Universidade de Aveiro 2016

Departamento de Eletrónica,
Telecomunicações e Informática

Vasco Alexandre Maia dos Santos

Infraestrutura Segura e Descentralizada para a Internet das Coisas

Secure Decentralized Internet of Things Infrastructure

Universidade de Aveiro Departamento de Eletrónica, Telecomunicações e Informática
2016

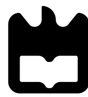
Vasco Alexandre Maia dos Santos

Infraestrutura Segura e Descentralizada para a Internet das Coisas

Secure Decentralized Internet of Things Infrastructure

"*The walls between art and engineering exist only in our minds*"

— Theo Jansen



Universidade de Aveiro
2016

Departamento de Eletrónica,
Telecomunicações e Informática

Vasco Alexandre Maia dos Santos

Infraestrutura Segura e Descentralizada para a Internet das Coisas

Secure Decentralized Internet of Things Infrastructure

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia de Computadores e Telemática, realizada sob a orientação científica do Doutor Diogo Nuno Pereira Gomes, Professor auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro, e do Doutor João Paulo Silva Barraca, Professor auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro.

o júri / the jury

presidente / president	Prof. Doutor André Ventura da Cruz Marnoto Zúquete professor auxiliar da Universidade de Aveiro
vogais / examiners committee	Prof. Doutora Ana Cristina Costa Aguiar professora auxiliar convidada da Faculdade de Engenharia da Universidade do Porto
	Prof. Doutor Diogo Nuno Pereira Gomes professor auxiliar da Universidade de Aveiro (orientador)

agradecimentos / acknowledgements

Gostava de agradecer, em primeiro lugar ao Professor Doutor Diogo Gomes e ao Professor Doutor João Paulo Barraca pela oportunidade de integrar o ATNOG e trabalhar nesta dissertação, bem como por todo o apoio dado ao longo do mestrado.

Queria também deixar um grande agradecimento para os meus colegas de casa Joel Pinheiro, José Sequeira, Miguel Vicente e Rui Monteiro pelas discussões e momentos de convívio durante os últimos cinco anos.

Aproveito ainda para agradecer aos meus amigos André Alves, André Jerónimo, Andreia Jacinto, Daniela Tomás, David Fontes, Henrique Cruz, João Abel, João Leitão, José Mendes, José Rosa, Miguel Campos, Miguel Valério, Rafael Saraiva e Ricardo Martins pelo apoio e amizade.

Por fim, um agradecimento especial aos meus pais, António Santos e Célia Santos, bem como à minha irmã Diana Santos, por toda o apoio e motivação que me deram ao longo do meu percurso académico. A todos dedico esta dissertação.

Palavras Chave

Internet das coisas, Infraestruturas descentralizadas, DHT, P2P, Privacidade, Segurança.

Resumo

Apesar de várias infraestruturas para IoT terem sido implementadas nos últimos anos, nenhuma delas está realmente preparada para ser utilizada a uma escala global, onde a escalabilidade e a tolerância a falhas são um requisito essencial. Esta dissertação apresenta um conceito alternativo de infraestruturas para a IoT, cujo foco consiste em evoluir a tradicional arquitetura centralizada, normalmente utilizada por uma entidade única, para uma arquitetura descentralizada, compatível com múltiplas entidades e casos de uso. Propomos uma infraestrutura auto-configurável, dinâmica e orientada à comunidade, construída em cima de uma rede *Peer-to-Peer* estruturada, baseada numa *Distributed Hash Table*. Além disso, a infraestrutura proposta é suficientemente flexível para permitir que cada *peer* da rede possa ativar diferentes serviços, mediante as suas capacidades. Por outro lado, um conjunto de protocolos de comunicação é providenciado, de modo a suportar dispositivos heterogéneos, bem como o acesso aos dados e a sua transmissão e persistência. Também é um foco importante desta proposta a disponibilização de mecanismos que garantam a privacidade e segurança da informação durante a sua transmissão e o seu armazenamento.

Keywords Internet of Things, Decentralized Infrastructures, DHT, P2P, Privacy, Security.

Abstract Despite many IoT Infrastructures having been implemented in recent years, none of them is truly prepared for a global deployment, where failure tolerance and scalability are an essential requirement. This dissertation presents an alternative concept for IoT Infrastructures, which focuses on enhancing the traditional centralized architecture, usually operated by a single entity, into a decentralized architecture featuring multiple entities and use cases. We propose a dynamic, community driven and self-configurable infrastructure on top of a structured Peer-to-Peer network, based on a Distributed Hash Table. Moreover, the proposed infrastructure is flexible enough to allow each peer of the overlay to enable a set of different services, according to its capabilities. In addition, a set of communication protocols is provided in order to support heterogeneous devices, as well as data access, streaming and persistence. It is also an important focus of our proposal to have mechanisms that guarantee the privacy and security of the information flow and storage.

Contents

Contents . . . i

List of Figures . . . v

List of Tables . . . vii

Listings . . . ix

Acronyms . . . xi

1 Introduction . . . 1
- 1.1 Objectives . . . 2
- 1.2 Contributions . . . 3
- 1.3 Dissertation Outline . . . 4

2 Internet of Things (IoT) . . . 5
- 2.1 Concept and Vision . . . 5
- 2.2 Scenarios . . . 6
- 2.3 Wireless Sensor Network (WSN) . . . 7
- 2.4 Machine-to-Machine (M2M) Communications . . . 8
  - 2.4.1 Representational state transfer (REST) . . . 9
  - 2.4.2 MQ Telemetry Transport (MQTT) . . . 9
  - 2.4.3 Constrained Application Protocol (CoAP) . . . 10
  - 2.4.4 Comparative Summary . . . 10
- 2.5 Infrastructure . . . 11
  - 2.5.1 Vertical Solution . . . 11
  - 2.5.2 Centralized Horizontal Solution . . . 12
  - 2.5.3 Decentralized Horizontal Solution . . . 16
- 2.6 Persistence . . . 18
- 2.7 Privacy and Security . . . 19

3 Distributed Systems . . . 21
- 3.1 Architectures . . . 21
  - 3.1.1 Centralized Architectures . . . 21
  - 3.1.2 Decentralized Architectures . . . 22
- 3.2 Distributed Hash Table (DHT) . . . 24
  - 3.2.1 Chord . . . 24

3.2.2 Kademlia 25
3.2.3 Pastry 26
3.2.4 Tapestry 27
3.2.5 Content Addressable Network (CAN) 28
3.2.6 Comparative Summary 29
3.3 Security 31
3.3.1 Hierarchical Certification 32
3.3.2 Web of Trust (WoT) 32
3.3.3 Blockchain-based PKI 33
3.4 Promising Decentralized Infrastructures 33
3.4.1 Ethereum 33
3.4.2 InterPlanetary File System (IPFS) 34
3.4.3 ZeroNet 34

4 Decentralized IoT Infrastructure 37
4.1 Problem Statement 37
4.2 Functional Requirements 38
4.3 Non-functional Requirements 40
4.4 Design Principles 42
4.4.1 Overlay Network 43
4.4.2 Management Interface 45
4.4.3 Data Interfaces 50
4.4.4 Data Persistence 54
4.5 Peer Architecture 54

5 Prototype Implementation and Technologies 57
5.1 Architecture 57
5.2 Twisted Application Communication 59
5.2.1 REST Application Programming Interfaces (APIs) 60
5.3 Register Server 61
5.4 Deployment and Integration 61
5.4.1 Configuration and Setup 62
5.4.2 Application Binding 63

6 Evaluation and Results 65
6.1 Sensor Simulators 65
6.1.1 Software Specification 65
6.1.2 Data Model 66
6.2 Web Application 67
6.2.1 Architecture 67
6.2.2 Features and Graphical User Interface 68
6.3 Deployment Scenario 73
6.3.1 Network Deployment 73
6.3.2 Performance Evaluation 74
6.3.3 Communication Protocols Evaluation 76
6.3.4 Data Security Evaluation 77
6.3.5 Final Overview 79

7 Conclusions 81
7.1 Final Considerations 81

7.2 Future Work 82

References 83

Appendix A: Infrastructure's Peer REST API 89

Appendix B: Register Server's REST API 97

Appendix C: Gateway Peer Configuration 102

Appendix D: Gateway Peer Configuration File 104

List of Figures

1.1 Infrastructure shared across the globe. 3

2.1 Devices growth prediction. 6
2.2 Wireless Sensor Network. 7
2.3 Vertical Solution for the IoT. 11
2.4 The IrisNet architecture. 12
2.5 Horizontal Solution for the IoT. 13
2.6 The Sensor Andrew architecture. 14
2.7 High level architecture for the Service oriented framework for IoT. 14
2.8 The OpenMTC architecture. 15
2.9 Overview of the WebDust Architecture [40]. 16
2.10 Overview of the ADEPT Architecture [43]. 17

3.1 Wireless Sensor Network. 22
3.2 Peer-2-Peer Model 23
3.3 Chord Topology organization. 24
3.4 Kademlia Topology organization. 26
3.5 CAN 2-d topology organization. 28

4.1 Network of the infrastructure. 38
4.2 Use case Scenario. 39
4.3 Process for a new node get in the overlay. 44
4.4 Messages exchange in the overlay network 45
4.5 Sign up a user in the infrastructure. 47
4.6 Log in a user, in order to request its data to the infrastructure. 48
4.7 Bind sensor to a pseudonym. 49
4.8 Share data from a sensor with other entity. 50
4.9 Sensor data to be inserted in the DHT. 52
4.10 Get data history from the distributed database. 53
4.11 Architecture for a Decentralized IoT Peer. 55

5.1 Gateway Peer prototype architecture. 58
5.2 Kademlia API Interactions. 59
5.3 Register Server prototype architecture. 61

6.1 HTTP Sensors Simulator. 66
6.2 Web Application. 67
6.3 Log in interface. 68

6.4 Sign up interface. 69
6.5 Main interface of the dashboard. 69
6.6 Sensor management Interface. 70
6.7 Share Data Interface. 71
6.8 Main interface of the user. 71
6.9 Sensor management Interface. 72
6.10 Data History of a Sensor Interface. 72
6.11 Prototype Scenario Diagram. 73
6.12 Percentage of CPU usage. 75
6.13 Percentage of Memory in Peers. 75
6.14 Response time of HTTP messages over time. 76
6.15 Response time of MQTT messages over time. 77
6.16 InfluxDB query for bound sensor. 78
6.17 Sensor data request. 78
6.18 Sensor data response. 79

1 Documentation Resume. 89
2 Method for getting the last received data of a sensor. 90
3 Method for getting the data history of a sensor. 90
4 Method for publishing new data in the Infrastructure. 91
5 Method for accepting a share request. 91
6 Method for getting the access list of a sensor. 92
7 Method for acknowledging a data sharing refusal. 92
8 Method for acknowledging a data unsharing. 93
9 Method for binding a new sensor. 93
10 Method for getting the private data of the user, when logged in. 94
11 Method for logging a user in the infrastructure. 94
12 Method for refusing a share request. 94
13 Method for sharing data of a sensor. 95
14 Method for finishing the sign up process by spreading the user's private credentials through the overlay. 95
15 Method for initiating a sign up process. 96
16 Method for unsharing sensor data. 96

17 Documentation Resume. 97
18 Method for getting a list o bootstrapping peers. 97
19 Method for getting the self-signed certificate of the Certification Authority. 98
20 Method for getting the data a specific peer. 98
21 Method for getting the public key of a peer. 98
22 Method for registering a peer on the Certification Authority. 99

List of Tables

3.1 DHT comparison summary (1). 30
3.2 DHT comparison summary (2). 30
3.3 DHT comparison summary (3). 30
3.4 DHT performance comparison. 31

4.1 Use cases description. 40

6.1 Virtual Machines' Specifications. 74
6.2 RaspberryPis' Specifications. 74

Listings

1 User Data Model for DHT . . . 46
2 Sensor Meta-data Model for DHT . . . 48
3 Sensor Data Model for DHT . . . 51
4 Menu output for gateway script. . . . 62
5 Sensor object from JSON file. . . . 66
6 Format of data sent from the simulator. . . . 67
7 Output for configuring a gateway with all services enabled. . . . 102
8 JSON Configuration File. . . . 105

Acronyms

ADEPT	Autonomous Decentralized Peer-to-Peer Telemetry
API	Application Programming Interface
CA	Certification Authority
CAN	Content Addressable Network
CoAP	Constrained Application Protocol
DBMS	Database Management System
DDoS	Distributed Denial of Service
DNS	Domain Name System
DHT	Distributed Hash Table
HMAC	Hash-based Message Authentication Code
HTTP	Hypertext Transfer Protocol
IEEE	Institute of Electrical and Electronics Engineers
IoT	Internet of Things
IP	Internet Protocol
IPFS	InterPlanetary File System
IV	Initialization Vector
JSON	JavaScript Object Notation
M2M	Machine-to-Machine
mDNS	Multicast Domain Name System
MQTT	MQ Telemetry Transport
P2P	Peer-to-Peer
PGP	Pretty Good Privacy
PKI	Public Key Infrastructure
PoC	Proof-of-Concept
QoS	Quality of Service
REST	Representational state transfer
RPC	Remote Procedure Calls
SDK	Software Development Kit
SPO	Small Programmable Object
TCP	Transmission Control Protocol
TSDB	Time series Database
UDP	User Datagram Protocol
URI	Uniform Resource Identifier
W3C	World Wide Consortium
WoT	Web of Trust
WSN	Wireless Sensor Network

CHAPTER 1

Introduction

The continuous evolution of wireless and mobile communications has allowed an ever increasing number of devices, such as computers, smartphones and daily items (for example, air conditioning and heating systems, or luminaries), to not only being remotely managed, but also to be connected to the Internet. That enhancement, together with the technological evolution of sensors, resulted in the development of the Internet of Things (IoT). Although the term is relatively recent, this concept has the same basis as the concept of Ubiquitous Computing.

Nowadays, the IoT is becoming an important subject in the technology industry, as it can be seen in the headline news of popular media [1] [2] [3]. Moreover, it is bringing into existence exceptional opportunities for a considerable number of new use cases, which promise to enhance our quality of life. Therefore, the approaching wave in the era of computing will be outside the realm of the traditional desktop.

In Computer Science, the IoT is a concept in which computing is made to appear everywhere, in order to improve the way the world works. IoT focuses on a wide range of topics, including Distributed Computing and Sensor Networks. Several technologies serve as the building blocks of this new paradigm, such as Wireless Sensor Network (WSN) and Machine-to-Machine (M2M) communications. Finally, it has an infinitude of application domains, such as health care, logistics, automotive, environmental monitoring, and many others.

Above all, the IoT aims to allow several heterogeneous devices to be sensed and controlled remotely through network infrastructures. These devices are locatable and addressable on the Internet, which allows them to communicate with other devices, but frequently IoT Gateways. As a result of the communications between the physical world and computers, it is possible to both enhance the efficiency and accuracy of things, as well as to decrease costs.

Technology advances so rapidly that it is difficult to understand what will make a remarkable impact on human life. There are some breakthroughs that have potential to make a difference, such as the IoT. Nonetheless, it raises significant challenges that could stand in the way of realizing its potential benefits. In conferences, reports and news articles [1]–[3], it has been discussed the potential impact of

[1] http://www.businessinsider.com/how-the-internet-of-things-market-will-grow-2014-10

[2] http://cloudtweaks.com/2016/04/growing-popularity-iot/

[3] http://betanews.com/2016/01/19/internet-of-things-rising-popularity-will-increase-security-risks-business-costs/

the `"IoT Revolution"`. Some observations refer to the IoT as a revolutionary fully-interconnected `"Smart"` world of progress, efficiency, and opportunity, with the potential for adding billions in value to industry, as well as to the global economy [1]. Others refer that it represents a darker world of surveillance, privacy and security violations, as well as consumer lock–in [2], [3].

Accordingly, critical problems have to be solved, in order to achieve the IoT success. It is crucial to build an open and uncensored network infrastructure, which has to be capable of interconnecting all types of IoT business processes, as well as to be efficient, secure and scalable. This way, each business model may bind their WSN to the infrastructure, without any network and privacy concerns.

Recently, developers and manufacturers have created a considerable number of different IoT infrastructures (either closed or not). This allowed the communication between their own devices and applications, without concerning with the devices' interoperability [4]. However, this advancement in the architecture of IoT platforms resulted in the appearance of the so called silos [5]. A data silo consists of a data repository, which is under the control of a single entity. It may lead to inflexible solutions, as well as to security and privacy problems.

Moreover, thanks to the interactions among machines without human intervention, the amount of information that flows in this networks is increasing significantly. Although each device does not send data to the infrastructure in a data stream, and the data sent each time is relatively small, the fact that there might be millions of sensors sending data at the same time, gives the sense that the network is being crossed by large streams of data.

Therefore, the infrastructures have to be flexible and scalable enough to accommodate such a level of diversity, concerning both the users and objects, as well as to support billions of smart objects, producing massive amounts of data. In addition, it is fundamental to preserve privacy and provide secure means for data sharing, in order to accomplish a secure and reliable infrastructure. However, with billions of devices being connected together, an infinitude of vulnerabilities will inevitably be uncovered. This results in the devices becoming vulnerable to exploits. Furthermore, IoT is present in many applications and industry use cases, each having its own security requirements but relying in the same fundamental IoT technologies. Therefore, designing security that applies to all use cases is an imposing task.

The current solutions rely on centralized architectures and not so open environments, in reality leading to the Internet of Platforms [6]–[8]. With this dissertation, we propose a new concept to truly enable an organically growing and secure Internet of Things.

1.1 OBJECTIVES

Traditional IoT infrastructures are built on top of a centralized architecture, which obtains information from sensors located in data retrieval networks, and provide it to data consumers. Therefore, these centralized platforms control the whole information flow. On the other hand, taking into consideration a distributed approach, entities at the edge of the network would exchange information and collaborate with each other in a dynamic way, providing a self-organized and scalable infrastructure.

Our work aims at taking a side step from current trends, in order to avoid the many issues created by a platform oriented to the Internet growth. In particular, we consider an Internet where an high number of loosely coupled devices, owned by users, or even Telecom operators, provide a decentralized infrastructure, which is responsible for interconnecting all IoT platforms, without the

need for any central entity. It focuses on storing and securing data at a global scale, instead of in a set of central entities. Accordingly, in a distributed approach, entities at the edge of the network exchange information and collaborate with each other in a dynamic way, providing a decentralized, self-organized and scalable operation.

We specifically aim at avoiding centralized data storage, and centralized data processing, even when considering Cloud infrastructures. We particularly consider that data should be strongly encrypted and only accessible to their rightful owners, or someone else with the appropriate authorization. Scalability and organic growth, by the addition of new sensors and systems, are also mandatory. This will have the potential of empowering local businesses and citizens to provide the services required for others to integrate their devices. In the end, we envision that everything will be connected to a global net, acting as a secure common data repository and communication channel, which we see as the real Internet of Things. This approach resides in a decentralized infrastructure that may be used across the globe for a diversity of business processes, each one with its own requirements, as illustrated in Figure 1.1.

Figure 1.1: Infrastructure shared across the globe.

All things considered, the centralized approaches have some properties that do not match the IoT philosophy. Some crucial challenges are scalability, fault tolerance and data silos. Accordingly, this dissertation proposes to solve the above challenges relying on a decentralized architecture, based on a DHT. Therefore, taking into account the characteristics of a decentralized architecture [9], we intend to design an efficient, scalable and secure infrastructure, which could be used by a wide range of IoT business processes, preserving the entities privacy.

1.2 CONTRIBUTIONS

The proposed and implemented solution contributes to the evolution of the IoT infrastructures. Essentially, it provides a new vision for scaling and securing this type of platforms.

The work inherent to this dissertation originated an article entitled `"Secure Decentralized IoT Infrastructure"`, which was submitted to the Wireless Days 2017, an international conference co-sponsored by IEEE Communications Society and IFIP.

1.3 DISSERTATION OUTLINE

This document is organized in 6 chapters, being the first chapter the already presented introduction. The next chapters of this dissertation are organized in the following sequence:

- **Chapter 2**: introduces the necessary background information about the IoT state of the art, regarding its scenarios, protocols, databases and architectures.
- **Chapter 3**: gives an overview of distributed systems architectures, as well as how to implement secure distributed systems. Additionally, several decentralized solutions are presented, in order to analyze the potential of these solutions;
- **Chapter 4**: provides a depth description of the problem that this dissertation aims to solve, as well as the requirements of the proposed solution. Moreover, the decisions and principles used for designing the solution are specified;
- **Chapter 5**: presents the implemented prototype, based on the designed solution previously depicted. Shortly, the software architecture and its communications are described, as well as the implemented Register Server;
- **Chapter 6**: provides an overview of the obtained results, as well as a discussion concerning the performance and security of the proposed solution;
- **Chapter 7**: sums up the outcomes of this dissertation, along with relevant conclusions and future work that may enrich this work.

CHAPTER 2

Internet of Things (IoT)

At the moment, the IoT is becoming a popular term of discussion and research. However, connecting a few embedded systems to the Internet is not sufficient to achieve its tremendous potential. Therefore, before confronting any other problems, it is important to define in this chapter, what is this IoT, as well as its vision and how it can be explored. Afterwards, it is presented a brief description of the state of the art for the IoT data flow (generation, transmission and persistence), as well as a detailed analysis of the IoT infrastructures state of the art. Finally, the state of the art regarding Privacy and Security in the IoT is described.

2.1 Concept and Vision

The term "Internet of Things" (IoT) was used for the first time in 1999. It was used by a British technology pioneer called Kevin Ashton, in order to describe a system in which objects could be connected to the Internet [10]. This concept promised to transform the way we work, live and play.

At the present time, the Internet of Things is an emergent topic of technical, social, and economic significance. It illustrates a variety of scenarios in which Internet connectivity and computing capability extends to a diversity of objects, sensors and devices, allowing these objects to produce, exchange and consume data.

The enhancement of technology is providing more processing power, as well as storage and battery capacity, at a relatively low cost. This trend is allowing the production of small electronic devices, which may be embedded in common objects. As a result, smart objects are getting to the market and the IoT is showing up is potential to the consumers.

Above all, the IoT illustrates the vision of a network of devices that collect data from an environment and share it through the Internet. This collected data can be processed, with the objective of transforming it into valuable information. Moreover, devices communicate through M2M communications, allowing the information to flow from machine to machine with no human intervention.

As the number of Internet-connected devices grows, the amount of traffic generated is presupposed to increase significantly. For instance, Cisco estimates that the amount of M2M connections will

increase from 24% of all connected devices in 2014 to 43% in 2019[11]. Consequently, the quantity of generated data in the context of the IoT will considerably grow.

Although they differ about predictions, most technology observers admit that billions of additional devices will be connected to the Internet in the near future. As a result of these numbers around IoT, numerous studies have been published in the recent years, which focus on the IoT potential and trends.

In a study [12] conducted by John Greenough, a Senior Research Analyst for Business Insider (American business, celebrity and technology news website) it was predicted that by 2020, there will be 24 billion IoT devices. The following Figure 2.1 aims to illustrate the expected growth of IoT devices.

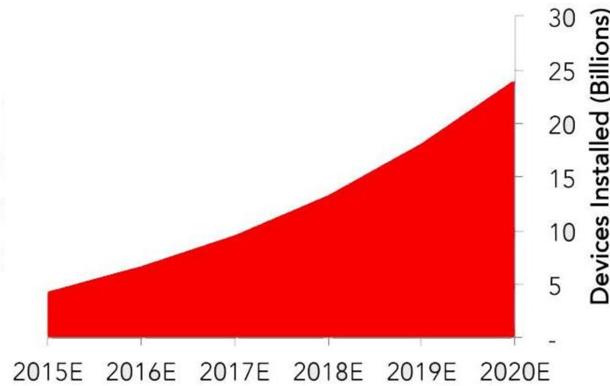


Figure 2.1: Devices growth prediction.

Oracle published a business outlook study [13], which estimates that the value of the global IoT market will be $14.4 trillion by 2022 [14]. Additionally, it is anticipated that the IoT will provoke a tremendous impact in the amount of digital data that needs to be stored [15]. Accordingly, it is expected that it will generate 4.4 trillion GB of data by 2020.

2.2 SCENARIOS

Unquestionably, the IoT vision offers a considerable set of opportunities to companies, as well as users, in a wide range of contexts. Therefore, this paradigm has an infinitude of application domains, such as the following ones.

The concept of `"Smart Cities"` has attracted considerable attention and investments recently, aiming to improve the citizens' quality of life and the socio-economic development [16]. Accordingly, there are several use cases regarding this topic, such as managing the rubbish levels in containers aiming to optimize the trash collection routes, monitoring parking spaces in order to determine the available parking spaces in the city, supervising the traffic congestion in the interest of improving the driving routes, among others.

Taking into account the enhancement of efficiency and security of houses, the `"Smart Homes"` approach has also appeared [17]. Therefore, there are some use cases in this field, such as intrusion detection systems, monitoring the energy and water consumption, remote control of devices in order to avoid accidents, as well as save energy and time, and so forth.

Considering that the Agriculture is an industry closely related to the welfare and the people's livelihood, it is also an important application domain for the IoT [18]. It is possible to control micro-

climate conditions to maximize the production, as well as its quality, study the weather conditions in fields in order to enhance forecasts, enhance the wine quality through the monitoring of soil water in vineyards, with a view to control the amount of sugar in grapes, as well as grapevine health, among others.

Recently, environment monitoring has also received much attention, focusing on using scientific and engineering principles to enhance the environmental conditions. This area has experienced a very rapid development thanks to the IoT rise [19]. Control air pollution in factories, as well as the pollution emitted by cars, fire detection through the monitoring of combustion gases, snow level measurements in real time, in order to predict avalanches, are some examples of IoT use cases regarding the environment monitoring.

The IoT may also provide several benefits in the Health-care domain [20], such as patients surveillance through the monitoring of patients, sportsmen care through vital signs monitoring in high performance centers, identification and authentication of people, automatic data collection and sensing, among others.

In a final analysis, it has to be noticed that each scenario has unique requirements, which must be taken into account. For instance, disaster predictions have real-time requirements, while in tracking scenarios, security and reliability are vital. Instead, scenarios like Smart Agriculture do not have substantial constraints regarding real-time and security. Therefore, the IoT requires an infrastructure able to handle different scenarios' requirements and paradigm modifications.

2.3 Wireless Sensor Network (WSN)

One of the key concepts associated with the IoT is the concept of WSN. A WSN [21] is a network consisting of a large number of autonomous nodes (devices), cooperating in order to collect, process, analyse and propagate valuable data. This data is sent through the network to a main location, i.e. Gateway Sensor Node. An example of a WSN is depicted in Figure 2.2.

Figure 2.2: Wireless Sensor Network.

A sensor is a hardware device that generates measurable response signals when a physical condition changes, such as temperature, pressure and location. This analog signal is digitalized by a analog-to-digital converter and sent to an embedded processor for being processed. Therefore, each node is

usually equipped with one or more sensors, as well as, a microcontroller, a wireless transceiver and an energy source (usually a battery). Thanks to the flexibility given by wireless networks, nodes can be in any location within a range, of one another. Despite the word "sensor" in the name, nodes can also be actuators, interacting with the physical world based on decision processes.

In the IoT context, it is possible to think in a WSN as a virtual layer where the information about the world can be accessed by a computational system [22]. Accordingly, combining the IoT and WSN has benefits regarding the heterogeneous environment inherent to the IoT, since a group of devices communicate with a single gateway, instead of directly with the infrastructure. Consequently, sensors may communicate with the gateway using a protocol within a set of different communication protocols, as described in the following section.

Finally, it is important to notice that the majority of the IoT devices are constrained in memory, CPU and power capacity. Therefore, it is crucial to have this characteristic in mind when developing software for the IoT.

2.4 Machine-to-Machine (M2M) communications

In the recent history of technology, the evolution of cellular networks allowed the creation of a new paradigm, which consists on connecting smartphones, tablets, sensors and wearables in a smart environment. Therefore, the evolution of M2M is providing the opportunity to create a new concept of smart environment, where simple devices like sensors and actuators are also connected through the Internet.

M2M Communications consists of a set of mechanisms for allowing a direct communication between devices through a certain channel, without human intervention. These mechanisms include algorithms to handle networked devices and to transmit data. The evolution of M2M results mainly from the technological progress in the sensors area, as well as the decreasing costs of semiconductor components.

Just like the IoT, this is not a new concept. It is already common to have communications between machines without human intervention, such as routers' and servers' communications. However, in the context of the IoT, machines not only are able to communicate with each other, but they are also capable of understanding the information received. This capacity provides a wide range of opportunities, as it is possible to trigger certain actions in real time, according to the data received [23].

As previously stated, WSNs are composed by a set of sensor nodes and a gateway, where sensor nodes must communicate with the gateway to publish its data, that is, M2M communications between sensor nodes and the gateway. In addition, these nodes operate on constrained devices, and consequently require low-bandwidth and energy efficient protocols for data transmission.

Considering a global deployment of a IoT infrastructure, which is composed by heterogeneous devices around the globe, the infrastructure must be able to communicate with a set of different protocols. The most used protocols for the IoT are REST, MQTT and CoAP, which are described next [24], [25].

2.4.1 Representational state transfer (REST)

The IoT intends to build a deployment where devices can communicate with each others using the Internet. Taking into account the adoption of REST communications in the World Wide Web, it is fundamental for the IoT gateways to support this type of communication. As a result, web services based on REST ease the IoT deployment and management.

REST is a software architectural style, which is built on top of the HTTP and, consequently, relies on a client-server architecture. Accordingly, this protocol uses the request-response model wherein a client sends a HTTP request to the server, the server handles the client request and returns a HTTP response message to the client. The response may be characterized by its code, which can indicate the completion of the request, as well as an error that occurred. Moreover, it offers a reliable transport and flow control, so that the sender cannot overflow the receiver, and it supports the use of a structured data type, such as JSON or XML.

This type of communication is characterized by stateless interactions between a server and a set of clients. Accordingly, the interactions between both parties are oriented to a particular resource, which has a specific state and a unique address. Thanks to the central server that handles clients' requests, the data processing may be executed in the server, instead of on the clients. As a result, the clients efficiency increases significantly.

In the context of the IoT, a sensor can be the client that reports data changes to a specific server, using a HTTP POST request. Thus, each smart device has to send its data through a specific URI, which contains its identifier. This protocol matches perfectly several use cases of the IoT. However, it does not satisfy the real time requirements of some business processes [24].

2.4.2 MQ Telemetry Transport (MQTT)

MQTT is an asynchronous messaging protocol that was introduced by Andy Stanford-Clark and Arlen Nipper in 1999 and was standardized in 2013 at OASIS [26]. It intends to connect embedded devices and networks with applications and middlewares. In addition, it consists of a lightweight publish-subscribe messaging protocol designed for resource-constrained devices and M2M communications [25], [26]. Finally, it is built on top of TCP/IP protocol.

This protocol is based on a publish-subscribe pattern, in order to provide transition flexibility and simplicity of implementation, as well as one-to-many message distribution. Moreover, MQTT is composed by three components, a subscriber, a publisher and a broker. Thus, an interested entity may register as a subscriber for a specific set of topics, with a view to be informed by the broker when a publisher publishes data to the topics previously subscribed. Regarding the IoT, a device publishes MQTT messages to the network. Those messages contain a topic, which generally consists of a device identifier, and the data generated by the sensor. Applications and services may subscribe sensors, in order to receive their generated data.

A MQTT [26] message is composed by a header and a payload. The header has a fixed length of 2 bytes and contains the protocol name and version, several flags, message type, among others. As a result, each type of MQTT message has an associated message type, such as CONNECT, DISCONNECT, PUBLISH, SUBSCRIBE, among others. The payload is responsible for the content of the message.

Regarding Quality of Service (QoS) for the transmission of messages, MQTT has three types of QoS control [26]. Firstly, QoS level 0 (at most once delivery) may be used when message loss can occur. Secondly, QoS level 1 (at least once delivery) should be adopted if message loss can not occur,

but message duplicates may exist. Finally, QoS level 2 (exactly once delivery) has to be used when it is intended that messages always arrive to the destination, without any duplicate.

Considering the devices with limited resources that compose the IoT, it is crucial for the IoT gateways to also support MQTT communications. However, it was not designed with security in mind and it only checks authorization of the publishers and subscribers through the use of a user-name and a password.

All things considered, this protocol is an important messaging protocol for the IoT and M2M communications, as a result of providing lightweight messaging between devices, as well as real time data acquisition.

2.4.3 Constrained Application Protocol (CoAP)

CoAP [27] was designed by the IETF Constrained RESTful environments working group. It consists of an application layer protocol for RESTful web services in resource-constrained devices. In addition, it aims to keep the protocol overhead as low as possible.

As practiced in REST, CoAP also uses a URI for identifying resources. Moreover, it provides several features, such as adaptability for constrained environments, multicast and asynchronous transactions support, built-in resource discovery, among others [28].

Taking into account its architecture, CoAP is divided into two layers, the messaging and the request/response. While the messaging layer provides reliable asynchronous communications over the UDP, the request/response layer handles the requests and responses exchanged on top of REST.

In spite of CoAP being based on HTTP, it relies on UDP instead of TCP [25]. As a result of the UDP lower overhead, CoAP is prepared for being used by constrained devices (processing, bandwidth, memory and power). Its header may be reduced to 4 bytes, in order to produce lightweight messages. In addition, some HTTP features are modified, in order to guarantee low power consumption and operation in the presence of lossy and noisy links. Nonetheless, since it has been built on top of REST, the conversion between these two protocols is straightforward [28] and its integration with the web is easy.

In summary, despite having a behavior similar to REST, it is more lightweight and, consequently, has a smaller overhead. However, it does not satisfy the real time requirements either.

2.4.4 comparative summary

After presenting all the previous M2M application protocols, it is important to compare them according to different metrics. These protocols are important for the efficiency of a IoT platform and should be analyzed according to their bandwidth and energy consumption, overhead, reliability, as well as the number and size of the messages that need to be exchanged.

Taking into account the performance of each protocol, the chosen transport layer has a crucial impact on it [29], [30]. In spite of providing more reliability to the communications, protocols based on TCP consume more bandwidth than the protocols built on top of UDP, as a result of exchanging messages frequently, in order to maintain its connection. Therefore, MQTT and REST use more bandwidth than CoAP for transferring the same amount of data. Consequently, CoAP should be used for constrained environments.

For transmitting a certain quantity of data, REST needs to transmit more bytes than CoAP and MQTT, as a result of not being optimized for constrained devices and, consequently, having a larger header with unimportant information in it. Despite the MQTT header being smaller than the CoAP header, both are small enough for the requirements of constrained communications.

With a view to prevent data loss, CoAP and MQTT have a QoS mechanism to enhance their reliability. Both mechanisms include retransmission time-out for solving this problem. However, increasing the number of exchanged messages across the network has a considerable impact on the performance of the protocols.

Considering the service discovery, CoAP has its own service discover, while MQTT and REST use the DNS for the discovery in the network. According to a study [31], the service discovery used in CoAP has a lower overhead than DNS.

All in all, the described protocols have their set of advantages and disadvantages. Shortly, CoAP and MQTT were conceived for constrained environments, while HTTP was designed for web scenarios. Consequently, an IoT gateway should be able to use all these protocols to communicate with sensors, so that it may be prepared for the heterogeneous environment inherent to the IoT, as well as the requirements of each use case.

2.5 INFRASTRUCTURE

2.5.1 VERTICAL SOLUTION

Taking into consideration that in a first instance WSNs were used in a limited scale, the first infrastructures created to manage these WSNs were not complex. As an initial approach, the vertical solution consists of a system capable of producing data, which can be used by a service or an application. Figure 2.3 illustrates a vertical solution deployment, where each business model needs its own infrastructure, in order to have communication with its WSN.

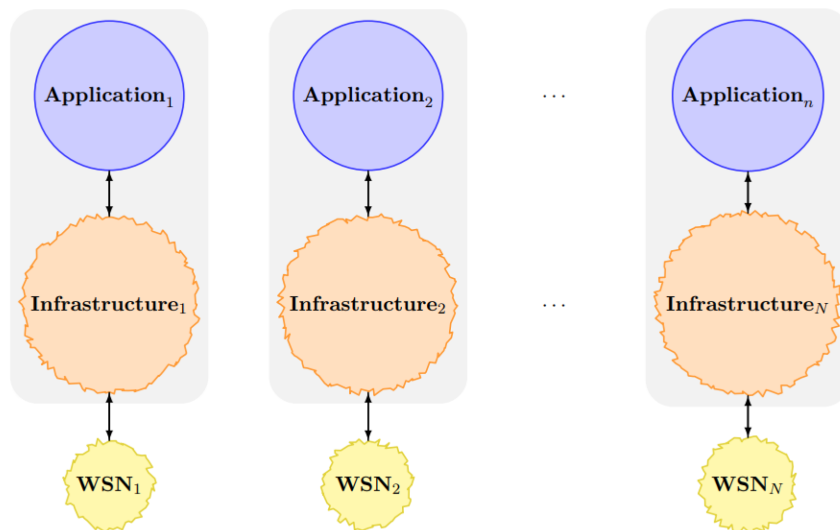


Figure 2.3: Vertical Solution for the IoT.

The first proposal for integrating networks of sensor was IrisNet [32]. This project intended to create a global system built on top of a WSN, which provides a set of services to an application. In other words, it aims to receive data from multiple heterogeneous sensors, as well as to provide it to an application developed by a third party.

The architecture of IrisNet is divided into two layers as represented in Figure 2.4, namely the Sensing Agents (SAs) layer and the Organizing Agent (OA) layer. The first one is responsible for accessing and fetching the data from sensors in a generic way, in order to allow heterogeneous sensors in the WSN. The other one has to handle the persistence of the retrieved data.

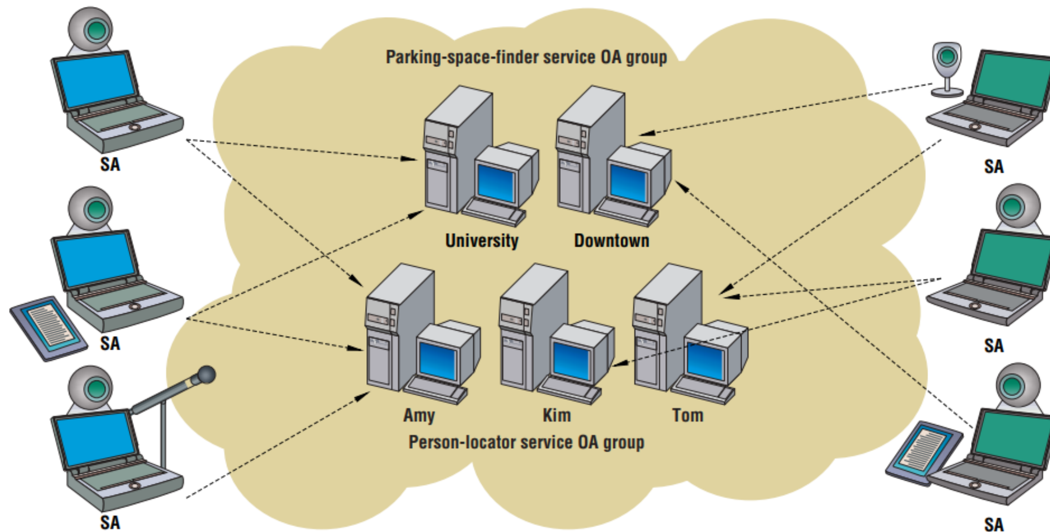


Figure 2.4: The IrisNet architecture.

In spite of IrisNet not presenting a relevant state of the art, it represented a transformative vision, regarding the use of sensors networks, as well as the possibility of getting information about the data generated by the sensors, in a comfortable way. Nevertheless, this project did not consider the possibility of combining the data received from multiple services, as a consequence of having a different service for each data source.

Despite being able to fill the requirements of some systems (at least, in an initial phase), vertical solutions do not match the IoT vision. Particularly, the evolution and scalability that these solutions provide is very limited, and lead to compartments and silos.

2.5.2 CENTRALIZED HORIZONTAL SOLUTION

With the aim of overcoming the main problems of the previous solution, horizontal solutions appeared in more complex systems. They introduced an additional layer, which is used to create an abstraction of the communication between applications and WSNs. In Figure 2.5, the deployment of a horizontal solution is illustrated. It is possible to verify that the network infrastructure is shared among the applications and WSNs, which contrasts with the previous solution.

As a result of this appearance, several highly flexible infrastructures appeared, more capable of scaling and easier to manage and maintain. Nowadays, commercial solutions from world class players (e.g., Amazon, ThingsWorks) provide highly available products, capable of integrating heterogeneous

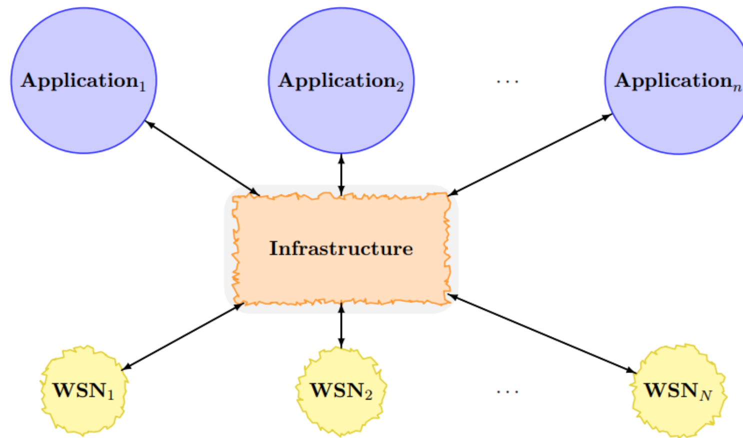


Figure 2.5: Horizontal Solution for the IoT.

devices with ease, and potentiating novel services. From an academic perspective, several other solutions kept innovating, by exploring more radical, and differentiated solutions. Several examples are described below.

MetroSense [33] follows a people-centric paradigm and is focused on urban sensing. It consists of an infrastructure based on an opportunistic sensor network [34], that allows it to scale to large urban areas together with the active users. At its core, it provides data from the interactions between people and their environment.

The MetroSense architecture consists of a hierarchical network structure, which allows a scalable people centric sensing at a relatively low cost. Moreover, it focuses on the use of sensors, as well as the range and scalability of the Internet to provide data gathered from and around people, to a set of applications.

In the context of MetroSense, a prototype was developed and tested. Therefore, a sensor network was installed in a university campus, which sent its data to an infrastructure. This infrastructure provided the generated data to a set of applications.

Sensor Andrew [35] presents a multi-purpose, scalable infrastructure, aiming to offer a platform where applications can be easily deployed, in order to handle the received data. It fails to offer a scalable data persistence, as well as to support a heterogeneous environment of sensors, but this work was pioneer in many aspects related to scalability.

As a consequence of focusing on a set of goals that had not been studied in depth before, this project was important to the advancement of IoT infrastructures. These design goals were ease of management and configuration, built-in security and privacy and data sharing among different services.

With a view to satisfy the requirements specified previously, as well as robustness and scalability, a architecture composed by three layers was designed. It is divided into front-end server layer, gateway layer and transducer layer. The transducer layer is composed of sensor devices with low capacities, while servers and gateways were part of a campus network. Moreover, the gateway layer aggregates and structures the data received from sensors, in order to make it available for being requested by the front-end server layer. Finally, the front-end server layer allows the subscription of sensor data, so that it may provide high-level services. The architecture described is depicted in Figure 2.6.

Regarding security and privacy, an access control mechanism in the application communications interface was considered, which is essential to protect sensitive data. In addition, this interface allows privileges sharing, so that the stored data may be accessed by a set of authorized entities.

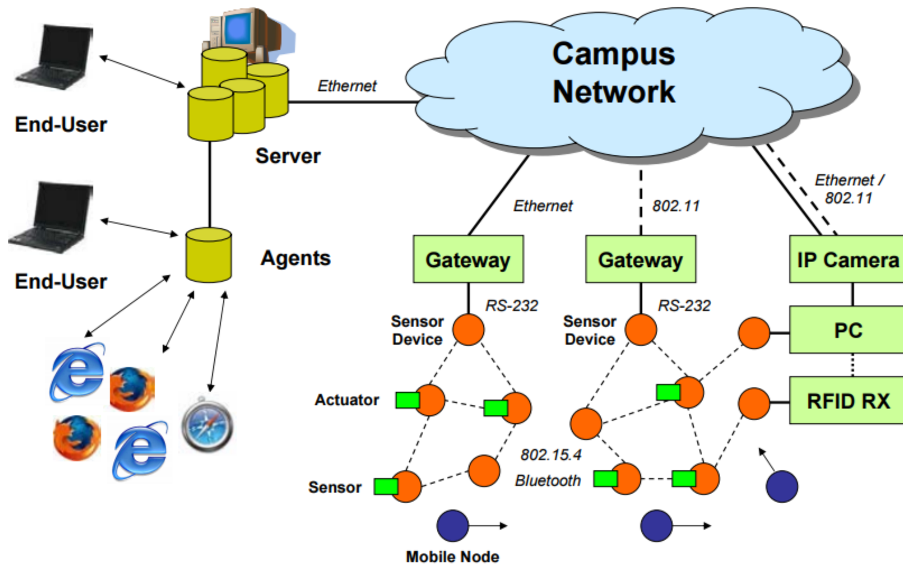


Figure 2.6: The Sensor Andrew architecture.

Considering another proposal, Service oriented framework for IoT [36] consists of an infrastructure that allows efficient and secure interactions between Small Programmable Objects (SPOs) and the Internet, as well as the management, maintenance and operation of those SPOs.

The main goal of this proposal consists of the management and monitoring of multiple networks of sensors. Moreover, its architecture is also divided into three layers. The SPOs are located in the lowest layer, the controller in the intermediate layer and the clients in the highest one, as presented in Figure 2.7.

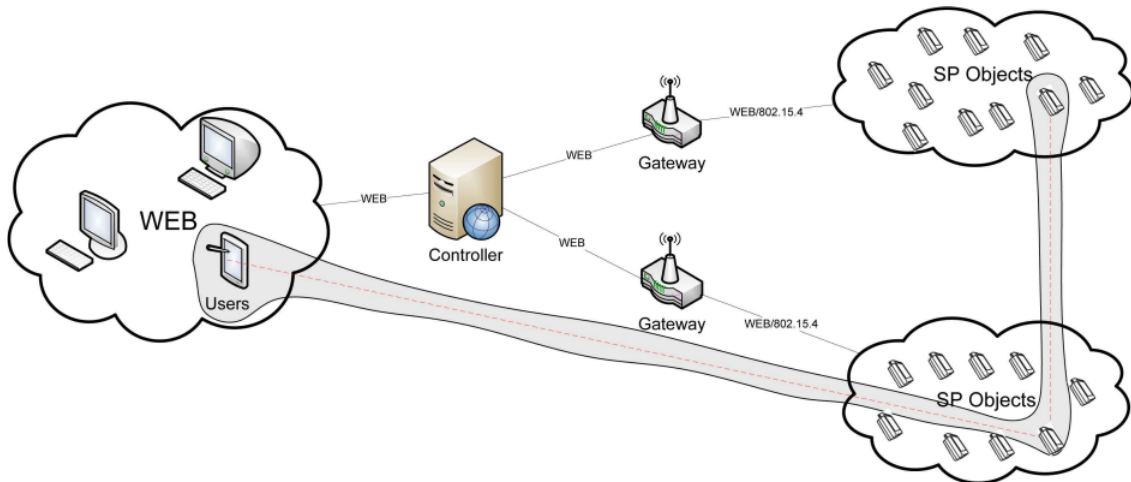


Figure 2.7: High level architecture for the Service oriented framework for IoT.

For exposing their features, SPOs provide web services, which are implemented using Senselets, a minimalistic adaption of Java Servlets that allows constrained devices to provide web services. Apart from Senselets, there are also mechanisms for interacting with the infrastructure in the SPOs.

The controller layer is divided into three modules. A WSN controller to perform administration and maintenance operations in the SPOs, a module routing that implements the virtual topologies among the networks of SPOs and the proxy module, which publishes and exposes the features provided by the Senselets to the client side.

Finally, the clients layer may offer a website or an application that consumes the services offered by the controller in order to interact with the SPOs.

On the other hand, some platforms were created aiming to offer a standards-compliant platforms for M2M services, with the aim of providing standardized environments, open to large scale integration of devices. One relevant example is OpenMTC [37], [38], which was developed by Fraunhofer FOKUS institute and Technische Universität Berlin.

OpenMTC is a middleware intended to interconnect IoT sensors from different scenarios using a cloud platform. Moreover, it provides communication and device management. However, it does not take care of data handling and persistence, as this is delegated to application specific software systems.

Taking into account that the OpenMTC aim to provide standardized environments, it offers a SDK containing core assets and services that may be used in third party applications. Therefore, Figure 2.8[1] illustrates the OpenMTC architecture.

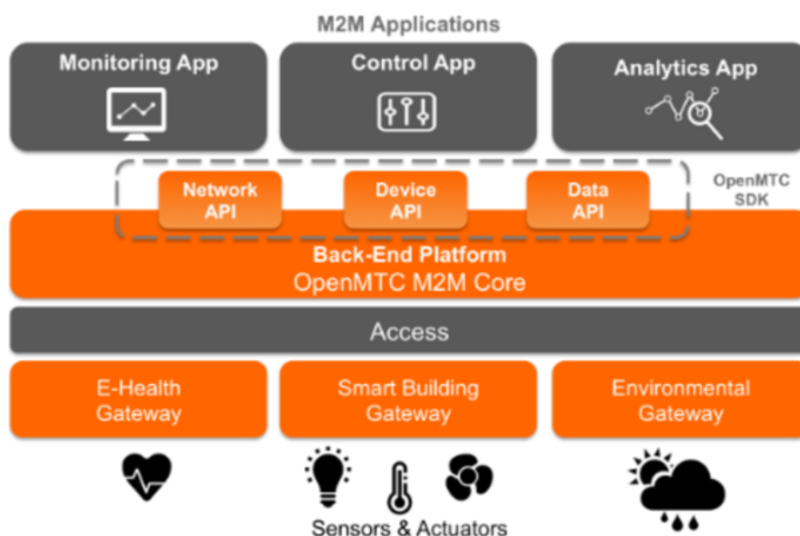


Figure 2.8: The OpenMTC architecture.

Summing up, traditional solutions are characterized by an infrastructure composed of a set of centralized servers, which receives WSNs' data from IoT gateways, providing it to applications through service exposure layers, pub-sub brokers and high level APIs. Consequently, these platforms control the whole information flow, and the model brings some disadvantages, which do not match the IoT vision [9]. Namely, scalability, single point of failure, surveillance, easy target for cyber-criminals and proprietary solutions without external security verifications. Accordingly, some approaches based on a decentralized infrastructure have appeared recently.

[1] http://www.openmtc.org/

2.5.3 DECENTRALIZED HORIZONTAL SOLUTION

One of the first approaches to decentralize traditional solutions was Hourglass[39]. This infrastructure aims to integrate different networks of sensors, which interact with computers and programmed agents. Moreover, it deals with the discovery, addressing and routing of data streams to client applications. Therefore, this proposal allows several applications to extract data from sensors positioned in different places, which are connected to the Internet.

Despite Hourglass allowing the communication with heterogeneous devices, as well as low bandwidth connections and data streams, it may have scalability problems, as a consequence of connections being established only at the time of each request. Thus, Hourglass opened a set of new problems, but provided a solid background for future research. In addition, it proved the feasibility of the concept.

WebDust [40] consists of a P2P framework for managing and monitoring multiple and heterogeneous WSN widely dispersed. Consequently, it provides a web-based dashboard for the administration and visualization of the network state.

This platform architecture encourages the implementation of customized applications. Moreover, it is based on three different sub-domains, namely the P2P network, Nano-Peers and Gateway-Peers. Firstly, the P2P network is composed by desktop applications that actuate as peers in the distributed environment. Nano-Peers is a cluster of sensors that communicate using wireless Communications. Finally, Gateway-Peers handle the communication between the Nano-Peers and the P2P network. In addition, Gateway-Peers have monitoring and sensing capabilities, and consequently act as stations that control the networks of sensors. An overview of WebDust architecture is depicted in Figure 2.9.

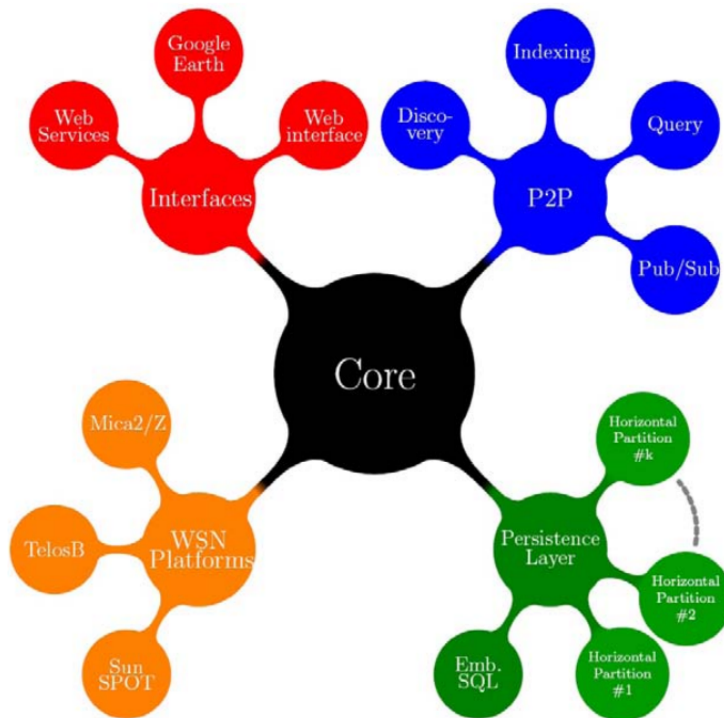


Figure 2.9: Overview of the WebDust Architecture [40].

Taking into consideration the peers that compose the network, each one has a two layer architecture containing an inner layer and an outer layer. The inner layer is responsible for keeping the sensor network information, as well as the results of the previous executed queries and the devices' specifications. The

outer layer takes care of the high level services provided by the peer. As a consequence, the outer layer requests information to the inner layer.

In this proposal, the data received from the networks of sensors is stored in a relational database. It is organized in different categories, namely by device, by query and by state of the sensor. Finally, this proposition offers other important services, such as a buffering for temporary storage to store data in connection failures, data aggregation by sets and a network performance monitoring.

More recently, large players like IBM and Samsung Eletronics developed the Autonomous Decentralized Peer-to-Peer Telemetry (ADEPT) Proof-of-Concept (PoC)[41]. It aims to demonstrate several capabilities, which are fundamental for building a full decentralized IoT, and clearly demonstrates that these solutions need research, having interest from the market. This proposal intended to design a system capable of providing a trust-less P2P messaging system and a distributed file sharing, as well as a autonomous coordination among devices

This PoC selected three open source protocols for its implementation. Firstly, Telehash (a DHT implementation of Kademlia protocol[42]), to provide encrypted peer-to-peer messages. BitTorrent was chosen for file sharing purposes, and finally Ethereum (a blockchain protocol), for the synchronization and security of the devices. A logical overview of its architecture is illustrated in Figure 2.10.

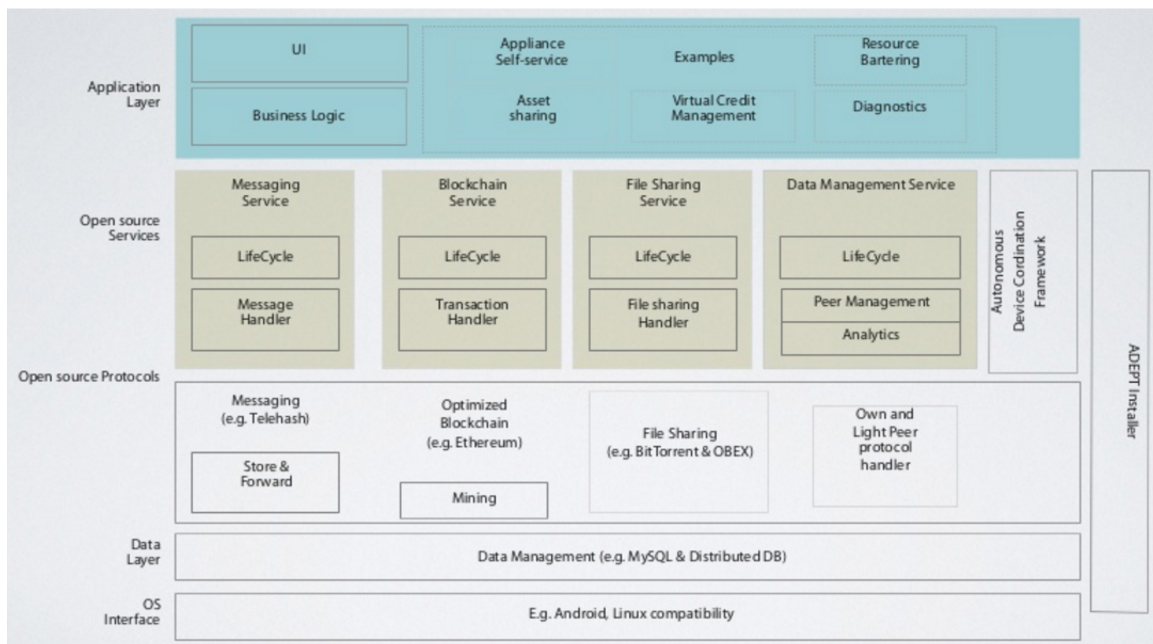


Figure 2.10: Overview of the ADEPT Architecture [43].

Combining the blockchain[41], [44] idea with the IoT may provide an attractive set of capabilities. Namely, maintaining sensors information, create an immutable history of transactions, as well as data security and privacy. ADEPT supports different types of devices, according to their performance and storage capabilities. However, devices require considerable resources, in order to take advantage of the security benefits of a blockchain.

The proposed use cases were tested using Samsung washers of the future, equipped with high storage and processing capabilities matching the blockchain requirements. This equipments are far away from the common user, as they represent a considerable investment. Moreover, no blockchain solution was tested at a world scale [45]. Even Bitcoin (world leader cryptocurrency) cannot scale to a

world currency, as a result of the large amount of time needed to validate each transaction. Finally, its consensus algorithm is under research, in order to prevent denial of service attacks [43].

All things considered, the solutions proposed pave the way to decentralized IoT systems, but they are not ready for the current view of the IoT, where the devices are expected to be inexpensive and their real-time communications may be crucial. In addition, even recent solutions like ADEPT are limited to Proof-of-Concepts (PoCs) and do support global scale operation, as consequence of the blockchain's slow validation and high resource requirements.

2.6 PERSISTENCE

After describing the communications between the WSNs and the gateway nodes, it is also crucial to describe how the infrastructure should process the received data, in order to generate valuable information. As previously mentioned, it is expected a massive growth in the amount of data generated by the IoT in the next years. Consequently, an efficient and scalable data persistence is necessary.

The main goals of a database consist of storing and retrieving data, as well as handling modifications. Currently, the databases are divided into NoSQL databases and relational databases. Shortly, NoSQL databases discard some features of relational databases, in order to enhance their scalability.

Data generated by sensors consists of a series of data points listed in time order, which is known by time series data. New NoSQL approaches are being used for handling this type of data, with advantages in flexibility and performance over relational databases [46]. Therefore, a new Database Management System (DBMS) is arising, which is know by Time series Database (TSDB). This approach has attractive characteristics, such as being able to scale and use less structured data than traditional database systems.

The Time series Databases (TSDBs) are optimized for taking care of large amounts of time series data, i.e. each entry of the database is associated with a timestamp [47]. Accordingly, the data entries are indexed by their timestamps, which results in a set of advantages that matches the IoT scenario [48].

At first, this DBMS provides efficient sequential writes, as well as efficient queries for a temporal result set. In addition, it provides a fast removal of samples that are no longer relevant (by its timestamp) [47].

Considering a large scale IoT infrastructure containing heterogeneous data from an enormous number of sensors, the TSDB should be a column oriented database, where each column of the table would contain the received data from a sensor, indexed by its timestamp. In spite of all its advantages, there are few stable implementations available, as a result of this approach being recent. The most popular TSDB are InfluxDB and RiakTS.

The number one in the db engines ranking [2], InfluxDB [3], is an open source database designed for high-performance writes and compact disk storage. It also provides clustering capabilities built-in. In addition, it offers a simple documentation and setup.

RiakTS [4] is an enterprise NoSQL database specifically optimized to store, query and analyze time series data. It is easy to manage and scale, as well as to distribute data through a cluster. However, it

[2] http://db-engines.com/en/ranking/time+series+dbms
[3] https://influxdata.com/time-series-platform/influxdb/
[4] http://basho.com/products/riak-ts/

does not provide an easy to use documentation and the majority of its features are paid.

2.7 PRIVACY AND SECURITY

In the context of Information technology, security considerations and problems are not a new topic. With the emergence of the IoT concept, a wide range of new and unique security challenges will appear. These challenges represent a critical factor for allowing the widespread adoption of IoT infrastructures. For this purpose, addressing them must be a vital priority for the IoT success.

As a rule, without assurances related to confidentiality, authenticity and privacy the stakeholders are unlikely to approve IoT solutions on a large scale. Furthermore, users have to trust that IoT devices are secure, particularly as this technology becomes omnipresent into users daily lives.

A set of attacks can be done in order to compromise an IoT platform, namely:

Distributed Denial of Service(DDoS) IoT infrastructures may be vulnerable to a wide range of DDoS attacks [49]. Besides traditional DDoS attacks, the WSN can also be targeted;

Physical damage In the context of the IoT, things may be accessible to anyone. In consequence, the attacker can simply target the `"thing"` hardware;

Eavesdropping. In a passive way, an attacker may intercept the communications, in order to retrieve data from the communication flow;

Controlling In an active way, an attacker may try to get control over an IoT device. Consequently, the stored data could be accessed compromising data privacy, as well as modified in order to cause damage to its owner.

Thus, IoT and WSNs are at a considerable distance from being secure. In essence, devices and services can be potential entry points for exploiting security vulnerabilities, which will expose user's data. Moreover, the interconnected nature of IoT means that a compromised device, which is connected online, may perturb the security and resilience of the entire network. For instance, according to [50] it is possible to identify three main issues that require new approaches: data confidentiality, privacy and trust.

However, adding an extra layer for security purposes requires appropriate decisions, in order to ensure that the operation of the device stays adequate. Failing to do so has consequences, such as the incapacity to achieve real-time requirements, the decrease of features and the need of more energy than expected. Doubtless, security results in an overhead, which we may not be inclined to accept.

All things considered, the IoT should be secure and privacy-preserving by design. To put it differently, security should be taken into account at the time of the design of architectures and methods for IoT infrastructures. However, relevant problems to be solved in this context are associated with scalability and energy consumptions of existing infrastructures, which may not match the usual requirements of the IoT infrastructures.

CHAPTER 3

Distributed Systems

All things considered regarding IoT horizontal solutions described in Section 2.5, it is important to analyze in detail the advantages and disadvantages of the described architectures for an IoT infrastructure. Therefore, in this chapter is presented a comparison of different categories of Distributed Systems' Architectures, which may be implemented in an IoT infrastructure. Taking into account decentralized architectures, several implementations of a DHT are also presented. Moreover, this chapter also presents the reader with a brief overview of the state of the art for Security in Distributed Systems. At last, some deployments, which were built on top of decentralized infrastructures, are briefly described.

3.1 ARCHITECTURES

There are several architectures, which may be used according to the specifications of the required system. They can be divided into two broad architectures - Centralized and Decentralized. It is important to analyze both regarding failure tolerance, management cost, performance and scalability, in order to verify which type matches the requirements of an IoT infrastructure. Therefore, both architecture types are described as follows.

3.1.1 CENTRALIZED ARCHITECTURES

The traditional centralized architecture is known as the Client-Server architecture, which has one server that provides a service and many clients that communicate with the server, in order to consume the provided service. The main disadvantages of this architecture are the inherent difficulty to system's scalability, server's susceptibility to congestion, system's availability and system's high cost of building and managing [51], [52]. An example of this architecture is illustrated in Figure 3.1.

As a general rule, in spite of a resource pretended by a client may be replicated across the web, the client will request this resource to a particular server located in a specific location of the network. However, if the client looked up the network in order to find the resource, as well as got it from a closer

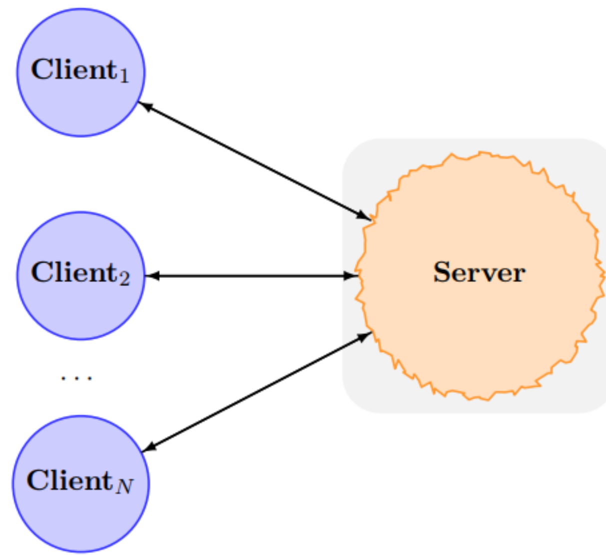


Figure 3.1: Wireless Sensor Network.

location, it will eventually result in a smaller consumption of bandwidth, as well as a lower response time.

For example, considering a room where a person uploads a video to a common video sharing platform and asks the other people in the room to watch the uploaded video. In consequence, a huge amount of bandwidth will be consumed, because all the people in the room will need to communicate and receive the video through the platform's data center. However, if the people in the room get the video from each other, the bandwidth consumed would be incredibly smaller.

All things considered, the properties of a centralized architecture do not match the needs of a global IoT Infrastructure. Particularly, the lack of scalability and performance, as well as its single point of failure. Therefore, the proposed system could be structured in a different way, in order to provide lower response times, as well as lower consumptions of bandwidth.

3.1.2 DECENTRALIZED ARCHITECTURES

Due to the disadvantages of centralized architectures, for some systems other types of architectures are used, known as Peer-to-Peer (P2P) architectures.

P2P systems and applications are distributed systems without any centralized control or hierarchical organization. Thus, P2P computing is a networking paradigm that provides high scalability by exploiting the resources of the participants (including computation, storage and bandwidth). In addition, it guarantees the autonomy of the system and has a low cost of ownership. Due to these desirable properties, P2P has been acclaimed as an encouraging technology that will reconstruct the architecture of distributed computing [51], [52].

A P2P network is usually composed by a set of equally privileged participants in the network. These participants are known as peers, which are both suppliers and consumers of resources, in contrast to the traditional client-server model in which the server supplies resources that are consumed by the clients. That network may be designated as an overlay network, which is a computer network that is built on top of another network. For example, distributed systems such as P2P networks are overlay

networks because their nodes run on top of the Internet [9]. An example of an overlay network is depicted in Figure 3.2.

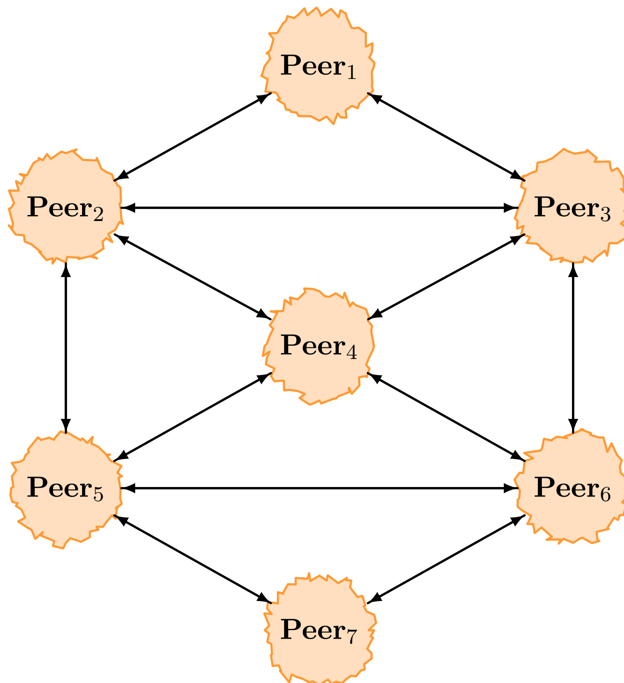


Figure 3.2: Peer-2-Peer Model

In the presented overlay network, there is no central entity. Each peer has a partial view of the P2P network and offers data or services that may be relevant to other peers. The biggest challenge of this architecture is to find which peers provide the desired resources quickly. Comparing with the previous analyzed architecture, this one removes the single point of failure and enjoys high performance and scalability [9], which are important advantages for the vision of the proposed infrastructure.

Decentralized P2P architectures may have a different logical network topology. It can be categorized as structured and unstructured overlays and their difference consists of how queries are being forwarded to other nodes.

In an unstructured P2P topology, there is no straight mapping between identifiers of objects and peers. Therefore, each peer is responsible for its own data, and keeps a list of neighbors that it may forward queries to. In this type of topology, locating specific data is complex, since it is difficult to precisely predict which peers have the data that is being queried. Consequently, there is no guarantee on the completeness of answers (unless all the network is searched), as well as no prediction on the response time of the query. The best known unstructured P2P systems are FreeNet[53] and the original version of Gnutella[54].

On the other hand, in a structured P2P topology, data location is controlled by certain strategies. In order to provide a mapping between data and peers, the strategy generally used is a distributed hash table (DHT), which is described in the following subsection.

3.2 Distributed Hash Table (DHT)

A DHT is a class of a decentralized distributed system that has a look up feature identical to a hash table: (key, value) pairs are stored in a DHT and any node can easily get the value associated with its key. Continual node arrivals, departures, and failures are easy to handle, as long as the responsibility for maintaining the mapping from keys to values is being distributed among the nodes [9], [55].

The structured P2P systems provide a guarantee (precise or probabilistic) on query cost, which makes possible that a request can be routed to a peer quickly and accurately. However, since the location of data is highly controlled, maintaining the structured topology is expensive, especially in a dynamic network environment, where peers may join and leave the network at will [9]. Chord, Kademlia, Pastry, Tapestry and Content Addressable Network (CAN) are some well-known solutions used to build such structured overlays.

3.2.1 CHORD

Chord [56] is a protocol for structured P2P overlays, which uses Distributed Hash Tables (DHTs) to locate specific data items, i.e. keys, in the nodes of a P2P network. Mainly, Chord uses consistent hashing for binding keys to nodes, in order to balance related workload of nodes, since it leads to nodes keeping approximately the same number of keys.

As shown in Figure 3.3, Chord's topology is organized into a virtual ring where each node always has an higher identifier than its predecessor and knows who is its successor (next hop). The consistent hashing assigns a key to each node identifier (node's IP address hash), whereas a key is also generated for every data item available on the overlay (data's hash).

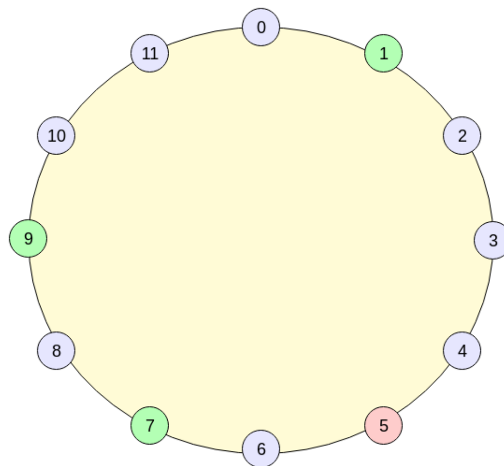


Figure 3.3: Chord Topology organization.

Taking into account that the length of the identifier is `m`, then the identifiers are ordered in a Chord ring of module `2m` size. When a new data item is on the overlay network and has a key `k`, its key is associated to the node in the Chord ring, whose identifier is either `k` or is the first to come in a clockwise order after `k`. Meanwhile, assuming that a new peer joins the overlay, its IP is hashed and its position in the ring is determined based on the produced key. This process may require a rearrangement of the key-node bindings [56].

Particularly, some keys that were previously handled by the successor of the new node in the ring will now need to be assigned to the new node itself, based on the previous principle of key-node binding. Furthermore, when a node leaves the overlay, all the keys that were assigned to it must be reassigned to its successor in the ring. For instance, considering Figure 3.3 in a given time and considering that the nodes whose hash is 1, 7 and 9 are part of the overlay network. Consequently, node 7 is responsible for keys whose hash is between 2 and 7. However, if a new peer whose hash is 5 joins the overlay, node 5 will be take responsibility for keys whose hash is between 2 and 5.

For implementing the consistent hashing in a distributed environment, each node only needs to be aware of its successor in the ring. Nonetheless, this solution may be inefficient for the reason that it may require to pass through all the nodes to find the appropriate mapping. To accelerate this process, Chord maintains additional routing information. Therefore, every node maintains a routing table called Finger Table, which is used to look up data items on the Chord overlay. The Finger Table is constructed so that for a node `n` its `ith` entry has a pointer to the successor of node $n + 2^{i-1}$ in the Chord ring, where $1 \leq i < m$ and m is the maximum size of the table. This allows Finger Tables to have information about the nodes in its neighborhood, as well as about a few remote nodes [56], [57].

Undoubtedly, the consistent modifications in the topology, caused by nodes joining and leaving the overlay, influences the Finger Tables, which should be updated constantly, in order to ensure correct lookups.

Finally, it is clear that nodes joining and leaving the overlay only influences their neighborhood and not the whole overlay. In consequence, a lookup procedure in a stable N-node Chord overlay requires O(log N) messages [56]. However, if the overlay topology is always changing as nodes join and leave the network, the performance of the system decreases as a result of its high cost of maintenance. In this case, an unstable overlay can have a performance of $(\log N)^2$ [56].

3.2.2 KADEMLIA

Kademlia [42] is one of the most famous structured, fully decentralized P2P overlay. Currently, it is used in several public networks, such as Kad Network and BitTorrent. It uses a consistent hash method to bind identifiers to keys and nodes, as well as a XOR-based metric to compute the distance between identifiers. Furthermore, Kademlia was designed based on observations regarding existing P2P overlays' activity. One of the main conclusions of these observations was that the longer a node remains connected, greater is the probability to stay connected for another hour. Accordingly, in order to ensure an higher probability in the overlay's connectivity, Kademlia promotes the use of "old" nodes as neighbors [42].

Regarding the topology of this protocol, nodes are considered leaves in a binary tree. Thus, each node's position consists of the shortest unique prefix of its identifier. Figure 3.4 illustrates a simple example, where a node with ID 10110 is represented by the node 10 of the tree. This is possible if the 10 prefix is unique for all the nodes that are part of the overlay.

An identifier from a 160-bit name-space is assigned to nodes and resources. Each node maintains several lists of nodes, which have a distance between 2^i and 2^{i+1} from itself, where $0 \leq i < 160$. These lists are called k-buckets and their size can grow up to k (typically 20). A k-bucket is a list of nodes, that are at a distance k from the node. The distance between two identifiers is the integer value obtained from their XOR [42].

The K-buckets of a node are updated when messages are received from other nodes: in case that

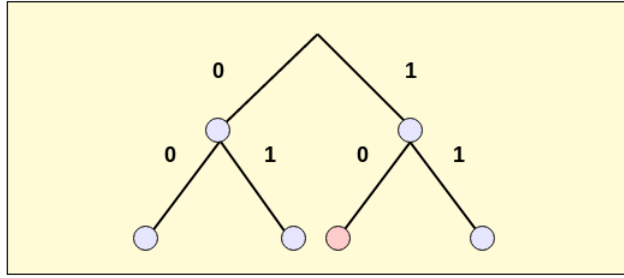


Figure 3.4: Kademlia Topology organization.

the message sender is known by the receiver, the k-bucket where the sender is currently present is updated, i.e. the sender ID is moved to the end of the list; whereas in case that the message sender is unknown, the sender ID is added to the end of the correspondent k-bucket, according to the distance between the sender and the receiver, and assuming that the k-bucket is not full. However, when the k-bucket is full, the least seen node is pinged to check its status. If it answers, the new node information is discarded, otherwise the least seen node is discarded and the new node is added to end of the list [42], [58], [59].

When a new node intends to join the overlay network, it needs to communicate with a bootstrap node, which is any node currently in the overlay network that provides initial configuration information. Therefore, the new node is added to the appropriate k-bucket of the bootstrap node and performs a lookup procedure for its own identifier, in order to get information on other nodes and populate its k-buckets accordingly. In this way, k-buckets promotes "old" nodes as neighbors in the overlay, with a view to ensure an higher availability and stability.

At the time that a certain node identifier is requested, it is located the k closest nodes for a given identifier. Firstly, the initiator of the lookup verifies in its own k-buckets for the closest α nodes (typical value for α is 3). Then, it sends requests to these α nodes in parallel and they return nodes that are close to the requested identifier (use of XOR operation), according to the entries in their k-buckets. Consequently, nodes are recursively contacted in order to collect further nodes that are closer to the requested one. This process continues until k nodes have been collected by the initiator [42], [58], [59].

With regard to the lookup efficiency, Kademlia allows redundancy and caching (keys can be stored to up to `k` different nodes in the overlay), as well as proper operation even under dynamic conditions, such as node failures. Kademlia has a performance of $\log N$, where `N` is the number of nodes in the network. It is an efficient algorithm, as a result of sending parallel requests and maintaining the network topology through the received messages [42].

3.2.3 PASTRY

Pastry is a scalable P2P overlay and routing network for P2P applications based on a DHT. In terms of functionality, it is similar to Chord, but it differs mostly in how it handles neighbor sets and routing. Each node in the pastry network has a unique identifier of 128 bits. When a node receives a request message with a key, it can efficiently route the request message to the node, whose unique identifier is closer to the requested key [60].

Nodes are organized into a virtual ring, ordered by their identifiers, like Chord. However, in Pastry,

each node has three data structures to store, namely a routing table, a neighborhood set and a leaf set.

The routing table has $\log_B N$ rows, $B-1$ columns and it contains one entry for each address block. To build the address blocks, the 128 bits key is divided up into digits with B bits length. This divides the nodes into different levels, with level 0 representing a zero-digit common prefix between two nodes, level 1 a one-digit common prefix, and so on. The neighborhood list of a node represents the closest peers, considering a routing metric (usually the size of the list is $2 * B$). Although it is not used directly in the routing algorithm, the neighborhood list is important to maintain the locality principles in the routing table [60]–[62].

When a new node wants to join the overlay network, it needs to communicate with a node currently in the network, which will provide a list of the closer nodes. Randomly, the new node contacts one node of the provided list and sends its join request, with its unique identifier as the key. As soon as the bootstrap node, as well as the nodes in the routing path, receive a join request, the bootstrap node sends a reply composed by its routing table content. Finally, the new node builds its routing table, based on the received information, and notifies all the interested nodes of its arrival.

On the other hand, if a node leaves the overlay, its neighbors will contact the remaining members of their neighborhood sets and it is found an alternative node to replace the departed one.

The data items have a key (unique identifier) from the same identifier space as the nodes. These keys from data items are handled by nodes whose identifiers are numerically close to each other. Moreover, when a certain data item key is requested, nodes forward lookup queries to nodes whose identifiers have a prefix with at least 1 bit similar [60]. Thus, the node that is responsible for the data item is located in a progressive manner.

Finally, a key can be located in up to $\log_B N$ overlay hops (number of rows in the routing table). It has great results in scalability (experiments show that it works efficiently for networks of even a hundred thousand nodes) and fault tolerance (it is necessary the departure of at least $L/2$ nodes from the overlay to cause failure problems to the overlay) [60].

3.2.4 TAPESTRY

Tapestry is a decentralized, scalable, fault-tolerant and location-aware routing infrastructure for distributed applications. Its main design goal consists of scalability and fault tolerance under dynamic network conditions. Tapestry is very similar to Pastry, essentially in terms of their common routing infrastructure. However, it has some differences, namely it replicates data objects for creating redundancy and has a different definition of networking locality [63].

Nodes and objects have a unique identifier of 160-bit. These unique identifiers are determined from the same 160-bit identifier space using a uniform distribution.

In this overlay topology, nodes have the minimum necessary information about the network. Every node has node identifiers and IP addresses only of its direct neighbors, i.e. neighbors with common prefixes, in the form of neighbor maps. Each row of a neighbor map contains information about the neighbors that have the same prefix as the node. This prefix has a length equal to the row's level in the map [62]. Moreover, it is used a redundant routing, which consists of adding two backup nodes in each row of the map, in order to ensure the connectivity between nodes in dynamic network environments [63].

When a new node gets in the overlay, it finds out the length of the longest prefix of the unique identifier it shares. Then it sends a multicast message to all the existing nodes sharing the same prefix,

and these nodes add the new node to their routing tables. After this, the new node performs an iterative nearest neighbor search to fill all levels in its routing table. Moreover, the new node based on its identifier might need to become responsible for some object references. When a node intends to leave the overlay, it sends a broadcast message informing about its departure and transmits a replacement node for each level in the routing tables of the other nodes. Object references at the leaving node are redistributed from redundant copies, to another node. An unexpected node failure is handled through redundancy in the network and backup object references to restore damaged references.

To add new objects to the overlay, the owner of the object sends a publish object message and the node, whose identifier is closer to the object's identifier, will be responsible for the object reference. This node, will keep information about the original owner of the object, in order to be able to return it to any requesting nodes.

When a concrete resource is requested, its unique identifier is used, in order to find a node that has information about the resource's owner. Therefore, resource discovery is made by progressively contacting a node with a closer unique identifier. Each node along the path checks the mapping and redirects the request properly.

Finally, routing in Tapestry occurs in at most $\log_B N$ hops (where N is the size of the identifier space). According to some experiments [63], Tapestry is an efficient solution, even under dynamic network environments. Its routing has a good performance, as a result of the stretch factor for locating resources remains low.

3.2.5 Content Addressable Network (CAN)

CAN is a distributed, decentralized P2P infrastructure system that is based on the DHT concept. It organizes the overlay nodes in a d-dimensional coordinate space, where `d` is a parameter of the CAN protocol. This coordinate space is used to store key/value pairs, however it is completely independent of the physical location of the nodes. All the coordinate space is dynamically distributed among all the nodes in the overlay, in order to guarantee that every node has at least one distinct zone of the coordinate space [64]. Therefore, Figure 3.5 illustrates a 2-dimensional coordinate space with 5 nodes.

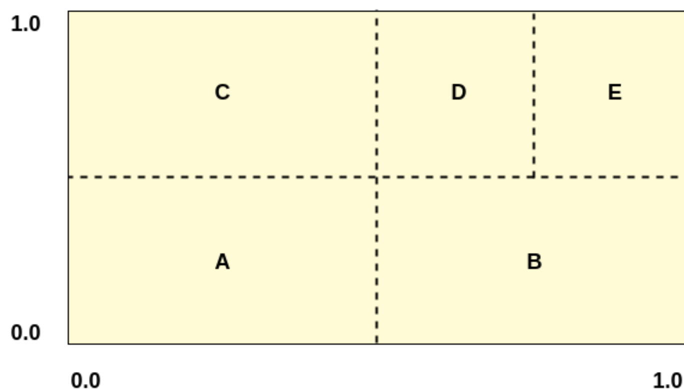


Figure 3.5: CAN 2-d topology organization.

Every node maintains a routing table with the IP address and virtual coordinate zone of each of its neighbors. A joining node has to find a node already in the overlay (bootstrap node), in order to get in the network. After contacting a bootstrap node, the joining node may join the overlay and

discover further nodes. Then, the new node chooses a random point of the coordinate space and sends a JOIN message to that point. The existing node, which is responsible for that point, divides its area in two and assigns one part to the new node. Finally, messages are sent in order to update the routing tables of the neighbors. On the other side, if a node leaves the network, its coordinate space area has to be assigned to one of its neighbors. For this, each node is continuously monitoring their neighbors, to update their routing information [64]–[66].

When a new resource is added to the overlay, a hash function is applied to it, in order to map its key to a certain point on the CAN space. Then, the node responsible for the area, which that point belongs to, becomes in charge of that key/value pair. Moreover, if a resource is requested, it is applied a hash function to the resource key, in order to get the location of the point and consequently, the node that is responsible for that key. After this, it is sent a request message to the discovered node, using a greedy forwarding routing mechanism, i.e. each node forwards this request to the closest neighbor in the coordinate space that it knows [64].

Assuming that every node of the overlay network maintains a list of $2d$ neighbors and that the d-dimensional space is divided in n equal zones, accordingly to [65], the average routing path is computed to be equal to $\frac{d}{4}n^{\frac{1}{d}}$. A CAN overlay is reliable because its construction provides alternative routing paths to connect nodes [64]. Nonetheless, it is susceptible to failure in case of area partitions and it does not guarantee load balancing, considering that is possible to have areas that contain much larger number of key/value pairs than others.

3.2.6 COMPARATIVE SUMMARY

After presenting all the previous implementations of a DHT, it is important to compare them according to different metrics.

Load Balancing is an important feature, since it provides scalability, availability and reduces the probability of bottleneck occurrences in the overlay. A tremendous advantage of most DHTs results in the use of consistent hashing algorithms to assign resources/services to certain nodes. Therefore, these algorithms provide a uniform distribution of resources through the nodes of the overlay, which promotes a load balancing for handling lookup and routing requests. Only CAN does not include Load Balancing in its features.

However, when changes in the overlay occur, hashing algorithms face a challenge, which consists of the need to reorganize resources. This need is expensive for processing and communication overhead, particularly in Chord and CAN, which have inflexible structures. In some cases, such as Kademlia, the notion of redundancy represents an enhancement in the performance of the hashing algorithms and lookup operations, as a consequence of having resources replicated through the overlay.

In the context of an overlay set up, nodes and resources have to be arranged in a topology by creating or updating existing connections of themselves, as well as, of other neighboring nodes. Consequently, when a new node joins the overlay or an existing node departs, some nodes of the overlay may be affected. Therefore, P2P Overlays' performance is affected in dynamic networks, where churn rates are quite high. This aspect was considered in some cases, where some approaches were adopted, in order to support the robustness and resilience of the overlay. This is the case of Kademlia, Pastry and Tapestry approaches [67].

P2P overlays based on DHT solutions have a great performance in locating specific resources (exact matching, as a result of the nature of consistent matching, which implies that identifiers are

uniquely mapped). Therefore, these solutions are not usually appropriate for range queries. Only Kademlia provides a concept of range queries, through the asynchronous lookup messages.

A comparison summary of the studied implementations of a DHT is presented in Tables 3.1, 3.2 and 3.3.

P2P	Architecture	Queries
Chord	Circular and uni-directional node identifier space	Regular Queries
Kademlia	XOR metric for computing distance	Range Queries
Pastry	Hypercube	Regular Queries
Tapestry	Hypercube	Regular Queries
CAN	Multidimensional ID coordinate space	Regular Queries

Table 3.1: DHT comparison summary (1).

P2P	Churn	Redundancy
Chord	Performance decrease and high overhead	N/A
Kademlia	Good support thanks to redundancy	Replication and caching
Pastry	Robust for a few changes	N/A
Tapestry	Replication and routing redundancy	Resource replication
CAN	Depends of the node degree	Routing redundancy

Table 3.2: DHT comparison summary (2).

P2P	Security	Load Balancing
Chord	N/A	Consistent hashing
Kademlia	Older nodes more trusted	Consistent hashing
Pastry	N/A	Consistent hashing
Tapestry	N/A	Consistent hashing
CAN	N/A	N/A

Table 3.3: DHT comparison summary (3).

Typically, most structured P2P overlays have a small diameter, which allows an efficient resource discovery [68]. Therefore, the greater part of the DHT solutions studied in this section have a logarithmic performance for lookup requests, as well as routing.

Chord and Kademlia have a $\log N$ lookup performance, whereas Pastry and Tapestry have a $\log_B N$ performance, which is associated with the base B of the defined identifier space. In contrast, CAN's performance does not depend on the number of nodes, but depends on the number of neighbors of a node (d) and on the total number of zones in the d-dimensional zones (n).

On the whole, a performance comparison of the implementations of a DHT is presented in Table 3.4. For a better understanding, β represents the base of a key identifier, **d** symbolizes the number of dimensions considered and **n** is the number of nodes in the overlay.

P2P	Lookup	Table Size	Join/Leave
Chord	O(log N)	log N	$(logN)^2$
Kademlia	O(log N)	log N	log N
Pastry	O($\log_\beta N$)	$2\beta \log_\beta N$	$\log_\beta N$
Tapestry	O($\log_\beta N$)	$\log_\beta N$	$\log_\beta N$
CAN	$\frac{d}{4}n^{\frac{1}{d}}$	2d	2d

Table 3.4: DHT performance comparison.

In the context of network security, all the presented solutions may suffer from security threats. The biggest threat is the placement of resources exclusively according to hashing algorithms, which can result in hosting resources on non-trusted nodes.

All things considered, the properties inherent to a structured P2P system provide great value for a new type of IoT Infrastructure, which is built on top of a decentralized paradigm. Namely efficient data queries, as well as the scalability and failure tolerance.

3.3 SECURITY

There is a wide range of security threats applicable to distributed environments that have to be considered, in order to implement a secure system. Even in a closed distributed environment, it is unrealistic to take for granted that none of the participating peers have been compromised.

Aiming to describe some attacks regarding distributed systems, we will take into account a scenario where some entities are part of a P2P network and provide a set of services to the overlay. When an entity is requested to provide one of its services, it can efficiently provide the demanded service or, on the other hand, it can provide a malicious service. It is possible to find in a P2P network individual malicious peers, as well as malicious groups of peers. Such peers, may corrupt messages and routing information, as well as assume the identity of other nodes, in order to provide corrupt services.

Considering the IoT paradigm described in Section 2.1, millions of internet-connected devices are predicted to be deployed, some of them from critical systems. Those systems need to keep their data properly safe. However, security risks grow with the number of deployed devices.

Therefore, considering the expected massive deployment of devices, it is necessary to have an infrastructure properly prepared. Currently, the solution for security in Decentralized Systems is know as Public Key Infrastructure (PKI).

PKI is a well-established standard, which may provide authentication, data access control, data confidentiality and information integrity [69]. It is widely used as a standard for Internet Security. According to a IEEE article [70], `"When you are looking at authenticating devices, the only real standards at the moment that offer any real interoperability tend to the PKI"`. It provides the authentication and encryption components needed by a distributed platform for data security, making it a proven solution. Consequently, it could be the groundwork for the deployment of a secure IoT Infrastructure.

In this standard, each infrastructure's entity has an associated public key certificate. This association is established by a registration process, as well as the issuance of certificates for identity

validation. There are three main approaches for certification, namely Hierarchical Certification, Web of Trust and Blockchain-based PKI. Following are described these three approaches.

3.3.1 HIERARCHICAL CERTIFICATION

In the context of this approach, there is a set of centralized servers, which represents the Certification Authority (CA) root [71]. It contains an RSA key pair, as well as a self-signed certificate. In addition, it handles the issuance and revocation of certificates.

The set of centralized servers listens requests from entities, so that it issue certificates for them. Accordingly, a request contains the entity's public key, which will be signed by the CA private key. The certificate binds a public key with a specified entity and is valid for a previously defined time, according to the CA policies. Thus, an entity that trusts a CA will trust an entity which has a certificate signed by it [72].

The hierarchical certification may have `n` levels [71]. In this case, certificates are issued and signed by the private key of an entity that is higher in the hierarchy. Therefore, certificates are validated using a chain of trust, which validates all the certification hierarchy until reaching a trustworthy entity.

In spite of the possible redundancy of the CA servers, its main disadvantage consists of not being completely tolerate to failures. On the other hand, it provides lower response times and does not need a considerable number of messages to issue and validate certificates. Finally, it is the most used approach for handling the certification in PKIs.

3.3.2 Web of Trust (WoT)

Considering the WoT [73], no central server is necessary for taking care of the certification. This approach does not have a hierarchical notion, as a result of the certification role being shared among all the entities that compose the infrastructure.

Taking into account the certification process, each peer that composes the WoT has a self-signed certificate, as well as information regarding other peers' certificates [73]. Therefore, several peers will have to participate in the validation of a peer's certificate process. An entity of the network establishes trust in others by checking that those others are trusted by at least one already-trusted entity. For instance, if an entity `A` trusts an entity `B` and the entity `B` trusts and entity `C`, `A` trusts `C` consequently.

This approach may be characterized as a flexible and generic solution, thanks to its distributed nature. However, this nature results in an infrastructure difficult to manage.

The Pretty Good Privacy (PGP) is the most popular implementation of this approach [73]. It was created by Zimmermann[74] and consists of a public key cryptography designed for being used in the e-mail. Each entity has a list of the public keys of the entities it communicates with. This list is signed by the entity's private key, so that it can not be modified by an attacker without leaving behind an evidence.

The main advantage of this approach consists of the absence of a single point of failure. However, the response time for the validation of certificates increases significantly compared to the previous approach, as a result of the necessity to exchange a considerable number of messages for the consensus algorithm. In addition, it is difficult for an entity to join the network without being previously trusted.

3.3.3 BLOCKCHAIN-BASED PKI

The blockchain was presented in 2008 as the groundwork technology of the Bitcoin cryptocurrency [75]. In this approach, as in the WoT, there is not any centralized set of servers for handling the certification. However, in the context of the Blockchain-based PKI, each entity has part of a distributed database, containing all the transactions previously performed. This distributed database has a previously defined replication factor for reliability [76].

Typical blockchain solutions provide appealing security characteristics. However, they are not scalable and suitable enough for many PKI infrastructures [77]. In addition, they do not provide privacy as a result of all the actions being recorded in the database.

This approach is mainly used in cryptocurrencies, being Bitcoin the most popular example of a blockchain. Shortly, Bitcoin [75] consists of a purely P2P electronic coin, which can be used for online payments without going through a centralized financial institution. The network keeps the transactions timestamps by hashing them into an ongoing chain of hash-based proof-of-work. This proof-of-work represents a record that cannot be modified without recomputing itself. Peers may leave and join the network at will, accepting the longest proof-of-work chain, which represents all the transactions history. Therefore, as long as the greater part of the computing power is not controlled by attackers, the blockchain will not be compromised.

As in the previous approach, the main advantage of this procedure is the absence of a single point of failure. Moreover, it provides an easy entrance in the blockchain for new entities, contrarily to the WoT. On the other hand, thanks to its complex algorithms for transaction validations, the response time of certificate validation is even higher than the WoT approach. In addition, any implementation of this protocol was tested in a large scale [45].

3.4 PROMISING DECENTRALIZED INFRASTRUCTURES

Taking into account the modern requirements, the current platforms have to change their paradigm, as a result of the problems stated in the beginning of this section, regarding centralized architectures. With this in mind, modern infrastructures should be built on top of a P2P architecture, in order to provide a more efficient service to its clients.

Accordingly, some infrastructures are being built according to this view. Following are described some examples of promising decentralized infrastructures.

3.4.1 ETHEREUM

The Ethereum[1] [78], [79] project consists of a platform that is built for developers and application users. It is a blockchain platform that allows developers to deploy distributed applications through their platform. These applications run with no possibility of downtime, censorship, fraud or third party interferences.

This platform monitors the state of every Ethereum accounts, as well as all the blockchain transitions, in order to validate the exchanged information among accounts. In other words, it may be characterized as a decentralized and cryptographically secure, transaction-based state machine [78].

[1] https://www.ethereum.org

In this context, each user of a deployed application also provides data of the deployed applications to other users. In exchange, the consumers receive ether (cryptocurrency) for paying the hardware, electricity and processing power that are using for maintaining the applications running. On the other hand, the developers who intend to deploy their applications have to buy ether to have it deployed in the platform.

Despite all its appealing characteristics, scalability remains an endless concern [78]. As a consequence of its state transition paradigm, it is difficult to parallelize the validation of transactions.

Currently, there are already several applications running on top of the Ethereum platform. An example is the Eth-tweet[2], which consists of a decentralized microblogging service similar to Twitter. As a consequence, there is no central entity who controls what is being published.

3.4.2 InterPlanetary File System (IPFS)

The IPFS [80] consists of a hypermedia distribution protocol that intends to interconnect all the file systems of the peers, which are part of a P2P network, building a distributed file system. It intends to revolutionize the web, so that it becomes faster, more open and safer.

Considering a traditional file sharing platform built on top of HTTP, a machine downloads a file directly from another machine. Contrarily, IPFS allows a machine to download a file through the transmission of data chunks from multiple machines dispersed over the network. Therefore, IPFS enables the distribution of high quantities of data efficiently thanks to its P2P approach.

The increasing centralization of data may lead to threats regarding cyber attacks and natural disasters. IPFS vision aims to prevent this problem dispersing the data through the network. Moreover, it allows the persistence availability with or without an Internet connection

In essence, the IPFS [80] may be described as a combination of Web, with BitTorrent and Git. When a file is inserted in the IPFS, it is identified by the hash of its content. Therefore, IPFS is able to guarantee the nonexistence of duplicated files, as well as a version history tracking for every file, similar to Git.

Taking into account data storage and routing, each node of the network is responsible for storing the content it is interested in, as well as indexing information. This information is used for discovering a set of peers which are storing a certain file, through its unique hash. It is also important pointing out that IPFS uses a decentralized naming system that maps human-readable file names with its hash identifier, in order to enable queries.

With a view to make this possible, IPFS combines a DHT, a content-addressed blocked storage and a keychain (concept similar to the web of trust).

3.4.3 zeronet

ZeroNet[3] is a decentralized web platform, which allows the deployment of open, free and uncensorable websites, using the Bitcoin cryptography, as well as the BitTorrent network.

[2] http://ethertweet.net/
[3] https://zeronet.io

When a visitor accesses a web site, its content is distributed using the BitTorrent network, instead of using a central server. Accordingly, the websites deployed using this platform are uncensored as a result of not being deployed on a specific location.

Every site deployed in this platform has a `content.json` file associated, which contains a signature generated using the site's private key. This signature is validated by the peers before starting to serve the web site contents to its visitors. This mechanism guarantees that only the site owner is able to modify the site.

In summary, this platform provides no hosting costs to websites, as well as no single point of failure and low response times. In addition, it matches real-time requirements of certain use cases and offers security, transparency and trust to the websites.

CHAPTER 4

Decentralized IoT Infrastructure

In this chapter, the motivation for decentralizing an IoT infrastructure using a DHT is described, followed by the identification of the requirements that such an infrastructure must comprise. Afterwards, the decisions and principles used for designing the intended infrastructure are presented, as well as an overview of the proposed architecture.

4.1 Problem Statement

In section 2.5 is stated that the current IoT infrastructures are built on top of horizontal solutions, which combine multiple WSNs and applications, in a single infrastructure. This single infrastructure usually consists of a centralized server, which receives WSNs' data from gateways. Accordingly, this distributed system has a centralized architecture, which is described in section 3.1.1. This type of architecture has disadvantages, which do not match the IoT vision described in section 2.1.

IoT infrastructures must be flexible and extensible enough to accommodate such a high level of diversity, concerning both the users and objects, as well as to support billions of smart devices producing massive amounts of data. Ideally, one could argue that infrastructures should not exist, as they contribute to the ever existing problem of data and management silos. The Internet grew from organic attachment of networks, end-devices and servers, and this proved to be a much successful approach, both by the amount of distribution it presents, and by the adaptation to new use cases and services. Moreover, common infrastructures store large amounts of data in a central location, which will have an ever increasing impact in case of a cyber attack, as more data is continuously being gathered and stored together.

Taking into account the objectives stated in section 1.1, the infrastructure proposed relies on a DHT to set up a decentralized infrastructure, as well as route messages among peers. It is a P2P overlay network, where IoT gateways (GW) are the network peers. This network is complemented by several WSNs, which will provide data to the infrastructure. In addition, a undefined number of applications may communicate with it, as depicted in the picture below.

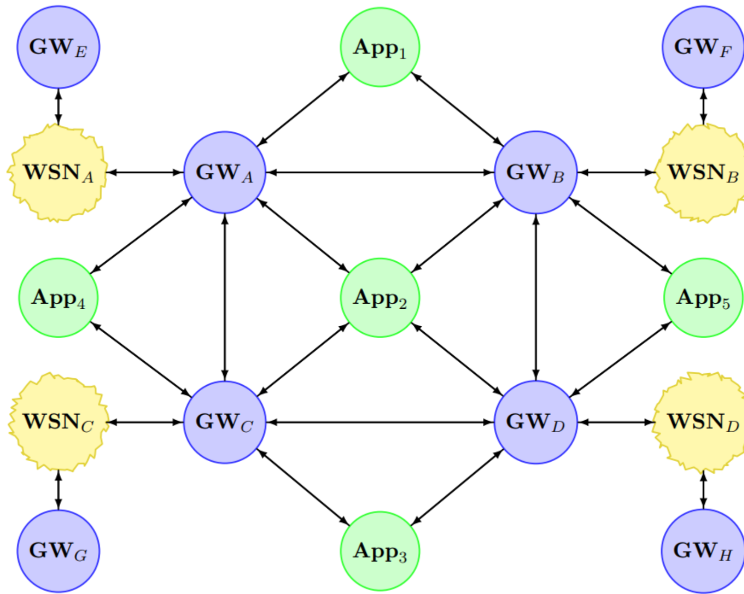


Figure 4.1: Network of the infrastructure.

With the proposed topology illustrated in Figure 4.1, it is possible to achieve the Internet level of scalability, once each new node joining the network shares its resources towards the overlay. Therefore, the topology consists of a community driven, decentralized network. Consequently, the system may provide different roles in the overlay, which will result in the share of computational power, storage capacity and network bandwidth, allowing the system to scale easily.

One of the biggest disputes that must be overcome in order to deploy IoT in a global scale is security. In section 2.7, the importance of having a secure IoT is described, as well as the impact of security mechanisms on system's performance. In the context of this dissertation, the decentralized and heterogeneous nature of the distributed approach increases the complexity of most security mechanisms, such as Access Control, Data Integrity, Identity Management, Network Security, Trust Management and Privacy [81].

In short, an efficient, scalable, secure and self-configuring decentralized infrastructure is proposed to integrate IoT platforms. This solution has a set of advantages, such as no single point of failure, data replication in different locations of the network and capacity to have different types of peers, each one with its roles, according to its hardware specifications.

4.2 FUNCTIONAL REQUIREMENTS

The proposed infrastructure may appeal to a wide range of entities. Firstly, telecom operators may improve their services, as well as create new business processes to provide secure and privacy-preserving IoT services to their clients. In addition, this solution provides a great opportunity for common corporations to enhance their quality of service, such as wine vendors, shipping enterprises, factories, among others. Moreover, public entities may also provide public data of certain cities, which can be used by developers, in order to improve the quality of life of those cities' residents. Finally, common people should also be allowed to make their things smarter, such as houses, by sending data to the

infrastructure.

Nevertheless, taking into consideration the community-driven approach of this infrastructure, all the described entities must provide resources to the infrastructure, so that it can continue its growth.

Briefly, sensors owned by the described entities send periodic data to the infrastructure. Afterwards, each entity, usually behind an application, may act with the infrastructure using different approaches. The use-case model illustrated in Figure 4.2 shows the possible interactions of some entities with the infrastructure. It is important to notice, that several other entities may also be presented in this diagram, according to the diverse use cases of the IoT.

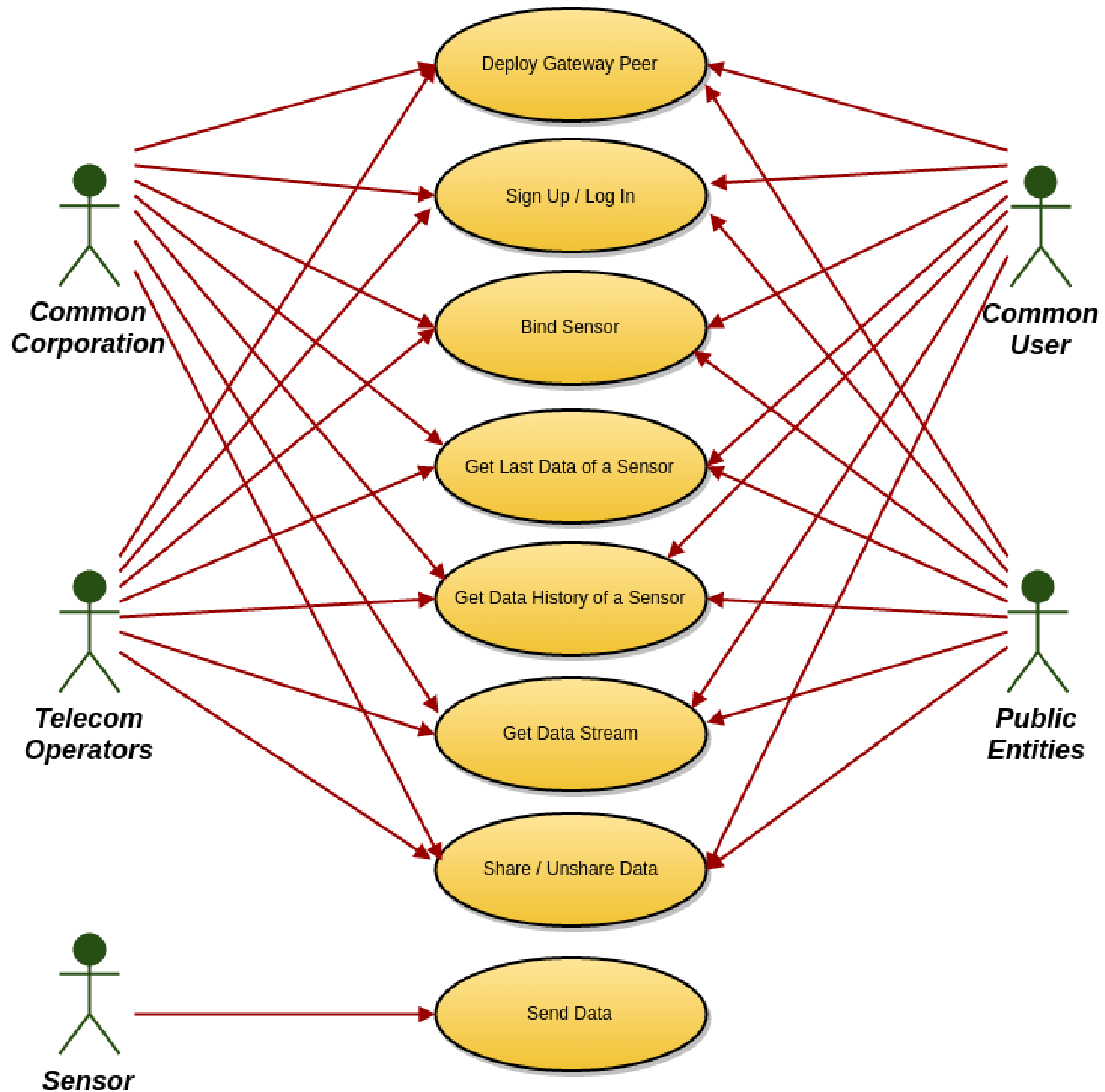


Figure 4.2: Use case Scenario.

The specified use cases are briefly described in Table 4.1.

Use Case	**Description**
Deploy gateway peer	Provide public gateways to the infrastructure, in order to enhance the infrastructure's performance and scalability, as well as enable its growth.
Sign Up	Sign up in the infrastructure, with a view to assign sensors to this account.
Log In	Authenticate in the infrastructure, aiming to interact with it.
Bind sensor	Assign a sensor to an entity.
Get last data of a sensor	Request the last received data of a sensor.
Get data history of a sensor	Ask the data history of a given sensor.
Get data stream	Request a data stream from a set of sensors.
Share data	Share data from a sensor with other entity.
Unshare data	Unshare data from a sensor with other entity.
Send data	Sensor send a new chunk of data to be persisted.

Table 4.1: Use cases description.

4.3 NON-FUNCTIONAL REQUIREMENTS

After presenting the scope of this solution, this sub-section intends to identify the main characteristics of a IoT infrastructure, pointing out their benefits and disadvantages towards its different architectures. Then, the non-functional requirements of this system are specified.

A generic IoT infrastructure must comprise the following characteristics [81]:

1. **Data Management:** As data is being generated, it is necessary to identify where to store it, as well as how it should be accessed;
2. **Fault Tolerance:** In the event of failures, the infrastructure must continue its proper behavior;
3. **Interoperability:** Within the IoT paradigm, the devices are expected to be heterogeneous;
4. **Performance:** The infrastructure needs to provide a minimum level of performance, so that it does not compromise the requirements of a business process;
5. **Reliability:** It must assure a minimum level of up-time to match the IoT requirements;
6. **Scalability:** The infrastructure must scale, in order to allow the number of connected devices, as well as the amount of data generated, to grow exponentially;
7. **Security:** There is a wide range of issues that must be addressed, in order to guarantee a secure and trusted IoT.

In a centralized IoT infrastructure, data management is simple, since all the infrastructure data is stored in the same point of the network. Consequently, the data access is easy to manage. Additionally, it is easy to guarantee the interoperability of the infrastructure, because all sources of data will

communicate with a single API, provided by the centralized system. Considering performance and reliability, the centralized infrastructures are usually built on top of a cloud platform, which allows a very good service up-time, as well as an acceptable performance. Therefore, the infrastructure scalability is limited by the available cloud resources, which are proportional to the available budget. Regarding Security, a single vulnerability may put the whole infrastructure in risk. In other words, the central entity consists in a single point of failure, as well as a possible security vulnerability. Finally, this type of architecture may provide a simple publish-subscriber system for real-time data acquisition.

On the other hand, in a decentralized IoT infrastructure, data management is more complex, as the data will be distributed through the network. As a result, the data access is more difficult to manage, but it can easily be replicated through the network. Then, due to the communications between heterogeneous devices, the interoperability is complex to achieve and should be standardized. Considering performance and reliability, the infrastructure up-time and efficiency depends on the number of peers maintaining the infrastructure, but a failure of a peer will not affect the whole system. In terms of scalability, it is considerably improved through the distribution of computational power, as well as data storage through the peers. Moreover, the information will be stored in different locations, which results in a smaller impact in case of a successful attack. However, the number of possible points of attack increases significantly. Finally, providing a global publish-subscriber mechanism is considered a complex task, as a result of the need for synchronizing peers efficiently.

Therefore, it is easy to understand that a centralized infrastructure is easier to implement and consequently, it was the first approach to get in the market. In spite of the numerous problems that need to be solved, a distributed approach has benefits that are important to accomplish the IoT potential, namely fault-tolerance, scalability and security.

Summing up, the following list of non-functional requirements has to be accomplished, in order to have a secure decentralized IoT infrastructure:

1. **Adaptability** - concerning an infrastructure composed by heterogeneous nodes, the infrastructure should accept nodes with different constraints, as well as take advantage of the main hardware characteristics of each type of node.
2. **Data Access Control** - with regard to protect data from attackers, an access control mechanism should be used, in order to determine if a certain entity is allowed to access the requested data;
3. **Data Anonymity** - the infrastructure must store the IoT data, considering that if an attacker accesses this data, he could not link the stolen data to its owner;
4. **Data Integrity** - the data stored in the infrastructure, as well as the data transmitted through it, should maintain its accuracy and consistency over the time;
5. **Data Replication** - regarding possible failures of nodes in the network, the infrastructure should have its data replicated through the network, in order to guarantee no loss of data;
6. **Fault Tolerance** - in spite of any failure in the infrastructure, it must be always available for receiving and storing data, as well as answering data requests;
7. **Identity Management** - the infrastructure must manage the identities of data consumers, assuring the entities authenticity and privacy;
8. **Interoperability** - with regard for the heterogeneous environment of IoT, as well as the lack of standards, the infrastructure must provide interface communications for a wide range of protocols;

9. **Performance -** the proposed infrastructure may comprise short response times, as well as high throughputs, in order to not compromise its expected behavior;
10. **Real-time Data -** the infrastructure must provide a secure data stream, as a result of the real-time requirements of many use cases;
11. **Scalability -** concerning the IoT vision, the proposed infrastructure must be prepared to an exponential growth of devices, as well as generated data;
12. **Secure Data Storage -** the collected data must be encrypted, in order to be useless in case of a successful attack to the infrastructure's persistence;
13. **Secure Communications -** concerning that the infrastructure is built on top of the Internet, its nodes must communicate in a secure way among them, in order to protect communications from man in the middle attacks.

4.4 DESIGN PRINCIPLES

In this section, considering the requirements defined in the previous ones, the design principles are outlined along with a brief motivation for making the associated decisions.

As the IoT continues expanding, researchers and companies search for economical and efficient solutions to secure the infrastructures. Mutual authentication, secure communications and integrity messaging will undoubtedly be necessary for the IoT success. As stated in section 3.3, PKI has been the backbone of security in the Internet since its inception, relying in the use of digital certificates. It is a concept that covers authentication, data access control, data confidentiality and information integrity. Above all, PKI is an economical, reliable and proven technology that can be used, in order to set up a secure and high-performance infrastructure [69].

In spite of PKIs extensive capabilities having never been completely explored, this is about to change due to the IoT emergence, which will test the real performance and scalability of PKIs. Unlike conventional PKI and connected devices, the IoT will be composed by constrained devices, which can compromise the infrastructure's performance. In addition, it is necessary to equip all potential devices with PKI-based credentials, which is not easy, since making good keys, in an efficient way, is not an easy task.

Therefore, the PKI is the obvious option for achieving the security requirements identified previously. For a secure infrastructure, it is required to guarantee the peers identity and authenticity. Currently, the three most frequently adopted approaches are considerable different and are known as Hierarchical Certification, WoT and Blockchain-based PKI, as described in section 3.3.

After analyzing the three mentioned approaches, as well as the ADEPT proposal in section 2.5 and the scaling problems regarding the blockchain implementation [45], it is possible to understand that the current implementation of the blockchain cannot be used to achieve the requirements of the proposed infrastructure. Above all, the interoperability, real-time data and scalability required will not be guaranteed using it. For instance, a bitcoin transaction takes about 10 minutes to be validated [45]. Moreover, the WoT approach can not achieve the requirements of the proposed solution either. In particular, the infrastructure growth, real-time data transmissions and scalability may be compromised, as a result of this approach taking a considerable time to validate an identity, as well as being difficult for new entities to join the network without being previously trusted. For these reasons, the proposed

infrastructure makes use of a Hierarchical Certification. Consequently, it will have a centralized dependency. However, considering the current implementations for a decentralized certification, it is not possible to achieve an interoperable and scalable solution with real-time capabilities.

While the number of networked devices is becoming larger, the capabilities of them will diversify. In consequence, the proposed infrastructure must be flexible enough to allow peers with different purposes in the overlay, according to their hardware constraints. For instance, the infrastructure should accept peers with a set of different services enabled, namely:

1. **Data Access:** provides an end-point for accessing the data stored in the infrastructure, which may be used by applications and services. This end-point has an access control mechanism associated, in order to verify if the requester has permissions to access the requested data. After the access control, it handles the data retrieval from the infrastructure network;
2. **Data Collector:** offers an end-point for reporting data, which is used by the sensors of the WSNs associated with the peer. As new data arrives to the end-point, this service forwards the received data to the data persistence and data stream services, which may be in other peers;
3. **Data Persistence:** manages and runs a distributed database combined among the peers with this service enabled. Moreover, it provides an end-point for storing data, which is used by the data collector service, and for retrieving data, which is used by the data access service;
4. **Data Stream:** provides a publish-subscriber mechanism to the infrastructure. As in the data access service, this service has an access control mechanism for allowing entities to subscribe sensors. Therefore, it receives subscribe messages from services and applications, as well as publish messages from the data collector and forwards the received data to the interested parties.

In order to accomplish an infrastructure as described previously, it is possible to divide the peers architecture into four different large categories, Overlay Network, Management Interface, Data Interfaces and Data Persistence.

4.4.1 OVERLAY NETWORK

The Internet protocol suite consists of a computer networking model, which has a set of communication protocols used on computer networks like the Internet [82]. The proposed overlay will be built on top of the Internet, so that a scalable infrastructure may be accomplished. Therefore, the network transport should be carried out through UDP or TCP, which are protocols that run on top of the IP.

UDP is a connectionless protocol, that is, one entity may send a set of packets to another, and the entities relationship would end afterwards. This protocol is faster because there is no guarantee of reliability nor ordering of packets. In consequence, UDP is appropriate for fast and efficient transmissions.

TCP is a connection-oriented protocol, in other words, one entity establishes a session with another entity, sending its packets afterwards. It also requires acknowledge messages, in order to guarantee that the messages arrived their destination. As a result, TCP is suited for networks that prefer high reliability to transmission time. Due to its nature, this protocol is slower than UDP.

On balance, while UDP should be applied in time sensitive networks, as well as in networks where small packets are exchanged by a huge number of clients, TCP should be applied for web browsing

and file transfer. All things considered, UDP is the best protocol for the proposed infrastructure as a result of the large quantity of small packets that are expected to be exchanged among peers.

As previously stated, a structured P2P system is the evident choice for the proposed infrastructure, since it provides a guarantee (precise or probabilistic) on query cost, which makes possible that a request can be routed to a peer, who maintains the desired data quickly and accurately [9].

The most used technique for building a structured P2P overlay is the DHT. Regarding the described implementations in section 3.2, Kademlia was the chosen DHT implementation, since it guarantees good performance, while also providing caching, resource replication and asynchronous range queries [42].

Kademlia ensures a fast and lightweight exchange of data by using UDP instead of TCP. It is based on four simple RPC, known as `PING`, `STORE`, `FIND_NODE` and `FIND_VALUE`. Each peer of the overlay is identifiable through its `node_id`, while each data resource is identifiable by the corresponded sensor identifier. The nodes communicate using the mentioned RPC over UDP communications.

According to the defined requirements, this DHT implementation needs to be enhanced, in order to build a secure and trusted network. In short, the communication among peers must both be secure and maintain the data integrity through the network. Considering the PKI, each peer of the overlay will keep a public key certificate. As represented in Figure 4.3, when a node intends to join the overlay, the following interactions will occur:

1. Get the information of a set of bootstrapping nodes, so that the new node may join the overlay;
2. A set of bootstrapping nodes containing their IP addresses, ports and public key certificates are returned to the new node;
3. Node information and digital signature are sent to a bootstrapping node, in order to request the entrance in the overlay;
4. The bootstrap node creates a session key (symmetric key) and sends it back, encrypted with the new peer's public key.

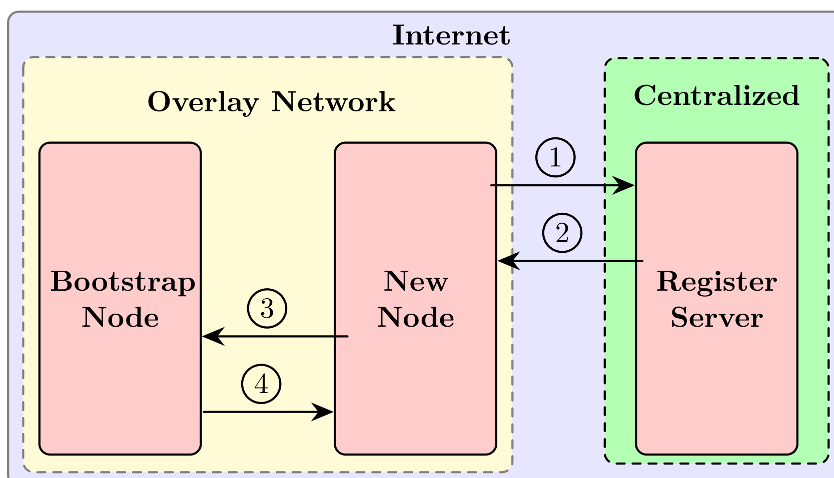


Figure 4.3: Process for a new node get in the overlay.

Since this moment, as illustrated in Figure 4.4, if the previous message is correctly authenticated, both peers will communicate with each other using lightweight encrypted messages, using the session

key previously created for the cryptographic operations. Moreover, when a peer intends to communicate with a new peer of the overlay, they need to negotiate a session key first, which will be persisted next to the node identifier in the kademlia's protocol k-bucket.

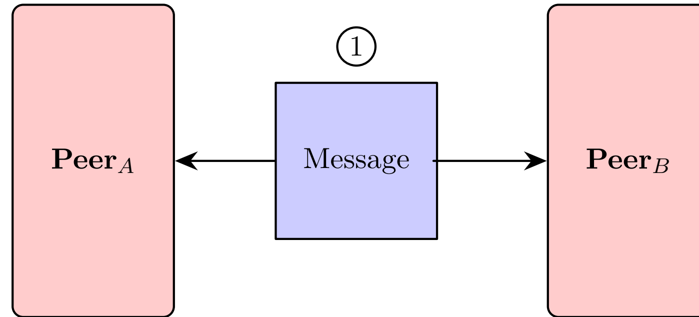


Figure 4.4: Messages exchange in the overlay network.

To summarize, the design choices at the overlay network level are crucial to achieve a high-performance, scalable and fault tolerant infrastructure. These decisions also provide data replication, as well as data integrity and secure communications through the network.

4.4.2 MANAGEMENT INTERFACE

The proposed infrastructure needs to be able to manage entities, who intend to access the data persisted in it, as well as devices, that will report data to it. Another important aspect is the fact that such infrastructure should guarantee data anonymity. As a result, an attacker cannot link a data set to its original owner. It must manage its entities, as well as their information, without compromising their privacy. To achieve this, each entity should have a pseudonym, which is a anonym identifier of an entity inside the infrastructure [83].

The entity's pseudonym must be generated on the application side, using, for example a specific hardware token or the hash of the entity's private key, in order to assure that only the entity can produce its own identity. The proposed infrastructure stores the entity's data through the DHT, where the pseudonym of the entity consists of its key. However, the private credentials of the entity must be stored in a different location of the network, so that no other entity may know who those encrypted credentials belong to. Moreover, all the entity data must be generated on the client side.

Taking into consideration the entities that may sign up in the infrastructure, their data must be properly secured, with a view to prevent identity theft and data tempering. The data model defined for this solution is known as User Data Model and is illustrated in Listing 1.

The User Data Model may be analyzed into three different sections. Firstly, the `"data"` and `"signature"` fields are used for keeping the credentials of the entity, as well as its integrity. Then, the `"sensors"` and `"sensorsSignature"` fields are responsible for storing the list of sensors the entity may access, as well as the lists integrity. Finally, the `"share_invites"` and `"share_requests"` fields are responsible for data sharing events.

```
{
    "data" :
            {
                "code": privateCredentialsDHTKeyCiphered,
                "iv": privateCodesIV,
                "salt": privateCodesSalt,
                "pub_key_DS": digitalSignaturesKey,
                "pub_key_ED": encryptionDecryptionKey
            },
    "signature": dataSignature,
    "sensors" :
            {
                "list" : userSensors,
                "shared" : userSharedSensors
            },
    "sensorsSignature": sensorsSignature,
    "share_invites": invitesDataSharing,
    "share_requests": shareRefusalOrUnsharedData
}
```

Listing 1: User Data Model for DHT

When an application assigned to the infrastructure intends to sign up a new entity, it must create two RSA key pairs, an RSASSA-PKCS1-v1_5 key pair for digital signatures and an RSA-OAEP key pair for encryption and decryption. In addition, it has to generate the entity's pseudonym, as well as a random hexadecimal value for storing the private credentials in the DHT. It is also required to derive the entity's key using a key derivation algorithm, such as PBKDF2. Afterwards, the resulting derivation is used to cipher the private keys and the random hexadecimal previously generated, using an encryption algorithm, such as AES-CBC (it needs a salt and an Initialization Vector (IV)).

Accordingly, the application may sign up the user to the infrastructure, as illustrated in Figure 4.5. This procedure is composed of the four following interactions:

1. The encrypted private credentials of the entity are sent to the infrastructure;
2. Credentials are inserted in the DHT, using the previously generated hexadecimal value as key (if nonexistent in the DHT), in order to keep the entity's credentials stored;
3. The User Data Model of the entity is sent to the infrastructure;
4. The User Data Model is inserted in the DHT, using the pseudonym as key.

It is important to notice that in case of a key overlapping attempt, the infrastructure must inform the application to create new credentials for the user.

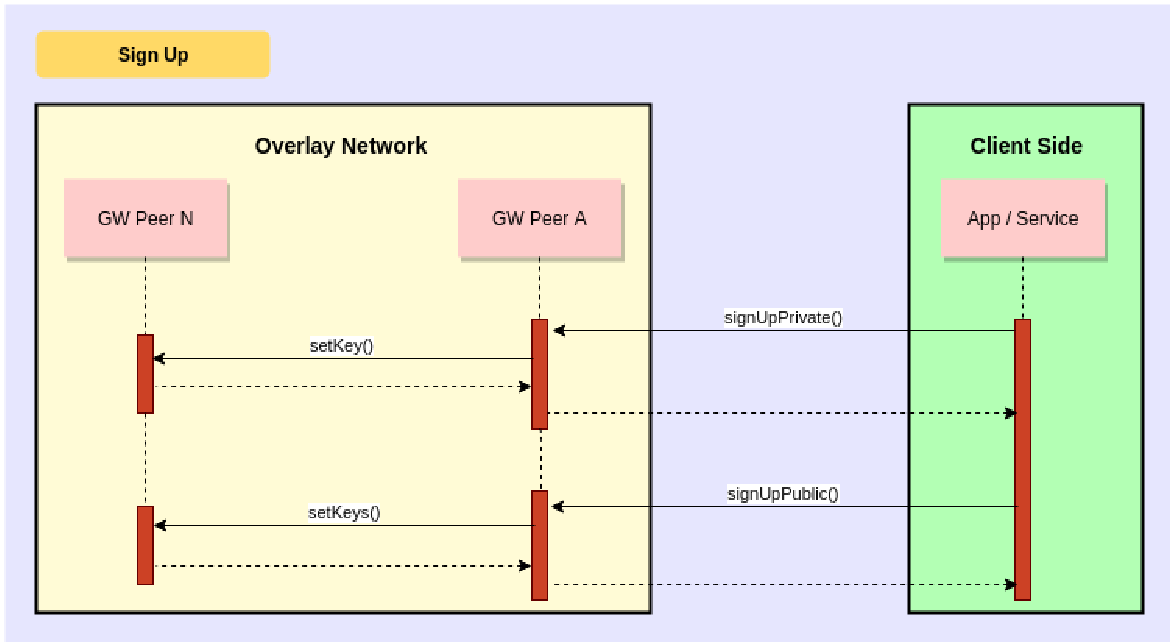


Figure 4.5: Sign up a user in the infrastructure.

After a sign up process has been made, when a user intends to log in the application, the infrastructure only has to receive the user's pseudonym, since only the entity is capable of generating it. The log in process is depicted in Figure 4.6 and comprehend the following interactions:

1. Requests to a infrastructure's peer the user data model of a specific pseudonym, which has been stored in the sign up process;
2. A gateway peer tries to get the user data model from the DHT;
3. The requester verifies the integrity of the previously received data, deciphers the private credentials hexadecimal code, using the password derivation and consequently, requests the user's private credentials, using this code;
4. A gateway peer tries to get the private credentials from the DHT.

When a new sensor is connected to the infrastructure, the user must bind it to its account, with a view to keep his data secure. As a result, the infrastructure has to receive meta-data information about the sensor, as well as to update the list of sensors in the user data model. The Sensor Meta-data Model is represented in Listing 2.

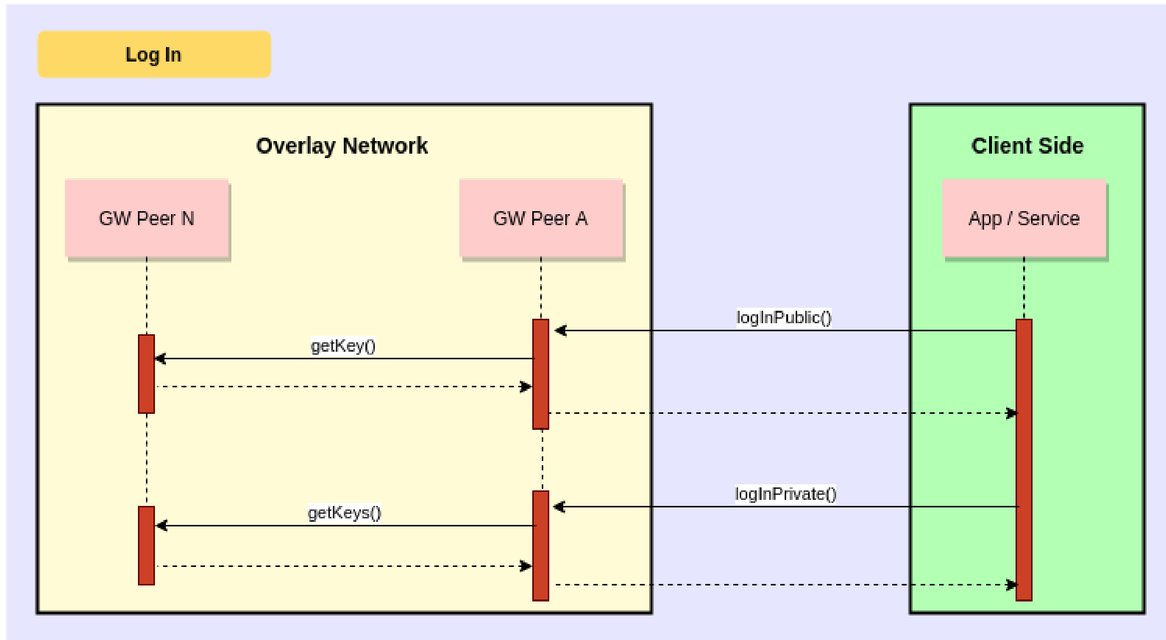


Figure 4.6: Log in a user, in order to request its data to the infrastructure.

```
1 {
2     "meta" :
3             {
4                 "owner": ownerPseudonym,
5                 "public": boolean,
6                 "access_list": [pseudonymsWhoMayAccess]
7             },
8     "meta_signature": metaSignature
9 }
```

Listing 2: Sensor Meta-data Model for DHT

For security reasons, the sensor bind operation, illustrated in Figure 4.7, must be authenticated using digital signatures, in order to guarantee the requests authenticity, as well as integrity. Aiming to prevent data tempering, as well as rewrites as a consequence of concurrent operations, this data model can only be modified by the sensor owner in atomic and signed operations. Moreover, the key used to insert the Sensor Meta-data in the overlay is generated by appending the letter `m` and the `sensor unique identifier`. Thus, this process is composed of the three following interactions:

1. The entity requests to a gateway a sensor binding. Therefore, it sends the Sensor Meta-data Model, composed by the meta-data of the sensor and the signature for validating the identity of the entity;
2. The gateway peer gets the entity data, as well as the sensor meta-data from the DHT and verifies if the received sensor identifier is already assigned, as well as the authenticity of the binding request;

3. The Sensor Meta-data is inserted in the DHT and the entity's data of the requester is updated with this sensor identifier in the access list.

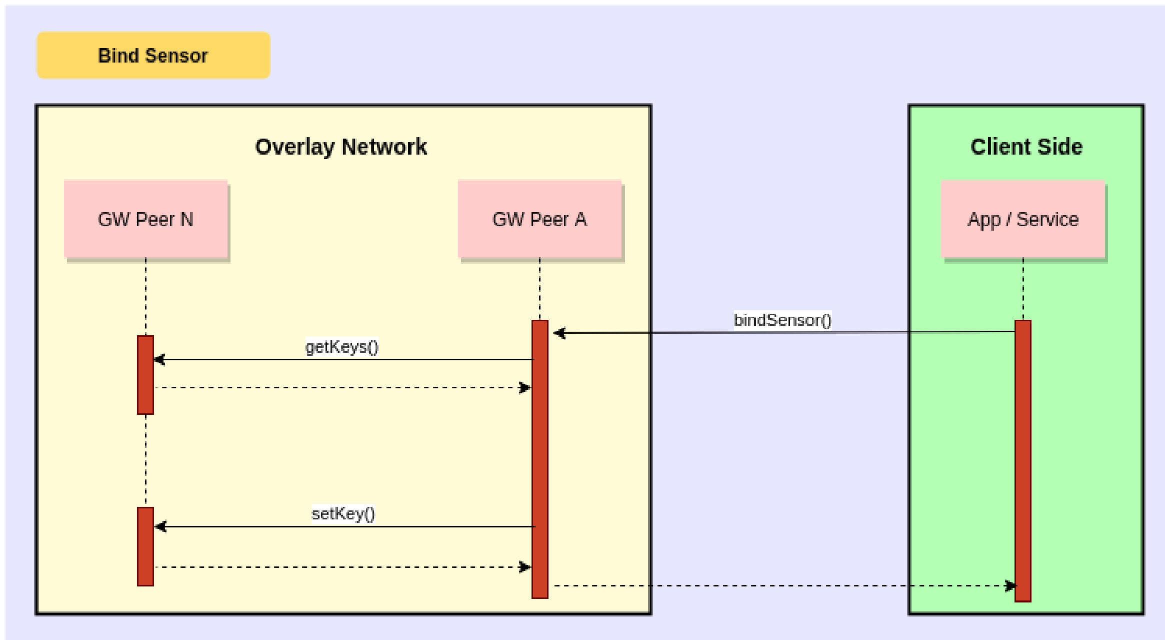


Figure 4.7: Bind sensor to a pseudonym.

Taking into account security purposes regarding authenticity, the sharing and unsharing of data have to be divided into two independent procedures. Firstly, the data owner sends a sharing invitation to another entity. Afterwards, the invited entity may accept or refuse the data share.

Both procedures have the same type of interactions and may be illustrated by Figure 4.8. As represented, the first process comprehend the three following interactions:

1. Requests to a gateway a data sharing to a certain pseudonym, sending the entities and sensor identifiers, as well as a signature of the request;

2. The gateway peer gets the entity data, the sensor meta-data and the intended entity for sharing from the DHT;

3. If all the requested data was found, the gateway peer verifies the authenticity of the sharing request, using the public Key of the requester, as well as if the requester entity is the sensor owner. Afterwards, it sends the updated sensor meta-data, with the new entity in the access list and the updated entity data model with the sharing notification.

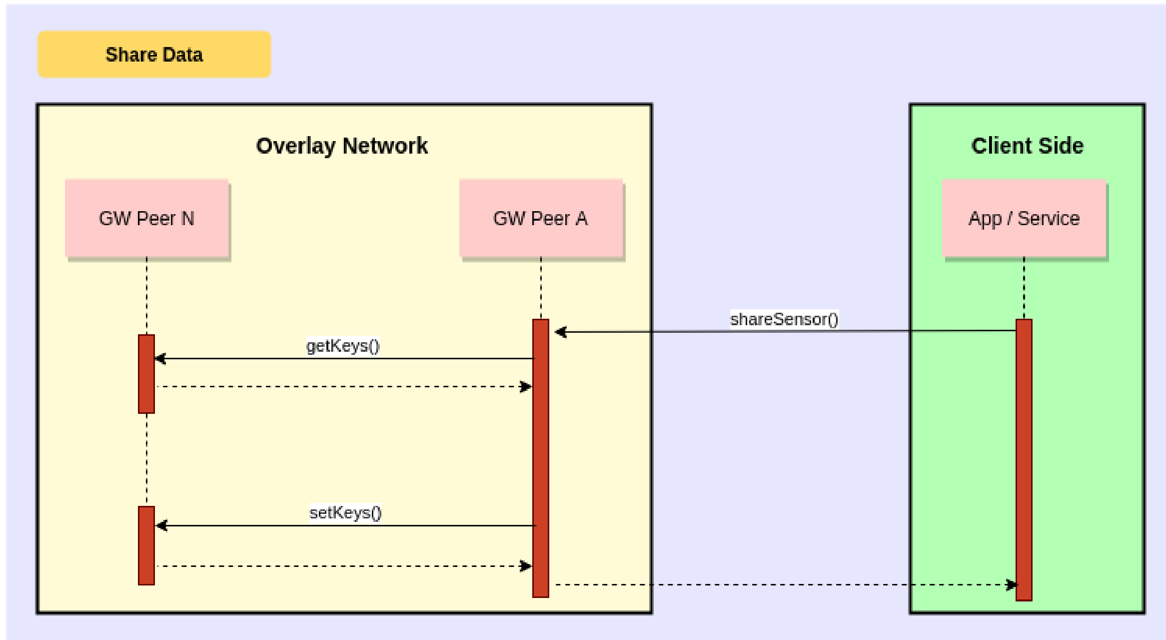


Figure 4.8: Share data from a sensor with other entity.

Therefore, when the user who received the sharing invitation logs in to the application, his data model will contain the received invitation, which may be accepted or refused. This process contains the following interactions:

1. Requests to a gateway peer the acceptance or refusal of a data sharing;
2. The gateway peer gets the user data and the sensor meta-data from the DHT;
3. If all the requested data was found, the gateway peer verifies if the received sensor identifier already has the requester pseudonym, as well as the authenticity of the request. If the user intends to accept the sharing invitation, the gateway updates the shared sensors list of the user, as well as the signature of the list. Finally, it removes the share notification from the user data, and sends the new data to the DHT.

The unshare process is similar to the previously described share. A user, who is the owner of the sensor, may unshare data with a previously shared user. This process is also divided into two independent procedures. Firstly, when a user `A` unshares data with a user `B`, the pseudonym of the user `B` is removed from the sensor access list and a unshare notification is added to his data model. Afterwards, when the user who received the unshare notification logs in the application, he receives and accepts the notification, where the sensor identifier is removed from his list of shared sensors and the notification is removed from his data.

4.4.3 DATA INTERFACES

The infrastructure must be able to receive data from WSNs and provide it to applications or services. It needs to have a data storage interface, as well as a data access interface. Both interfaces

must have security concerns, so that the infrastructure guarantees data security, as well as user's privacy.

Considering that the data produced by sensors may be an attack target of an IoT infrastructure, which may compromise the privacy of business processes, and the intended infrastructure will be in an environment with multiple entities, each one with its privileges, access to information must be limited to the entities who are part of an access list. Therefore, the requests for data retrieval must go through and access control mechanism fist.

The tremendous number and diverse nature of IoT deployments brings plenty new considerations to the table concerning how to actually implement flexible and interoperable IoT infrastructures. Every IoT device is uniquely identifiable by its embedded computing system and must be able to communicate within its WSN.

The store interface consists of a public interface through which the sensors of all WSNs will send their gathered data to the infrastructure. In other words, this public interface is the entrance door for the generated data and should accept different protocol communications, in order to be prepared for the heterogeneous environment of the IoT deployments. Therefore, the proposed infrastructure must accept data from a set of protocols, such as HTTP, MQTT and CoAP.

Taking into consideration that the application layer will update sensors' meta-data, it is necessary to have an additional data model, in order to avoid synchronization problems when both applications and sensors make requests for the same sensor. As a result, the Sensor Data Model is presented in Listing 3.

```
1 {
2     "data" : cipheredDataOrPlainText,
3     "codes": cipherAccessCodes,
4     "distribution": boolean
5 }
```

Listing 3: Sensor Data Model for DHT

This data model only contains information related to the data received by the sensor and consequently, it is only updated by a new message from it. Firstly, the `"data"` field contains the data received by the sensor. If the sensor is subscribed to a pseudonym, this data should be previously encrypted, using a random generated key (symmetric encryption). Secondly, the `"codes"` field consists of a list of access tokens for each pseudonym who may access this data. The access tokens result from the encryption of the random key used for the symmetric encryption, with the public key of each pseudonym (asymmetric encryption). Finally, the `"distribution"` field is used for the distribution of data through the peers, which have the data stream service enabled.

As illustrated in Figure 4.9, the insertion of new data in the infrastructure is composed by three interactions, which may be described as follows:

1. The applications requests the insertion of data in the infrastructure to a GW Peer, which has the data collector service enabled;
2. The GW peer tries to get the Sensor Meta-data from the DHT. If the sensor meta-data does not exist in the overlay, then this sensor is not subscribed by any pseudonym (public data). Otherwise, the GW peer gets the public keys' of the entities who may access this data.

3. If the sensor data is not public, the received data is encrypted and an access code for each entity, who may access the sensor's data, is computed. Finally, the new sensor's data is inserted in the DHT. Otherwise, it is inserted in the DHT in plain text.

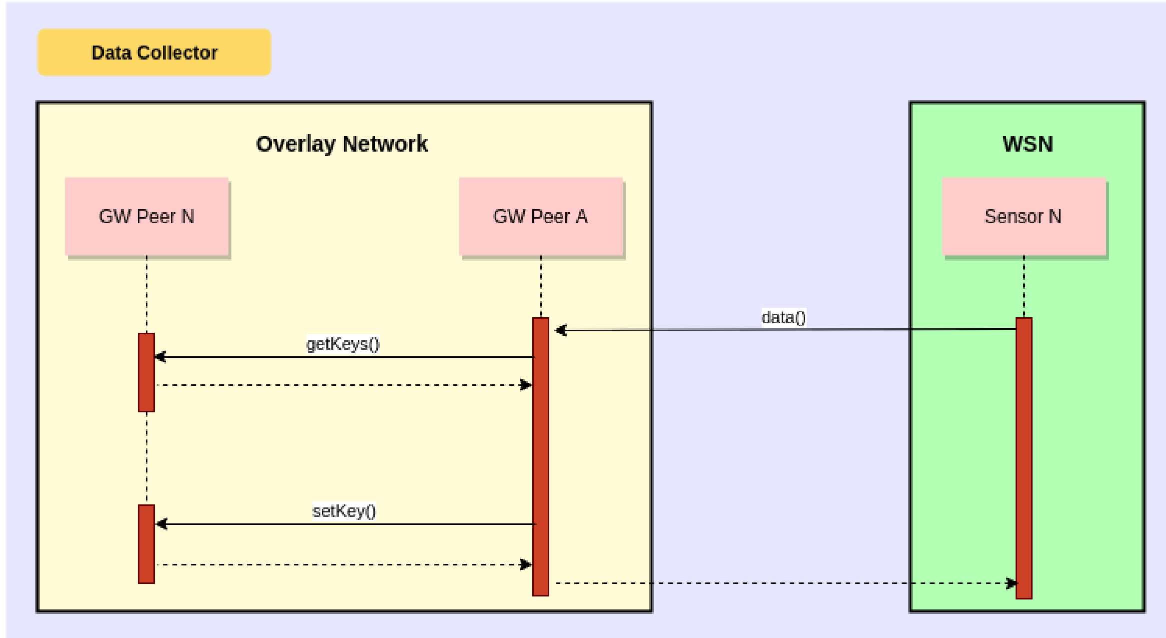


Figure 4.9: Sensor data to be inserted in the DHT.

However, the GW peer has also to persist this new data sending it to a peer, which has the persistence service enabled. Moreover, it also has to forward the new data to the peers, which have the stream service enabled, so that all the subscribers of real-time data of this sensor may receive the subscribed data. Both of this communications are performed asynchronously.

The global scale deploy of IoT devices promises to change the way we live. Accordingly, the data generated by sensors has to be retrieved by those who have the permissions to access it. Therefore, the proposed infrastructure must provide a flexible access interface, so that all business processes needs may be accomplished.

This access interface may be divided into three different interfaces, each one behind the same access control mechanism, in order to verify if the requester holds certain privileges to access the solicited data. The infrastructure should be able to provide the last known data of a sensor (value of the sensor in the DHT). In addition, it must be able to provide the history data of a sensor, i.e. all the measured data during a temporal interval (data persistence). Finally, it should provide a data stream, that is, a publish/subscriber mechanism, so that data consumers may subscribe events from their data.

It is also relevant pointing out the need for data integrity in this context. Accordingly, integrity guarantees that the transmitted data remains consistent and unmodified during its transmission through the network. When it is intended to transmit data, this data must be also signed by the emitter, in order to ensure to the receiver, the authenticity of the message. When an entity requests data from a sensor, it must send the sensor identifier, as well as the pseudonym along with the signature of the request.

Regarding Figure 4.10, a data history access procedure comprehends the following interactions:

1. The application requests the data history of a certain sensor to a GW Peer, which has the data access service enabled. This request contains the entity pseudonym, the sensor identifier, the intended query and a signature;
2. The GW peer tries to get the entity's data of the requester, as well as the sensor meta-data, from the DHT;
3. If the sensor-meta data and the entity data are found, the GW peer validates the request signature and verifies if the requester may access the requested data. Accordingly, the queried history data is requested to a peer, which has the persistence service enabled and forwarded to the application.

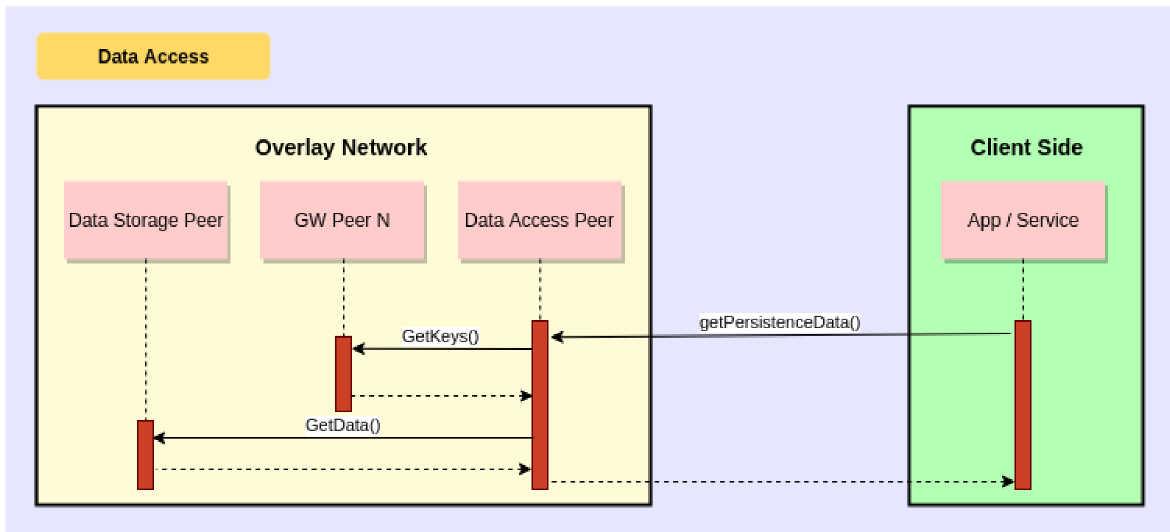


Figure 4.10: Get data history from the distributed database.

When the requested set of data is returned to the requester, he will need to decrypt the access code with its private key. Afterwards, it can use the resulting access code to decrypt the received data and parse it. Therefore, only the intended recipient can decrypt the received data through its private key, and the entity privacy is guaranteed.

Taking into consideration the publish-subscriber mechanism needed for real-time data acquisition, three important aspects must be taken into account. When a user tries to subscribe data from a sensor identifier, this subscription must be granted by an access control mechanism, such as in the data access previously described. Moreover, when a sensor publishes data to the broker, this data must be encrypted before being propagated by its subscribers. Finally, regarding the decentralized nature of the infrastructure, it must behave as a single broker. Each peer has to synchronize its publishers and subscribers with the other peers. Thus, each one subscribes all the other peers, which have the stream service enabled, in order to exchange information among them.

The design choices made aim to provide data access control and privacy, as well as data anonymity to the infrastructure. In addition, they intend to achieve a flexible and interoperable infrastructure.

4.4.4 DATA PERSISTENCE

Considering the IoT premise where millions of devices will be connected and will be producing massive volumes of data, the capacity and efficiency of data storage are crucial. The stored data must be encrypted, in order to prevent unauthorized access to it, in case of an attacker getting access to a gateway.

Just after the Store Interface generates the encrypted data, as well as the entities' access codes, this layer will get in scene, in order to persist this information along the network, for keeping the history state of the IoT.

Considering the expected amount of data that will be generated, it is crucial to distribute the specified data through the overlay. Therefore, this data could be stored in a distributed file system or, in a distributed database. A distributed database provides data consistency, as well as efficient data access and replication. On the other side, a distributed file system consumes considerably less hardware resources.

As stated in Section 2.6, time series databases are optimized for sequential writes, which will be indexed by timestamps. Moreover, they also provide an efficient retrieval for data in a time interval, as well as a fast removal of data that is no longer relevant.

Considering the tremendous advantages of time series databases for IoT infrastructures, as well as the flexibility to have nodes with different capabilities in the overlay, it was decided to use a distributed time series database for storing the received data.

This database will consist of a table oriented by columns, where each column will correspond to a sensor identifier. As a result, each entry of this column will contain the sensor data model for a certain time-stamp, and all the data received from a sensor will be stored sequentially, ordered by its relevance, that is, the most recent data is in the top of the column.

To summarize, the design decisions of this layer aim to provide a secure data storage to the infrastructure, guaranteeing the data anonymity. Moreover, the distributed database allows the scalability and performance of the infrastructure, as well as data replication.

4.5 PEER ARCHITECTURE

The above design principles aim to accomplish all the objectives and requirements previously stated. These principles lead to the following system architecture for the gateway peers, which is represented in Figure 4.11:

Peers with different constraints may cohabit in the overlay network and, consequently, the infrastructure has to be flexible to accommodate such a diversity of nodes. Therefore, when a peer is started, it has to select its roles in the overlay. The Network Transport, DHT, Overlay, Management, Middleware and Data Security layers are mandatory for all the peers, so that the infrastructure runs properly.

On the other hand, the Access Interface, Store Interface, Stream and Data Persistence layers are optional for the peers. In consequence, they should be enabled according to the peers hardware specifications. A peer capable of storing large quantities of data should enable the Data Persistence service, while an IoT gateway peer has to enable the Store interface, in order to listen for IoT events of its assigned WSN. Furthermore, the Access interface may be enabled, if the peer has good computing capabilities, so that it may provide the requested data.

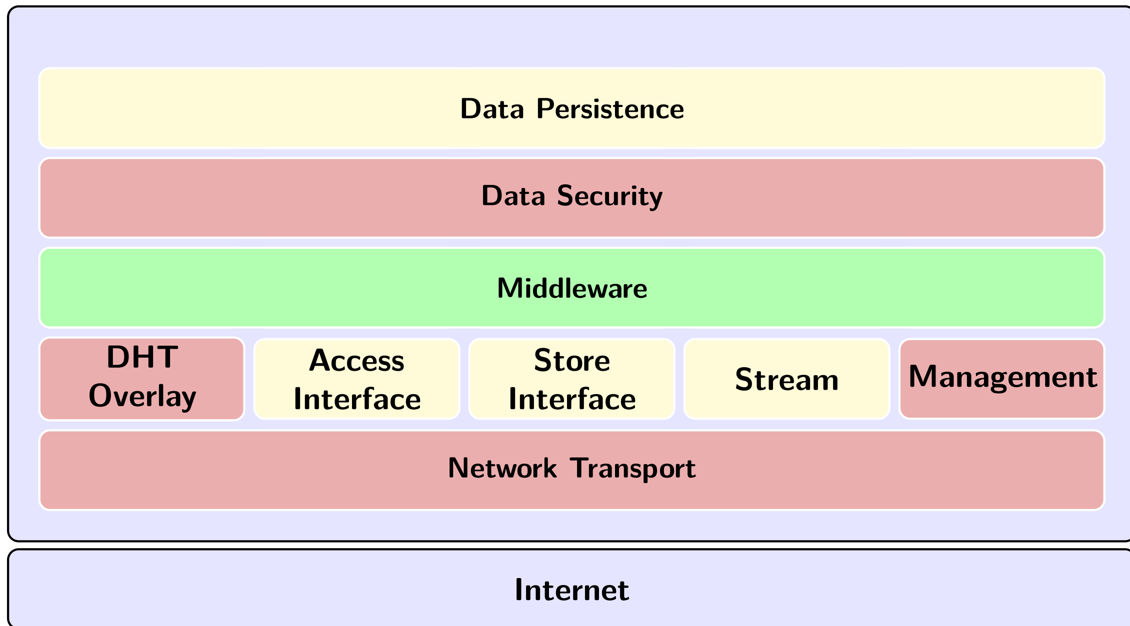


Figure 4.11: Architecture for a Decentralized IoT Peer.

The Middleware layer is responsible for enabling the layers of the services selected in the peers configuration. Moreover, this layer handles the communication between the different services of the architecture. For instance, when an application requests data to the Access Interface end-point, the Access Interface asks the middleware to handle the communications with the other components, in order to retrieve the requested data in a secure manner.

CHAPTER 5

Prototype Implementation and Technologies

In this chapter, the prototype architecture is presented, as well as an overview of the technologies that have been used in its implementation. Some important aspects regarding the implementation components, as well as its deployment and integration are described.

5.1 Architecture

The implemented prototype, known as dIoTi, was developed regarding the requirements previously specified in section 4.3. It may be composed of three different processes, a Twisted Application Infrastructure, an MQTT Broker (optional) and a Database Engine (optional), according to the enabled services of the gateway. The software architecture of this prototype is illustrated in Figure 5.1.

The main process was developed using Python Twisted[1], an event-driven networking engine. As a result of its asynchronous nature, it can handle thousands of connections in a single thread, providing a great performance and scalability, even in constrained devices. The Twisted Application Infrastructure implemented acts as a container of different services. Consequently, the implemented application is modular and easy to update, allowing new services to be easily developed and integrated in the application.

In the context of this prototype, the developed Twisted Infrastructure contains several services. The Overlay Service is used to build a secure and trusted overlay network, on top of a DHT, as described in section 4.4.1. In addition, the Management Service implements a REST API for managing the user entities as specified in Section 4.4.2 for the Management Interface. The Access Service and the Collector Service also implement REST APIs for handling data flows between sensors and applications, as described in the Data Interface in section 4.4.3. Furthermore, the Collector Service also receives data from the MQTT Broker and forwards it to the Persistence Service, which persists the received data in a local or remote InfluxDB Database, according to Section 4.4.4. Finally, the Stream Service is

[1]https://twistedmatrix.com

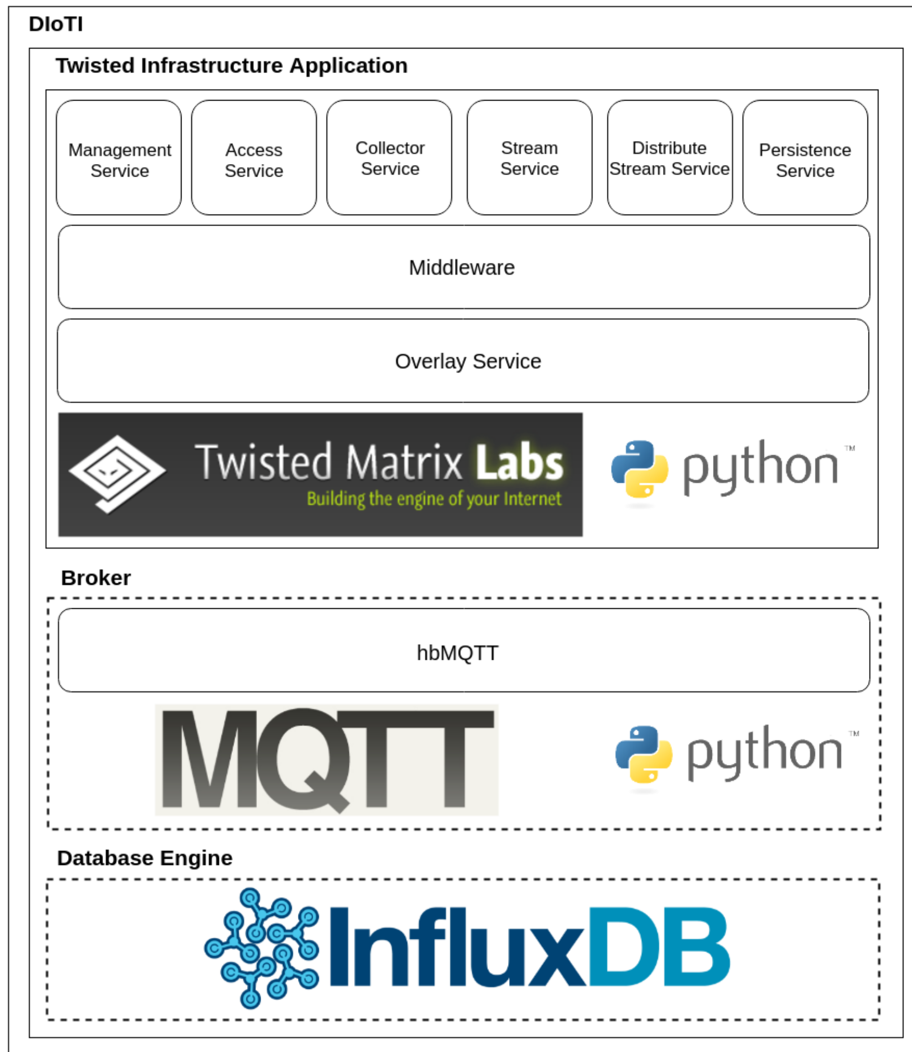


Figure 5.1: Gateway Peer prototype architecture.

used by the MQTT broker for communicating with the infrastructure, whereas the Distributed Stream Service is used for synchronizing all the MQTT Broker processes of the overlay network.

All the developed services use auxiliary modules, such as modules for encryption and decryption, or certification. Combining those modules, with the developed services and its communications, we achieve a Peer Architecture similar to the specified in Section 4.5.

Taking into consideration the decentralized nature of this prototype, as well as its security requirements, it is fundamental to have a logging system in the proposed infrastructure. Thus, a logging instance is provided to each service, which is bound to the developed application. With the resulting log files, it is possible to identify attack attempts to certain sensors, as well as to detect applications misuses or even errors and loss of availability. In the context of this prototype, the logging messages have a tag where the service is identified and the associated message.

Since developing a fully working broker was not part of this dissertation' scope, we opted for modifying an already existing open source implementation, in order to suit our needs. Accordingly, the HBMQTT Broker[2] implementation was modified, so that we can achieve the proposed infrastructure

[2]https://github.com/beerfactory/hbmqtt

requirements. Shortly, this implementation has also an access control mechanism for data subscriptions and a data encryption mechanism for data publishes. This broker is executed in its own process, separated from the main process.

5.2 TWISTED APPLICATION COMMUNICATION

The networking infrastructure developed, needs a set of different types of communication, each one with its own properties.

The overlay service uses an open source implementation of the Kademlia Protocol[3], properly modified so that the security requirements may be achieved. Therefore, all the communications between the overlay peers are properly encrypted. This API provides two main methods to the developed infrastructure, as depicted in Figure 5.2. It has a `set` method, which is used for inserting data in a certain key of the DHT. Moreover, it provides a `get` method, which allows the peer to get data stored in the DHT by its key. Both of this methods are made on top of RPC over UDP, which consists of another open source project[4].

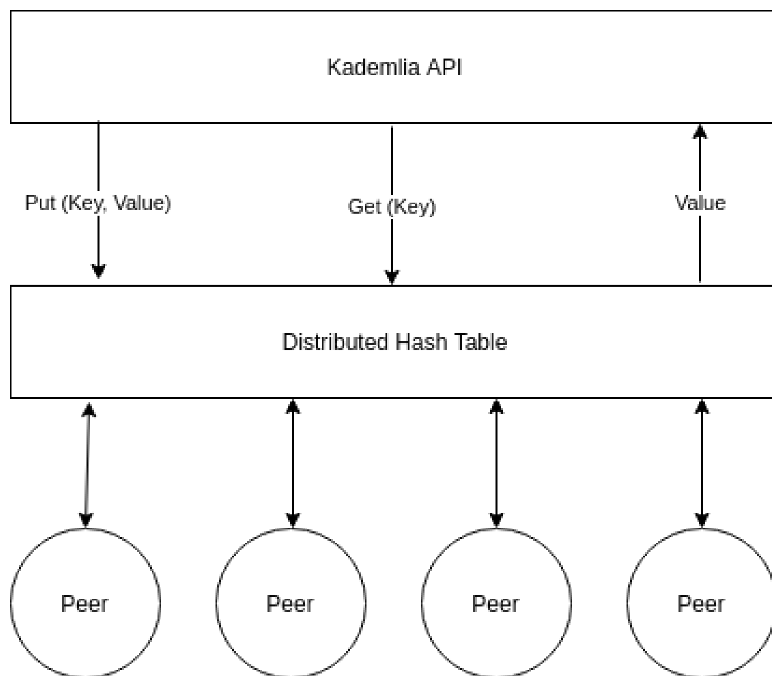


Figure 5.2: Kademlia API Interactions.

As previously stated, the modified HBMQTT broker has an access control, as well as a data encryption mechanism. In the context of the access control mechanism, the broker process communicates with the Twisted Application, so that it can verify if a pseudonym may subscribe a certain topic (sensor identifier). Moreover, considering the data encryption mechanism, the encryption is requested to the Twisted Application, in order to encrypt the received data before spreading it to its subscribers,

[3]https://github.com/bmuller/kademlia
[4]https://github.com/bmuller/rpcudp

as well as persist it in the infrastructure. Both communications described are made on top of HTTP communications.

Taking into account the peers which have a local database enabled, it is necessary to have a driver that allows the communications between the Twisted Application and the database engine. There is a Python client for interacting with the database, which was developed by its creators. However, this client is not compatible with twisted, thanks to its asynchronous nature.

Accordingly, an open source driver was developed for the InfluxDB database[5], which is compatible with Twisted. Therefore, the Persistence Service uses this driver to communicate with the Database, in order to insert new data, as well as query it.

Considering the different services that each peer may have activated, it is necessary a form of discovering what services each peer offer. Consequently, a good solution to solve this problem is the mDNS[84]. It provides the ability to look up DNS resource record data types in the network.

In the context of this prototype, an open source Twisted plugin[6] is used for using the Avahi/Bonjour service[85]. Thus, when a gateway peer starts, it starts broadcasting its enabled services, using the services' name, in order to have its services available for being discovered.

5.2.1 REST APIs

The implemented peer's architecture is composed of five different services, which may implement a REST API. However, only Management Service must be enabled in every gateway peer. Therefore, the Access Service, Collector Service, Stream Service and Persistence Service will be enabled according to the peers' configuration.

The Management Service, which is responsible for managing the infrastructure entities, has twelve methods that may be used by applications. Briefly, it has methods for authentication, associations between sensors and pseudonyms, as well as methods for handling data sharing.

Considering the HTTP interface, which is used by sensors to report their data, the Collector Service has a method for publishing data in the infrastructure. Moreover, as the applications need to get sensors' data, the Access Service provides two different methods for accessing data. It is provided a method for retrieving the last received data of a certain sensor, as well as a method for obtaining the history data of a sensor (may be filtered by timestamps).

Bearing in mind the needed communication between the MQTT Broker and the application, the Stream Service has two methods to be used by the broker. For subscription requests, the application provides an access control method, while for publishing requests, it is used a method for data encryption.

Finally, when a peer has the Persistence Service enabled, it also provides methods for data querying and data insertion to the peers that have the considered service disabled. In addition, the REST APIs described are properly documented in the Appendix A.

[5] https://github.com/vasco-santos/Twisted_InfluxDB_Driver
[6] https://github.com/jdcumpson/txbonjour

5.3 REGISTER SERVER

Complementing the IoT Gateway peers, a CA server was developed using Python's AsyncIO library, as depicted in Figure 5.3. In addition, aiming to persist the peers' certificates, it is used a RedisDB.

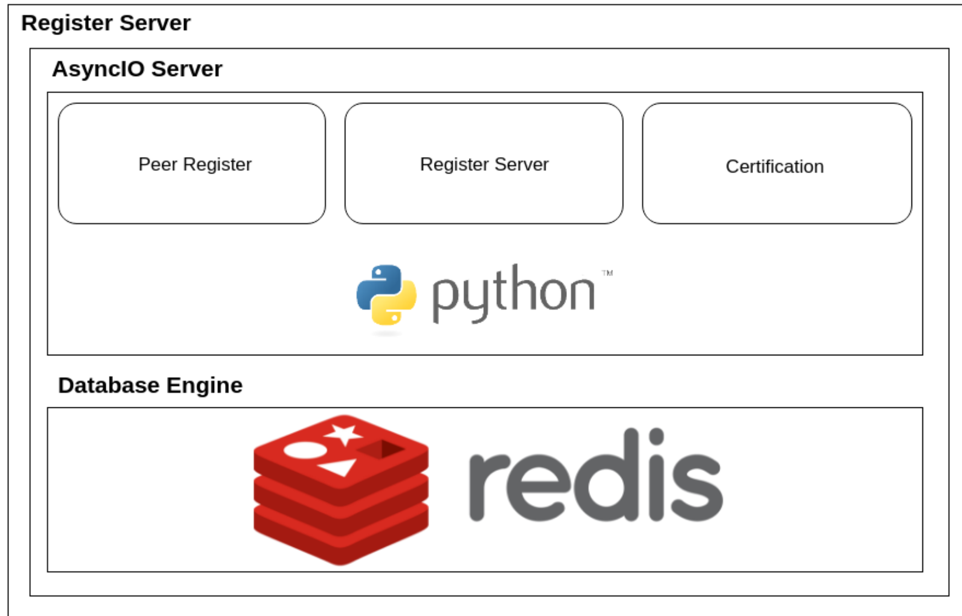


Figure 5.3: Register Server prototype architecture.

This server's main feature consists of generating signed X.509 certificates to the overlay peers. This certificates are persisted in the database for a short limited time (previously defined expiration time). Consequently, the overlay peers have to register in the CA periodically, in order to keep a fresh list of bootstrapping nodes.

Taking into account the register server communications, it has a REST API implemented using an open source HTTP server known as aiohttp[7], which provides a set of methods to the overlay peers. It provides methods for getting bootstrapping nodes and to register a peer, as well as for getting peers' data. This API is fully described in the Appendix B. Finally, it uses an open source Redis client, known as aioredis[8], for communicating with the database.

5.4 DEPLOYMENT AND INTEGRATION

Considering the complexity of this prototype, it is essential to automate its installation and setup. Moreover, it is important to provide a good support for the application developers, so that they can start developing applications, which use the proposed infrastructure.

The implemented prototype contains a set of modules and scripts, for handling the configuration, set up and profiling of the peers. These modules and scripts aim to automate the installation and setup of the peers, requesting only a few inputs to the entity that is installing the software.

[7]https://github.com/KeepSafe/aiohttp
[8]https://github.com/aio-libs/aioredis

According to the prototype architecture previously presented, this prototype may have multiple processes running. The Twisted Application Infrastructure was developed using Python2, as a result of the cryptographic libraries compatibility, while the broker was developed using Python3, thanks to the use of the AsyncIO library. Consequently, peers must have both versions of python installed.

Aiming to offer an easy deployment to gateway managers, a machine_setup shell script is provided, as well as a requirements file for Python2 and a requirements file for Python3. Therefore, the shell script starts by installing the required Linux packages. Afterwards, it installs the required packages for both Python versions, as well as the InfluxDB database.

It is also important pointing out that it is used the version 0.11 of the InfluxDB database, which was released during the implementation of the presented prototype. In spite of a new version being already available, the version 0.11 was the last open source version that included the clustering feature[9].

5.4.1 CONFIGURATION AND SETUP

After executing the machine setup script for installing all the necessary packages, it is necessary to configure the gateway. In the interest of providing an easy to use software, it was developed a shell script for configuring and starting the peer. When this script is executed with any parameter, it prints out a menu with its available options, as illustrated in Listing 4.

```
> ./dioti.sh

Usage:  ./dioti.sh <command> [<arg>]

GATEWAY COMMANDS:

  config [<conf>] -> Change Configuration (Addresses OR Register OR Services)
  init -> Initalize Gateway Local Configuration
  show [<conf>] -> Show Configuration details (Addresses OR Identity OR
       Register OR Services)

NETWORK COMMANDS:

  daemon -> Start a long-running daemon process
  reload -> Restart Server
```

Listing 4: Menu output for gateway script.

For installing the peer software, it is executed the command with the `init` argument for configuring the Gateway, as illustrated in the Appendix C. The presented execution starts by creating the directory structure, where all the configuration files and logs of the application are stored. Then, openssl is used for generating an RSA key pair and the peer identifier for the DHT is generated, using the `sha1sum` of the generated Private Key. Afterwards, the network environment of the peer is configured and the CA's self-signed certificate is requested. Finally, the gateway manager may decide what services he intends to enable, as well as its configurations. An example of the generated configuration file is presented in the Appendix D.

[9]https://docs.influxdata.com/influxdb/v0.12/concepts/011_vs_012/

From this moment on, the peer only has to execute the script with the `daemon` argument, in order to be ready for being part of the overlay network.

5.4.2 APPLICATION BINDING

Taking into consideration the application layer, which will be composed of a large number of different applications, the developers must be focused on their features, instead of the integration of the application with the infrastructure. Consequently, it was developed two JavaScript snippets, which may be included in web applications, in order to provide an easy integration for web applications.

The cryptographic snippet abstracts the necessary logic for all the cryptographic operations inherent to the authentication processes, as well as to the integrity validations and data encrypting/decrypting. This snippet uses the Web Cryptography API[10], which was implemented by the Web Cryptography Working group from the W3C, which has the participation of companies like Google and Netflix. This snippet processes the needed cryptographic approaches and generates the necessary data to be sent to the Infrastructure. It provides a wide set of methods, including `signUp`, `logIn`, `sign`, `verifySignature`, `decipherData`, among others.

The mDNS snippet may be used to discover the IP addresses of peers, which have the Access Service and Management Service. This snippet uses an open source implementation of the mDNS for JavaScript[11]. It monitors the network, in order to keep an updated list of the peers, which have the Management Service enabled, as well as a list of the peers, whose Access Service is enabled. Furthermore, it has methods for retrieving peers with the intended service enabled.

[10] https://www.w3.org/TR/WebCryptoAPI/
[11] https://github.com/agnat/node_mdns

CHAPTER 6

Evaluation and Results

The present chapter aims to evaluate the designed and implemented solution. A deployment scenario has to be built, in order to create a real test scenario for evaluating the proposed IoT infrastructure. Therefore, this scenario has to include sensors sending data to the infrastructure and an application requesting data to it.

At the beginning of this chapter is presented the sensor simulators implemented, which were developed so that the infrastructure may receive data from sensors. In addition, a web application that was built for simulating entities requesting data to the infrastructure is presented. Finally, the deployment scenario which was used to evaluate the proposed infrastructure is outlined and evaluated, regarding performance, data security and the use of different communication protocols.

6.1 Sensor Simulators

Taking into account that the proposed infrastructure must receive data, in order to be properly evaluated, two sensor simulators were developed. Shortly, these simulators send periodic messages to the infrastructure, simulating the behavior of a large set of different sensors.

6.1.1 Software Specification

Two identical simulators were developed for evaluating two different communication protocols, as well as the infrastructure. Both were implemented using Python Twisted and use a JSON file, which contains a list of sensors. The main difference between these simulators consists in the protocol used for reporting data to the infrastructure. One uses the HTTP protocol, whereas the other uses the MQTT protocol.

When a simulator is started, it loads the sensors list from the JSON file and finds peers, which have the collector service enabled, using the mDNS. For each sensor in this list, a chunk of random data is generated, according to the sensor object specified, and forwarded to the infrastructure. Finally, a new chunk of random data is generated and forwarded periodically. Accordingly, each sensor has a

previously defined period, which is specified in the sensor object. Thanks to the asynchronous nature of twisted, multiple requests can be efficiently sent at the same time.

An example of the HTTP simulator deployment is illustrated in Figure 6.1.

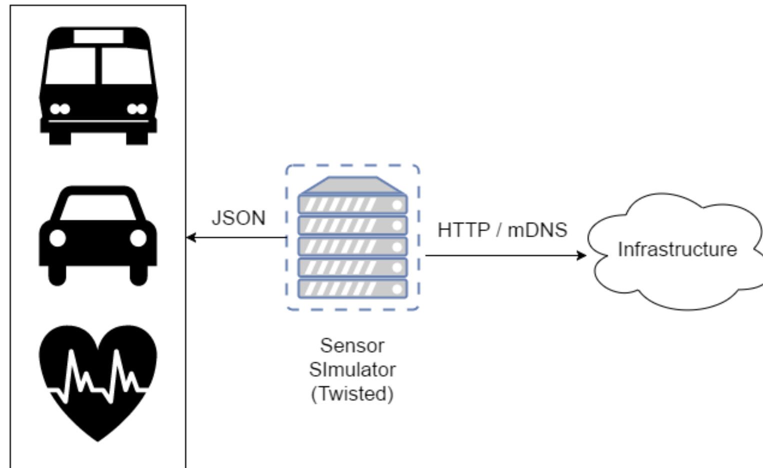


Figure 6.1: HTTP Sensors Simulator.

6.1.2 DATA MODEL

Considering the JSON file previously mentioned, which contains a list of sensor objects, the Listing 5 exemplifies an entry of this list. It is possible to verify that this object contains the sensor identifier, the range of values it may generate and its periodicity, in seconds. In addition, it has an object field, which is used by the application layer to manage different types of sensors.

```json
{
    "device": "56e0849db14f9ecd0d51bbae",
    "object": "temperature",
    "value": [-20, 50],
    "periodicity": 20
}
```

Listing 5: Sensor object from JSON file.

The simulator verifies the value range and generates periodic random values accordingly. An example of the data, which may be sent to the infrastructure, is presented in Listing 6. In the HTTP simulator, the sensor identifier is passed in the URI, whereas in the MQTT simulator, the sensor identifier corresponds to the MQTT topic used.

```
{
    "object": "temperature",
    "value": 30
}
```

Listing 6: Format of data sent from the simulator.

6.2 WEB APPLICATION

In the context of the IoT, it is crucial to obtain the data generated by sensors easily and safely. Consequently, common IoT platforms have applications, which allow the users to visualize their data, even in real-time. These applications may be desktop, mobile or web-based. Considering the advantages of web applications, regarding the multiple operating systems compatibility and the easy access in all types of devices, it was the chosen type of test application to be implemented.

6.2.1 ARCHITECTURE

This web application was designed taking into consideration the functional requirements previously defined in section 4.2. Its main goal consists in testing all the implemented features of the infrastructure. Accordingly, this web application was developed, composed of a server-side and a client-side, as illustrated in Figure 6.2.

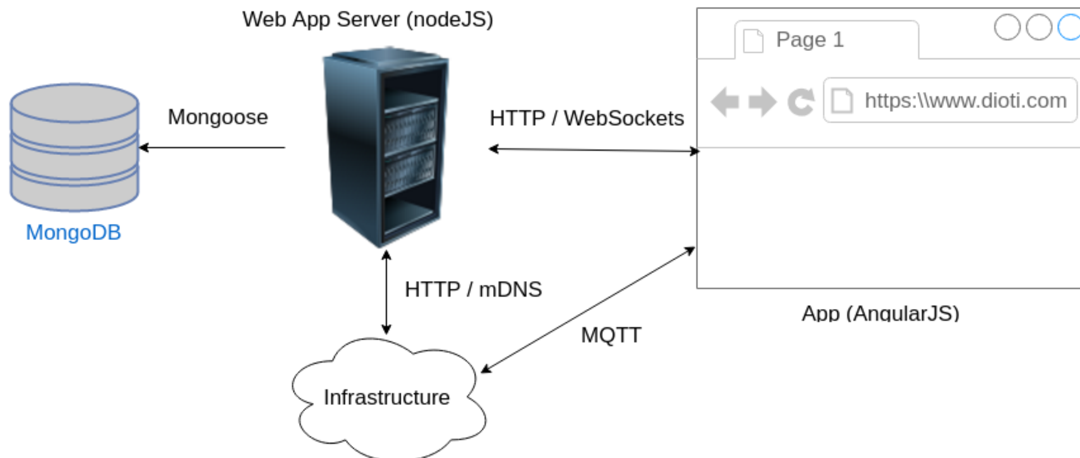


Figure 6.2: Web Application.

The server-side handles the logic inherent to the application context, as well as the communications between the client-side and the proposed infrastructure. It is composed of a Node.js[1] server and a MongoDB[2] database. This database is responsible for the data specifically related to the application.

[1]https://nodejs.org/en/
[2]https://www.mongodb.com/

For the client-side, it was developed a single-page application, using the AngularJS[3] framework. It is structured according to the model-view-controller (MVC) pattern. Taking into account the Graphical User Interface development, the Bootstrap Framework was used, as well as the AdminLTE[4] theme. Chart.js[5] was also used for the creation of dynamic graphs for representing the data history of a sensor. Moreover, MQTT.js[6] client was used for the browser, in order to allow the subscription of MQTT topics from the browser, on top of web sockets, directly to the infrastructure. Finally, it is important pointing out that the developed application uses the Cryptographic Snippet described in section 5.4.2 for the cryptographic related operations.

Taking into consideration that this application is not the focus of this dissertation, the entity's pseudonym is stored in the application's database. As a result, the entity has to trust the application to use the infrastructure. However, the ideal solution consists of the entity's pseudonym being only in his hands. For instance, the entity could have a hardware token for providing his pseudonym, which could be protected with a password or a pin code.

6.2.2 FEATURES AND GRAPHICAL USER INTERFACE

In the IoT realm, current applications have a user management system for authentication and access control. For the proposed infrastructure, it is also essential for the applications to have an authentication mechanism, so that the entity's privacy and security are guaranteed.

Presented in Figure 6.3 is the first interaction that dIoTi provides to the user. It consists of a login screen, where the user is prompted to enter his credentials. If the user has no account in the application, he has to click on the `"Register New Membership"` button.

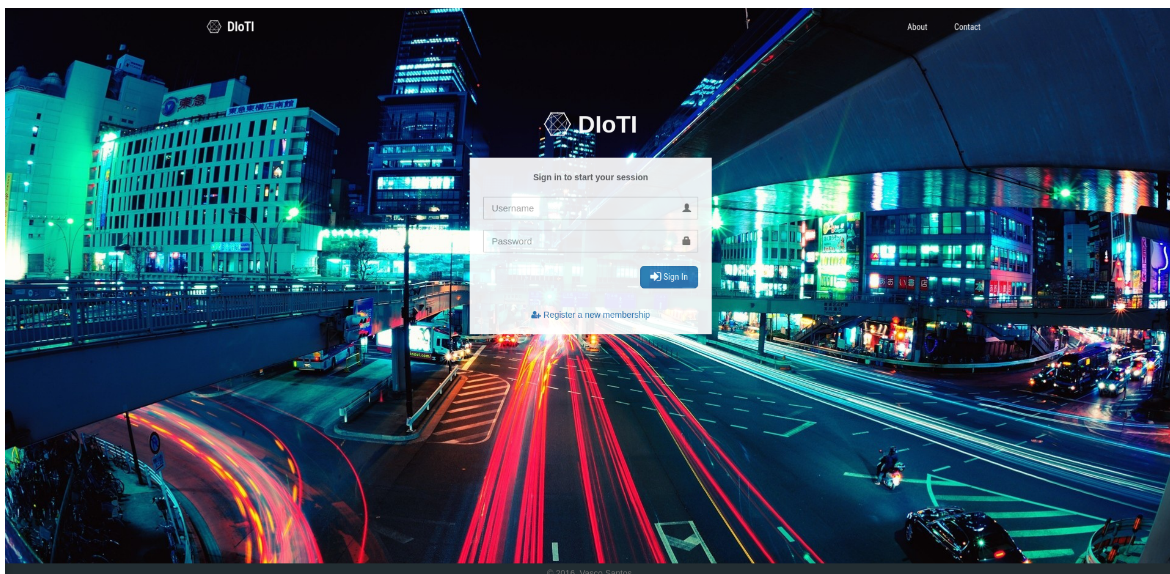


Figure 6.3: Log in interface.

[3]https://angularjs.org/
[4]https://almsaeedstudio.com/
[5]http://www.chartjs.org/
[6]https://github.com/mqttjs/MQTT.js

Figure 6.4 illustrates the sign up form. In this process, all the user credentials are generated on the client-side and sent to the infrastructure, as specified previously.

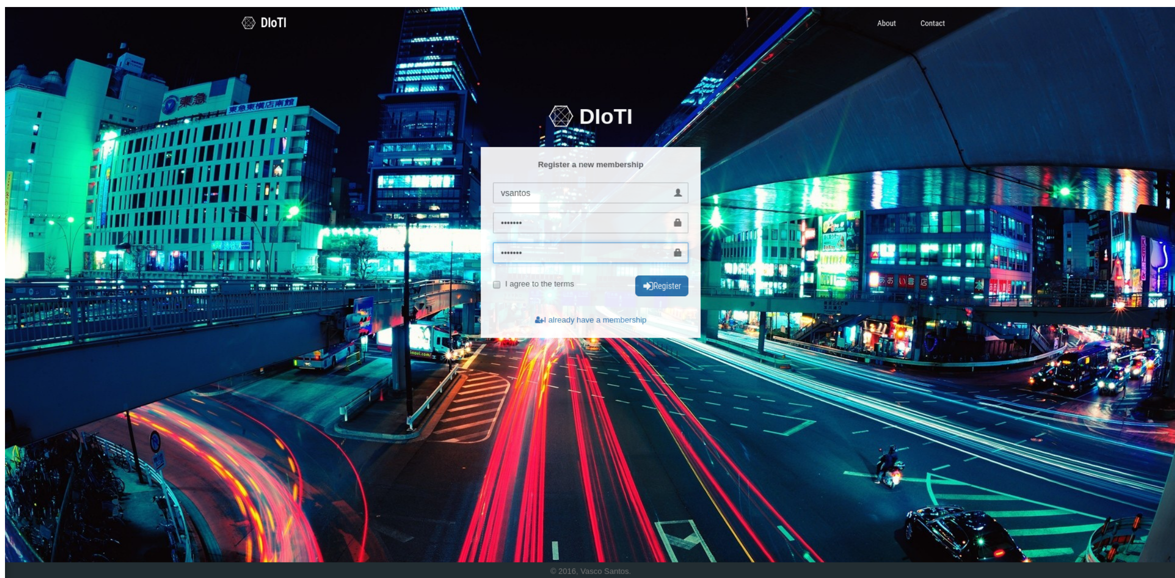


Figure 6.4: Sign up interface.

In case of a successful login, the user is redirected to the main page of the dashboard, and all his data is requested to the infrastructure and decrypted on the client-side. Afterwards, as illustrated in Figure 6.5, a resume of the entity's account is presented. The last received data of each sensor owned by the entity is presented in the first panel, while the second panel shows the last received data of each sensor shared with the entity. Finally, the entity's notifications are shown in the last panel. All the sensors that may be accessed by the entity are subscribed and consequently, the first two panels of this page are updated in real-time.

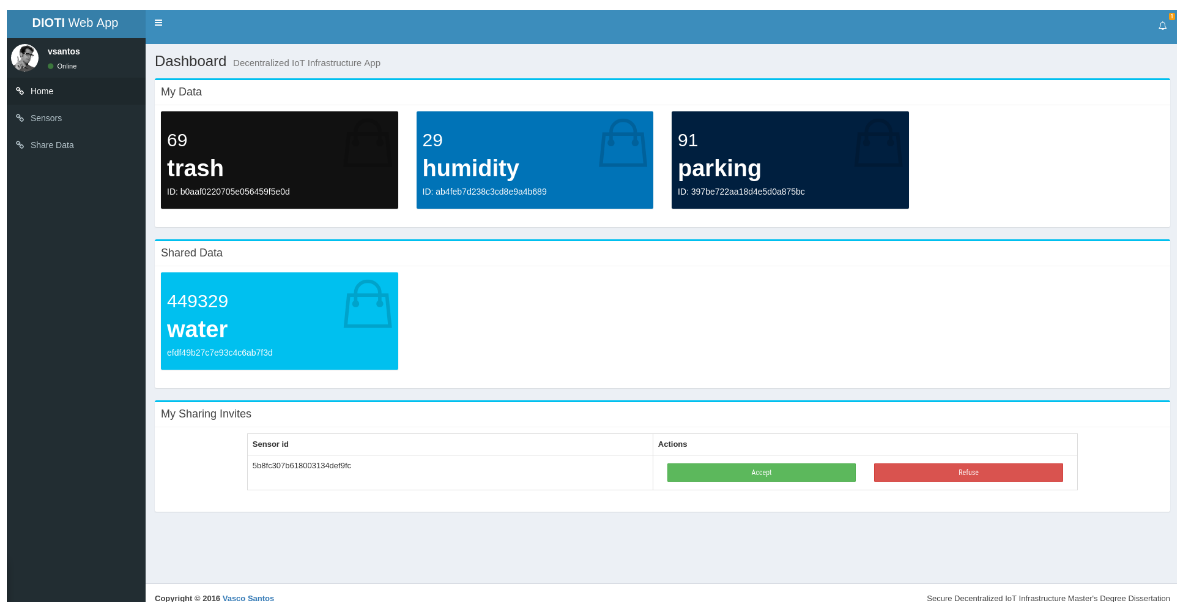


Figure 6.5: Main interface of the dashboard.

The dashboard provides a set of other pages that may be accessed by the user through the left bar. The entity can bind his new sensors by selecting the `Sensors` option. As illustrated in Figure 6.6, the user has to insert the sensor identifier and, if it was not previously bound by another entity, the infrastructure binds this identifier to the entity.

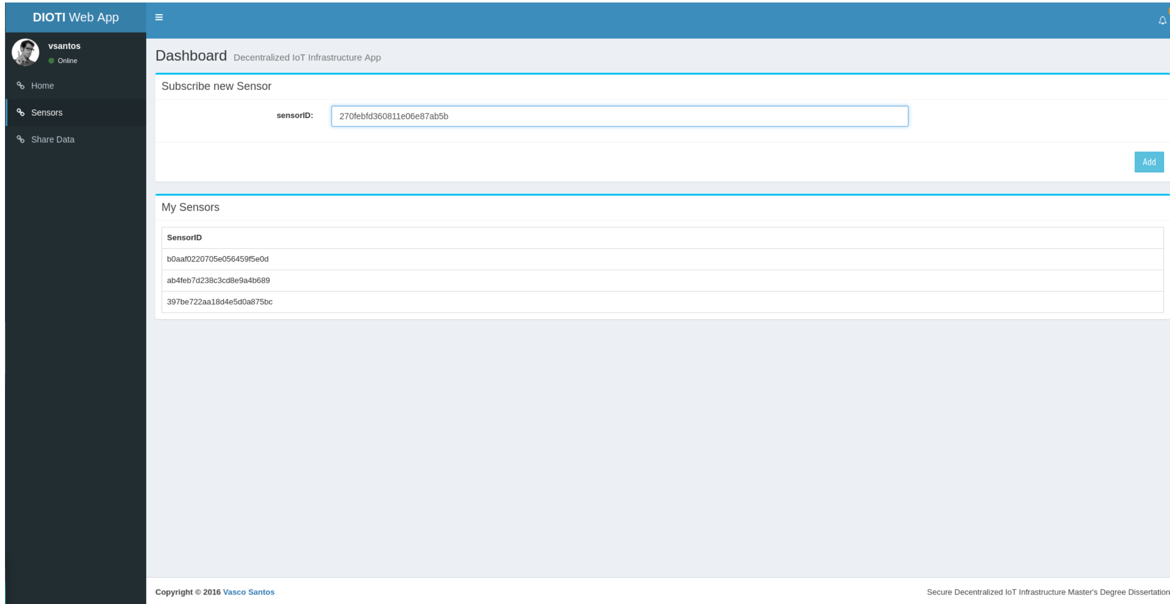


Figure 6.6: Sensor management Interface.

For data sharing, as well as data unsharing the user has to select the `Share Data` option. In this page, the entity may share data of a specific sensor with another entity, as well as unshare data with a entity, who currently has access to it. In Figure 6.7, it is illustrated a data share of the sensor `"397be722aa18d4e5d0a875bc"` to the entity `"maria10"`. As a consequence of any entity having access to this sensor yet (besides its owner), there is no users to unshare the sensor data with.

As a result of the previous data share, the entity `"maria10"` receives a notification of data sharing, as illustrated in Figure 6.8. The entity may accept or refuse this data sharing.

In the case of a data unsharing, the entity that no longer has access to a sensor data receives a notification, as depicted in Figure 6.9.

Finally, it is possible to analyze the data history of the sensors. In the main page of the dashboard, the user may click on a sensor to get their data history. The user is redirected to another page and it is presented a chart containing the received data over the time, as exemplified in Figure 6.10. This chart is updated in real-time, as new data of the sensor is received by the infrastructure.

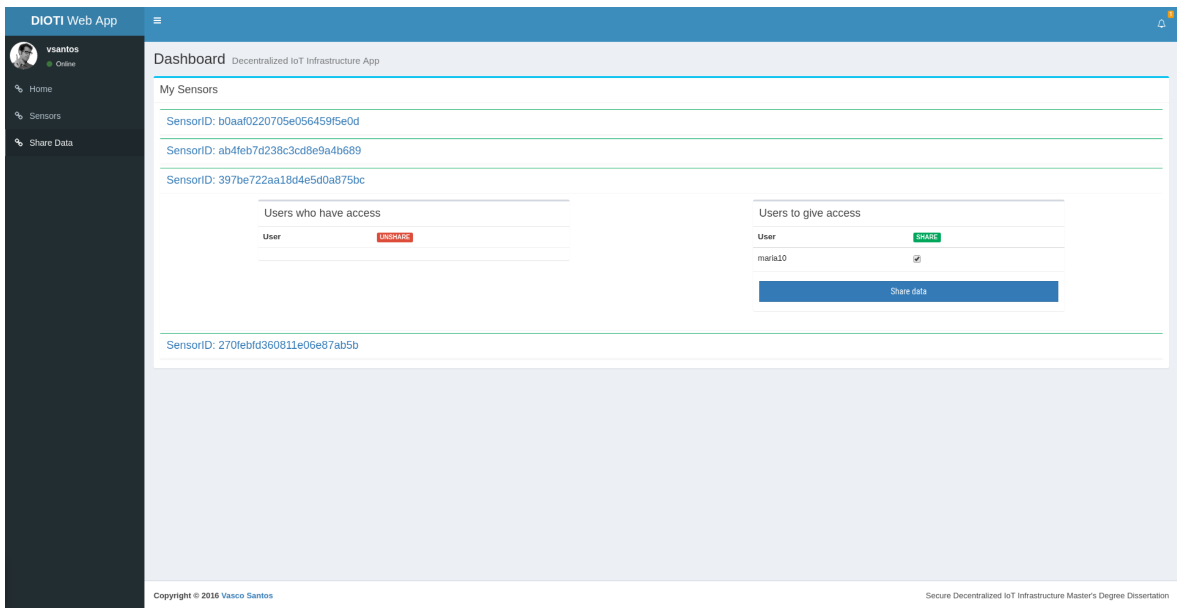


Figure 6.7: Share Data Interface.

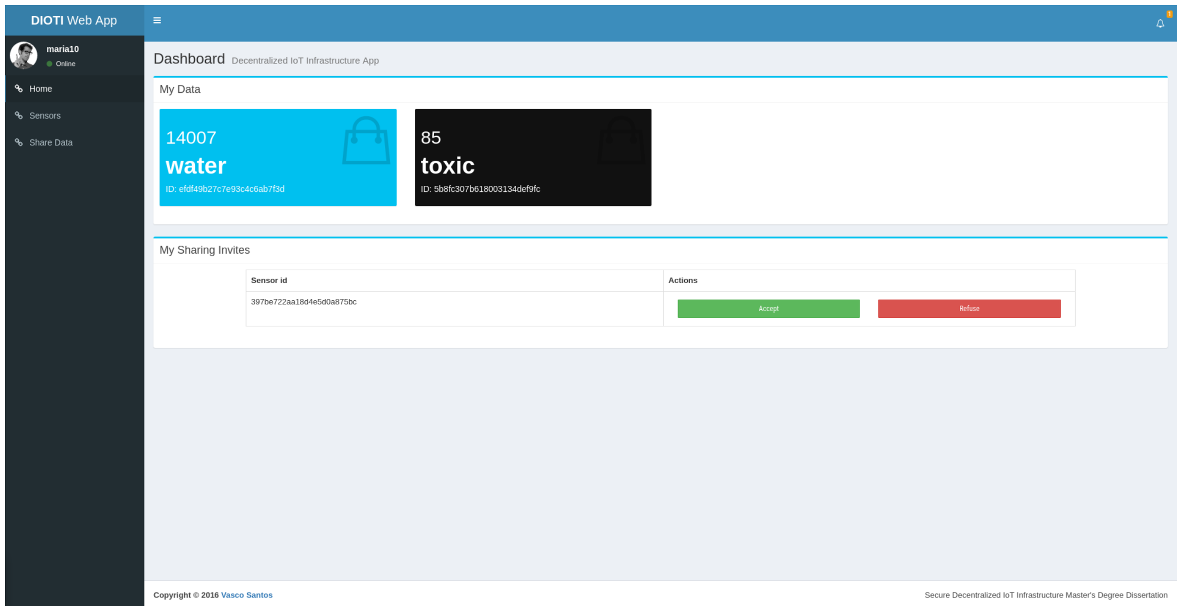


Figure 6.8: Main interface of the user.

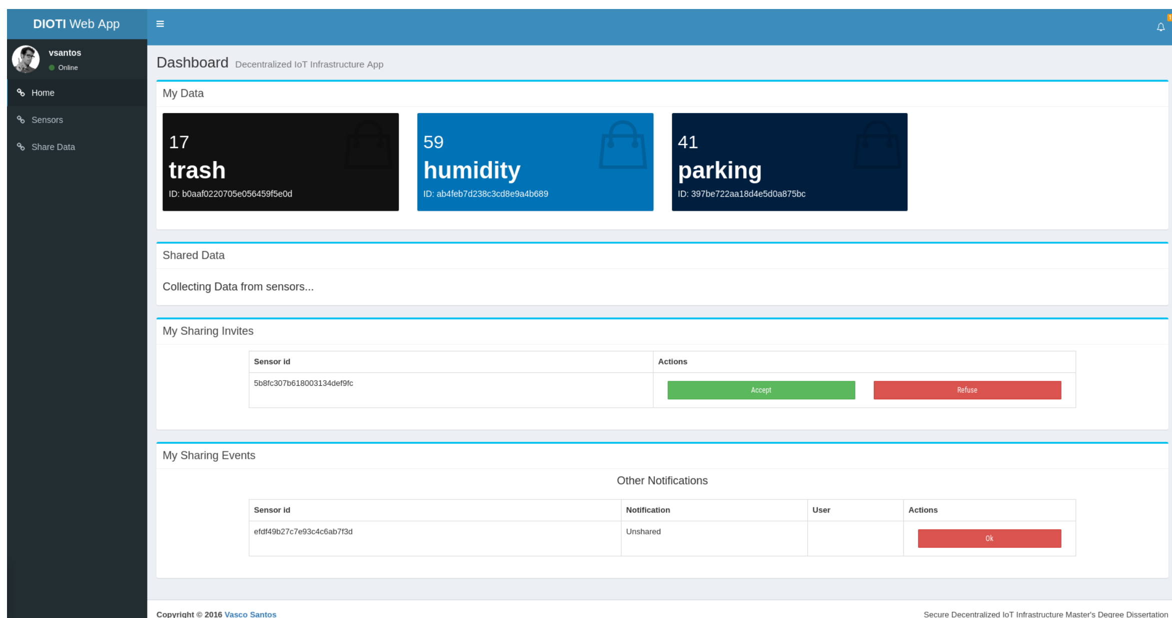


Figure 6.9: Sensor management Interface.

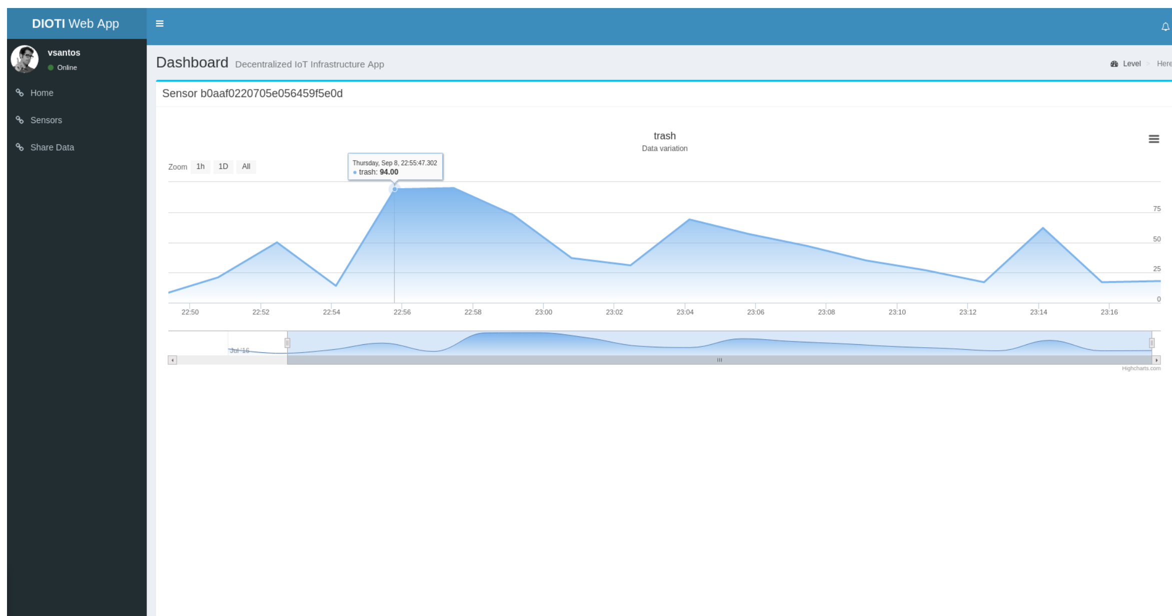


Figure 6.10: Data History of a Sensor Interface.

6.3 DEPLOYMENT SCENARIO

With a view to simulate a real scenario for testing the proposed infrastructure, or at least a sequence of events close to what might be real, a scenario was designed, which is composed of seven Gateway Peers, a Register Server, two Sensor Simulators and a Web Application. It was used to analyze and test the infrastructure, regarding its non-functional requirements, previously specified in section 4.3. Briefly, the tests presented essentially aim to validate the infrastructure interoperability, performance, security and scalability.

Considering a 7 days test scenario in a local network, the infrastructure received messages from the simulators, where each sensor had its own periodicity varying between 20 and 150 seconds. The simulator that uses the MQTT protocol for communicating with the infrastructure contains 66 sensors, while the the simulator that uses the HTTP has 77 sensors. The first simulator sent a total of 998889 messages, while the second one provided 908445 data messages.

6.3.1 NETWORK DEPLOYMENT

Aiming to deploy the designed scenario in the network, ten machines were used. Four of them (Peer 2, 3, 5 and 6) are RaspberryPi's, which have hardware limitations and consequently, do not have persistence, nor do they have data stream services enabled. In addition, one machine is used for the simulators, as well as one for the Register Server and one for the Web Application. The remaining ones are virtual machines with capabilities to have all the services enabled. The described scenario is illustrated in Figure 6.11.

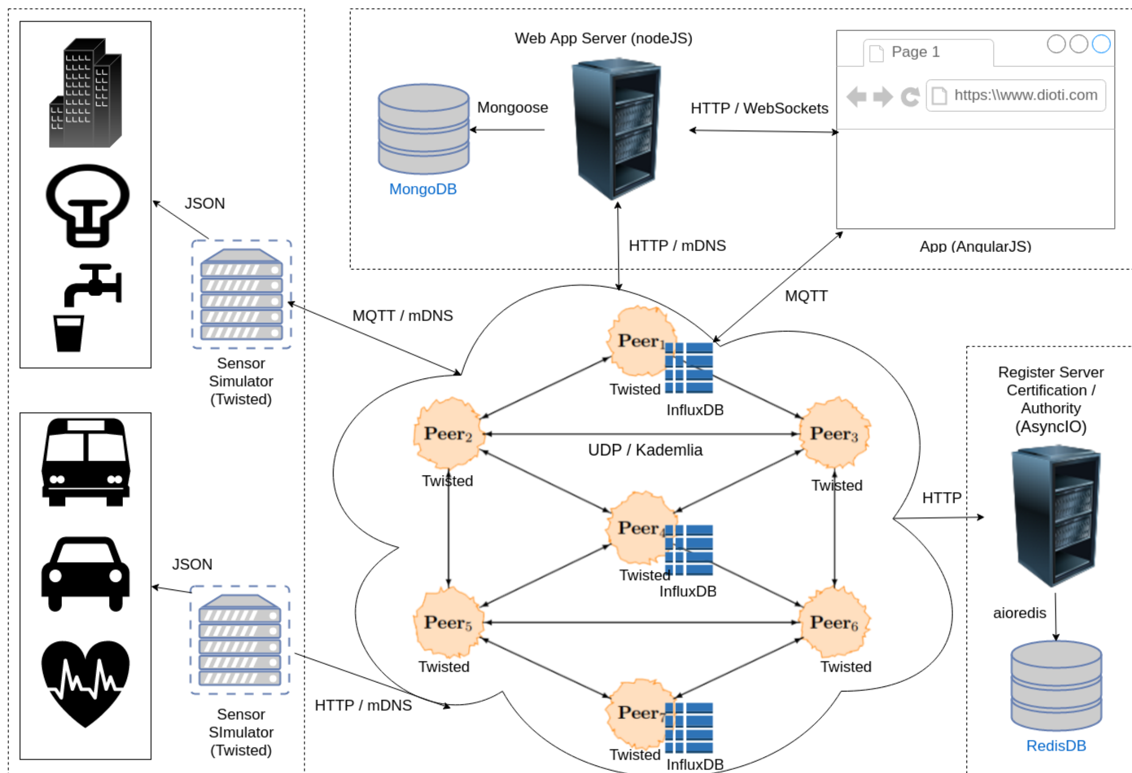


Figure 6.11: Prototype Scenario Diagram.

The specifications of the machines, which were used are discriminated in Table 6.1. For a better understanding, the services column represent the list of services enabled by the peer (A - Access, C - Collector, M - Management, P - Persistence, S - Stream).

VM	vCPU	MEM (MB)	Storage (GB)	Services
Peer1	1	1024	25	A, C, M, P
Peer4	1	1024	25	A, C, M, P, S
Peer7	1	2048	25	A, C, M, P, S
Register Server	1	512	10	NA
Simulators	1	1024	25	NA
Web Application	1	512	10	NA

Table 6.1: Virtual Machines' Specifications.

The specifications of the RaspberryPy's used are detailed in Table 6.2.

RaspberryPi	CPU	MEM (MB)	Storage (GB)	Services
Peer2	1	512	16	A, C, M
Peer3	1	512	16	A, C, M
Peer4	1	512	16	A, M
Peer6	1	512	16	C, M

Table 6.2: RaspberryPis' Specifications.

6.3.2 PERFORMANCE EVALUATION

Taking into account the distributed and heterogeneous environment of the proposed infrastructure, it is essential to build efficient software. Each peer of the overlay was monitored during this test scenario, in order to obtain their CPU and Memory usage over the time. During the test scenario, all these measures were obtained each 60 seconds, using psutil (Python System and Process Utilities). Therefore, the results obtained for Peer1, Peer2 Peer4 and Peer7 are presented as follows.

Starting with the CPU, Figure 6.12 shows the hourly average CPU usage along the experience. From the traces of Peer1 and Peer4, it is possible to verify that the Persistence service is heavy, as a result of the need to have a local database running. It is important to notice that InfluxDB recommends a minimum of 2-4 CPUs, and it was used only one. Moreover, Peer4 uses more CPU than Peer1 as a consequence of having the broker process running. Peer2 has an average CPU usage of 42%, as a result of the lower hardware requirements of its enabled services.

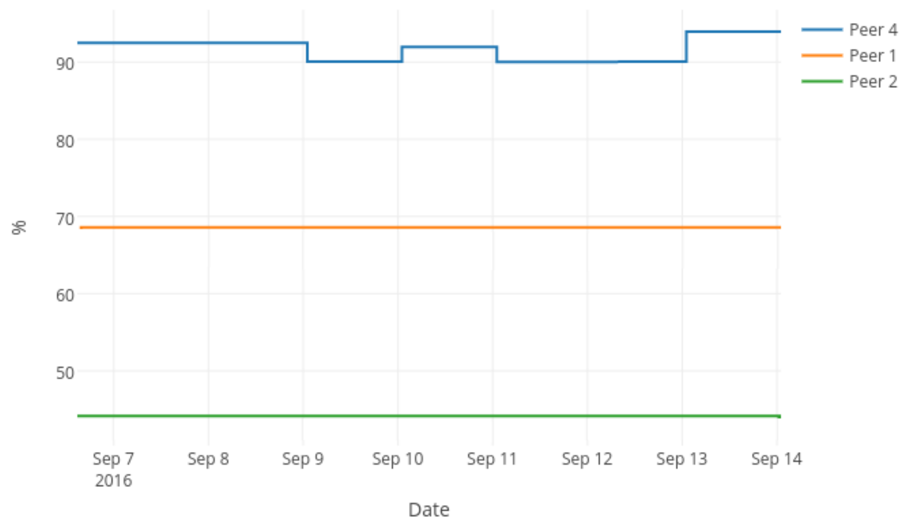


Figure 6.12: Percentage of CPU usage.

Regarding memory, the graph that presents the hourly average memory usage along the tests is presented in Figure 6.13. Taking into consideration the Memory graphs, Peer1 presents a growth in the memory consumption over the time. As in the CPU, it is important to consider that InfluxDB recommends a minimum of 2-4 GB of RAM, and Peer1 has only 1GB. Moreover, peer7 maintained its memory consumption as a consequence of having 2GB of memory available. Peer2 maintained its memory consumption, from 36% to 48% during all the test.

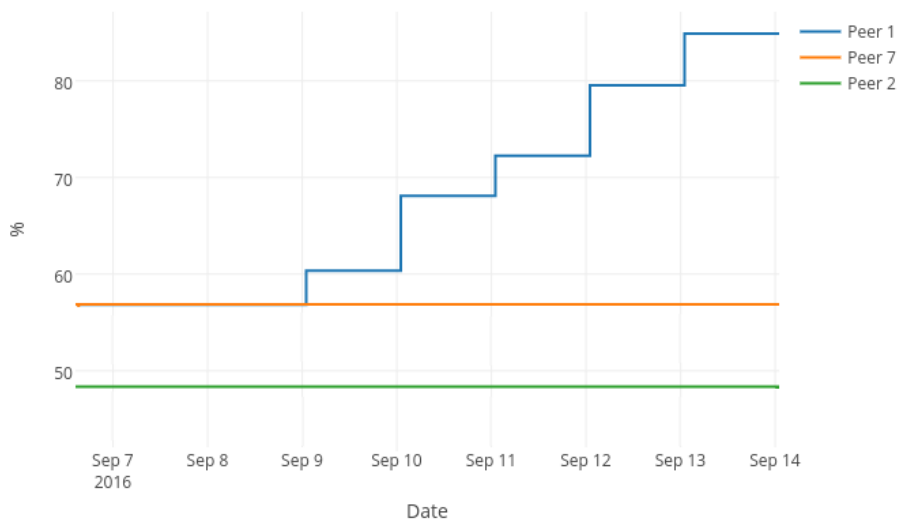


Figure 6.13: Percentage of Memory in Peers.

In the final analysis, we may conclude that the proposed infrastructure provides a satisfying efficiency for a IoT deployment, which will be composed of devices with different capabilities. This results from the developed architecture, which is divided into different services that may be enabled according to the hardware constraints of each peer.

6.3.3 COMMUNICATION PROTOCOLS EVALUATION

An important aspect of a IoT infrastructure is the response time of the communication protocols, which are used to communicate with the infrastructure. In this test scenario were transmitted 908445 MQTT messages and 998889 HTTP messages. For each message, the round-trip time of the request was measured. The obtained results are presented in Figures 6.14 and 6.15. It illustrates the response time of each message that was sent to the infrastructure over time.

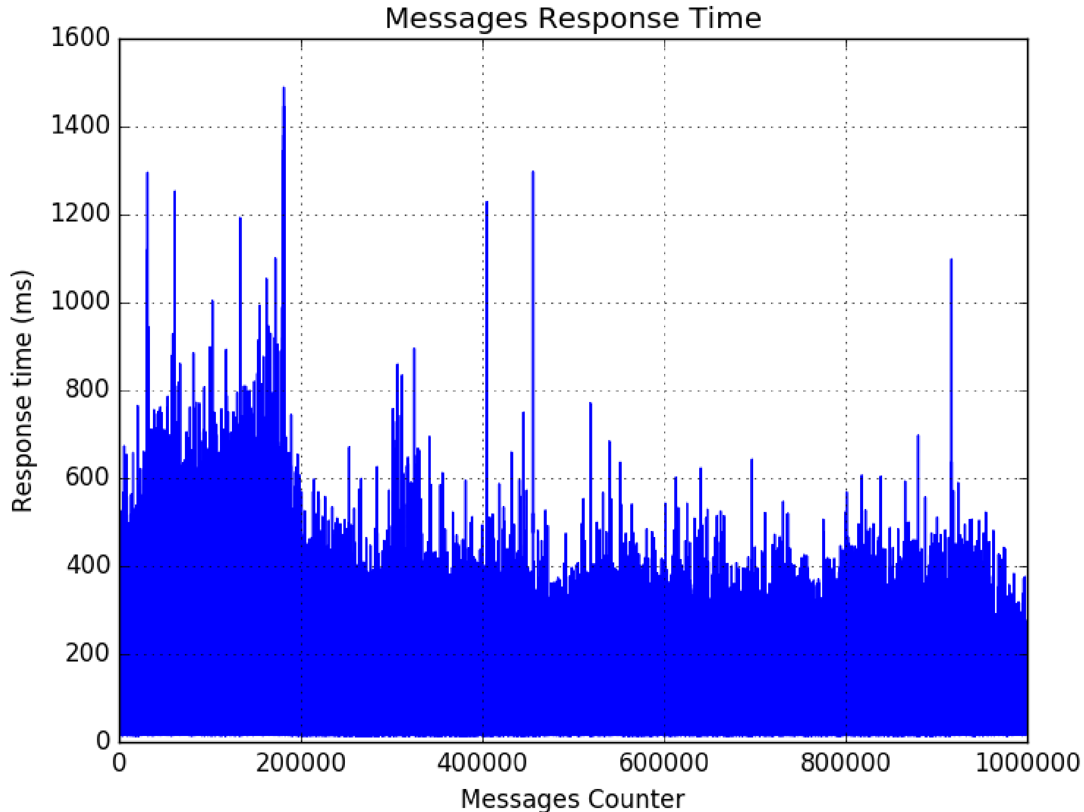


Figure 6.14: Response time of HTTP messages over time.

Analyzing the results presented, the average round-trip time for the HTTP was approximately 116.2ms, while for the MQTT was about 80.58ms. Thanks to its connection-oriented nature, MQTT provided better and uniform results, with a standart deviation of about 7. In contrast, HTTP presented more disperse response times, with a standart deviation of approximately 101. As a consequence of the asynchronous nature of the peers, in rare occasions the obtained response time is considerable high. This occurs when the peer receives a considerable number messages in a short period of time.

It is expected that in the Internet scale, the response times will increase. However, as it is illustrated in the presented figures, the response times will be continuous over time, as a result of the use of an efficient time-series database. In addition, it is predicted that those response times will not compromise the real time requirements of critical systems.

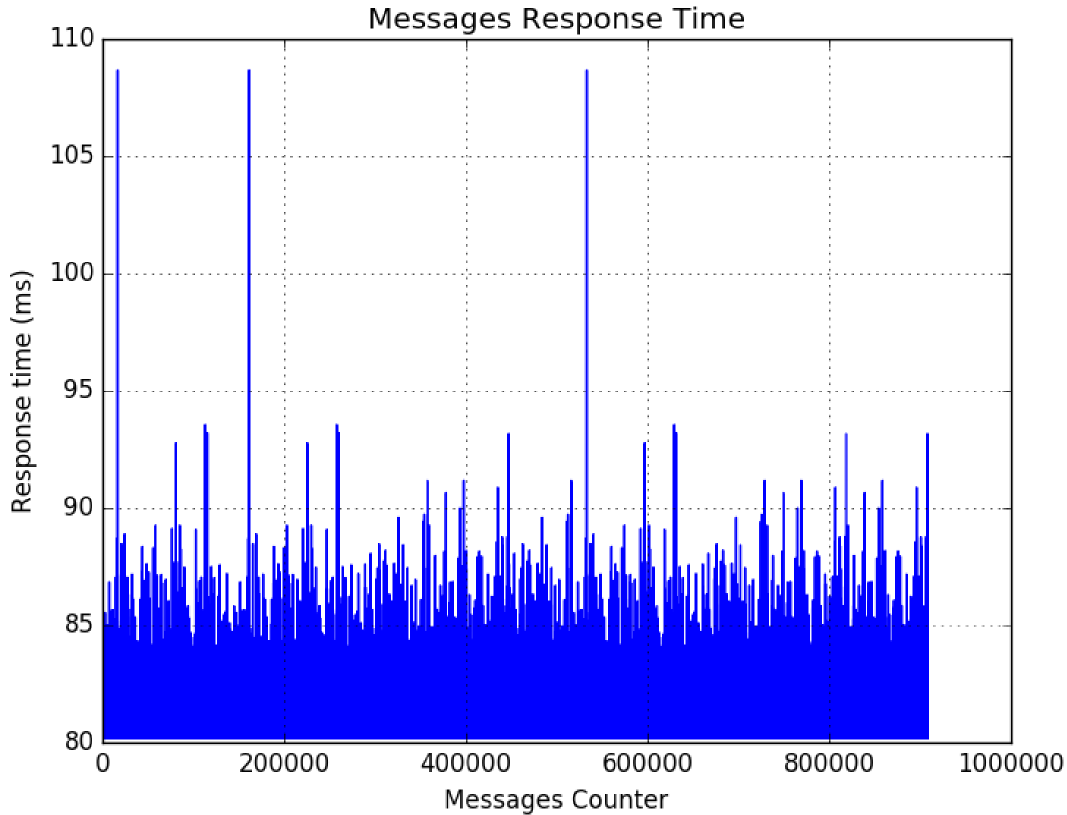


Figure 6.15: Response time of MQTT messages over time.

6.3.4 DATA SECURITY EVALUATION

In the context of the security and privacy requirements defined for this infrastructure, it is important to verify the security of the data during its flow and storage. For this test, an user bound a sensor with identifier `"397be722aa18d4e5d0a875bc"`. Afterwards, the simulator was started, so that the infrastructure may store data from the bound sensor.

Aiming to analyze the stored data, it was started the InfluxDB shell. In this shell, it was executed a query to get the data of the bound sensor. As illustrated in Figure 6.16, it was executed the query `select * from "397be722aa18d4e5d0a875bc"`. For the presented query results, it is possible to verify that for each timestamp, there is a list containing one code and the sensor data, properly ciphered. The code consists of a symmetric key, appended with a IV, a salt and a HMAC code, ciphered with the user's public key, which is needed to decipher the data cryptogram, as well as to verify its integrity. For this evaluation, it will be considered the last received data from this sensor (first entrance of the query).

Using google chrome, the user started the web application, which requested to the infrastructure the last received data of the previously bound sensor. Using the developer tools of the browser, as presented in Figure 6.17, the request contains an `"accessData"` parameter. This parameter is composed of the sensor identifier, as well as a signature, in order to verify the authenticity of the request. In other words, this data is used for the access control mechanism of the access service.

```
> select * from "397be722aa18d4e5d0a875bc"
name: 397be722aa18d4e5d0a875bc
------------------------------

time codes data distribution

1474041990380952440 ['QYilPHm/iQKlgDw5zNByQAb4nfyYha4ZDAZwt2SboBBtMRCNNdA7EvFtow7kJe+Dhz
ufeyID9S+PW++TxzjysTX/jEe4e+loxg6znDnoYiN2VggggPLBCMogW1Marnp/boWPSe0IR6drWt1lHCp9DKzSzr
TSS8PrW9FpqtmqxkmxAT9WL8eULbmlgkmZ08k3onwp2k8kX6pN9wxPRCb
ZafL8EmVgK4KOI5qIV2B6oImxvXYvbF3aIYScdVUdhkETcWDZ5RL/NwBFwHBYtahsagkF3CNacLPuMpVUfq8JFfS
o+5xyTv5gqJw9Tee/nJrLJkNZWbGXrMIUmTPJpwgaqA=='] mHH6PJXLqQWnY9h5xBgQKNTuZJHefBaCwgoKoB/V
n8xvgjfteuX9Hr9VoAki+zhA True

1474042030287161300 ['pHcJBH3IgbZqJuLUdg9qrHDEoXIrDHTTgDoVZe+a7K/bTPpJURJTSw7+XKINOAQSc7
qLfB6c+F6kMhTVImrLPgubHJWN9aITGBdIoKAJwZRFYvuhJKMxfMXUSZGeAVcLIkr5QFpBtE2JkBeoRAMaQUN64Q
aXgReI8JRWfhAk70p7JK9TMatowMbFWhJJeIWeAp6GqI2qIIG3n02BWE5
ldZECTNLA6ILHR38qkWVrB3L33yh3NxIPQwecsDuQH3Jh9CIUEsOWOsJywf30hQL34yXRY6WhmnuARaIQoCLXUvr
s9ZDkSBIQrxJJ/o3mgPpSurNUNdwrLBLfsOUNaLUalw=='] vAQQnslHwsOX/HZNXqaZZqR5ADEWKEIn56TOh7cq
KJKO+HI7OIQAqqE/g5MLmOnT True

1474042070293703140 ['XJOSSs/b9KhuDqmPvRgTgQV3C7ucVrJ29pCwhfcrBbKlrdr5YEFFwS8gvZF6LIOoOJ
FV5XwXmPMJVpIGMYd5GkbkIneW6+u96oBuUJaXLEVNRYAn/spthmQUVS62e2FcWKgxSURNbsxcDt6qtZZt8lCHCf
6g6+Q5bvCLePm3E93r44NmhJ5WTpJswPLoCblQC4l4n2oD8JCrdnElevc
SuSF7otnZGOa2cfVWPvDl9ZealTb3We069Z93ve4S9ec/mKB8A20amVtEtv3gGpkWlZqXhdfGlSbEl8paZ+uBEJM
SknEduaFmg2acd4JH+RetOYshOtKXKPJs5RK9h7S2fw=='] JAsGBcktGON/ZTlykwNL3LZTfvaPw7gUTEJsXI8b
ZMEZKTgJY8bP9F7n9Wb3re+n True
```


Figure 6.16: InfluxDB query for bound sensor.

× Headers Preview Response Cookies Timing

▼ General

Request URL: http://localhost:3000/data/getCurrentData?accessData=ewAiAHAAcwBlAHUAZABvAG4AeQBtACIAOg
YAZAA4AGUAZQBhADQAZgBmADgAMwA0ADQAMgAyADAANQA3AGIAZABmAGQAMAA0ADMAOABlAGIAIgB9AA%3D%3D&sensorId=397b
tMue4avItnMKepwg6Yp9wUJSmBZtY2EiqYo%2F%2BdhIFcS8fZosz5%2ByDuDf3IMlaxKx3qtBnqLbqhZtsbEhHJyzRRfEqAIQb
PpiKjfZ%2FHpxHJZmjINo6QkLOarryLz7E%2BKgr0CkcdAbRcghHwvbWY1Xbp5FtrgYsjisJLYABaMhJ%2B8HeR%2FA%3D%3D

Request Method: GET

Status Code: 304 Not Modified

Remote Address: [::1]:3000

▼ Response Headers view source

Connection: keep-alive

Date: Fri, 16 Sep 2016 19:18:54 GMT

ETag: W/"1b7-c1f5d840"

X-Powered-By: Express

▼ Request Headers view source

Figure 6.17: Sensor data request.

Moreover, using the developer tools, it was also possible to verify what was received from the infrastructure. As illustrated in Figure 6.18, the received data is equal to the first entry stored in the database, which was shown in Figure 6.16. The received data is decrypted on the client side, using the user's private key to decrypt the code, which will be used to decipher the data.

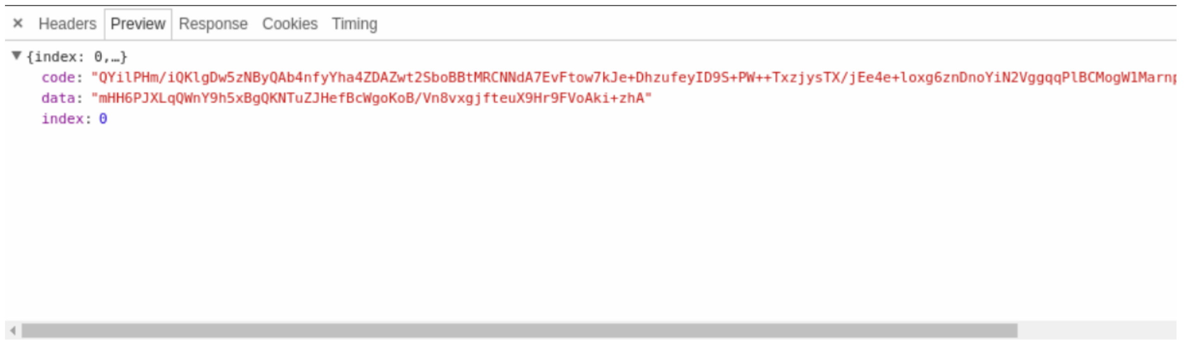


Figure 6.18: Sensor data response.

6.3.5 FINAL OVERVIEW

All in all, the results obtained prove that the requirements of the designed solution were accomplished. The infrastructure can adapt its behavior according to the peers constraints, resulting in using the available hardware in an efficient manner. In addition, it maintains its performance while receiving data from simulators, as well as data accesses and streams from the application. Furthermore, the proposed infrastructure is fault tolerant and scalable thanks to its decentralized architecture and has its data replicated through the network, as a benefit of the chosen database.

Regarding security, it was shown that the infrastructure is keeping the data properly encrypted while stored and during its flow, as well as maintaining its anonymity and integrity. The infrastructure authenticates the requests of the users and pass them through a data access control.

Nonetheless, it would be important to test this infrastructure in a uncontrolled environment, composed of thousands of peers and WSNs, in order to verify the real scalability of the infrastructure.

CHAPTER 7

Conclusions

In this chapter, a brief overview of the work done is presented, as well as its academic contributions. It is also discussed how the proposed infrastructure can evolve.

7.1 Final considerations

Nowadays, the IoT infrastructures are built on top of centralized architectures [86]. This type of architectures results in an infrastructure that is not scalable nor fault tolerant. In addition, thanks to the data silos created, these infrastructures may have privacy and security problems.

Accordingly, the main objective of this work was to solve the specified problems, taking advantage of a decentralized architecture. Consequently, this architecture ensures a scalable and fault tolerant infrastructure, as well as the nonexistence of data silos. We essentially aimed to interconnect multiple business processes and use cases in a single infrastructure. Thus, a global decentralized infrastructure has potential to reduce the infrastructure and maintenance costs, as well as to enhance the privacy of its clients, as a result of their data being encrypted and dispersed over the network.

Some other problems resulted from this architectural decision, such as the different requirements of each business process, real-time data acquisition, as well as the heterogeneous environment of sensors, regarding their communication protocols and hardware capacity. In the context of this dissertation, all this problems were studied and properly solved.

This dissertation consists of a proof of concept that matches all the specified requirements. Consequently, it is a good initial solution for a global IoT infrastructure. This results from the current state of the IoT, where devices are intended to be inexpensive and lightweight, as well as the current state of blockchain, which does not scale to the Internet level. However, some research is currently being done, in order to scale the blockchain. Whenever this is possible, combining a scalable blockchain with the enhancement of devices' computing capabilities, will for sure provide a better infrastructure, as a result of the possibility of end-to-end encryption, which is not supported by the proposed infrastructure.

Regarding the objectives and requirements of this dissertation, all of them were properly accomplished. This work contributes for the decentralized architectures in IoT research, regarding several fields. Firstly, it was designed a flexible solution, considering the heterogeneous environment of the

IoT. The proposed solution also provides a decentralized publish-subscribe mechanism for real-time data. Finally, it was integrated a time-series database cluster inside the overlay network, which allows for efficient data insertions and queries, as well as data security and privacy.

7.2 FUTURE WORK

In spite of this dissertation fulfilling all the specified objectives and requirements, there are several aspects to explore that would allow its further enhancement. The following list identifies some of the most relevant aspects that should be considered in the future:

- **Broker Implementation**: implement a broker integrated in the Python Twisted Application, in order to remove the overhead of the communication between processes and enhance the synchronization among peers;
- **Certification**: evaluate the Pretty Good Privacy Algorithm for replacing the hierarchical certification, in the event of achieving the real-time requirements and maintaining the performance of the infrastructure;
- **Sharing Permissions**: allow sensor data-sharing with different permissions, such as permission for receiving data, permission for sharing with other users, permission for retrieving data between specific time intervals, among others;
- **Software Updates**: the infrastructure may propagate software updates through its peers, so that new features may be provided to the applications by updating the installed software;

Some other aspects could be considered, in order to transform this work in a product. It would be necessary to elaborate a business process, where for example, users would have to deploy a public gateway, for each set of n sensors that they intend to bind.

References

[1] A. Thierer and A. Castillo, "Projecting the growth and economic impact of the internet of things", George Mason University, 2015. [Online]. Available: http://mercatus.org/sites/default/files/IoT-EP-v3.pdf.

[2] A. Greenberg, *Hackers remotely kill a jeep on the highway—with me in it*, 2015. [Online]. Available: http://www.wired.com/2015/07/hackers-remotely-kill-jeep-highway/.

[3] D. Ackerman, *Samsung smart tv's voice recognition creates privacy concerns*, 2015. [Online]. Available: http://www.cbsnews.com/videos/samsung-smart-tvs-voice-recognition-creates-privacy-concerns/.

[4] M. Blackstock and R. Lea, Iot interoperability : a hub-based approach:79–84, 79–84, 2014.

[5] S. Kubler, M.-j. Yoo, C. Cassagnes, and K. Fr, Opportunity to leverage information-as-an-asset in the iot – the road ahead, (September 2014), 2015. DOI: 10.1109/FiCloud.2015.63.

[6] A. Riahi, Y. Challal, E. Natalizio, Z. Chtourou, and A. Bouabdallah, A systemic approach for iot security:351–355, 351–355, 2013. DOI: 10.1109/DCOSS.2013.78.

[7] M. Floeck, A. Papageorgiou, A. Schuelke, and J. Song, Horizontal m2m platforms boost vertical industry : effectiveness study for building energy management systems:15–20, 15–20, 2014.

[8] V. Gazis, M. Goertz, M. Huber, A. Leonardi, K. Mathioudakis, A. Wiesmaier, and F. Zeiger, Short paper : iot : challenges , projects , architectures:145–147, 145–147, 2015.

[9] Q. H. Vu, M. Lupu, and B. C. Ooi, *Peer-to-Peer Computing: Principles and Applications*. Berlin, Germany: Springer, 2010.

[10] W. Alex, *The internet of things is revolutionising our lives, but standards are a must*, 2015. [Online]. Available: http://www.theguardian.com/media-network/2015/mar/31/the-internet-of-things-is-revolutionising-our-lives-but-standards-are-a-must.

[11] Cisco, *Cisco visual networking index: forecast and methodology, 2014-2019*, 2015. [Online]. Available: http://www.cisco.com/c/en/us/solutions/collateral/service-provider/ip-ngn-ip-next-generation-network/white_paper_c11-481360.pdf.

[12] J. Greenough, *How the 'internet of things' will impact consumers, businesses, and governments in 2016 and beyond*, 2015. [Online]. Available: http://www.businessinsider.com/how-the-internet-of-things-market-will-grow-2014-10.

[13] Oracle, *Energize your business with iot enabled applications*, 2015. [Online]. Available: http://tamarafranklin.com/wp-content/uploads/2015/09/Oracle-Internet-of-Things-Cloud-Service_RGB.pdf.

[14] J. Bradley, J. Barbier, and D. Handler, *Embracing the internet of everything to capture your share of $14.4 trillion*, 2015. [Online]. Available: https://www.cisco.com/web/about/ac79/docs/innov/IoE_Economy.pdf.

[15] EMC, *The digital universe of opportunities*, 2014. [Online]. Available: http://www.emc.com/collateral/analyst-reports/idc-digital-universe-2014.pdf.

[16] H. Schaffers, N. Komninos, M. Pallot, and B. Trousse, Smart cities and the future internet : towards cooperation frameworks for open innovation:431–446, 431–446.

[17] M. Wang, G. Zhang, C. Zhang, J. Zhang, and C. Li, An iot-based appliance control system for smart homes:744–747, 744–747, 2013.

[18] Y. Bo and H. Wang, The application of cloud computing and the internet of things in agriculture and forestry:168–172, 168–172, 2011. DOI: 10.1109/IJCSS.2011.40.

[19] S. Fang, L. Da Xu, Y. Zhu, J. Ahati, H. Pei, J. Yan, and Z. Liu, An integrated system for regional environmental monitoring and management based on internet of things, *IEEE Transactions on Industrial Informatics*, 10(2):1596–1605, 1596–1605, 2014.

[20] L. Atzori, A. Iera, and G. Morabito, The internet of things: A survey, *Computer networks*, 54(15):2787–2805, 2787–2805, 2010.

[21] Q. Wang and I. Balasingham, *Wireless sensor networks - an introduction*, 2010. [Online]. Available: http://cdn.intechweb.org/pdfs/12464.pdf.

[22] C. Alcaraz, P. Najera, J. Lopez, and R. Roman, Wireless sensor networks and the internet of things: Do we need a complete integration?, in *1st International Workshop on the Security of the Internet of Things (SecIoT'10)*, 2010.

[23] M. Chen, J. Wan, X. Liao, S. Gonzalez, and V. Leung, "A survey of recent developments in home m2m networks", Vol 16, 98-114, 2013. [Online]. Available: http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6674156&newsearch=true&queryText=A%20Survey%20of%20Recent%20Developments%20in%20Home%20M2M%20Networks.

[24] X. Zhang, Z. Wen, Y. Wu, and J. Zou, The implementation and application of the internet of things platform based on the rest architecture, *BMEI 2011 - Proceedings 2011 International Conference on Business Management and Electronic Information*, 2:43–45, 43–45, 2011. DOI: 10.1109/ICBMEI.2011.5917838.

[25] D. Thangavel, X. Ma, A. Valera, H. X. Tan, and C. K. Y. Tan, Performance evaluation of mqtt and coap via a common middleware, *IEEE ISSNIP 2014 - 2014 IEEE 9th International Conference on Intelligent Sensors, Sensor Networks and Information Processing, Conference Proceedings*, (April):21–24, 21–24, 2014, ISSN: 978-1-4799-2843-9. DOI: 10.1109/ISSNIP.2014.6827678.

[26] D. Locke, Mq telemetry transport (mqtt) v3. 1 protocol specification, *IBM developerWorks Technical Library], available at http://www. ibm. com/developerworks/webservices/library/ws-mqtt/index. html*, 2010.

[27] Z. Shelby, K. Hartke, and C. Bormann, "The constrained application protocol (coap)", Tech. Rep., 2014.

[28] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, Internet of things: A survey on enabling technologies, protocols, and applications, *IEEE Communications Surveys & Tutorials*, 17(4):2347–2376, 2347–2376, 2015.

[29] S. Bandyopadhyay and A. Bhattacharyya, Lightweight internet protocols for web enablement of sensors using constrained gateway devices, in *Computing, Networking and Communications (ICNC), 2013 International Conference on*, IEEE, 2013, 334–340.

[30] K. Kuladinithi, O. Bergmann, T. Pötsch, M. Becker, and C. Görg, Implementation of coap and its application in transport logistics, *Proc. IP+ SN, Chicago, IL, USA*, 2011.

[31] B. C. Villaverde, R. D. P. Alberola, A. J. Jara, S. Fedor, S. K. Das, and D. Pesch, Service discovery protocols for constrained machine-to-machine communications, *IEEE communications surveys & tutorials*, 16(1):41–60, 41–60, 2014.

[32] P. B. Gibbons, B. Karp, Y. Ke, S. Nath, and S. Seshan, Irisnet: An architecture for a worldwide sensor web, *IEEE pervasive computing*, 2(4):22–33, 22–33, 2003.

[33] A. T. Campbell, S. B. Eisenman, N. D. Lane, E. Miluzzo, and R. a. Peterson, People-centric urban sensing, *Proceedings of the 2nd annual international workshop on Wireless internet - WICON '06*:18–31, 18–31, 2006, ISSN: 1089-7801. DOI: 10.1145/1234161.1234179.

[34] Q. Liang, X. Cheng, and D. Chen, Opportunistic sensing in wireless sensor networks: theory and application, *2011 IEEE Global Telecommunications Conference - GLOBECOM 2011*, 63(8):1–5, 1–5, 2011, ISSN: 1930-529X. DOI: 10.1109/GLOCOM.2011.6134471.

[35] a. Rowe, M. E. Berges, G. Bhatia, E. Goldman, R. Rajkumar, J. H. Garrett, J. M. F. Moura, and L. Soibelman, Sensor andrew: large-scale campus-wide sensing and actuation, *IBM Journal of Research and Development*, 55(1.2):6:1–6:14, 6:1–6:14, 2011, ISSN: 0018-8646. DOI: 10.1147/JRD.2010.2089662.

[36] O. Akribopoulos, I. Chatzigiannakis, C. Koninis, and E. Theodoridis, A web services-oriented architecture for integrating small programmable objects in the web of things, in *Developments in E-systems Engineering (DESE), 2010*, IEEE, 2010, 70–75.

[37] S. Wahle, T. Magedanz, and F. Schulze, Demonstration of openmtc – m2m solutions for smart cities and the internet of things:3–5, 3–5.

[38] M. Corici, H. Coskun, A. Elmangoush, A. Kurniawan, T. Mao, T. Magedanz, and S. Wahle, Openmtc: Prototyping machine type communication in carrier grade operator networks, in *2012 IEEE Globecom Workshops*, IEEE, 2012, 1735–1740.

[39] J. Shneidman, P. Pietzuch, J. Ledlie, and et al, Hourglass: an infrastructure for connecting sensor networks and applications, *Harvard Technical Report TR-21-04*, 2004. [Online]. Available: http://www.eecs.harvard.edu/syrah/hourglass/papers/tr2104.pdf.

[40] I. Chatzigiannakis, C. Koninis, G. Mylonas, U. Colesanti, and A. Vitaletti, A peer-to-peer framework for globally-available sensor networks and its application in building management, in *2nd international workshop on sensor network engineering (IWSNE 2009)*, 2009.

[41] IBM, Empowering the edge - practical insights on a decentralized internet of things, 2015. [Online]. Available: http://www-935.ibm.com/services/multimedia/GBE03662USEN.pdf.

[42] P. Maymounkov and D. Mazières, Kademlia: a peer-to-peer information system based on the xor metric, *Revised Papers from the First International Workshop on Peer-to-Peer Systems*:891–921, 891–921, 2002. [Online]. Available: https://pdos.csail.mit.edu/~petar/papers/maymounkov-kademlia-lncs.pdf.

[43] B. Panikkar, S. Nair, P. Brody, and V. Pureswaran, *Adept: An iot practitioner perspective*, 2014.

[44] T. Mcconaghy, Blockchain, throughput, and big data, 2014. [Online]. Available: http://trent.st/content/2014-10-28%20mcconaghy%20-%20blockchain%20big%20data.pdf.

[45] K. Croman, C. Decker, I. Eyal, A. E. Gencer, A. Juels, A. Kosba, A. Miller, P. Saxena, E. Shi, and E. Gün, On scaling decentralized blockchains, *Proc. 3rd Workshop on Bitcoin and Blockchain Research*, 2016. [Online]. Available: http://fc16.ifca.ai/bitcoin/papers/CDE+16.

[46] T. Dunning and E. Friedman, Time series databases, 2015.

[47] T. Pelkonen, S. Franklin, J. Teller, P. Cavallaro, Q. Huang, J. Meza, and K. Veeraraghavan, Gorilla: a fast, scalable, in-memory time series database, 2015. [Online]. Available: http://www.vldb.org/pvldb/vol8/p1816-teller.pdf.

[48] D. Namiot, Time series databases, 2015. [Online]. Available: https://pdfs.semanticscholar.org/bf26/5b6ee45d814b3acb29fb52b57fd8dbf94ab6.pdf.

[49] Q. Gu and S. Marcos, Denial of service attacks department of computer science texas state university – san marcos school of information sciences and technology pennsylvania state university denial of service attacks outline:1–28, 1–28.

[50] D. Miorandi, S. Sicari, F. D. Pellegrini, and I. Chlamtac, "Internet of things: vision, applications and research challenges", Ad Hoc Networks, 2012. [Online]. Available: https://www.researchgate.net/publication/235642082_Internet_of_Things_Vision_Applications_and_Research_Challenges.

[51] A. Gupta and L. Awasthi, "Peer-to-peer networks and computation: current trends and future perspectives", Computing and Informatics, Vol 30, 559-594, 2011. [Online]. Available: http://www.cai.sk/ojs/index.php/cai/article/viewFile/184/155.

[52] J. Kangasharju, *Peer-to-peer networks*. [Online]. Available: https://www.cs.helsinki.fi/u/jakangas/Teaching/PrintOuts/08s-P2P-01-Introduction.pdf.

[53] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong, Freenet : a distributed anonymous information storage and retrieval system.

[54] C. D. S. Services, The gnutella protocol specification v0.4.

[55] J. Buford and K. Ross, "P2p overlay design overview", IETF P2P-SIP ad hoc, 2005. [Online]. Available: http://www.cs.uml.edu/~buford/irtf-p2prg/JBufordKRoss-IETF-Overlay-Systems-v4.pdf.

[56] I. Stoica, R. Morris, D. Liben-Nowell, D. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan, Chord: a scalable peer-to-peer lookup service for internet applications, *ACM SIGCOMM Computer Communication*, 31:149–160, 149–160, 2001. [Online]. Available: https://pdos.csail.mit.edu/papers/chord:sigcomm01/chord_sigcomm.pdf.

[57] B. Karp, *Distributed hash tables: chord*. [Online]. Available: http://www0.cs.ucl.ac.uk/staff/B.Karp/gz06/s2012/lectures/gz06-lecture5-dhts.pdf.

[58] A. Payberah and S. Haridi, *Kademlia: a peer-to-peer information system based on the xor metric*. [Online]. Available: https://www.sics.se/~amir/files/download/p2p/kademlia.pdf.

[59] A. Kogan, *Distributed systems (tutorial 7 - kademlia)*. [Online]. Available: http://webcourse.cs.technion.ac.il/236351/Winter2012-2013/ho/WCFiles/236351-win12-tut7.pdf.

[60] A. I. T. Rowstron and P. Druschel, Pastry: scalable, decentralized object location, and routing for large-scale peer-to-peer systems, *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg*:329–305, 329–305, 2001. [Online]. Available: http://research.microsoft.com/en-us/um/people/antr/PAST/pastry.pdf.

[61] J. Song and S. Wang, The pastry algorithm based on dht, *Computer and Information Science*, 2, 2009. [Online]. Available: http://www.ccsenet.org/journal/index.php/cis/article/viewFile/4282/3729.

[62] M. Welzl, *Peer-to-peer systems: dht examples, part 2 (pastry, tapestry and kademlia)*. [Online]. Available: https://heim.ifi.uio.no/michawe/teaching/p2p-ws08/p2p-5-6.pdf.

[63] B. Y. Zhao, L. Huang, J. Stribling, S. C. Rhea, A. D. Joseph, and J. D. Kubiatowicz, Tapestry: a resilient global-scale overlay for service deployment, *IEEE Journal on selected areas in*

Communications, 2004. [Online]. Available: http://www.srhea.net/papers/tapestry_jsac.pdf.

[64] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, A scalable content-addressable network, *Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*:161–172, 161–172, 2001. [Online]. Available: http://conferences.sigcomm.org/sigcomm/2001/p13-ratnasamy.pdf.

[65] A. Popescu, D. Ilie, and D. Kouvatsos, On the implementation of a content-addressable network. [Online]. Available: https://www.diva-portal.org/smash/get/diva2:836192/FULLTEXT01.pdf.

[66] T. Strufe, *Peer-to-peer networks - chapter 3: dht.* [Online]. Available: http://www.p2p.tu-darmstadt.de/fileadmin/user_upload/Group_P2P/share/p2p-ws10/Lecture_3-2.pdf.

[67] B. Zhao, L. Huang, and J. Stribling, Exploiting routing redundancy via structured peer-to-peer overlays, *Proceedings of 11th IEEE international conference on network protocols*, 2003. [Online]. Available: http://oceanstore.net/publications/papers/pdf/tapestry_icnp.pdf.

[68] J. Xu, A. Kumar, and X. Yu, On the fundamental tradeoffs between routing table size and network diameter in peer-to-peer networks, *IEEE J Select Areas Communications*, 2006. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.142.4641&rep=rep1&type=pdf.

[69] Digicert, *White paper: pki - the security solution for the internet of things.* [Online]. Available: https://www.digicert.com/internet-of-things/iot-pki-whitepaper.htm.

[70] M. Anderson, Looking for the key to security in the internet of things, 2014. [Online]. Available: http://spectrum.ieee.org/riskfactor/consumer-electronics/standards/looking-for-the-key-to-security-in-the-internet-of-things.

[71] W. E. Burr, N. A. Nazario, and W. T. Polk, A proposed federal pki using x. 509 v3 certificates, *NIST Gaithersburg*, 1996.

[72] AICPA/CICA, Trust service principles and criteria for certification authorities, 2011. [Online]. Available: http://www.webtrust.org/homepage-documents/item54279.pdf.

[73] G. Caronni, Walking the web of trust, in *Enabling Technologies: Infrastructure for Collaborative Enterprises, 2000.(WET ICE 2000). Proeedings. IEEE 9th International Workshops on*, IEEE, 2000, 153–158.

[74] P. R. Zimmermann, *The official PGP user's guide.* MIT press, 1995.

[75] S. Nakamoto, *Bitcoin: A peer-to-peer electronic cash system*, 2008.

[76] T. McConaghy, D. Jonghe, R. Henderson, R. Marques, T. McConaghy, S. Bellemare, A. Muller, M. Greg, and A. Granzotto, Bigchaindb: a scalable blockchain database, 2016. [Online]. Available: https://www.bigchaindb.com/whitepaper/bigchaindb-whitepaper.pdf.

[77] L. Axon, Privacy-awareness in blockchain-based pki, 2015.

[78] G. Wood, Ethereum: A secure decentralised generalised transaction ledger, *Ethereum Project Yellow Paper*, 2014.

[79] V. Buterin, A next-generation smart contract and decentralized application platform, *White paper*, 2014.

[80] J. Benet, Ipfs-content addressed, versioned, p2p file system, *ArXiv preprint arXiv:1407.3561*, 2014.

[81] R. Roman, J. Zhou, and J. Lopez, On the features and challenges of security & privacy in distributed internet of things, *Computer Networks, vol. 57*, 2013. [Online]. Available: `http://dl.acm.org/citation.cfm?id=2494697`.

[82] I. E. T. Force, Requirements for internet hosts – communication layers, *Networking Group*, 1989. [Online]. Available: `https://tools.ietf.org/html/rfc1122`.

[83] R. Lu, X. Lin, T. H. Luan, X. Liang, S. Member, and X. S. Shen, Pseudonym changing at social spots(an effective strategy for location privacy in vanet), 61(1):86–96, 86–96, 2012.

[84] I. E. T. Forsce, "Multicast dns", 2013. [Online]. Available: `https://tools.ietf.org/html/rfc6762`.

[85] D. Kaiser, A. Rain, M. Waldvogel, and H. Strittmatter, A multicast-avoiding privacy extension for the avahi zeroconf daemon, 2015.

[86] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, Internet of things (iot): A vision, architectural elements, and future directions, *Future Generation Computer Systems*, 29(7):1645–1660, 1645–1660, 2013.

Appendix A: Infrastructure's Peer REST API

Decentralized Internet of Things Public API

Each peer provides a set of services, according to its hardware constraints.
More information: https://helloreverb.com
Contact Info: vasco.santos@ua.pt
Version: 1.0.0
All rights reserved
http://apache.org/licenses/LICENSE-2.0.html

Methods

[Jump to Models]

Table of Contents

1. POST /access/dht/{key}
2. POST /access/persistence/{key}
3. POST /collector/data/{key}
4. POST /management/acceptSensor
5. POST /management/accessSensor
6. POST /management/answerRefusal
7. POST /management/answerUnshare
8. POST /management/bindSensor
9. POST /management/logInPrivate
10. POST /management/logInPublic
11. POST /management/refuseSensor
12. POST /management/shareSensor
13. POST /management/signUpPrivate
14. POST /management/signUpPublic
15. POST /management/unshareSensor

Figure 1: Documentation Resume.

POST /access/dht/{key}

Get last received data of a specific sensor (**accessDhtKeyPost**)

Method for getting data from a specific sensor, which is contained in the DHT of the infrastructure. This data access has a control access mechanism, in order to verify if the da

Path parameters

key (required)
Path Parameter — sensor identifier.

Query parameters

pseudonym (required)
Query Parameter — Pseudonym of the user for access control.

signature (required)
Query Parameter — Signature of the received pseudonym, in order to verify its integrity. format: base64

Return type
Object

Produces
This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

- `application/json`

Responses
200
Success. Object
401
Not allowed. Object
404
Sensor Not found. Object

Figure 2: Method for getting the last received data of a sensor.

POST /access/persistence/{key}

Get the data history of a specific sensor (**accessPersistenceKeyPost**)

Method for getting a filtered data set from a specific sensor, which is contained in the infrastructure's persistence. This data access has a control access mechanism, in order tc requested data.

Path parameters

key (required)
Path Parameter — Sensor identifier.

Query parameters

data (required)
Query Parameter — Composed by the user pseudonym and query conditions.

signature (required)
Query Parameter — Signature of the received data, in order to verify its integrity. format: base64

Return type
Object

Produces
This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

- `application/json`

Responses
200
Success. Object
401
Not allowed. Object
404
Sensor Not found. Object

Figure 3: Method for getting the data history of a sensor.

POST /collector/data/{key}

Push data from a sensor to the infrastructure (**collectorDataKeyPost**)

Insert data received from a sensor to the infrastructure. It is stored in the DHT, as well as in the persistence cluster. In addition, it is streamed by the subscribers.

Path parameters

key (required)
Path Parameter — sensor identifier.

Query parameters

data (required)
Query Parameter — Sensor data.

Return type
Object

Produces
This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

- `application/json`

Responses
200
Success. Object
404
User Data not Found. Object

Figure 4: Method for publishing new data in the Infrastructure.

POST /management/acceptSensor

Accept a share invitation for accessing data from a sensor. (**managementAcceptSensorPost**)

When a user receives a share notification for data sharing from another user, he must accept to add this new sensor to the list of sensors which the user may access.

Query parameters

dataToShare (required)
Query Parameter — Composed by the user pseudonym, sensorID, new user sensors data object and its signature.

signature (required)
Query Parameter — Signature of the received data, in order to verify its integrity. format: base64

Return type
Object

Produces
This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

- `application/json`

Responses
200
Success. Object
401
Not allowed. Object
404
User Not found. Object
500
Invalid Sensors list for sharing. Object

Figure 5: Method for accepting a share request.

POST /management/accessSensor

Get an access list of the sensors, which the user is granted to access. (**managementAccessSensorPost**)

Requests the list of users who may access each sensor, which the user may access.

Query parameters

pseudonym (required)
Query Parameter — Pseudonym of the user for access control.

sensors (required)
Query Parameter — Users sensors data object, which consists of a list of the user's sensors, the sensors shared and a signature of those lists.

signature (required)
Query Parameter — Signature of the received sensors, in order to verify its integrity. format: base64

Return type
Object

Produces
This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

- `application/json`

Responses
200
Success. Object
401
Not allowed. Object
500
Invalid Sensors List. Object

Figure 6: Method for getting the access list of a sensor.

POST /management/answerRefusal

Refuse data sharing invitation from other user. (**managementAnswerRefusalPost**)

Refuse data sharing, removing the user notification and the informing the user who intended to share his data.

Query parameters

dataToNotification (required)
Query Parameter — Data to the answer, which is composed by the pseudonyms of the user and the requester, as well as the sensor identifier.

signature (required)
Query Parameter — Signature of the received dataToNotification, in order to verify its integrity. format: base64

Return type
Object

Produces
This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

- `application/json`

Responses
200
Success. Object
401
Not allowed. Object
404
User Not found. Object

Figure 7: Method for acknowledging a data sharing refusal.

POST /management/answerUnshare

Acknowledge data sharing refusal from other user. (**managementAnswerUnsharePost**)

Acknowledge the data sharing notification, in order to update the access list of the sensor according to this refusal.

Query parameters

dataToNotification (required)
Query Parameter — Data to the answer, which is composed by the pseudonyms of the user, as well as the sensor identifier and the updated sensors object.

signature (required)
Query Parameter — Signature of the received dataToNotification, in order to verify its integrity. format: base64

Return type
Object

Produces
This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

- `application/json`

Responses
200
Success. Object
401
Not allowed. Object
404
User Not found or Sensor Not Found. Object
500
Invalid Sensors List for unsharing. Object

Figure 8: Method for acknowledging a data unsharing.

POST /management/bindSensor

Bind a sensor to its owner. (**managementBindSensorPost**)

Associate a new sensor to a pseudonym in the infrastructure.

Query parameters

pseudonym (required)
Query Parameter — Pseudonym of the user for access control.

sensors (required)
Query Parameter — User sensors data, which is composed by the list of sensors, which may be accessed ny the user (owned and shared).

signature (required)
Query Parameter — Signature of the received sensors, in order to verify its integrity. format: base64

Return type
Object

Produces
This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

- `application/json`

Responses
200
Success. Object
401
Not allowed. Object
404
User Not found. Object
500
Sensor already bound by another user. Object

Figure 9: Method for binding a new sensor.

POST /management/logInPrivate

Get the User (Private Data) (**managementLogInPrivatePost**)

Get the user private data, which is in the infrastructure DHT. It is composed bt the users' Private Keys encrypted.

Query parameters

codes (required)
Query Parameter — DHT random secret keys present in the User Public data (encrypted).

Return type
Object

Produces
This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

- `application/json`

Responses
200
Success. Object
404
Data Not found. Object

Figure 10: Method for getting the private data of the user, when logged in.

POST /management/logInPublic

Log In (Public Data) (**managementLogInPublicPost**)

Log in a user in the infrastructure. It is requested the user's public data

Query parameters

logInData (required)
Query Parameter — necessary data for logging in, which consists of pseudonym and the password hash.

Return type
Object

Produces
This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

- `application/json`

Responses
200
Success. Object
404
Data Not found. Object

Figure 11: Method for logging a user in the infrastructure.

POST /management/refuseSensor

Refuse data sharing from other user. (**managementRefuseSensorPost**)

Refuse the data sharing invitation and inform the requester user of this decision.

Query parameters

dataToShare (required)
Query Parameter — Data to the refusal, which is composed by the pseudonym of the user, as well as the sensor identifier.

signature (required)
Query Parameter — Signature of the received dataToShare, in order to verify its integrity. format: base64

Return type
Object

Produces
This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

- `application/json`

Responses
200
Success. Object
401
Not allowed. Object
404
User Not found or Sensor Not Found. Object

Figure 12: Method for refusing a share request.

POST /management/shareSensor

Share data from a sensor to a user. (**managementShareSensorPost**)

Send a data sharing invitation and add the intended pseudonym to the sensors access list.

Query parameters

dataToShare (required)
Query Parameter — Data for data sharing, which is composed by the pseudonym of the user, as well as the sensor identifier.

pseudonymsToShare (required)
Query Parameter — List of pseudonyms to share data.

signature (required)
Query Parameter — Signature of the received dataToShare, in order to verify its integrity. format: base64

Return type
Object

Produces
This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

- `application/json`

Responses
200
Success. Object
401
Not allowed. Object
404
User Not found or Sensor Not Found. Object
500
No pseudonyms to share data. Object

Figure 13: Method for sharing data of a sensor.

POST /management/signUpPrivate

Sign up a User (Private Data) (**managementSignUpPrivatePost**)

Sign up a user in the infrastructure DHT. It is stored the users' Private Keys, in different points of the DHT. Each one is properly ciphered with a symmetric key generated on the cli

Query parameters

code_DS (required)
Query Parameter — key generated on the client side to store the Digital Signatures Private Key in the DHT.

priv_key_DS (required)
Query Parameter — Encrypted private key for digital signatures. format: base64

code_ED (required)
Query Parameter — key generated on the client side to store the Encryption Private Key in the DHT.

priv_key_ED (required)
Query Parameter — Encrypted private key for data encryption. format: base64

Return type
Object

Produces
This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

- `application/json`

Responses
200
Success. Object
401
Not allowed. Object

Figure 14: Method for finishing the sign up process by spreading the user's private credentials through the overlay.

POST /management/signUpPublic

Sign up a User (Public Data) (**managementSignUpPublicPost**)

Sign up a user in the infrastructure DHT. It is stored the user data and the user's sensors list, as well as two digital signature for authenticating, both the user data and the senso

Query parameters

data (required)
Query Parameter — Countains public key for digital signatures validation and public key for data ciphering, as well as the codes for proceed to the private sign up. format: base64

signature (required)
Query Parameter — Signature of data paramater. format: base64

sensors (required)
Query Parameter — Countains a list of owned sensors, shared sensors, as well as a digital signature of both lists. format: base64

signature_sensors (required)
Query Parameter — Signature of sensors parameter. format: base64

Return type
Object

Produces
This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

- `application/json`

Responses
200
Success. Object
401
Not allowed. Object

Figure 15: Method for initiating a sign up process.

POST /management/unshareSensor

Unshare sensor data from a previously shared user. (**managementUnshareSensorPost**)

Unshare data from a sensor to a previously shared user, removing the user from the sensor's access list and notifying the user.

Query parameters

dataToUnshare (required)
Query Parameter — Data for unsharing data, which is composed by the pseudonym of the user, as well as the sensor identifier.

usersToUnshare (required)
Query Parameter — List of pseudonyms to unshare data.

signature (required)
Query Parameter — Signature of the received dataToUnshare, in order to verify its integrity. format: base64

Return type
Object

Produces
This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

- `application/json`

Responses
200
Success. Object
401
Not allowed. Object
404
User Not found or Sensor Not Found. Object

Figure 16: Method for unsharing sensor data.

Appendix B: Register Server's REST API

Decentralized Internet of Things Certification Authority API

Certification Authority entity, which is responsible for the bootstrapping of the overlay peers.
More information: https://helloreverb.com
Contact Info: vasco.santos@ua.pt
Version: 1.0.0
All rights reserved
http://apache.org/licenses/LICENSE-2.0.html

Methods

[Jump to Models]

Table of Contents

1. GET /getBootstrappingPeers
2. GET /getCertificate
3. GET /getPeer/{id}
4. GET /getPeerPublicKey/{id}
5. GET /getStoragePeers
6. POST /setPeer/{id}

Figure 17: Documentation Resume.

GET /getBootstrappingPeers

Get a list of the peers, which are present in the DHT. (**getBootstrappingPeersGet**)

Method for getting a list of peers, wich are part of the infrastructur eoverlay. A new peer uses this list to join the overlay network.

Return type
Object

Produces
This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

- `application/json`

Responses
200
Success. Object
404
No peers found. Object

Figure 18: Method for getting a list o bootstrapping peers.

GET /getCertificate

Get the public key certificate of the Certification Authority. (**getCertificateGet**)

Get the public key certificate of the Certification Authority.

Return type
Object

Produces
This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

- `application/json`

Responses
200
Success. Object

Figure 19: Method for getting the self-signed certificate of the Certification Authority.

GET /getPeer/{id}

Get the certification data of a peer. (**getPeerIdGet**)

Method for getting the certificate of a specific peer, which is signed by the CA.

Path parameters

id (required)
Path Parameter — Peer identifier.

Return type
Object

Produces
This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

- `application/json`

Responses
200
Success. Object
404
Peer Not found. Object

Figure 20: Method for getting the data a specific peer.

GET /getPeerPublicKey/{id}

Get the public key of a peer. (**getPeerPublicKeyIdGet**)

Method for getting the public key of a peer.

Path parameters

id (required)
Path Parameter — Peer identifier.

Return type
Object

Produces
This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

- `application/json`

Responses
200
Success. Object
404
Peer Not found. Object

Figure 21: Method for getting the public key of a peer.

`POST /setPeer/{id}`

Register a peer in the infrastructure. (**setPeerIdPost**)

Add a new peer certificate to the CA database for a limited time.

Path parameters

id (required)
Path Parameter — Peer identifier.

Query parameters

data (required)
Query Parameter — Countains Public Key Certificate (signed by the CA).

Return type
Object

Produces
This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

- `application/json`

Responses
200
Success. Object

Figure 22: Method for registering a peer on the Certification Authority.

Appendix C: Gateway Peer Configuration

```
> ./dioti.sh init
Initializing DIoTI at /home/vagrant/.dioti

Generating 2048-bit RSA keypair...
Generating RSA private key, 2048 bit long modulus
......+++
............+++
writing RSA key

Peer Identity:  f20aa2a0e57e3c35fe6941c5c8077639151960af

Getting your network definitions...

IPv4 addresses available:
10.0.2.15
192.168.33.20
127.0.0.1
- Select which IP address to use (line number, default value:  0):  1
- UDP Port (Default Value:  8468):
- TCP Port (Default Value:  8080):
- MQTT Port for Broker (Default Value:  1883):
- Register Server Address (Default Value:  192.168.33.30):
- Register Server Port (Default Value:  8080):
- Management TCP Port (Default 2000):
- Enable Data Access (Y/N, Default:  Y):
- TCP Port (Default 6000):
- Enable Data Collector (Y/N, Default:  Y):
- HTTP Port (Default 7000):
- MQTT Broker Address (Default 127.0.0.1):
- Enable Data Persistence (Y/N, Default:  Y):
- TCP Port (Default 8087):
- Enable Data Stream (Y/N, Default:  Y):
- TCP Port (Default 9000):

Everything is ready!
```

Listing 7: Output for configuring a gateway with all services enabled.

Appendix D: Gateway Peer Configuration File

```
1  {
2      "Addresses": {
3          "ipv4": "192.168.33.20",
4          "mqttPORT": 1883,
5          "tcpPORT": 8080,
6          "udpPORT": 8468
7      },
8      "Identity": {
9          "Passphrase": "/home/vagrant/.dioti/passphrase",
10         "PeerID":
              "f20aa2a0e57e3c35fe6941c5c8077639151960af",
11         "PrivKey": "/home/vagrant/.dioti/peer_priv.pem",
12         "PubKey": "/home/vagrant/.dioti/peer_cert.pem"
13     },
14     "Register": {
15         "ca_cert": "/home/vagrant/.dioti/ca_cert.pem",
16         "ipv4": "192.168.33.30",
17         "tcpPORT": 8080
18     },
19     "Services": [
20         {
21             "function": "Data Access",
22             "tcpPORT": 6000
23         },
24         {
25             "function": "Data Collector",
26             "httpPORT": 7000,
27             "mqttIP": "127.0.0.1"
28         },
29         {
30             "function": "Data Persistence",
31             "ip": "localhost",
32             "tcpPORT": 8087
33         },
34         {
35             "function": "Data Stream",
36             "tcpPORT": 9000
37         },
38         {
39             "function": "Management",
40             "tcpPORT": 2000
41         }
42     ]
43 }
```

Listing 8: JSON Configuration File.