesirable data exfiltration/infiltration or privacy breaches, internetworking firewall

functions, authentication services for IoT devices in a pairwaised way, intercommunication and protocol-conversion facilities to intermediate the possible heterogeneity in device-to-device operations, cryptographic processing functions, and other application-specific modules for local-processing requirements of particular IoT applications.

A relevant function in Smart Hubs is the provisioning for supporting multiple protocols, suitable for serving different edged-based IoT devices and IoT environments, particularly addressing subsets of protocols in the panoply of "IoT standards" that have been promoted: for wired or wireless communication protocols and internetworking infrastructures, device-level identification and addressing, transport-level protocols, discovery protocols, data-dissemination protocols, request/response protocols and remote-calling operation environments, as well as, data-semantic and representation protocols [44], [42].

In a complete design and implementation of such smart-hubs, those multi-protocols can be addressed as required supports in the context of multi-layer frameworks, targeted for different IoT markets, devices, and applications.

Although we can expect that the hub, with this generic perspective for industrial approach, can have several features in a richer support, we are more interested in regarding smart-hubs, for the implementation purpose in prototyping out proposed platform, as possible low-cost SW/HW appliances, implemented from cheap and convenient software/hardware. Our primary goal is the support of functions for local-operation REST-based proxy-service, provisioning natively TCP/IP supported IoT devices, to process local operations (requests) forwarded as blockchain-enabled transactions.

3.1.4 System Operation

Figure 3.2 represents the system operation. The first step in the system operation is for the client to perform its registration. Next, the service provider will contact the Cluster Head using the implemented REST interface in order to set the application-specific parameters, represented as contract rules, which could be, for example, price/time value of the provided service. This way, all Smart Hubs will be informed of this information and will be able to construct and propose a transaction to Hyperledger Fabric. A confirmation will be sent to the Smart Hubs. Following the confirmation, the system will use the its implemented REST interface in the Smart Hubs in order to consult the contract rules. The Smart Hub will propose a transaction in order to view this information which is stored in Hyperledger Fabric. The response is then sent to the Smart Hub and the required information is shown to the user. The user, will then send the transaction information to be able to use the system. The payment is then processed between the Smart Hub and the Service Provider and, when accepted, the transaction proposal is built and sent to Hyperledger Fabric. A response will then be sent to the Smart Hub, which will send a confirmation to the client and send an application-specific event to the IoT device. In the case of the locker management system, this event could represent the opening of a lock.

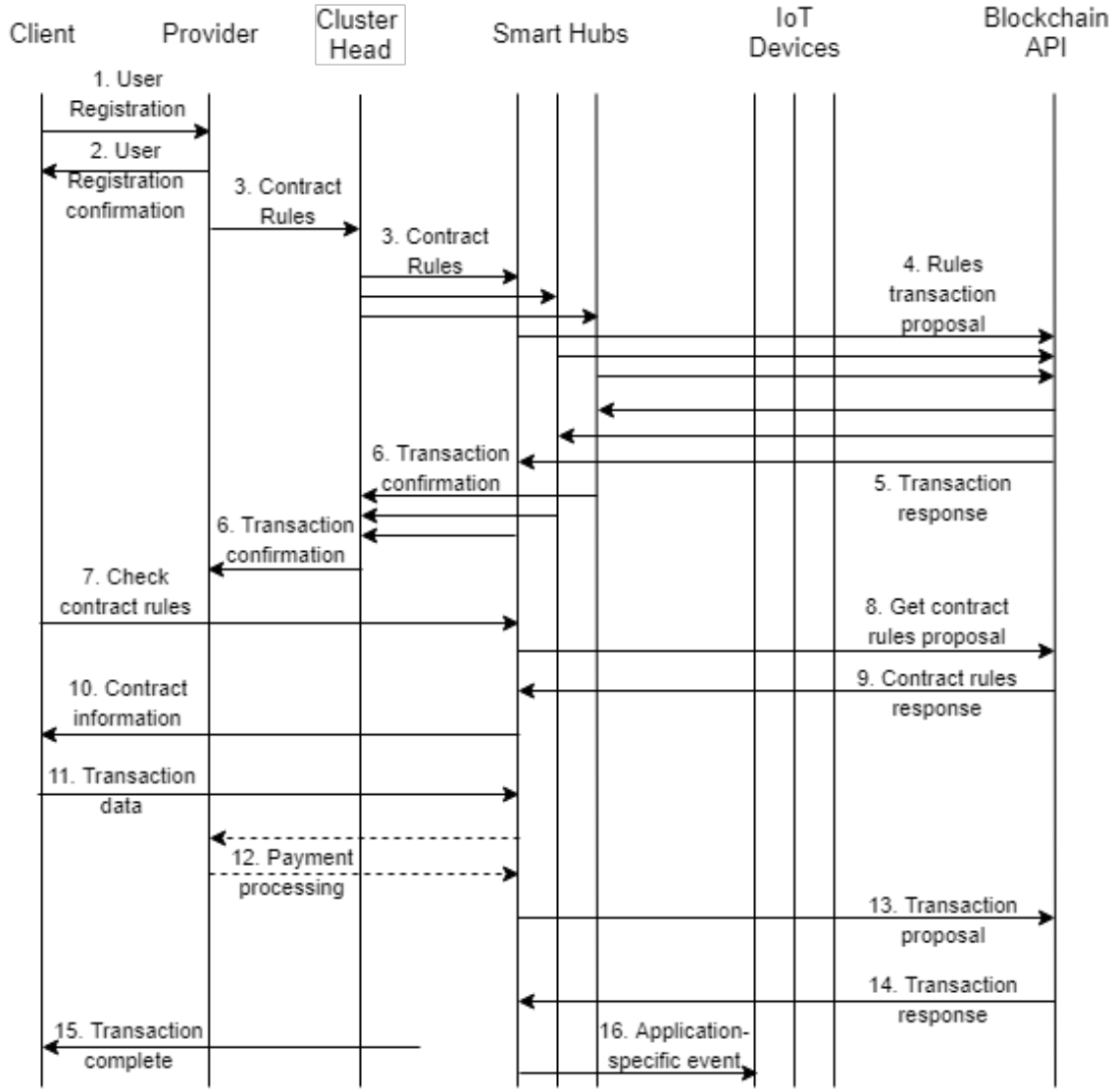

Figure 3.2: Representation of the system operation.

3.1.5 Application-level Support

In order to support applications, three REST interfaces are provided. The first one will be used by Service Providers and is available through the Cluster Head, in order to set application-specific details. The methods implemented and their respective descriptions are represented in Table 3.1. The second interface is directed to the user and is available through the Smart Hub and is used to register user proposed transactions. The implemented methods are represented in Table 3.2.

The use of the first interface enables the possibility of creating a concept of an extended chaincode. This is a chaincode with parameters that can be set via an exterior method. For example, in the locker use case, it can be set different sectors for locker placement and for each sector an associated price. The set of this value permits that the same chaincode can be used for different sectors.

Method	Description
POST /initPrice/{sector}/{price}	Sets the price per hour value for the specified sector.
GET /readSectorPrice/{sector}	Reads the price per hour value for the specified sector.
GET /getSectorsByRange/{startId}/{endId}	Retrieves the sectors and the respective price whose ID is between the specified.

Table 3.1: Cluster Head's Interface.

Method	Description
POST /initRental/{lockerid}/{sector}/{duration}	Starts a rental for the specified locker, sector and duration.
POST /endRental/{rentalId}	Ends the rental for the specified rental id.
GET /getRentalsByRange/{startId}/{endId}	Retrieves the rentals whose ID is between the specified.

Table 3.2: Smart Hub's Interface.

Method	Description
POST /openLocker	Opens the locker.
POST /closeLocker	Closes the locker.
GET /getLockerState	Retrieves the state of the locker.

Table 3.3: Locker's Interface.

Table 3.3 illustrates the REST interface of a locker.

There are two data structures that are created with chaincodes are invoked. The sector data structure, represented in Table 3.4 has two attributes with the Sector's identifier and the associated price. The rental data structure, represented in Table 3.5, is composed by the transaction's identifier which is a sequence, the locker's identifier which is being rented, the sector in which the locker is located, the duration of the rental and the price which is calculated from the duration and the price of the associated sector.

Field	Type
Sector	String
Price	String

Table 3.4: Sector's data structure.

3.2 Application Scenario

The application scenario, which represents one of the initial motivations for this dissertation is a Locker Management System. The description of this application scenario together with the system model entities and interactions, enable a better understating of

Field	Type
TransactionID	String
LockerID	String
Sector	String
Duration	String
Price	String

Table 3.5: Rental's data structure.

the next section, System Operation, where the interaction between the different system entities will be described in more detail. This interaction is represented in Figure 3.3.

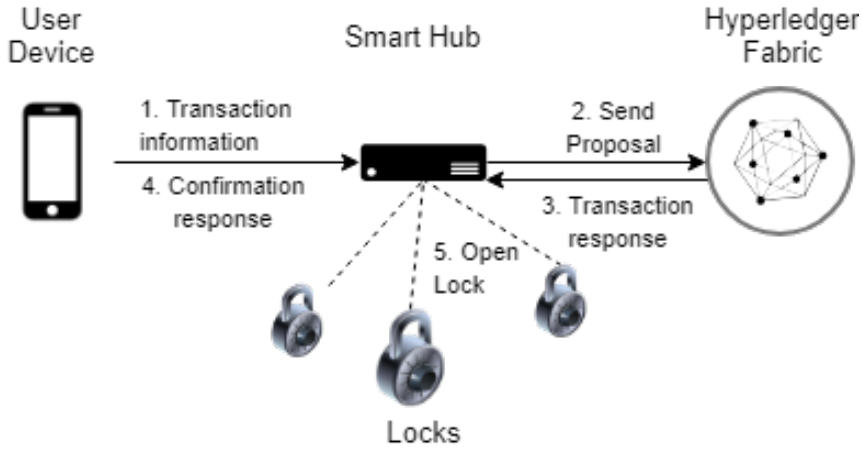

Figure 3.3: Representation of the application scenario.

The Locker Management application represents a system without a central trust model and authority, capable of functioning both for users and service providers in a decentralized ledger manner. Service providers, which would be the owners of sets of lockers, possibly installed in different places of cities, as a smart-city application context, would set up their lockers, in the specific locations, as IoT devices interconnected by Smart Hubs. Each set of lockers would have a Smart Hub and all Smart Hubs from the same provider would be grouped by a Cluster Head. This Cluster Head is responsible for transmitting Service Providers' information to all Smart Hubs. Smart Hubs are responsible for:

1. Acting as an intermediary for communications between users and the blockchain. This way lowering complexity from the user's application, as well as the software running in lockers, which are obviously resource-constrained devices;

2. Establishing communications with the Cluster Head which would lower the communications necessary for the Service Provider;

3. Process orders from Clients for renting (open/close) the lockers, order a specific locker to open or close, as the result of client-orders;

Service Providers would contact the Cluster Head in order to set up the price of renting a locker in that sector, as well as, to setup other additional information and conditions, which would then be transmitted to the Smart Hubs in order to locally pre-process orders, to execute the required transactions.

Users would contact a Smart Hub in order to initialize a rental. The Smart Hub would take the user's request and information and dispatching the order as a transaction to the backend blockchain. After validation, the user would be informed and, in case of validation, the rented locker would open. According to the specific smart-hub control, the locker could be locked, after the validated renting period, informing the user under new request for open or close that, in this case, she/he must contact a contact-point for some specific reason.

In our application scenario we are not concerned with complementary processing requirements behind the validation of transactions in the blockchain, except those executed via chaincodes in the processing of blockchain transactions. In specific applications, the execution of transactions in the blockchain executed by extended verification procedures, in the context of smart-contract executions could fire a composition of remote notifications in other external services, or can execute remote operations in external services of possible different entities.

Naturally, these external operations can condition orthogonally the full validation of the transactions, in addition to the checks on the internal processing of consistent logging of the state of operations in the blockchain. This is possible regarding additional interoperability requirements between blockchain processing and the interaction with such orthogonal services, out of the scope of our basic system model assumptions.

The installation of lockers, enabled as IoT devices interconnected by a Smart Hub, form a local infrastructure that can be installed in different city places, such as metro or railway stations, airports, supermarkets, malls, football stadiums, a university campus, etc. Figure 3.3 represents a user that interacts with the smart hub with a personal smartphone (using a IoT specific app), to rent a locker (controlled by the represented smart hub) that it is associated to a node in the backend blockchain.

3.3 Dependability Issues

3.3.1 Threat Model and Security Trends

The threat model set is related to the protection of communications between the different entities of the system and the protection of the data that is stored in the Blockchain. The protection of the communications is done using TLS. The communications that are protected are the communications between the client and the SH, the SP and the CH, the SH and the Blockchain, the SH and the device and between the different components that compose the Blockchain network.

The communications between client, SH, CH and devices are all done via the REST interfaces and uses TLS, which means that the traffic is encrypted. The communications between the network entities are done using the gRPC protocol with TLS. All connections between different system entities are done with one-way authentication.

The protection of the stored data is guaranteed by the encryption of the data itself. In the chaincode, the payload of the proposed transaction is encrypted to hide its content. The method used is the same as in [27] which is the method provided by Hyperledger Fabric which consists in encrypting data in the chaincode itself using AES 256.

3.3.2 Failure Model and Intrusion Tolerance Guarantees

The failure model which is supported is related to the failure of Hyperledger Fabric network entities and the failure of the Smart Hub. Since the system implements a BFT ordering service on top of the BFT-SMaRt [64] protocol the replicas, which will store the information saved in the blockchain, are tolerant to Byzantine failures.

The ordering service implementation [8], relies on the modular architecture of Hyperledger Fabric to implement a state machine replication protocol on top of BFT-SMaRt's consensus algorithm [64].

Clients send their requests which are collected by the frontend delivered to the ordering service. Ordering service nodes will create a block containing the envelopes and the hash of the previous block, when it gathered enough envelopes. The node then generates a digital signature for the block and disseminates it to all frontends, which will collect $2f + 1$ matching blocks. These blocks are then delivered to peers to be added to the blockchain.

The Smart Hub also implements a recovery mechanism in case it fails. The users that were previously registered are stored in a permanent storage which is then used to recover the state. The existent channel is then reconstructed and the system is ready to operate again.

3.3.3 Scalability Issues

The scalability of the system follows the method which is described in [21]. This method consists in organizing the system architecture in such way that the devices are organized in clusters being and the communications to the devices are all done by one device, which, in the case of this architecture is the Smart Hub. This way, it is possible to greatly reduce network overlay. Another method consists in using Cluster Heads in order to reduce the communications that have to be done to Smart Hubs that belong to the same Service Provider.

3.4 Summary Remarks

In this chapter, it was defined the system model as well as the system architecture and operation for a proposed blockchain-enabled IoT platform. It was also illustrated the use of the proposed model in the context of an application scenario, to clarify the system operation and the provided application-level support. Finally, we discuss complementary dependability issues in the proposed architecture. In the next chapter we will describe the implementation of the discussed ideas in a prototype that we used for conducting the validation of our proposal and to support the related experimental assessment.

Chapter 4

Implementation

This chapter presents the implementation of the proposed system's architecture, following the system model assumptions for the proposed blockchain-enabled IoT platform and architecture. First, an overview of the implementation is addressed, followed by a description of the components of the proposed architecture and the technology used for the prototype. Then, some relevant system development issues and setup mechanisms are described. The chapter is concluded with a summary of the implementation issues and related remarks.

4.1 Implementation Overview

The implementation of the proposed architecture follows the base system model as proposed in Chapter 3, with the background inspired by the Locker Management System, as an Application-Scenario, as a simple concretization for a smart-city application context.

In this application, IoT devices represent as lockers and the Service Provider (SP) represents providers. These SPs will set different sectors of lockers (as locker cabinets) that can be installed in different places of one or more cities. For this setup, SPs will configure associated pricing and payment conditions. This application and its exemplificative requirements are supported in the proposed blockchained IoT platform (regarded as a reusable architecture for other IoT applications), structures in the seven main components: Client, Service Provider, Cluster Head, Smart Hub, Devices, the Blockchain-enabled backend services and related network infrastructure.

4.2 Implementation Components and Technology

The Client was implemented in a mobile application that was developed using Ionic Framework 3 [43]. Ionic Framework 3 is a development framework that enables the creation of hybrid mobile applications. The application enables users the possibility to use the system, and the motivation was to create an application environment similar to real mobile Apps, that will be used to support users' interactions in the context of edged-based IoT ecosystems, interconnected and served by the Smart Hub (SH) proximity functions.

The App is able to read sector prices or rental conditions, initiate a new rental and retrieve the rentals, after the rental validation and acceptance conditions. The Client communicates with the SH via the provided REST interface using HTTP or HTTPS requests, depending on the specific setup.

The Service Provider was also implemented in a mobile application using the same framework Ionic 3. The purpose of this application is to enable Service Providers the setup functionality during the insertion of sectors for the promoted lockers, and its correspondent rental conditions. The Service Provider communicates with the Cluster Head (CH) via its provided REST interface.

Thus, the interactions of Clients and SPs are done in a web-based internet environment, anticipating a common interaction model that can be used for the enhancement of Client Applications and their specific extension of functionalities, as well as, other Web applications and services for SPs, used as orthogonal Backend environments for business management functions in a Business-to-Client (B2C) application context, or client-relationship management (CRM) functions.

The Cluster Head implements a REST interface which will receive the requests from the Service Providers and communicate these requests to the Smart Hub. The Cluster Head (CH) is implemented in Java, using the Java 8 development framework.

Figure 4.1: Representation of the Smart Hub's stack.

The Smart Hub is implemented with the support of three main components, as illustrated by Figure 4.1:

1. The *REST API*, which was implemented using the Spring Boot [65] framework, dispatches operations from Clients or SPs, managing the related information flows

from the mobile application, processing and forwarding such operations as transaction to processed and controlled, by the backend service planes provided by the blockchain environment.

2. The *Java application* components, responsible for constructing the transaction proposal that must be processed by the Hyperledger Fabric leveraged functions, at the Blockchain level. There are two main steps involved in the forwarded SH transaction processing: proposal step and the query itself, depending on the functionality required in the case of read or a write transactions.

3. *Hyperledger Fabric Java SDK* functions for SH runtime, where the required methods for communicating with the Hyperledger Fabric services and network are processed, to support the required information control features and transactions' states in the blockchain.

In the prototype we made two different setups for the SH, using both variants for validation, assessment purposes and experimental observations, as presented in Chapter 5.

One setup is based on a *"Software-Emulated SH"*, that we can run in different computers, with the solo requirement of a setup installation of the related runtime support components (e.g., Java 8 Runtime Environment, Java SDK for Hyperledger Fabric 1.1, Apache Maven 3.5.0 and the required cryptographic material generated by the HLF network for the authentication of entities).

The second setup is based on a "Sofware/Hardware SH appliance"implemented with a Raspberry Pi computer, with the motivation for testing as a cheap and convenient SH option for real-deployments.

For this implementation, we used a Raspberry Pi Model 3 B+ [57], with a Broad-Com chipset, 1.2GHz Quad-Core ARM Cortex-A53, 802.11 B/G/N Wireless LAN and Bluetooth 4.1 (Bluetooth Classic and BLE enabling support), 1GB RAM, 32 Bit CPU and additional 4 x USB ports. The chipset board includes a native 10/100 BaseT Ethernet connector, supported by an independent controller, avoiding the use of USB-emulated wireless Ethernet dongles.

The setup for the SH implementation includes the use of the Raspbian (Linux) OS distribution, the Stretch lite 9 version (Kernel version 4.14). We used Java version 8 and it was installed the required runtime support for the SH, including: Apache Tomcat Native SSL library version 2.0.9 which requires compilation due to the Raspberry Pi's ARM 32 architecture.

Despite the two setup variants for the SH developed functions, the interaction provided for Clients and related Mobile Apps, is transparent and only have repercussions in the evaluation parameters for the support of the REST-based operations, as we analyzed in our experimental observations.

The implementation of the Blockchain in the backend of our prototype include all the entities in the Hyperledger Fabric service planes and network support, that will deal with the processing of transactions sent by the SH. This includes peers, endorsing peers, orderers, CAs (Certification Authorities) and replicas. Peers will store information. Endorsing peers will process transaction by signing the respective endorsement signatures. Orderers will collect envelopes and execute the BFT-SMaRt's replication protocol to order them consistently. The CA entity at the blockchain level will deal with the registration of identities and the replicas, managing and storing the related information, which includes the public-key certification policy management, for trustable public-keys as pure and anonymous identifiers representing the blockchain nodes.

With the background of the application-scenario, we implemented two chaincodes, as the expression of smart-contracts, with the included validation executions, for the locker management system example: one chaincode for locker sectors, as a chaincode for the aggregated IoT devices in a sector, and a chaincode for rentals.

The sector chaincode enables the creation of a sector and associated pricing conditions. It enables the possibility to service providers to define the price of renting a locker for a specific sector. This chaincode holds the data structure for sector, which is defined by the Sector ID and the associated pricing conditions, considered for validation purposes. The rental chaincode initializes a rental of a locker, which is located in a certain sector. The price is based on the sector price, which was previously inserted by defining the first chaincode. A rental is composed by a Transaction ID, a Locker ID, the Sector reference, the Rental duration and the price.

The Fabric CA is the entity responsible for the management of the identities and public-key certified associations of the HLF participants, through the management of X.509 certificates for ECDSA certification signatures, maintaining the information on a SQLite database repository [66], in order to store and to retrieve this information and the associated certificates.

The Blockchain represents a decentralized ledger, where all operations are recorded in as a total ordered and consistent sequence and transactions (contained in their processed blocks) are immutable. However, in an application it is useful to use a state database which records the current state of the stored data in each peer. This way, we decided to use CouchDB as the state database organized as a key-value storage component.

In the developed application, each key is the ID of the entry, this is its unique identifier, and the value is a JSON object containing the data. The ID of the sector is given by Service Providers and its uniqueness is verified. The ID of the rental is a concatenation of the Sector ID, Locker ID of the rental and a timestamp. The data of the transaction are all the fields from the data structures defined in the chaincodes. As CouchDB supports queries based on keys, such as setting and querying operations given the key, and it includes support queries within a specified range [16], the option (among other alternatives for peer state-storage in HLF) reveled the required flexibility and functionality.

4.3 System Development and Setup

The starting point for the implementation was the BFT-SMaRt's implementation for Hyperledger Fabric [8]. This implementation was used and adapted so that each component executes in individual Docker [19] containers. Another adaptation that was implemented was the creation of a script that generates the required certificates for entities, genesis block for the channel where the transactions take place, the channel configuration transaction, and the generation of anchor peers. The channel configuration transaction is a defined transaction that translates the specified network configuration properties, such as, settings for orderers, the control for different organizations involved in the blockchain network (in the related mapping on the channel) and the parameterized conditions to close a block, upon the respective validation.

The configuration of the network, this is the entities that compose the Blockchain network are all specified in YAML files. YAML is a data serialization standard for all programming languages [74]. Figure 4.2 illustrates the organization of the Blockchain network. The Certificate Authority, which is described above will deal with the registration of identities and the issuance, revocation and renewal of certificates, for the correct use of the public keys, relevant for the verification of signatures in the supported transactions.

The peers mainly serve two purposes to maintain a ledger containing the current state of the stored values and to run chaincode operations on this ledger.

As represented in Figure 4.3, BFT-SMaRt is the algorithm executed within the support of the consensus protocol, to establish the total order of the transactions [8] and the consistency protocol of the state-machine replication environment. The support for BFT-SMaRt is composed by an ordering service, which will receive endorsed transactions (from the endorsement peers) and the frontend, to builds and to send the envelopes of transactions to the replicas. These replicas will receive these envelopes and execute the BFT-SMaRt replication protocol to order them [64]. After a certain number of envelopes is gathered, the replica will build a block, and send the block to the frontend, which will then relay them to the distributed peers to be appended to the ledger.

As introduced before, we implemented two chaincodes, one for the definition of sector pricing condition and another one for the rental condition.

The sector chaincode has a structure composed by the object type, the identification of the sector and the rental price per hour. In this chaincode we provide three functions, executable in the verification of transactions: initPrice, readSectorPrice and getSectorsByRange. The function initPrice enables the creation of a sector setting its price and readSectorPrice enables reading the price of a certain sector. The function getSectorsByRange retrieves all the sectors whose IDs are between the two provided values.

The rental chaincode, related to the rental conditions of a locker has a structure that includes: the object type, the transaction identification, the identification of the locker, the sector in which the locker is located and the duration and price of the rental. The provided

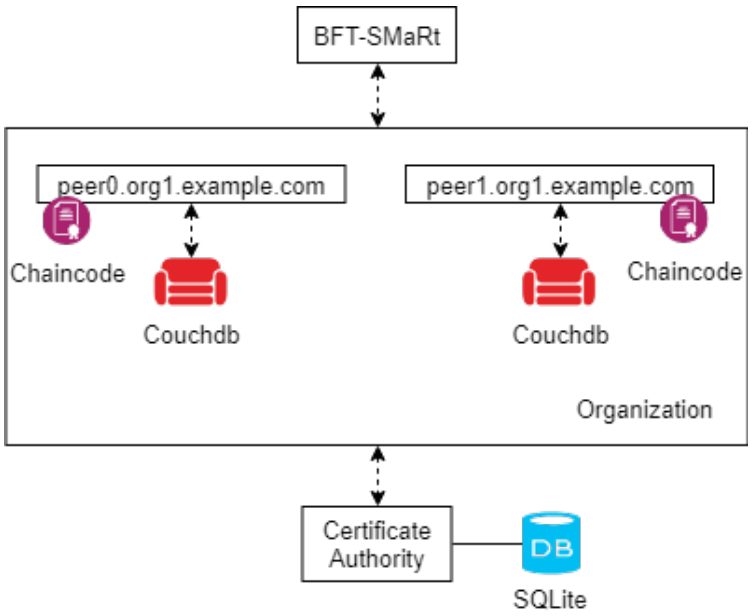

Figure 4.2: Representation of the Blockchain network.

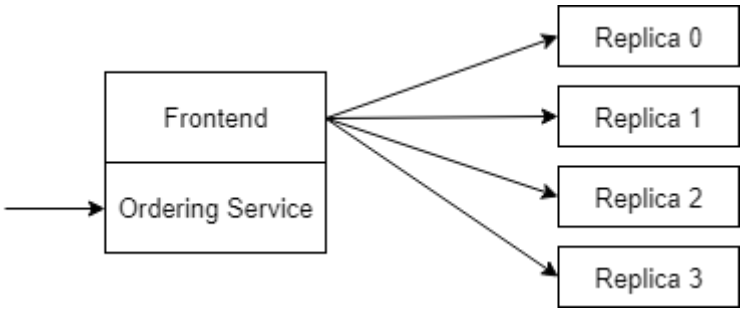

Figure 4.3: Representation of BFT-SMaRt.

functions are similar in functionality to the ones provided in the sector chaincode which are: initRental and getRentalsByRange. The object is stored in JSON format, with the respective identification.

In order to implement chaincodes, encryption functions are used These functions are provided by cryptographic primitives in the EncCC library [27]. The library allows for data encryption to be stored as protected data. The JSON object in chaincodes are encrypted with AES 256. The key for the encryption is provided by the Java application, which generates the key and initialization vectors (for CBC encryption mode), during the system initialization, and sends it to the chaincode. This key generation and establishment process is used as a bootstrap sequence, and the security issues usually related to the problem of key-generation and secure distribution, according to more complex and secure components and protocols, are out of the scope of our dissertation. As a matter of fact, we use the base key-generation and distribution mechanism, as provided by the HLF implementation.

Another relevant implementation issue is the definition of endorsement policies,

which include the rules used by peers to decide if a transaction is correctly endorsed. We must notice that, since there is only one user in the system, this user has to endorse the transaction. These policies are defined in a YAML file which states how many or which peers should endorse a transaction in order for it to be considered as valid.

The implemented application as well as all the necessary documentation are available in https://bitbucket.org/jpgalmeida/bep-iot-fctunl/.

4.4 Other Implementation Issues and Final Remarks

There are some other issues related to the implementation details, namely related to the way the Hyperledger Fabric mechanisms are provided and used.

For the assessment purposes described in the Chapter 5, the deployment of such implementation details also involve consequences that can affect the evaluation mechanisms, as they were materialized for the purpose of the current dissertation. In Chapter 5 we will describe the setups and implementation support for the conducted experimental evaluations, but some implementation details at the Blockchain level, are not able to be tested with other extended variable setups.

One relevant issue is related to the CA support and the process of issuing certificates. At the same time, in HLF certificates are limited to the usage of the ECDSA signature algorithm, not allowing the usage of other public-key algorithms and keys in different representation variants or sizes. ECDSA is a good solution, as a lightweight and secure digital signature standard, being the better option to avoid processing penalties, in the execution and validation of transactions.

However, there are other concerns related to the use of other digital signature algorithms, naming and binding services for the identification of peers externally (regarded as external entities to the blockchain) and internal identifiers (mapped in public-keys), as well as, the advantages of de-conflating membership management functions from the internal consensus-algorithms in the HLF consensus plane. This separation of concerns is a way for a better management of entities and roles, in the context of IoT applications promoted by IoT consortia, involving different partners with different responsibilities.

Another implementation issue is related to the encryption library that limits encryption to AES 256, not allowing that neither the key size nor the encryption mode or secure padding mechanisms to be easily changed. These issues are out of the scope of the development issues in the present dissertation, and in some sense, are out of the primary purposes of the validation and assessment of the proposed platform.

Finally, we must notice that REST APIs implemented for the Cluster Head and Smart Hub were initially tested with Swagger UI [69]. Swagger UI enables the creation of HTTP (or HTTPS) requests, with the insertion of the associated testing arguments and instrumentation parameters, not limiting the use of the REST interface to a mobile application setting.

Chapter 5

Evaluation

In this chapter, we present the experimental validation of the prototype implementation described in Chapter 4. The presented observations and results are centered in two types of benchmarks for the extraction of quantitative and qualitative metrics on the system operation and performance. First, we conducted benchmarks for the evaluation of the Blockchain based services. In this evaluation we include the evaluation impact on throughput and latency conditions, having a failure free environment, a fail-stop behaviour and a byzantine-failure setting, when using the byzantine fault tolerance environment promoted by the BFT-SMaRt ordering service. This evaluation give us the indications on the comparative settings between the native Kafka consensus plane solution (as natively supported by the HLF implementation) and the enhanced consensus layer, a comparison achieved for failure-free and fail-stop behaviors. Next, we conducted a second state of benchmark observations, at the client-side level, with observation results on the client-side invocation performance through the Smart Hub intermediation, also observing the impact on having concurrent clients performing operations causing the consequent workloads, dispatched by the Smart Hub to the Blockchain enabled services.

5.1 Testing environment

We considered three testing environments:

1. The first testing environment is a dedicated server in a Cloud hosted by the OVH Cloud computing and storage provider, using a dedicated server instance to run the Blockchain services. The hosting server has a CPU Intel Xeon D-1520 – 4c/8t - 2.2GHz /2.7GHz, 128GB DDR4 of RAM and 2x2TB in SoftRaid Disks, available for support via SATA or SSD. In our tests we used the SATA based SoftRaid solution. The sever runs Linux Debian 9.0, 64 bit Stretch version (considered as the OVH

Debian Stable Version). This server is used in order to test different blockchain network configurations, emulating a higher number of entities in the network.

2. The second environment is a Raspberry Pi 3b+ running the Linux Raspbian distribution (from June 2018) in order to test Smart Hub capabilities, as a specific Smart Hub appliance solution.

3. The last environment is a personal computer running Ubuntu 17.10 on a VM with an Intel Core i7 using 8 processors and 8Gb of memory in order to be able to compare benchmarks with the Raspberry Pi, when running the Smart Hub provided services. Finally, we used the last computer configuration to run Client-Side tests.

5.2 Evaluation Criteria

In this chapter it is described the evaluation for this dissertation. There are three main groups of tests: Network benchmarks, Failure benchmarks and Client-side benchmarks.

Network benchmarks, which refer to the tests that were made in order to find the optimal network configuration. In these tests the main objective was to find the best number of transactions per block and maximum block size in bytes that would maximize the throughput and minimize the latency in the baseline implementation which uses Hyperledger Fabric out-of-the-box with Kafka as an ordering service. It is also benchmarked the implementation which uses BFT-SMaRt as an ordering service with the optimal configuration to test the impact of tolerating byzantine faults. The values of throughput and latency were extracted from proposing two types of transactions, a write and a read operations. The variations done in this test are related to the way blocks are constructed in Hyperledger Fabric. A block is constructed when one of the following three conditions is met. First, a pre-specified number of transactions is achieved. Second, the size of the transactions is bigger than a pre-specified value. Third, there is a timeout and a block is built.

Failures benchmarks consist in the evaluation of the impact when failures occur. In this study it is varied the number of brokers in case of the baseline implementation and the number of replicas in the case of BFT-SMaRt. It will be tested a failure-free environment as well as crash and byzantine faults through replica failure simulation and the injection of wrong values, respectively.

Client-side benchmarks relate to the tests made when the Raspberry Pi was used to deploy the Smart hub and whether there is an impact in deploying the Smart Hub in a conventional personal computer configuration, with typical resources. The network benchmarks were completed using Hyperledger Caliper [11]. Hyperledger Caliper is a framework that allows to test different HLF blockchain configurations to predefined use cases, allowing the support of observable benchmarks, including the throughput of transactions under scalability conditions, and also latency observations, as we present next.

5.3 Network benchmarks

In this section, it is compared the performance results of the baseline implementation of Hyperledger Fabric which uses Kafka as an ordering service with the proposed system which relies on BFT-SMaRt.

The goal of this section is to find the optimal number of transactions per block and the size of the block in bytes. Initially, it was tested the throughput and latency of the out-of-the-box Hyperledger Fabric implementation which corresponds to 10 transactions per block and 512 KB of maximum block size. The results from this experiment are represented in Figure 5.1 as the entry for 0,5 MB. Next, as an experiment to test whether these parameters would increase the throughput it were tested block sizes of 0.5, 1, 2, 3 and 4 MB. As Figure 5.1 suggests, setting the block size at 2 MB allows better throughput and lower latency values and having a block size bigger than 3 MB will decrease the throughput and higher the latency.

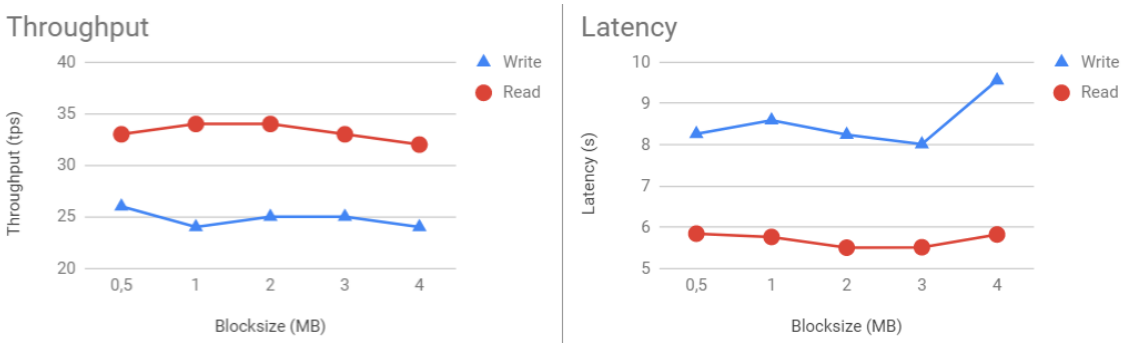

Figure 5.1: Variation of block size throughput and latency results.

This way, for the next tests the value of the block size was fixed at 2 MB and it were tested the maximum number of transactions per block. The results of this experiment is represented in Figure 5.2. The throughput is higher when there are 100 transactions per block.

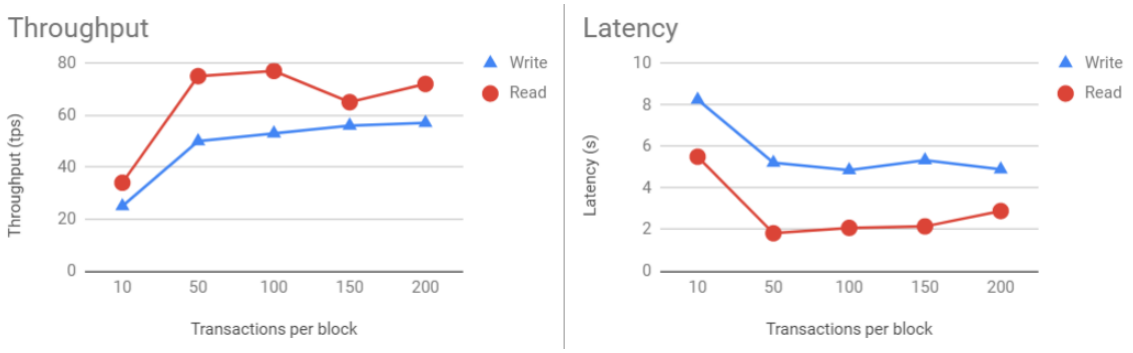

Figure 5.2: Variation of maximum number of transactions throughput and latency results.

According this value and the block size at 2MB were used for further experiments.

This way, it were also tested different block sizes varying its size and the number of transactions contained in each block in order to find the optimal throughput. The size that was used for the following tests is 50 transactions per block and a maximum size of 512 KB.

Following the previously done tests, it were tested the baseline and BFT-SMaRt implementation, with the optimal values obtained. It were performed tests to networks composed by 2, 4, 8, 10, 20 and 30 peers, in order to test the impact of increasing the network size would have in throughput and latency. As expected, the throughput when using the baseline implementation is higher when compared to BFT-SMaRt by about 20% and has a latency lower by about 13%.

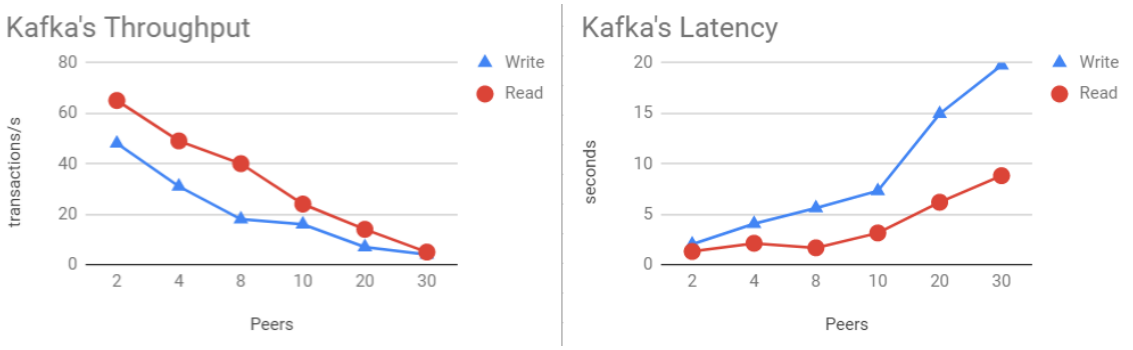

Figure 5.3: Kafka's throughput and latency.

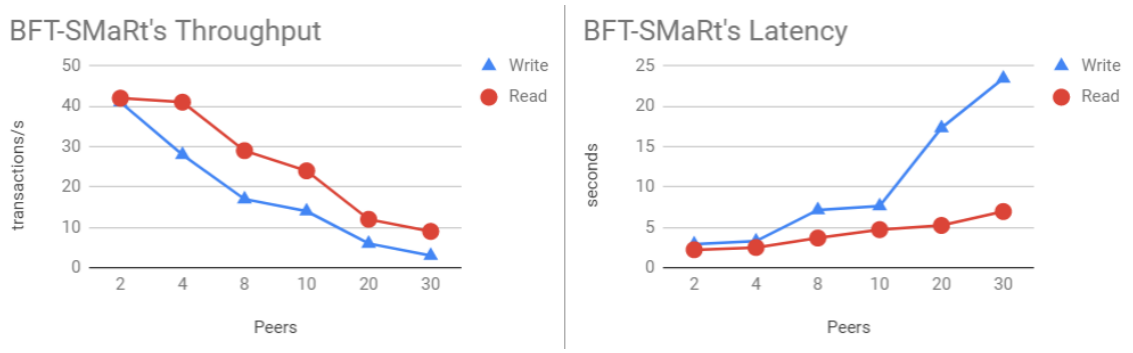

Figure 5.4: BFT-SMaRt's throughput and latency.

Our evaluation results follow the expectable trend, and we can say that the results are aligned with previous experiments showing that the BFT-SMaRt algorithm is not only the specific component in the aggravation of the throughput, as we evaluated with the HLF Caliper benchmark factory.

In [64], running a more "empty"environment adapted to HLF, by reducing the overhead of the blockchain nodes (to just one orderer and one peer in each node), and using a set of four distributed nodes in a Gigabit LAN environment to run the ordering service in a cluster of 4, 7 and 10 orderers, we can expect to have respectively throughputs of approximately 50 K to 5 K transactions per second. The more relevant factors affecting the performance for the considered number of orderers is the envelope size and the number

of envelopes used per blockchain block. It is observable in those experiments and results that even though throughput drops when increasing the number of receivers, the impact of the number of receivers is considerably smaller when using larger transactions, and this is expectable due to the envelope sizes. Also, the overhead of the replication protocol is greater than the overhead of transmitting small blocks.

However, we must notice that the previous observations are only somewhat visible in our achieved results and we must consider the differences in the use of the BFT-enabled consensus layer, in our implementation and our benchmarks. First of all, our "blockchain is virtualized"in the OVH cloud, in a unique computing instance. In this case, we have a faster "communication environment"connecting the blockchain peers and the nodes involved in the consensus layer in the related ordering service. On the other hand, the workload of a large number of nodes in one solo machine, independently of the provided resources, is a considerable degradation factor.

Additionally, to understand the validity of our results in Figures 5.1, 5.2, 5.3 and 5.4. We must compare our observations with other benchmark evaluations of HLF, only using the native services, i.e., using the native Kafka consensus plane service. In this study we find that there is a space for internal performance improvements in the HLF service planes. In [71], for example, the authors perform a comprehensive empirical study to characterize the performance of Hyperledger Fabric and identify potential performance bottlenecks, to gain a better understanding of the system and in order to promote possible optimizations. Further, the authors enhanced and measured the effect of bulk read/write optimization in HLF using CouchDB, in a distributed cluster of 32 peer nodes (32 vCPUs) in a computing cluster environment, with nodes connected by 3 Gbps links. They improved the initial observations, limited to 140 transactions per second (a reference metric that we can use to infer a comparison with our observation), to their optimized version, achieving 2250 transactions per second. The same authors also found significant differences when the endorsement process run with different endorsement policies, number of HLF channels, as well as, a significant impact on transaction arrival rates and block sizes (including the number of transactions, and their payload sizes).

5.4 Benchmarks under failure conditions

In this section, it is compared the impact of failures in the base implementation of the blockchained enabled services. For these tests, we varied the number of replicas for supporting an increasing number of replica failures. As illustrated in Figure 5.5, latency is lower than when a less number of brokers are up. This is given to the fact that a less number of follower brokers have to be in-sync with the leader broke, consequentially, causing less internal messages to be exchanged, during the consensus processing functions.

First we show in Figure 5.5 the impact in latency observations, when we use the variable number of brokers, in a failure free and fail stop environment.

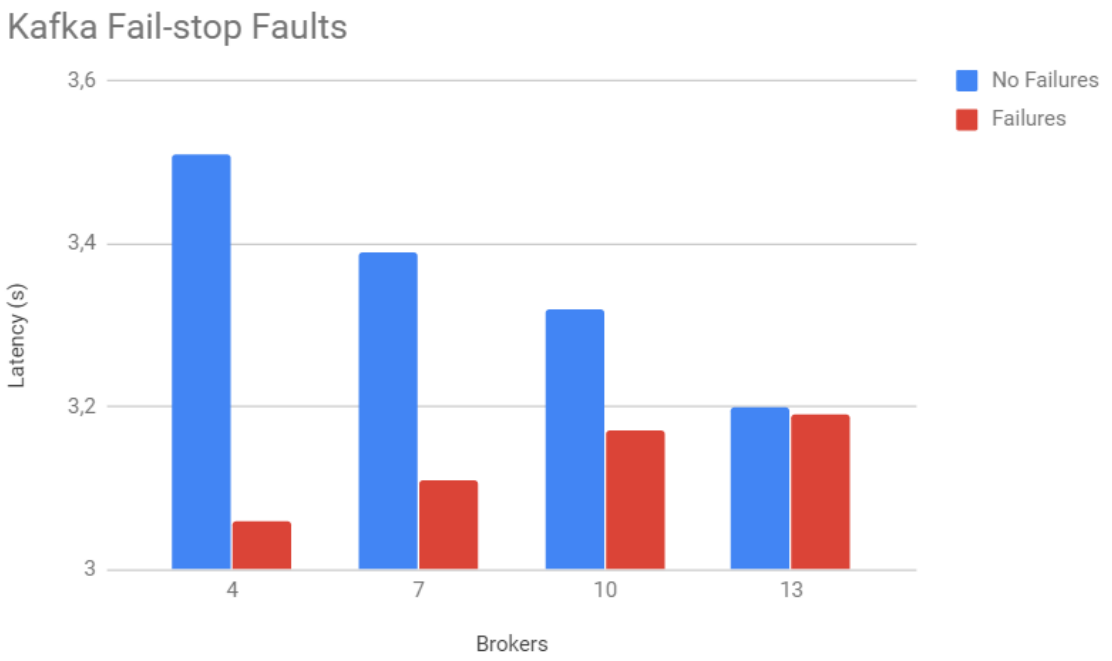

Figure 5.5: Kafka's latency with broker failures.

Figure 5.6 represents the performance impact in BFT-SMaRt when simulating a fail-stop behavior. Performance is widely affected in fail-stop simulations in BFT-SMaRt, given its synchronization phase. This phase takes 20 seconds to complete and due to this, during this phase, the throughput is strongly affected, because no transactions and blocks can be completed [7].

Figure 5.7 represents the latency when increasing the number of replicas in the ordering service provided by BFTSMaRt. In this tests, it were tested the impact in performance when we have a byzantine failure environment, following the fault model of using n = 3f + 1, n being the number of replicas and f the number of faulty replicas. It were created byzantine replicas that would inject wrong values in the consensus process. In this test it is noticeable that there is not a significant difference between the latency in fault-free and byzantine faults. This is explained by the fact that BFT-SMaRt's consensus protocol requires $\frac{n+f+1}{2}$ write and accept message and since the faults do not surpass this value, the malicious nodes are unable to disrupt the service.

5.5 Client-side benchmarks

In this section, it is shown the results measured at the client side. It was developed a Java application that simulates a normal behavior of the system by using one or multiple clients. The simulated tests include the definition of a sector price, the reading of the previously inserted value and the insertion of a rental.

The main objective of this test is to determine the latency of such operations when

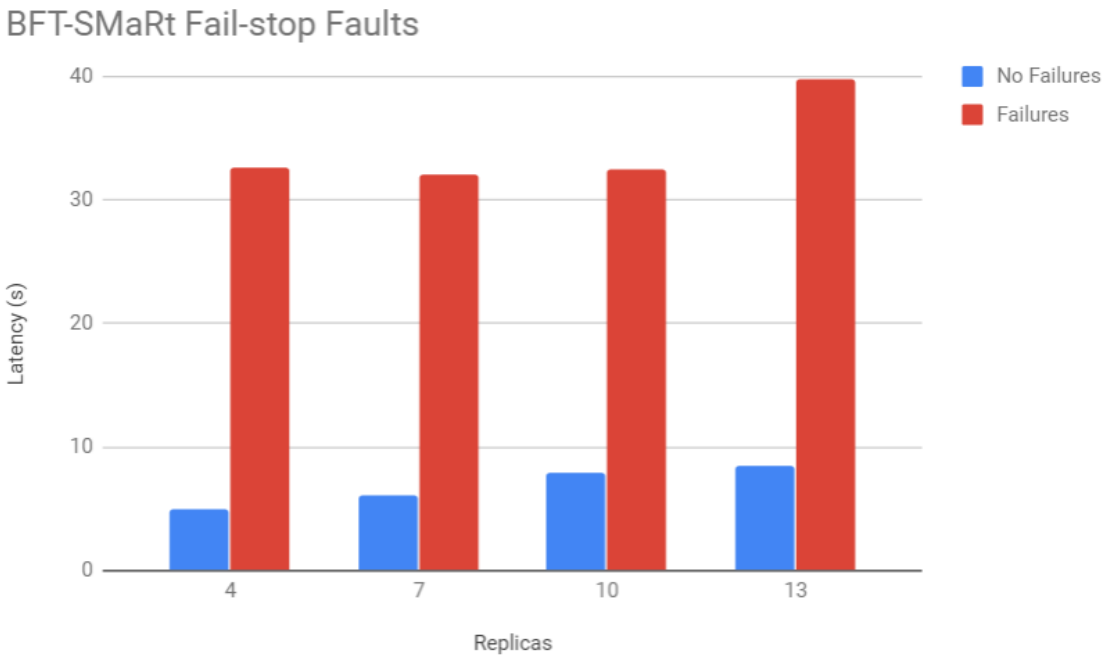

Figure 5.6: BFT-SMaRt's latency with fail-stop failures.

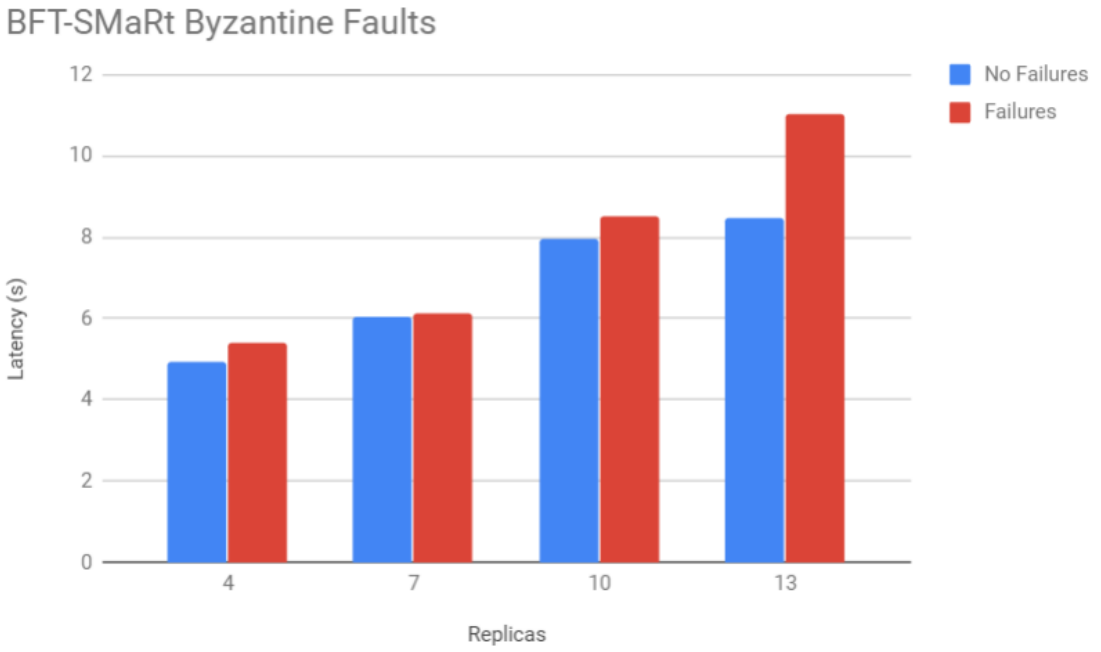

Figure 5.7: BFT-SMaRt's latency with replica failures.

increasing the number of clients, according to the dispatching services at the level of a Smart Hub. As Figure 5.8 suggests there is a substantial increase in the communication latency when using less than 5 clients. After that, the latency does not change very much. We explain the fact because transactions are being constantly received and blocks are built faster. Our results also show that a Raspberry Pi 3b+ is able to handle 12 simultaneous clients, with acceptable performance, and that there is a slight increase in latency when the results obtained in the Raspberry Pi (first setup explained in Chapter 4) are compared to an emulated Smart Hub in a conventional portable computer (second setup, as explained in Chapter 4).

We must clarify that these results are obtained in supporting synchronous remote REST invocations in the Smart Hub, with call results obtained after the entire cycle of blocks' and transactions' processing in the backend blockchain. In an optimistic implementation, we can address an asynchronous interaction model in the permissioned model, in which we can return a promising result and then execute the transactions, taking the necessary measures to deal with the required implications in such asynchronous setting.

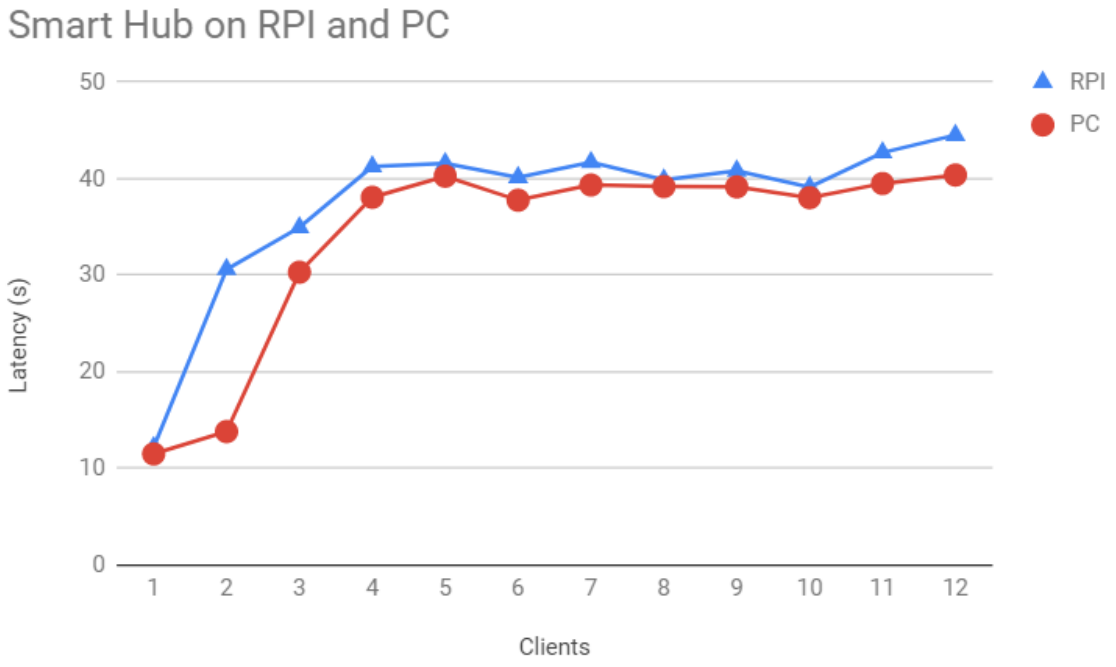

Figure 5.8: Comparison of Smart Hub's latency on a RPI and a PC.

5.6 Evaluation summary

In this chapter it was demonstrated the operation and we discuss some validity observations, in three types of evaluation benchmarks. First, the evaluation of the Blockchain network conditions, analyzing the impact of the provided services in the implementation planes. In this study we observed that by varying the conditions for closing blocks

or using the different consensus plane services, the performance of the system can be substantially improved or affected. Second, we evaluated the performance of the implemented solution, in behaviors under different failure models: failure-free, fault-stop and byzantine faults. These observations show the impact of tolerating crash and byzantine failures in the performance conditions. Finally, we observed the latency conditions in client-side operations, when Smart Hubs are deployed with two different setups: a Raspberry Pi used as the Smart Hub appliance and an emulated Smart Hub running in a conventional laptop computer. We also evaluated the client-side performance with multiple concurrent client executions of the developed application, used as a reference to measure the impact in final client-supported applications.

Chapter 6

Conclusions

In this chapter, it is described the main findings from this dissertation, as well as the existing open issues and introducing some future work directions.

6.1 Main Conclusions

Information is currently the most valuable resource and privacy concerns have been rising among people. People are becoming more aware of dangers and are looking for solutions that enhance their security and privacy. The Blockchain and the Internet of Things, have been one of the focus of many recent studies in the Distributed Systems field. These technologies are the answer to some of these demands since both technologies are able to provide solutions in terms of a more connected, independently verified and trustable world. We believe that Blockchain enabled IoT platforms are able to provide advantages in a clean-slate approach for new generation IoT architectures.

The main problem to be solved is the improve of verifiable trustability in smart devices and at the same time minimize the role played by central authorities in the verification of data and transactions consistency. The use of blockchain's logging capabilities is a possible solution for these problems. However, one of the main issues with current large scale applications that use Blockchain technology is scalability. The number of transactions that are processed, when the number of nodes in the system increases, are low and incomparable to systems that do not use blockchain technology. This way, dealing with scalability issues is a main priority for this dissertation. And, in this matter, it were studied several approaches that try to minimize the impact in throughput.

The main goal of this dissertation was the design, implementation and evaluation of a Blockchain-Enabled IoT platform proposal, as a reference architecture and guidelines, in providing, reliability effectiveness and trustability foundations based on a decentralized

ledger model, that must fit on the scalability criteria for IoT applications and services.

In the Related Work we survey the foundations and design principles of the Blockchain, as well the characteristics, properties and drawbacks related to scalability issues. In the study we analyzed relevant differences in approaching the base services in different blockchains, and we present different Blockchain platforms, considering their different design model assumptions and different condiments associated to their components, services and related operation. The study oriented the choice of the blockchain-enabled services to leverage the objectives of the thesis. Although the chosen platform (Hyperledger Fabric) to be considered as the base for our proposal, does not officially provide it, it is possible to integrate ordering services implemented for Byzantine Fault Tolerant consensus protocol, as a pluggable component, taking the advantage of the modular architecture. For this purpose, it was used the implementation of an ordering service implemented by the BFT-SMaRt algorithm, a state machine replication protocol for byzantine fault tolerance distributed consensus. We also studied some approaches in the research direction of designing Blockchain-supported IoT applications.

From the studied related work, we proposed a System Model and Architecture for Blockchain-enabled IoT Platforms. Architectural features of Blockchain drawbacks and IoT limitations in current IoT platforms were considered, to overcome such limitations by the added-value of merging Blockchain-enabled services with recent IoT systems using Smart Hubs as gateways offering proximity services to edge-based IoT environments.

The system model and the proposed architecture are mapped in a developed prototyped, providing an implementation environment used to conduct the validation and experimental evaluation. The results from the experimental observations show the viability and validity of our ideas, under the careful choice of the blockchain-enabled services, in terms of provided extensibility, openness, and base foundations, as well as, in allowing for the use of the permissioned model in the decentralized ledger functions and arguments. Moreover, the services developed and enabled in the developed prototype, allow us to observe relevant implications of fine-tuning of parameterizations in the blockchain enabled services in the performance of transactions and scale conditions, namely, throughput and latency conditions, under different failure-settings.

Even considering the possible limitations of the current state of the prototype and other relevant dimensions not achieved in our proposal, we found some interesting answers in addressing the problem statement initially defined for the dissertation: to improve the reliability and trustability guarantees of IoT platforms and software architectures, by introducing Blockchain-enabled mechanisms for decentralized logging and ledgering, auditable information flow control of operations involving IoT applications, as well as, to provide an independent environment in supporting data sent or received by user's smartphones and IoT devices in such applications, under decentralized ledger models and better trustability properties.

6.2 Future work directions

Although the provided solution shows interesting results, there are a few matter which could be improved. Namely, the evaluation environment could have a big impact in the performance of the system. Testing the Hyperledger Fabric's network in a more resourceful and distributed environment, distributing the different blockchain nodes in different physical nodes, could bring performance improvements and better scalability conditions.

One issue that was not addressed in this dissertation, given time limitations, was the integration of the CoAP - Constrained Application Protocol [14], as a lightweight Application protocol for web-based remote invocations to constrained devices, such as a considerable number of IoT devices. This protocol would be used to complement the Smart Hubs's developed REST interface, in order, to create an application that would not only consume less resources, but also be reliable in lossy networks. Other research directions in the same line, is the support of other interoperability protocols that are designed for implementation in IoT devices. This effort must be seen as a natural evolution and extension of Smart Hub modules, in convergence with the vision expressed in the discussion of the System Model in the background of our proposed architecture.

Another issue that was not dealt with in this dissertation, is the comparison between the implementation of BFT-SMaRt and the officially supported BFT algorithm that will be launched for HLF in the near future, in more extensive benchmarking environments. Finally, there is another research direction in other aspects with relevant impact for scalability purposes. We summarize some of these research directions in two different groups of concerns: scale-in and scale-out concerns.

- For scale-in concerns we must observe the on-going research proposals in addressing better performance figures for HLF (and other permissioned blockchains in the ongoing research agenda), as well in the approach of separation concerns decoupling membership services, consensus planes for consistency control and decentralized hierarchies of blockchains, for example in a tree-based architectural model or by using sharding models interconnecting different blockchain domains;

- For scale-out purposes, we can extend the current system model in the dissertation by interconnecting Smart-Hubs in edged-based blockchains, possibly composed in upper-level hierarchies.

Finally, there is a space to research on better expressiveness conditions of smart-contracts to support IoT operations enabled by Blockchain transactions. Smart contracts can be used to define parameters, rules and invariants for different levels of execution requirements. These requirements range from specific IoT application-level validation guarantees, to parameters and conditions regulating the internal services of Blockchain

planes (including storage, aggregation of transactions and management of stored blocks), in a more reconfigurable and possibly dynamic reconfigurable environment.

Bibliography

[1] *Amazon Echo*. Sept. 2018. URL: https://www.amazon.com/Amazon-Echo-Bluetooth-Speaker-with-WiFi-Alexa/dp/B00X4WHP5E.

[2] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. D. Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich, S. Muralidharan, C. Murthy, B. Nguyen, M. Sethi, G. Singh, K. Smith, A. Sorniotti, C. Stathakopoulou, M. Vukolic, S. W. Cocco, and J. Yellick. "Hyperledger Fabric: Distributed Operating System for Permissioned Blockchains." In: *EuroSys '18 Proceedings of the Thirteenth EuroSys Conference* abs/1801.10228 (2018).

[3] A. M. Antonopoulos. *Mastering Bitcoin: Unlocking Digital Crypto-Currencies*. 1st. O'Reilly Media, Inc., 2014.

[4] *Apache CouchDB*. Mar. 2018. URL: http://couchdb.apache.org.

[5] *Apple HomeKit*. Sept. 2018. URL: https://developer.apple.com/homekit/.

[6] M. Banerjee, J. Lee, and K.-K. R. Choo. "A blockchain future to Internet of Things security: A position paper." In: (2017).

[7] A. Bessani, J. Sousa, and E. Alchieri. "State machine replication for the masses with BFT-SMART." In: *Proceedings of the 44th IEEE/IFIP International Conference on Dependable Systems and Networks, Atlanta, GA, USA* (2014).

[8] *BFT ordering service for Hyperledger Fabric*. Sept. 2018. URL: https://github.com/jcs47/hyperledger-bftsmart.

[9] A. Bogner, M. Chanson, and A. Meeuw. "A Decentralized Sharing App running s Smart Contract on the Ethereum Blockchain." In: *IoT '16: Proceedings of the 6th International Conference on the Internet of Things* (2017).

[10] V. Buterin. "Ethereum White Paper - A Nest Generation Smart Contract & Decentralized Application Platform." In: (2014).

[11] *Caliper*. Sept. 2018. URL: https://github.com/hyperledger/caliper.

[12] M. Castro and B. Liskov. "Practical Byzantine Fault Tolerance." In: (1999).

[13] *Chaincore - Whitepaper*. Sept. 2018. URL: https://chain.com/docs/1.2/protocol/papers/whitepaper.

[14] *CoAP Technology – RFC 7252 Constrained Application Protocol, references and other related IETF RFCs*. Sept. 2018. URL: http://coap.technology.

[15] *Corda Documentation*. Sept. 2018. URL: https://docs.corda.net/.

[16] *CouchDB as the State Database*. Sept. 2018. URL: https://hyperledger-fabric.readthedocs.io/en/release-1.1/couchdb_as_state_database.html.

[17] K. Croman, C. Decker, I. Eyal, A. E. Gencer, A. Juels, A. Kosba, A. Miller, P. Saxena, E. Shi, E. Gun Sirer, D. Song, and R. Wattenhofer. "On Scaling Decentralized Blockchains." In: *Financial Cryptography and Data Security: FC 2016 International Workshops, BITCOIN, VOTING, and WAHC, Christ Church, Barbados* (2016), pp. 106–125.

[18] *CryptoNote*. Sept. 2018. URL: https://cryptonote.org/whitepaper.pdf.

[19] *Docker*. Sept. 2018. URL: https://www.docker.com/.

[20] A. Dorri, S. S. Kanhere, R. Jurdak, and P. Gauravaram. "Blockchain for IoT security and privacy: The case study of a smart home." In: (2017), pp. 618–623.

[21] A. Dorri, S. S. Kanhere, and R. Jurdak. "Towards an Optimized BlockChain for IoT." In: (2017), pp. 173–178.

[22] S. Duan, H. Meling, S. Peisert, and H. Zhang. "BChain: Byzantine Replication with High Throughput and Embedded Reconfiguration." In: (2014).

[23] M. Duggan. *P. I. R. Privacy and Information Sharing, Technical Report*. Jan. 2016. URL: http://www.pewinternet.org/2016/01/14/privacy-and-information-sharing/.

[24] C. Dwork and M. Naor. *Pricing via Processing or Combatting Junk Mail*. Ed. by E. F. Brickell. Springer Berlin Heidelberg, 1993, pp. 139–147.

[25] K.-K. E, P. Javanovic, L. Gasser, N. Gailly, E. Syta, and B. Ford. "Enhancing Bitcoin Security and Performance with Strong Consistency via Collective Signing." In: *USENIX Security Symposium 2016* (2016).

[26] K.-K. E, P. Javanovic, L. Gasser, N. Gailly, E. Syta, and B. Ford. "OmniLedger: A Secure, Scale-Out, Decentralized Ledger via Sharding." In: *39th IEEE Sympsium on Security and Privacy* (2018).

[27] *EncCC*. Sept. 2018. URL: https://github.com/hyperledger/fabric/tree/master/examples/chaincode/go/enccc_example.

[28] A. Gervais, G. O. Karame, K. Wust, V. Glykantzis, H. Ritzdorf, and S. Capkun. "On the Security and Performance of Proof of Work Blockchains." In: CCS '16 (2016), pp. 3–16.

[29] D. Gillmor. *As we sweat government surveillance, companies like Google collect our data*. Apr. 2014. URL: https://www.theguardian.com/commentisfree/2014/apr/18/corporations-google-should-not-sell-customer-data.

[30] M. Herlihy. "Blockchains and the Future of Distributed Computing." In: *PODC '17: Proceedings of the ACM Symposium on Principles of Distributed Computing* (2017).

[31] P. N. Howard. *How big is the Internet of Things and how big will it get?* Sept. 2018. URL: https://www.brookings.edu/blog/techtank/2015/06/08/how-big-is-the-internet-of-things-and-how-big-will-it-get/.

[32] *Hydrachain - Github*. Sept. 2018. URL: https://github.com/HydraChain/hydrachain.

[33] Hyperledger. *Hyperledger Architecture, Volume 1*. 1st. 2017.

[34] *Hyperledger*. Sept. 2018. URL: https://www.hyperledger.org/.

[35] *Hyperledger Fabric ACL*. Sept. 2018. URL: https://hyperledger-fabric.readthedocs.io/en/release-1.2/access_control.html.

[36] *Hyperledger Fabric Blockchain*. Sept. 2018. URL: https://hyperledger-fabric.readthedocs.io/en/release-1.2/blockchain.html#what-is-hyperledger-fabric.

[37] *Hyperledger Fabric Chaincode*. Sept. 2018. URL: http://hyperledger-fabric.readthedocs.io/en/release-1.1/chaincode.html.

[38] *Hyperledger Fabric CID*. Sept. 2018. URL: https://github.com/hyperledger/fabric/blob/master/core/chaincode/lib/cid/.

[39] *Hyperledger Fabric Ledger*. Sept. 2018. URL: https://hyperledger-fabric.readthedocs.io/en/release-1.2/ledger/ledger.html.

[40] *IBM - Blockchain basics: Hyperledger Fabric and Hyperledger Compose*. Mar. 2018. URL: https://developer.ibm.com/articles/cl-blockchain-hyperledger-fabric-hyperledger-composer-compared.

[41] *IFTTT*. Sept. 2018. URL: https://ifttt.com.

[42] *Internet of things*. Sept. 2018. URL: https://en.wikipedia.org/wiki/Internet_of_things#Enabling_technologies_for_IoT.

[43] *Ionic Framework*. Sept. 2018. URL: https://ionicframework.com/.

[44] *IoT Standards and Protocols*. Sept. 2018. URL: https://www.postscapes.com/internet-of-things-protocols/.

[45] *Kafka*. Sept. 2018. URL: https://kafka.apache.org/.

[46] S. King and S. Nadal. "PPCoin: Peer-to-Peer Crypto-Currency with Proof-of-Stake." In: (2012).

[47] N. Kshetri. "Can Blockchain Strengthen the Internet of Things?" In: 19.4 (2017), pp. 68–72.

[48] J. Kwon. "Tendermint: Consensus without Mining." In: (2014).

[49] *LevelDB, A light-weight, single-purpose library for persistence with bindings to many platforms*. Mar. 2018. URL: http://leveldb.org.

[50] W. Li, A. Sforzin, S. Fedorov, and G. Karame. "Towards Scalable and Private Industrial Blockchains." In: *BCC '17: Proceedings of the ACM Workshop on Blockchain, Cryptocurrencies and Contracts* (2017).

[51] I.-C. Lin and T.-C. Liao. "A Survey of Blockchain Security Issues and Challenges." In: *I. J. Network Security* 19 (2017), pp. 653–659.

[52] *Monax*. Sept. 2018. URL: https://monax.io/.

[53] *MongoDB*. Mar. 2018. URL: https://www.mongodb.com/.

[54] S. Nakamoto. "Bitcoin: A peer-to-peer electronic cash system." In: (2008).

[55] R. D. Petro, X. Salleras, M. Signorini, and E. Waisbard. "A Blockchain-based Trust System for the Internet-of-Things." In: *SACMAT '18: Proceedings of the 23nd ACM on Symposium on Access Control Models and Technologies* (2017).

[56] *Quorum - Whitepaper*. Jan. 2018. URL: https://github.com/jpmorganchase/quorum-docs/blob/master/Quorum%20Whitepaper%20v0.1.pdf.

[57] *Raspberry Pi*. Sept. 2018. URL: https://www.raspberrypi.org/.

[58] M. Salek-Ali, K. Dolui, and F. Antonelli. "IoT Data Privacy via Blockchains and IPFS." In: *IoT '17, Proceedings of the Seventh International Conference on the Internet of Things* (2017).

[59] M. Samaniego and R. Deters. *Blockchain as a Service for IoT*. 2016, pp. 433–436.

[60] *Samsung SmartThings*. Sept. 2018. URL: https://www.smartthings.com.

[61] L. S. Sankar, S. M., and M. Sethumadhavan. "Survey of Consensus Protocols on Blockchain Applications." In: (2017).

[62] H. Shafagh, L. Burkhalter, A. Hithnawi, and S. Duquennoy. "Towards Blockchain-based Auditable Storage and Sharing of IoT data." In: *CCSW '17: Proceedings of the 2017 on Cloud Computing Security Workshop* (2017).

[63] J. Sousa and A. Bessani. "Separating the WHEAT from the chaff: An empirical design for geo-replicated state machines." In: *Proceedings of the IEEE 34th Symposium on Reliable Distributed Systems, Montreal, Quebec, Canada* (2015).

[64] J. Sousa, A. Bessani, and M. Vukolic. "A Byzantine Fault-Tolerant Ordering Service for the Hyperledger Fabric Blockchain Platform." In: *SERIAL '17 Proceedings of the 1st Workshop on Scalable and Resilient Infrastructures for Distributed Ledgers* (2017).

[65] *Spring Boot*. Sept. 2018. URL: https://spring.io/projects/spring-boot.

[66] *SQLite*. Sept. 2018. URL: https://www.sqlite.org/.

[67] Y. L. Sun, Z. Han, W. Yu, and K. J. R. Liu. "A trust evaluation framework in distributed networks: Vulnerability analysis and defense against attacks." In: *Proceedings - IEEE INFOCOM* (2006).

[68] D. Svetinovic. "Blockchain Engineering for the Internet of Things: Systems Security Perpective." In: *IoTPTS '17: Proceedings of the 3rd ACM International Workshop on IoT Privacy, Trust, and Security* (2017).

[69] *Swagger*. Sept. 2018. URL: https://swagger.io/.

[70] N. Szabo. "Smart Contracts: Building Blocks for Digital Markets." In: (1996).

[71] P. Thakkar, S. Nathan, and B. Viswanatham. "Performance Benchmarking and Optimizing Hyperledger Fabric Blockchain Platform." In: *in https://arxiv.org/pdf/1805.11390.pdf* (May 2018).

[72] *The world's most valuable resource is no longer oil, but data*. May 2017. URL: https://www.economist.com/news/leaders/21721656-data-economy-demands-new-approach-antitrust-rules-worlds-most-valuable-resource.

[73] *There are officially more mobile devices than people in the world*. Sept. 2018. URL: http://www.independent.co.uk/life-style/gadgets-and-tech/news/there-are-officially-more-mobile-devices-than-people-in-the-world-9780518.html.

[74] *YAML*. Sept. 2018. URL: http://yaml.org.

[75] Z Zheng, S Xie, H Dai, X Chen, and H Wang. "An Overview of Blockchain Technology: Architecture, Consensus, and Future Trends." In: *2017 IEEE International Congress on Big Data (BigData Congress)* (2017), pp. 557–564.