University of Padova

Department of Information Engineering

Master Degree in Telecommunication Engineering

Internet of Things for Smart Cities: user interface and security issues

Candidate

Chiara Pielli

Supervisor

Prof. Michele Zorzi

Assistant supervisor

Prof. Andrea Zanella

Academic Year 2014/2015

A mamma, a papá. A Laura. Perché ci sono sempre.
Agli amici di ieri e di oggi e ai coinquilini di questi anni,
che hanno dato luce ai giorni piú grigi.
Agli ingegneri di Patavina Technologies, che mi hanno accolto.
A chi non c'é ma vorrei ci fosse.
A Pier, ovviamente.

"The hardest thing of all is to find a black cat in a dark room,
especially if there is no cat."

Confucius

Contents

Abstract

The Internet of Things revolution is drastically changing the concept of networking: everyday objects are becoming smart and interconnected, forming a ubiquitous and densely populated network. The work presented in this thesis is part of a bigger project concerning the development of a complete architecture for the Internet of Things. It consists in the improvement of an existing Internet of Things middleware, providing a way to seamlessy connect the final user interface to the 'things' world, and in the development of a security framework to manage read and write permissions owned by the users with respect to the 'things' and their observations. The overall project considers every element of an Internet of Things network, from the physical devices capable of generating data to the web interface through which the final user accesses and controls those data. The deployment of such architecture is envisaged in a Smart Cities context.

Introduction

'The Internet of Things has the potential to change the world, just as the Internet did. Maybe even more so'. This is what Kevin Ashton[1] states in an article appeared in the RFID journal in June 2009 [1]. The Internet of Things (IoT) is a recent communication paradigm which is deemed to be the next stage of the information revolution after the massive spreading of the Internet in every field: intelligent sensors are extended into the world of everyday objects: machines, buildings, vehicles, plants, people themselves, etc. and they are becoming an integral part of the Internet [2]. Everything is becoming linked to everything else and capable of providing us feedback as the *things* are getting equipped with sensors and microcontrollers. The IoT shall therefore be able to seamlessy incorporate a large number of different and heterogeneous end systems, while providing open access to selected subsets of data for the development of digital services [3]. The IoT is representing a central step towards the realization of Mark Weiser's vision of ubiquitous computing: a world where technology vanishes in the background [4]. A similar concept has been stated by Eric Schmidt [5], Executive Chairman of Google, for whom *'the Internet will disappear... you won't even sense it, it will be part of your presence all the time'.* The impact of this technology shift is supposed to be much heavier than the one caused by the integration of Internet in our lives through smartphones and other technological mobile devices. Very recent data from Juniper Research have revealed that the number of IoT connected devices is predicted to be 38.5 billion in 2020, up from 13.4 billion in 2015, a rise of over 285% [6].

In spite of the very fast growth of the IoT technology, many challenges still need to be addressed, mainly concerning the new transmission paradigm, security issues and energy efficiency. The Internet of Things aroused in me a vivid interest, which pushed me to do this thesis work in collaboration with Patavina Technologies[2], a spin-off of the Information Engineering Department of the University of Padova. It is a system and software house that

[1] Cofounder of the Auto-ID Center at the Massachussets institute of Technology (MIT).

[2] `http://www.patavinatech.com/`

deals with Internet of Things and provides services related to monitoring, actuation, indoor real time positioning and smart cities by using technologies such as IEEE 802.15.4 [7], IEEE 802.11 [8], 6LoWPAN [9], CoAP [10], Lo-Ra^{TM} [11]. They are currently working at the development of a complete IoT system with the target of Smart Cities (and smart buildings or, in general, home automation) which involves every element of the IoT architecture, from the physical sensors to the web interface for the final user. I took part in this project, and more specifically I worked on the integration of a middleware to fit in the architecture they developed and in the definition of some security measures at the application level, including the possibility of assigning different privileges to the users.

Thesis outline

This document is organized as follows. Chapter 1 introduces the Internet of Things and presents the state of the art of IoT for Smart Cities; Chapter 2 describes the project of Patavina Technologies; Chapters 3, 4 and 5 focus on the work I did and concern the choice of the middleware to use, the adaptations I implemented to make the platform fit the existing system and the security issues and measures concerning the interaction with the Internet, respectively. Chapter 6 analyzes the results of some measures I made for evaluating the work I did. Finally, in Chapter 7 the conclusions are drawn.

1. Internet Of Things

Although the concept of Internet of Things is quite recent and still not universally known, the idea of an appliance connected to the Internet was discussed as early as 1982: a Coke machine at Carnegie Mellon University (Pittsburgh, Pennsylvania) was modified in order to report its inventory and whether newly loaded drinks were cold [12]. In the following years the idea of an extremely connected world got through till the concept of the Internet of Things shaped up at the end of the XX century: a ubiquitous network of physical objects able to collect and exchange data.

Any object may be part of the Internet of Things; in his speech [13] at the TEDx conference of 2012 John Barrett[1] explains that three steps are needed to make an object 'smart': it must be given a unique identifier, the ability to communicate with the Internet, and senses for giving you feedback about what you are interested in. This is clearly a simplification which abstracts from all the communication and security issues that arise when thousands of devices are connected to the Internet. However, the basic concept of IoT is simple: connecting physical objects to the Internet in order to access remote sensor data and to control the physical world from a distance [14]. Nowadays, many embedded systems are already connected to the Internet so the real novelty of the IoT is not in the functional capabilities of smart objects but rather in the huge number of connected devices and in the heterogeneity of both types of devices and types of applications. The IoT will in fact have an impact in various areas which range from personal interests or comfort such as home automation, to medical services such as health care monitoring and elderly care, to collective benefits like smart cities, intelligent transportation or environmental monitoring [15]. However, such a heterogeneous field of application makes the identification of solutions capable of satisfying the requirements of all possible application scenarios a formidable challenge. The realization of a complete IoT network still lacks of a well-established and wid-

[1] Head of Academic Studies at the Nimbus Centre for Embedded Systems Research at Cork Institute of Technology and Group Director of the Centre's Smart Systems Integration Research Group.

ley acknowledged best practice and many issues still need to be addressed [3].

1.1 Open challenges

So far, the main driver for the development of communications technologies has been Human-to-Human (H2H) communication. Nonetheless, with the advent of smartphones and the new communication paradigm introduced by the IoT, human interaction is currently giving way to Machine-To-Machine (M2M) communication, which is expected to grow exponentially in the near future. Because of its distinct and unique features, Machine-Type Communication (MTC) raises numerous challenges which are the object of a intensive research in both academia and industry, which are trying to keep up with the fast and increasing growth of IoT technologies [16].

A considerable concern IoT systems need to cope with, is massive access management. So far, researchers have been investigating cellular based approaches: in particular they have focused on a revival of GSM [17] [18], the first generation of standards for cellular systems, and on adaptations of the fourth generation standards, LTE [19] [20]. However, an efficient solution still must be found as the requirements of IoT and MTC are considerably different from H2H communication needs. A general guideline is to implement an access scheme in between coordinated and completely random, for avoiding both a too high physical layer signaling and collisions, which would exacerbate the network congestion already threatened by a sudden surge of connecting devices. Other well-known issues touch energy efficiency, exacerbated by the strong constraint of the devices, which are typically battery equipped [16]. The heterogeneity of the devices introduces other complications such as the management of different Quality of Service (QoS) requirements. Finally, when even all these challenged were optimally addressed, system properties like robustness, scalability and especially security should also be integrated in the architecture [21].

It is important to mention that various solutions oriented to M2M services already exist, such as ZigBee [22] and the Third Generation Partnership Project (3GPP) Proximity-based Service protocol [23], but all of them are mostly valuable for local area services, while MTC aims at providing ubiquitous access to the devices across a geographically wide service area. This thesis concerns the development of an IoT architecture which relies on LoRaWAN™, a technology standardized by the LoRa alliance (see Section 2.2).

1.2 Protocols

Any IoT application must specify how things are interconnected, starting from the bottom of the OSI stack. The IoT poses numerous challenges in creating the basic interconnections between two things: network topology, cost, latency, application throughput and security.

The standardization of communication protocols for resource constrained devices is currently led by the Internet Engineering Task Force (IETF), which developed the 6LowPAN protocol suite [9] for low power devices allowing IPv6[2] packets to be transmitted over IEEE802.15.4 networks. IEEE 802.15.4 [7] is a radio technology standard for low-power and low-data-rate applications with a short radio coverage (few meters), and specifies both physical and MAC layers. In [24] the authors have proved that also low-power WiFi may be counted as a valuable solution, as it considerably reduces latency and energy consumption with respect to typical WiFi.
As regards the network and transport layers, the Internet Protocol (IP) in its new version IPv6 and the Transport Control Protocol (TCP) are actually among the most considered solutions. The User Datagram Protocol (UDP) instead offers lower energy consumption and traffic overhead than TCP but grants no reliability at all. Another protocol to consider is the already mentioned ZigBee, which is not compatible with the Internet stack but has been specifically developed for industry and home automation applications.
At the application layer, thing-driven protocols have been developed as the HyperText Transfer Protocol (HTTP), which represents the basis of the Web, does not suit well M2M communication. A protocol similar in HTTP from an application layer perspective has recently been designed: CoAP, the Constrained Application Protocol [10], which uses UDP as underlying transport protocol thus reducing the transmission overhead, and supports a basic notification mechanism. The most worth mentioning protocols targeting IoT applications are MQTT and AMQP: Message Queuing Telemetry Transport [25] is a fast and lightweight protocol based on a publish/subscribe mechanism and working on top of TCP - it will be thoroughly described in Section 4.1.1 -, whereas the Advanced Message Queuing Protocol is a binary application layer protocol designed to efficiently support a wide variety of messaging applications and communication patterns and uses TCP as the transport protocol.

[2]Since the very beginning of researches on IoT, IPv6 has been selected as communication protocol for providing an identification and location system for the devices as it uses addresses of 128 bits, much more than the 32 bits provided with IPv4.

1.3 Smart Cities

A possible IoT application in which the IoT community is spending effort is the Smart City vision. There is no formal definition of Smart City but its meaning is quite intuitive: a city which exploits the most modern communication technologies - like the MTC paradigm - to increase the quality of the services offered to citiziens whilst boosting local economies and decreasing the public administrations' costs [26]. Integrating IoT in an urban context brings benefits to a lot of services: public transportation, traffic management and parking, urban lighting, public safety and monitoring of public areas, garbage collection, hospitals and many others. The same citiziens are expected to be actively involved in city services and more aware about the status of the city where they live. Smart Cities are in fact based on three cornerstones: technology innovation, environmental sustainability and social involvement. Navigant Research forecasts that global smart city technology revenues will grow from 8.8 billions of dollars annually in 2014 to 27.5 billions in 2023 [27].

Smart City have not really established yet because despite the entusiasm generally shown by both city administrations and citiziens, some barriers still prevent their penetration in the existing contexts [28]. These obstacles concern political, financial and technical points of view; technical issues revolve around the current inoperability of the technologies actually used in the cities. Nonetheless some cities are already in the forefront of this urban technologies shift, leading the actualization of the Smart City vision: Padova itself [29] [30] is spending effort in becoming 'smart', offering services for an overall improvement in the citiziens' lives. Another worth-mentioning example is SmartSantander [31] [28], which with more than 1100 active sensors is probably the largest Wireless Sensor Network in the whole world.

In the development of an urban IoT, the following entities play a fundamental role [3]:

- *peripheral nodes*: the proper 'things' capable of generating data; they may be sensors or actuators and are typically cheap. They must support link layer technologies for communicating with the control entities and they must be univocally addressable, e.g., by using Radio-Frequency IDentification (RFID)[3] [32];

- *gateways*: intermediary devices in charge of interconnecting the sensors to the rest of the network. They must provide protocol translation

[3] RFID uses wireless electromagnetic fields for tracking tags attached to objects.

for mapping the constrained protocol used by the nodes to the unconstrained and more sophisticated protocols of the main communication infrastructure of the system;

- *backend services*: responsible for collecting, processing and storing the data received from the end nodes. By means of a graphical interface, they represent the access channel for the human users to the things' world.

Patavina Technolgies spent a lot of effort in the implementation of an IoT infrastructure in the context of a project called 'Padova Smart City'. The next chapter briefly introduces the main elements of its architecture.

2. The Patavina Technologies project

The Patavina Technolgies team developed an architecture that spans all the elements belonging to an IoT system, from the physical sensors to the final user graphical interface, with the target of smart environments, i.e., home automation or Smart Cities. Figure 2.1 is a schematic representation of the architecture they set up.

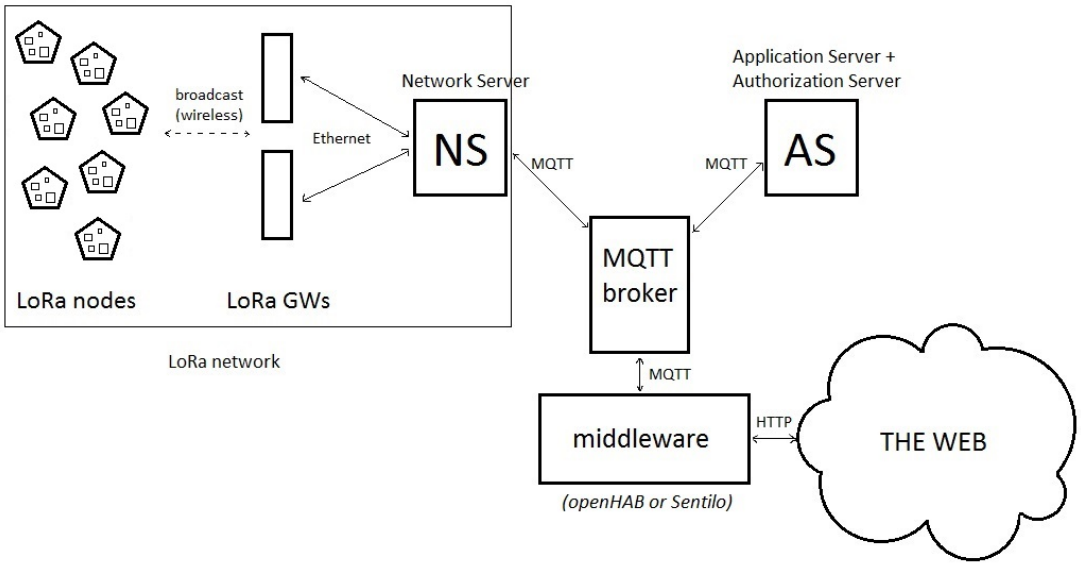


Figure 2.1: The system architecture

It is possible to identify two main parts in the architecture: the LoRa™ network, which includes sensors, gateways and a Network Server, and a second part comprehending the devices that allow the LoRa network (the 'Things') and the final user (the 'Internet') to interact.

I will illustrate all the elements involved and how they communicate among each other.

2.1 System architecture

The architecture designed by Patavina Technologies is based on LoRa™ technology, LoRaWAN [33], a Low Power Wide Area Network (LPWAN) specification intended for wireless battery operated devices, which may be either mobile or mounted at a fixed location. LoRaWAN is a network protocol for networks typically laid out in a star-of-stars topology[1]. The key elements are the end-devices, a central network server, and an arbitary number of gateways which relay messages among the backends. A thorough description of LoRa specification is given in the next section.

The elements of which the whole system consists are the following:

- *LoRa nodes*: they are the most peripheral elements of the network and are the devices able to generate observations. Each node can be equipped with multiple sensors of different types. Currently, nodes are incapable of supporting an operating system, so network services and applications are implemented as a monolithic piece of firmware to install into the devices. The packets generated by nodes (according to the LoRa specification) are sent in broadcast, via wireless; thus, any LoRa gateway in the coverage area can overhear them;

- *LoRa gateways (GW)*: they are in charge of forwarding the packets received from the LoRa nodes to the LoRa Network Server making use of IP tunnels. They also add some extra information to the packets, mainly regarding statistical data such as the SNR (Signal-to-Noise Ratio) or the signal frequency;

- *LoRa Network Server (NS)*: it is a unique element in a LoRa network and implements the LoRa protocol stack up to the MAC layer in order to seamlessly interact with the nodes through the gateways. It can also support other communication protocols, so it represents the point of access for the LoRa network. In the Patavina Technologies project, for example, the NS supports also the MQTT communication protocol to transmit the LoRa messages to the MQTT broker;

- *MQTT broker*: it is the server in a MQTT application, as will be explained in Section 4.1.1. It receives the messages from the NS and forwards them to the other clients through a publish/subscribe mechanism. The key feature of MQTT is its being a push protcol, so a client does not

[1] In star networks each host is connected to a central hub with a point-to-point connection.

need to continuously ask for the information it wants, but it just needs to subscribe to the desired service and the server will be in charge of sending him the related messages;

- *Application Server (AS)*: it acts as a MQTT client and subscribes to all the resources published by specific LoRa nodes[2]. Thanks to the MQTT addressing structure, the AS is able to discriminate the application used by the node that generated the data, and this is necessary for interpreting and processing each message in the proper way, according to the specific application;

- *Authorization Server (AUS)*: it manages the authorization and the join procedure of the nodes. The join procedure is used for the LoRa key exchange, as described further;

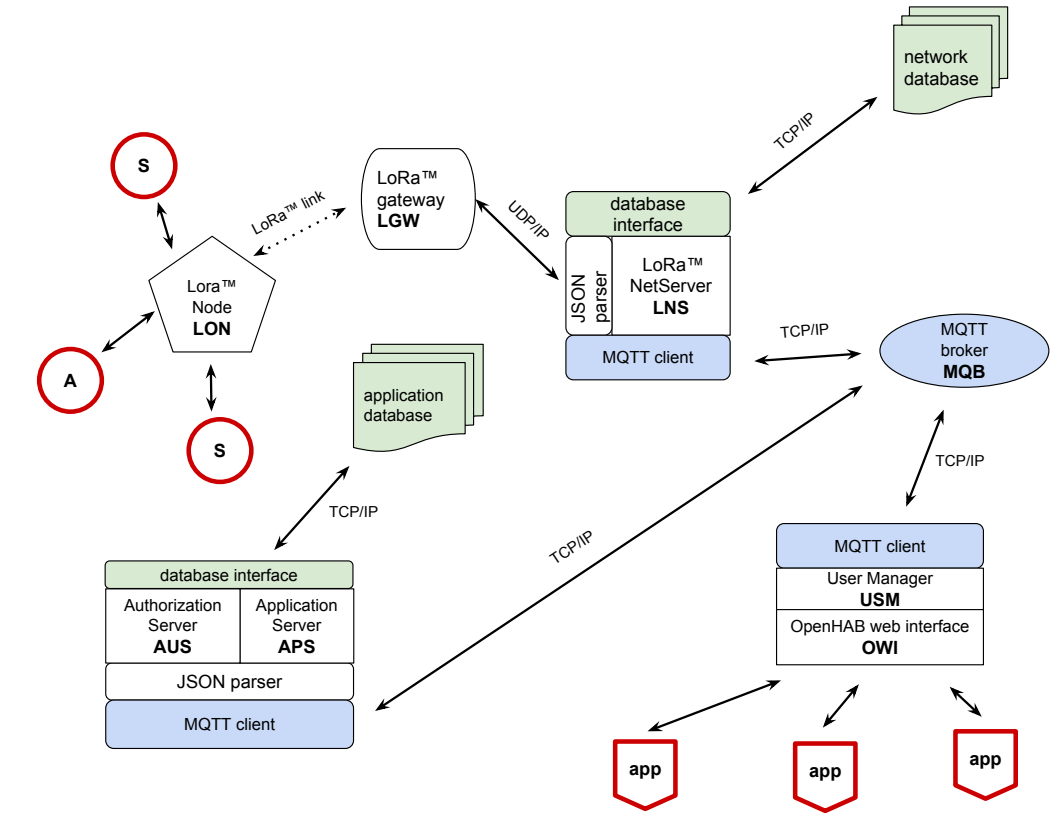
- *middleware*: it is a platform acting as intermediary between the AS, which is the representative of the 'Things', and the final user, the 'Internet';

- *graphical user interface (GUI)*: it is a web front-end that offers a friendly interface to the final user for accessing the data generated by the nodes applications, change some settings and send commands to the sensors. Such website is accessible via authentication and displays both the latest values registered and the graphs of historical data.

The listed elements are all distinguished from a logical point of view, but they may be physically located in the same device, as typically happens for the Application and the Authorization Servers. A possible configuration sees the Network Server residing in a single device to serve multiple gateways and the other services all running on a cloud.

Figure 2.2 represents a detailed scheme of the whole architecture. The openHAB reference is due to the fact that the first middleware chosen by Patavina Technologies was openHAB, a software built for home automation which supports many different technologies. The current implementation of the Patavina Technologies considers the possibility for the customers to choose whether to use openHAB or Sentilo, a platform designed to fit Smart City architectures. The differences between openHAB and Sentilo are described in Chapter 3, in which I will also explain why I was in charge of investigating about new platforms to use in place of openHAB. Anyway, whatever the platform included in the system, its task is to connect the MQTT broker to the final user interface.

[2] For example, it may be interested in all nodes belonging to a certain company.




```
S = Sensor
A = Actuator
app = openHab application
```

Figure 2.2: Detailed system architecture

All message payloads are in the Json format, as deducible from the Json parser elements of the scheme in Figure 2.2. Nodes communicate with the GWs through the LoRa wireless technology, receive and transmit in broadcast. All the other links use standard Internet technologies; the connection between a GW and the NS is wired because it needs to be quite reliable since the transport protocol used is UDP, which is faster and lighter than TCP but less reliable. The connections among the other elements are instead TCP based.

2.2 LoRa network

The LoRa™ alliance developed a specification for networks with *Lo*ng *Ra*nge and *Lo*w (power) *Ra*dio networks, providing a network architecture that fits very well the IoT systems requirements (support for transmission of thousands of short messages, security issues, low energy consumption of the devices, ...). All communication is generally bi-directional, although uplink communication from an end-device to the Network Server is expected to constitute the predominant traffic. Communication between nodes and GWs is spread out on different frequency channels and data rates (from 0.3 kbps to 50 kbps), chosen as a trade-off between communication range and message duration. LoRa supports an adaptive data rate scheme that specifically manages each node individually, so as to maximize the battery life of the nodes and the overall network capacity.

All the elements belonging to a LoRa network have some common minimum functionalities aiming at efficiently coping with the energy constraints of the devices, and they are said to implement the Class A. There exist two other classes of functionalities named Class B and Class C, which both add some features to the basic LoRaWAN Class A:

- *Class A*: it identifies characteristics that all nodes must implement or, at least, that have to be compatible with the characteristics of a node. The end-devices are able to both transmit and receive messages, but not simultaneously: an uplink tranmission window is followed by two shorter downlink receive windows. These communication slots are spaced out by sleep phases of semi-random duration, which are meant to save energy and battery life. Downlink communications from the server have to wait until the next scheduled uplink as they can happen only when the node is listening;

- *Class B*: this class has been thought for nodes that must be more reactive to the commands and messages received from the GWs. It in fact allows the devices to have more receive slots: in addition to the Class A random receive windows, Class B devices open extra receive windows at scheduled times determined by time-synchronized beacons sent by the GWs;

- *Class C*: the receive slots are maximized as the downlink windows are almost continuously open and interrupted only for transmitting. Of course, this behaviour consumes much more energy but it offers a minimal latency for communications from the server.

The LoRa network set up by Patavina Technologies does not include class C devices, as the predominant communication is in uplink and implementing class C features consumes a lot of energy, which is undesirable if the devices are not connected to a power source.

In order to participate in a LoRaWAN network, each end-device has to be personalized and activated. Personalization means that each node must be given a globally unique end-device identifier, an application identifier and a AES-128 key, which is used for the encryption mechanism and whose control is up to the NS. The activation of a node can be achieved in two different ways, namely via Over-The-Air Activation (OTAA), when a node is deployed or reset, or via Activation By Personalization (ABP), in which the two inizialization steps are done as a single one. For OTAA, end-devices must follow a join procedure prior to participating in data exchanges with the NS. Before the join procedure takes place, the personalization has to be done; after that the node sends a *join-request* message to the Network Server containing its unique ID and the application ID. If the request is not accepted, no response is given to the end-device; otherwise it receives a *join-accept* message. The ABP procedure, instead, directly ties a node to a specific network by-passing the join message exchange. Anyway, in this case the end-device must be equipped with the required information for participating in a specific LoRa network when started.

The end-devices and the gateways are equipped with a time counter, i.e., a clock that measures the time passed since the node activation, but it is not related to an absolute time. GWs may also have a GPS clock, whose presence is necessary for the beacon synchronization required by nodes implementing the Class B. An absolute timestamp is inserted in the message payload by the Network Server, which can for example make use of the Network Time Protocol (NTP). The timestamps of the gateways (which are relative timestamps) have the only goal of calculating the receive windows of the end-devices. However, this is an extremely important operation, as messages can be sent neither too late (otherwise the window will be closed) nor too early, as the node would not be in listening mode and the GW would not detect the failed transmission (no ACK feedback is implemented).

2.3 Security

Chapter 5 gives an overview of the importance of security measures in an IoT system. Data privacy should be granted in every element of the architecture as a single leak would disable the security of the system, representing a point of access for security attacks, e.g., capture of information, duplication of

existing packets or insertion of new fake packets.

For what concerns security in LoRa, the payloads of data frames are encrypted, but the MAC headers are left in clear. The encryption scheme is based on the generic algorithm described in IEEE 802.15.4/2006 Annex B using AES with a key length of 128 bits. AES [34] is a symmetric encryption algorithm whose key may be of length 128, 196 or 256 bits. For each end-device there is a specific application session key (AppSKey), which is used by both the NS and the end-device to encrypt and decrypt the payload field of application-specific data messages. In the OTAA, the AppSKey is derived from an application key (128 bits long) assigned by the application owner to the end-device and most likely derived from an application-specific root key exclusively known to and under the control of the application provider. In the ABP process the AppSKey is directly stored in the node and provided by the NS. In both cases, the key control is up to the NS, which is also responsible for deciding whether the AES key can be known by the gateways or not. Since the GWs may be unaware of the key used, the information they add about LoRa statistics are not encrypted. Patavina Technologies is currently working at the implementation of a Virtual Private Network (VPN), which essentially extends a private network across a public network thus allowing to benefit from the security policies of the private network [35].

Security about the MQTT connections is granted by the enabling of both authentication and TLS: the MQTT broker used provides username and password authentication as well as limiting access to topics by using access control lists, preventing abusers from subscribing to topics, whereas the Transport Layer Security (TLS) [36][3] is a cryptographic protocol that ensures a safe transmission. TSL uses X.509 certificates and hence asymmetric cryptography to authenticate the counterpart with whom they are communicating, and to negotiate a symmetric session key, which will be used for the encryption of data flowing between the parties. Thus, MQTT messages are quite safe against sniffing attacks and are not forwarded to non authenticated clients.

[3] TLS is the successor to Secure Sockets Layer (SSL)

3. The choice of the platform

The Network Server is the point of access to the LoRa network and the messages from the 'Things', and supports other protocols (such as MQTT, for instance) so to transmit LoRa messages outside of the LoRa network. Anyway, the final user communicates through websites and graphical interfaces, hence HTTP, and therefore an element to connect the final user to the 'Things' world is needed: it is the component labeled as *middleware* in Figure 2.1.

IoT systems often deal with different types of devices, each of them with its own communication protocol and different requirements, that need to somehow interact with the final user [37]. In order to meet this demand, IoT architectures require a software platform defined as middleware which fundamentally provides abstraction to applications from the 'things', and offers multiple services. Middleware represents an intermediate layer between the 'Internet' and the 'things' and acts as a bond joining the heterogeneus domains of applications communicating over heterogeneous interfaces [38]. A middleware is in charge of masking the heterogeneity and distribution problems that are faced when interacting with devices, so that even the average technology-inexperienced user is able to enjoy IoT services effortlessy [39].

The middeware platform originally chosen by the Patavina Technologies team was openHAB, an open source automation software specifically built for smart houses (see Section 3.2.1). Since openHAB currently lacks some useful features, e.g. the possibility to define different roles for distinguishing among users privileges, my first task consisted in investigating some existing middlewares for IoT applications in order to find out the one that better fitted Patavina requirements.
The development of a middleware in the IoT context requires the support of some functionalities, such as dealing with huge amounts of data, security and privacy aspects, scalability, device discovery and management. Hence, a number of challenges needs to be addressed in order to build an efficient, robust, scalable and real-time platform. For these reasons, we decided not to develop a custom middleware from scratch, but rather to use an existing

and tested platform and adapt it to fulfill our requirements, if needed.

In the following sections I will outline the requirements we looked for and describe three existing platforms we took into consideration, pointing out the strenghts and the weaknesses of each of them. All of them are open source projects, which are usable under the constraints of different software licenses, whose examination falls outsides the objectives of this thesis.

3.1 Required features

The main objective of an IoT middleware platform is to make the sensors and the final user interact. The element in charge of sending the sensors observations to the end user is the AS, or better, the MQTT broker to which the AS keeps sending sensors info and data after the processing of the MQTT messages coming from the LoRa network.
The following list summarizes the key features we identified for the middleware to use:

- *modularity*: the possibility to add functionalities without altering the existing core is essential in order to customize the platform in a plug-and-play fashion for accomodating missing features;

- *data format and protocol*: since the sensors observations are sent via MQTT, whereas the final user communicates via HTTP, the chosen middleware is asked to support both protocols. Moreover, sensors data are sent in the message payload in the Json format and therefore the already built-in capability of handling such format is appreciable;

- *data storage*: it is necessary to have historical data stored, mainly in order to create time-series graphs and to make it possible for the user to retrieve old observations;

- *user active interaction*: the user is expected not only to read data from sensors or retrieve information about the resources registered in the platform, but also to give orders to the sensors. Such orders may come from a spontaneous will of the user to change some parameter or the sensor state, or may be some kind of alarms triggered by the retrieved observations (e.g, when a monitored quantity goes beyond a certain threshold);

- *customizable sensor representation*: we would like to be able to assign custom properties to the registered items and to group them in a hierarchi-

cal way, mimicking the LoRa structure in which many different sensors belong to the same node and nodes may support different applications;

- *data timestamp*: as the sensors used send the timestamp at which data were originated along with the value of the observation, the chosen middleware should support the possibility of transmitting the timestamp in the data messages payload. This demand is important since transmission may considerably delay the message arrival at the AS with respect to the moment the message was originated. Moreover, it could allow the implementation of message piggybacking to send different sensor observations in the same message;

- *secure and conditional access*: as security represents a central issue in IoT systems, it is fundamental to have a secure and reliable middleware. An already implemented role division structure also represents a valuable feature: in such a way, it is possible to give users access according to the permissions they own, allowing to have users with different privileges;

- *administration console interface* to manage sensors, i.e., adding new ones, deleting or modifying existing ones.

Other minor but creditable features are the presence of an already implemented user interface (UI) and the presence of a growing community, which may play an important role to solve some unexpected issues and to make the chosen product better, as improvements only come when there is demand.

3.2 The candidates

The three platforms we considered are openHAB, Sentilo and Parse. OpenHAB, the original Patavina choice, principally addresses the needs of enterprising people that want to make their own house smart and provides them a complete tool for the administration and management of the connected devices. On the contrary, Sentilo has been developed with the target of smart cities, where many more users are implied. Finally, Parse is a cloud-based data management system mainly built to directly communicate with the hardware devices. All softwares are open source, which is an essential requirement since Patavina Technologies products should be sold. In this regard, the analysis of the three middlewares involved also the study of the licenses under which such tools are available. Anyway, as license investigation falls outside the purpose of this thesis, I will not go into its details. I will rather describe the main features of each platform and compare them with respect to the requirements mentioned above.

3.2.1 OpenHAB

OpenHAB [40] is a software platform for home automation born in 2008 from the need of its creator, Kai Kreuzer, to integrate sensors and actuators in his own house in Darmstädt (Germany). OpenHAB target is home automation and therefore it is thought to be used by a restricted number of users with complete access to all available information. The implementation of user differentiation according to given roles is on the future work list and will allow to authorize users with different read and write permissions. Anyway such feature is still missing and this is one the main concerns that made Patavina team willing to explore other possibilities after initially choosing openHAB as middleware.

OpenHAB is highly modular and extendible through a plug-and-play principle. Many modules have already been implemented, such as the MQTT binding, a component that allows openHAB to act as an MQTT client. A key concept for openHAB is the notion of an 'item', a data-centric functional atomic building block: all openHAB resources are represented using this 'item' abstraction, which is independent of the technology used. It is possible to register, modify or delete items, read and publish data, create triggers and alarms according to the sensors observations. Historical data can be stored in relational databases, NoSQL or round-robin databases, in IoT cloud services or simple log files, according to the user needs.

The strenghts of the openHAB platform are its high modularity, the presence of the MQTT binding, the possibility of using Json data format for payloads. Moreover, an already implemented and partially configurable user interface is available.

The main inadequacies of openHAB concern items, which are supposed to be an abstract concept not bound to a specific context. It is not possible to add custom properties to characterize them and hierarchy is feasible only by gathering them into groups. Moreover, as already pointed out, there still is no architecture to differentiate users and condition their access to the data stored. Another problem faced by using openHAB is the impossibility to send the timestamp at which sensors data are originated[1].

[1] This lack is actually a conscious decision of Kai Kreuzer, founder of openHAB. In response to a user request to add this option, Kreuzer stressed the 'item' abstraction described previously and stated: 'I do not really like the idea to add these additional properties [...] An item (including its state) is so far not bound in time and space and this is on purpose - it simply describes a situation without fixing it at a certain point in time'.

3.2.2 Sentilo

Sentilo [41] is the product of a project started in November 2012 by the Barcelona City Council and conceived for making Barcelona as a reference point in the field of Smart Cities. The name Sentilo was chosen because it means *sensor* in Esperanto, underlying the intention of openness and universality in the use of a platform.

Sentilo is a platform that offers an open source API based on REST interface and provides a data publication and subscription Java system based on Redis. In fact, even if the poll protocol HTTP is used, Sentilo developers implemented a sort of push system so that users can subscribe to the services about which they want to receive updates. The REST API defined for Sentilo allows to register new items, modify them or query their characteristics, to read, publish or delete sensors observations and send orders to the sensors. An alarm mechanism is also provided, in order to trigger commands when some conditions defined by the user about sensor data are met. Items in Sentilo are organized according to a hierarchical structure. They are identified by a unique ID and can be given other information, even custom ones. The standard data format used for message payload is Json, but future releases are going to support XML, too. All information about Sentilo resources is stored in MongoDB, while sensors observations and orders are stored in Redis, but relational databases can also be used. Moreover, it is possible to extend Sentilo's functionalities by implementing internal modules that do not affect Sentilo's core. For what concerns security, Sentilo creators developed a token-based authentication system to identify the petitioner of the request.

The strong points of Sentilo are the possibility of extending its functionalities, the presence of a hierarchical and slightly customizable items representation and the implementation of an authorization and role-based permission mechanism that facilitates the interaction in the same context of multiple users with different roles. Sentilo also provides a graphical interface for the administrator. An interesting feature is the possibility to send the sensor location in the data message, in order to visualize items on the map and track their path if they are mobile devices. Unfortunately such feature cannot be exploited in the Patavina project for the moment, as the sensors used are not able to send their GPS location.

As regards its weak points, Sentilo lacks an MQTT interface and of an already implemented graphical user interface.

3.2.3 Parse

Parse [42] is a cloud-based data management system that allows people to quickly develop web and mobile apps. More specifically, it is a Backend as a Service (BaaS) solution, a turnkey service that adds user authentication, push notifications, social media integration, location data, and data analytics into any app. It was acquired by Facebook in 2013 with the aim of adding Mobile BaaS capabilities to the existing platform and, as IoT backends are the logical extension of mobile backends, in the end of March 2015 Facebook announced Parse for IoT. Parse for IoT is a line of Software Development Kit (SDK) for connected devices, such as Arduino Yún, a microcontroller board with built-in WiFi capabilities. Parse SDKs are directly put on hardware platforms and provide a simple interface for the REST API. Such SDKs make devices able to receive push notifications, save data and take advantage of the Parse Cloud. All Parse resources are represented as Parse Objects, uniquely identified and customizable. Particular objects are roles, which group users with common access privileges in order to support role-based access control. Also storing data on Parse is built around the Parse Object: there is no need to explicitly create a database or tables to use Parse, data will be automatically stored in the cloud. Finally, it is possible to extend Parse for IoT functionalities by creating the so called Cloud Modules.

The strengths of Parse are the high customization available for Objects and the presence of a solid permissions and roles structure to condition user access.

The major weak points are instead the need of installing the SDK on each device and the lack an existing MQTT binding. Moreover, being Parse for IoT so recent (it was officially announced just a few months ago), there still is no consolidated user experience to highlight pros and cons of this tool.

3.3 Platforms comparison and final choice

Table 3.1 summarizes the presence or lack of the desired features (see Section 3.1) for the three platforms. All the three presented platforms represent valuable middlewares for IoT, but at the same time all of them lack of some useful feature. We chose not to use Parse for IoT, since it requires the SDK installation on the hardware devices, whereas Patavina team needed a platform more or less independent of the particular sensors and network topology used.

FEATURE	OPENHAB	SENTILO	PARSE
modularity	✓	✓	✓
MQTT support	✓	×	×
Json support	✓	✓	✓
data storage	✓	✓	✓
send orders to sensors	✓	✓	kind of [1]
items hierarchical representation	not really [2]	✓	not really [2]
items custom representation	×	✓	✓
user conditional access	×	✓	✓
add timestamp to transmitted data	×	✓	kind of [3]
administration console	✓	✓	✓
already implemented UI	✓	×	×
growing community	✓	✓	✓

[1] This service is not explicitally implemented, but it is possible to send push notifications after objects are saved
[2] Items can be gathered into groups
[3] Not contemplated, but feasible

Table 3.1: Comparison of the platforms

All the features currently missing in openHAB are instead present in Sentilo: it is able to handle the data timestamp, to customize the items representation and it also provides an authentication infrastructure to manage the coexistence of multiple users with different privileges. In view of these considerations, we decided to keep using openHAB and meanwhile adapting Sentilo to fit Patavina requirements. In this way the most suitable platform can be chosen time by time according to the use context: openHAB better fits small contexts, such as automated houses or buildings, with few users with same privileges, whereas Sentilo better serves wider scenarios such as Smart Cities.

I was in charge of adapting Sentilo to the Patavina Technologies project. The very first thing I had to do was the implementation of an MQTT interface over REST, in order to make Sentilo able to communicate with the MQTT broker.

4. The protocol bridge

Sentilo, the chosen platform, makes the Things and the Web interact via REST API over HTTP, a software architecture style illustrated in the following sections. On the contrary, the Application Server acts as a MQTT client that processes and interprets the messages from the sensors and sends these 'new' messages to an MQTT broker. HTTP and MQTT are quite different protocols, even just at a macroscopic scale as they use a pull and a push technology, respectively. Hence, I added Sentilo a 'bridge' module that works in a plug-and-play fashion in order to make it fit the existing architecture developed by Patavina Technologies. The objective of the bridge is to connect Sentilo to the MQTT broker by supporting both REST and MQTT and acting in this way as a protocol bridge that transforms MQTT messages in HTTP messages and viceversa.

Interoperability between protocols and devices is not a new problem, albeit it is a key factor for the success of an IoT system. Hence, before starting to create an architecture from scratch, I firstly searched for already existing solutions in the literature. The only contribution worth mentioning is *Ponte*[1] [43], a M2M bridge framework that speaks CoAP, MQTT and HTTP and that is based on *Node.JS*. In spite of its many excellent and customisable features, *Ponte* has been built in order to work by replacing the publish/subscribe broker or a REST API provider. But, we did not want to modify the existing MQTT broker, mainly considering that a complete system with openHAB as IoT middleware was already functioning. Moreover, the REST provider is part of Sentilo, which we decided to modify as little as possible in order not to violate the license under which the software is released: we therefore tried to improve it by exploiting its modular and extensible structure rather than altering the existing core. Hence, we opted for not using *Ponte* and rather develop an *ad hoc* framework.

In the following sections I will firstly describe the REST API and the MQTT protocol, highlighting the differences between the two, and then I will outline how I developed the protocol bridge.

[1] *Ponte* comes out from the Ph.D thesis of Marco Collina (2014).

4.1 The protocols

At the application layer, thing-driven approaches leverage binary protocols and data formats that are specifically designed for M2M communications. Such protocols typically introduce little overhead and minimize battery consumption, as they are implemented to optimize the exchange of many short messages, but they are rarely reused for other services. The most widespread open protocols specifically designed for the IoT are MQTT and the Constrained Application Protocol (CoAP), which are based on TCP and UDP, respectively. Patavina Technologies team decided to use MQTT, because of some useful features that are not supported by CoAP, such as the possibility of using wildcards for subscribing to many services with a single request and the possibility of easily having the Network behind a firewall or NAT.

4.1.1 MQTT

As its developers state [44], the Message Queuing Telemetry Transport protocol is a lightweight event and message oriented protocol allowing devices to asynchronously communicate efficiently across constrained networks to remote systems. MQTT has been implemented for easily connecting the Things to the Web in a M2M scenario, and therefore it can support unreliable networks with small bandwidth and high latency. The shortest message employs only 2 bytes, as in Machine Type Communication messages are typically short and a heavy header may become the bottleneck of the communication. The protocol has a client-server pattern: the server part is represented by a central broker that acts as intermediary among the clients, i.e. the entities that produce and consume the messages.

MQTT revolves around the concept of topic. Topics are UTF-8 strings used by the broker to filter messages for each connected client. A topic consists of one or more topic levels separated by a forward slash and in this way they provide a logical structure of common themes to be created, like any file system. Topics are used by clients for publishing messages and for subscribing to the updates from other clients. This pub/sub mechanism avoids a continuous polling to a consumer entity that wants to receive updates from the data producers: through a topic subscription, a MQTT client is constantly receiving all the messages sent to that topic by other clients. When a client subscribes to a topic, it can use the exact topic the message was published to or it can subscribe to multiple topics at once by using wildcards[2]. There exist the single-level wildcard (represented by a +) and

[2] Notice that a wildcard can only be used when subscribing to topics and its use is not

the multi-level wildcard (represented by a #) that are substitutes for exactly one topic level and an arbitrary number of topic levels, respectively.

As to reliability, MQTT is based on TCP, so it provides standard TCP delivery reliability, in addition to its own Quality of Service (QoS) mechanism, for which there exist three different levels of message delivery quality: the message can be successfully sent at most once, at least once or exactly once. The client connection can be interrupted without consequences as the broker is persistent. Furthermore, MQTT supports the concept of session, which is maintaned by a continuous exchange of keep-alive messages between the clients and the broker.

MQTT libraries have been provided for all major IoT development platforms, for the two major mobile platforms, i.e. Android and iOS, and for several programming languages (Java, C, PHP, Python, Ruby, Javascript). The MQTT protocol has recently been standardized at the Advancing open standard for the information society (OASIS) consortium.

4.1.2 REST

The Representational State Transfer (REST) [45] is a a software architecture style[3] for building scalable web services, typically over HTTP. It originated from the Ph.D thesis of Roy Fielding in the year 2000.

For a service to be identified as RESTful, the following five constraints must be respected:

- *client-server*: a RESTful service follows a client-server model, with separation of concerns;

- *stateless*: at the server side, no information about session and client context is retained and each request is an independent transaction that is unrelated to any previous request. So, each client request needs to contain all the information necessary to service the request and the client is the one that holds the session state. In this way servers are simpler and this enforces scalability;

- *layered system*: client and server may not be directly interconnected. Intermediary servers may improve system scalability by enabling load balancing and by providing shared caches and may also enforce security policies;

permitted when publishing a message.

[3] Notice that it is not a standard.

- *cacheable*: clients and intermediaries can cache responses, allowing an improvement in scalability and performance;

- *uniform interface*: the uniform interface between client and server allows each part to evolve independently. This constraint is based on four notions: first of all, individual resources must be identified in the requests and, secondly, these resources can be manipulated according to the CRUD pattern: *Create, Retrieve, Update* and *Delete*. The third constraint binds messages to be self-descriptive thus including all the information needed to be processed. Finally, the concept of Hypermedia as the Engine of Application State (HATEOAS): a hypermedia-driven site provides information to navigate the site's REST interfaces dynamically by including hypermedia links with the responses allowing the REST client to have no prior knowledge about how to interact with any particular application or server.

Actually, a sixth constraint has been defined, but, as it is tagged as optional, it does not really represent a constraint. It says that servers can temporarily extend or customize the functionality of a client by the transfer of executable code.

The concept of resource is central in RESTful services. Every resource is globally and uniquely identified by a Uniform Resource Identifier (URI) and resources are an abstract entity disconnected from their representation: the server may send data as Json or XML for example, but none of these formats is the server's internal representation.

Currently, thousands of businesses offer REST APIs for creating new applications and the majority of them (included the API provided in Sentilo) communicate over HTTP.

4.1.3 Comparison

As the above descriptions tell, REST and MQTT have very different characteristics. In an MQTT system, clients subscribe to a pool of topics that represent the 'data' about which they want to receive updates. There exist no different methods for performing different actions on items, everything is about publishing and subscribing to topics, whereas in REST there are resources and operations that can be done over these resources. It is possible to mimic in MQTT the different actions that REST allows to perform over resources by including the action in the meaning of the topic, thereby representing with a single topic a specific operation over a specific item, rather than a resource.

Through subscriptions in MQTT the pushing mechanism is activated. On the other side, REST requires the client to autonomously ask the server for the information he wants. It is possible to build a sort of pushing technology on REST, like Sentilo developers did, but nonetheless it requires the capability of having a HTTP server at the client side, which may require a lot of effort, especially in mobile devices. Another technique called pushlet exploits the persistence of HTTP connections and leaves the response unterminated so that the server can keep sending information to the client. The response is perpetually kept open and the server periodically sends snippets of JavaScript to update the content of the page, thereby achieving push capability. The major drawback to this approach is the lack of control the server has over the browser timing out.

4.2 The bridge

Trying to respect the Patavina Technologies will of not altering Sentilo platform, I developed an external module that acts as an MQTT-HTTP bridge. When it is not running, Sentilo keeps working but it has no way to communicate with the Application Server (through the MQTT broker) and hence with the physical sensors network.

The bridge is made up of two components:

- the *HTTP component* includes a client, which sends the sensors messages to Sentilo, and a server, which is needed to handle Sentilo orders[4];
- the *MQTT component* is made up of just a client which interacts with the MQTT broker.

So, each component is responsible for communicating with an external element and they internally interact with each other.

Considering that the whole Sentilo project has been written in Java, I also developed the bridge module in Java. It is implemented as a Sentilo agent, i.e., Sentilo is independent of it but it needs Sentilo to work, since it is dependent on it through Maven dependencies. The HTTP elements make use of the Apache library, whereas I used the Eclipse Paho library for the MQTT client. Mimicking Sentilo style, I also made use of the Java Spring framework, an application framework and inversion of control container for the Java platform.

[4] Remember that Sentilo's feature of generating alarms over data is not used because the physical network does not have the capability of handling alarms.

4.2.1 Patavina MQTT messages

As outlined in Section 2.2, LoRa nodes are uniquely identified by an address of 8 hexadecimal numbers. Each node is associated to a pool of different sensors, like humidity and temperature, and to some available commands. The sensors can send data, receive orders and send responses to these orders, and the message payloads are in the Json format.

The following list summarizes and describes the available MQTT services:

- *getNodeList*: when Sentilo publishes this message,

  monitoring.patavinatech.com/getNodeList

  it triggers several *newNode* messages from the Application Server, one for each node. The choice of sending multiple messages aims at simplifying the transmission behaviour when a new node appears when the architecture is already running: rather than sending a message containing information about all nodes stored in the database, the broker just needs to publish a message for the single new node;

- *newNode*: this type of message contains all the information for characterizing a node: its address, the sensors belonging to it together with their unit of measure, the commands supported by the node and the corresponding responses.

  monitoring.patavinatech.com/<AS>/<NS>/newNode

  where AS idenitifes the Application Server and NS the Network Server. The structure of the Json payload is the following:

```
{
    "addr" : "<addr>",
    "data" : [
        "sensorString" : "<sensorUnitOfMeasure>",
        "sensorString" : "<sensorUnitOfMeasure>",
        ...
    ],
    "commands" : ["<cmdString>", "<cmdString>", ...],
    "responses" : ["<respString>", "<respString>", ...]
}
```

In the data field, there are the names of the sensors, e.g., temperature and illuminance, and the data associated to LoRa statistics, such as the SNR, the timestamp, the frequency, etc;

- *data*: this service is used for publishing (and receiving) the sensors observations. The sensor that is sending data is univocally identified in the topic through the topic levels of the NS, of the node and of the sensor string.

  monitoring.patavinatech.com/<AS>/<NS>/<nodeAddr>/data/<sensorString>

  Sentilo needs to subscribe to the *data* messages of all nodes (by using wildcards), thereby receiving all the observations from every sensor. The Json payload has the following structure:

```
{
    "value" : "<sensorValue>",
    "timestamp" : "<timestamp>"
}
```

  where the timestamp field is optional;

- *command*: whenever the user wants to send an order to a specific node, Sentilo has to publish a *command* message, whose topic is:

  monitoring.patavinatech.com/<AS>/<NS>/<nodeAddr>/command/<cmdString>

  where the <cmdString> topic level refers to the command name specified in the *newNode* message of that node.
  The Json payload is in the following format:

```
{
    "value" : <value>
}
```

  with the value of the value field usually being an integer number;

- *response*: after receiving a command, a node publishes the corresponding response; hence, Sentilo needs to subscribe to the response service of all nodes.

  monitoring.patavinatech.com/<AS>/<NS>/<nodeAddr>/response/<respString>

  where the <respString> topic level refers to the response name specified in the *newNode* message of that node.
  The structure of the Json payload is the same of the command messages but with the value that can be 0 or 1, according to the command having been succesful or not, respectively.

4.2.2 Sentilo REST API

In Sentilo, the items are organized in a hierarchical structure. Any entity capable of generating data is classified as a *sensor* and sensors belong to *components*, which correspond to hardware or software elements with a known geospatial location[5], which may be fixed or even mobile. Components, in turn, are gathered into *providers*, which represent logical entities that 'administer' sensors with common features. The decision of a strategy for assigning providers to sensors, i.e., how to group sensors into providers, is up to the user. Providers play a relevant role in Sentilo for two reasons: REST messages from and to sensors must contain the corresponding provider ID in the URI, and users permissions are given upon providers, rather than upon the single sensors.

Sentilo provides the following services:

- *catalog*: allows to register, update or delete sensors and other entities and to retrieve information about them. An administration console accessible via authentication lets the administrator perform CRUD operations about all the entities, i.e., not only sensors but also components, providers, sensor types, users and the permissions they have. On the other hand, the REST API allows only some operations: for example it is not possible to create new providers with a REST request;

- *data*: this service is for publishing and retrieving the observations of the sensors (also historical data);

- *order*: for sending commands to single sensors or to all the sensors of a provider;

[5] For instance, a weather station that contains several sensors.

- *alarm*: it allows the end user to set internal alerts, e.g., depending on the last data received, and to the sensors to send alarms associated to these alerts ;

- *subscription*: for performing subscriptions to the data, order and alarm services in order to be updated in the case of new data. In fact Sentilo developers defined a sort of pushing technology over REST and the subscription service allows the platform clients to subscribe to system events. For this service to be effective, the platform client needs to have an HTTP server available, to which the Sentilo platform sends the messages related to the subscriptions the client performed. In this way the client does not need to perpetually ask the server for updates but it is directly informed by Sentilo. Notice that this workaround does not provide a publish/subscribe mechanism as complete as in MQTT and requires the capability of having an open socket at the client side.

Sentilo's URIs mainly have the following pattern:

http://<domain>:<port>/<service>/<entityId>/<sensorId>

where <service> is one of the services just described, <entityId> can be either the id of the alert for the alarm service or the id of the provider for the other services, and <sensorId>, i.e., the id of the sensor, is specified only in the latter case and may be optional.

4.2.3 The bridge implementation

The bridge's role is to interact with the HTTP Sentilo server and the MQTT broker. When it is started, it needs to send some messages to both sides in order to be properly connected. At the HTTP side, the bridge needs to know which sensors and providers are already registered in Sentilo because when performing catalog operation it is important not to interchange POST and PUT methods: the former is used for inserting new elements, the latter for updating existing ones, and using the wrong HTTP method generates an error response. To retrieve information about all the items registered in the platform, the bridge performs a *GET catalog* request and parses the payload of its response to extrapolate the information it needs. In this way the bridge can initialize a map containing the associations between sensors and providers, needed for building the correct URIs when converting MQTT messages. Also the subscriptions to existing providers orders' must be done, assuring in this way the reception of all the orders directed to the sensors from an external user. For what concerns the MQTT side, the bridge needs to

perform all the subscriptions necessary to receive the data. It firstly makes a subscription to all *newNode*, *data* and *response* messages by taking advantage of the wildcards and thereby ensuring it will receive all messages from all the existing nodes. After that, it publishes a *getNodeList* message, which will launch the sending of *newNode* messages from the broker, one for each node. Any MQTT publish message received from the MQTT component is duly handled and sent to Sentilo through the HTTP client. When the HTTP server receives a command message, the bridge appropriately manages it and publishes the corresponding MQTT message to the broker.

The MQTT protocol provides some features that are not present in HTTP, like the QoS level associated to messages: since HTTP does not support this option, the bridge always sets the QoS to level 2 (exactly one delivery) for MQTT messages. As the connection is supposed to be wired, the higher accuracy implied by level 2 does not represent a big deal. Other flags contemplated by the MQTT protocol, such as the retained option that allows to keep a message 'stored' in the broker, are simply not set since they find no correspondant in HTTP. I will now give a detailed illustration of the message conversion mechanism and the choices it implied. Elements concerning security and users roles and permissions will instead be described in the next chapter.

New sensors

The MQTT topic and the Sentilo syntax for the registration of new nodes are, respectively:

	<MQTTprefix>/<AS>/<NS>/newNode
POST/PUT	<HTTPprefix>/catalog/<provId>/<sensorId>

LoRa *newNode* messages include information about the new node, i.e., the node address (unique within the NS domain), the supported commands and corresponding responses, and the types of data the node can generate. For each sensor, the bridge registers in Sentilo a new sensor through the catalog service. The component associated to the sensors is the same for all the sensors of the same node and I chose its unique ID to be <NS>-<nodeAddr>, so as to make recoverable both the Network Server and the node IDs. A new sensor is created also for each available command. This is due to the fact that the payload of order messages is supposed to contain only the value field; to know to which command the value is associated, it is necessary to address it somewhere, and I decided for the URI[6]. The mapping between the

[6] Another possibility could have been the inclusion of the command name in the value

triplet (NS, nodeAddress, sensorString) that identifies a sensor in the LoRa domain and the ID of the sensors registered in Sentilo must be bijective, so all elements of the triplets should be included in the sensor ID and be recoverable. My choice has been the following:

(NS, nodeAddr, sensorStr) $\longleftrightarrow$ sensorID = <NS>-<nodeAddr>-<sensorStr>

where the delimiter '-' has been chosen because it is not used for other purposes yet. Notice that, although the component ID I chose is a prefix of the sensor ID, the REST URIs do not make reference to the component the sensor belongs to, so all information needs to be provided in the sensors' IDs. Albeit it is unlikely to have sensors belonging to different Network Servers with the same name, I decided to include this information in the sensor ID to avoid troubles in the case of homonimity. In fact, there exists a unique NS in each LoRa network, but there could be more LoRa networks in the whole system, e.g., one for each company. However, the reference to the Application Server is not necessary: first of all there typically is only one, but, even in the case of multiple ASs managing the same LoRa networks, the physical sensors would be exactly the same and two nodes with the same name and the same NS would certainly be the same. So there is no need to include the AS reference to discriminate over nodes. Anyway, the name of the Application Server needs to be known for sending messages to the sensors (such as orders), so I decided to save it in Sentilo under the *additionalInfo* field provided in the sensors profile. Each new sensor is assigned to a provider and this distribution is done according to the logic implemented, as illustrated further.

Other information registered in Sentilo for each new node includes the unit of measure indicated in the MQTT payload and the type of the sensor, which is the sensor name for data sensors (e.g., temperature), *statistics* if the sensor is associated to LoRa statistics or *commandSensor* for the 'fake' sensors related to the commands. Since new sensor types cannot be created by simply using the REST API, sensors whose type does not exist in Sentilo are marked as 'generic' (more information is given further). When inserting a new sensor in Sentilo, it is necessary to know if it was already registered or not: in the former case the HTTP request must be a PUT, indicating an update, in the latter a POST, which is typically used for creations. For

field. For example, to send the value 3 in relation to the *changeReadingInterval* command, we could have used the syntax "value": "changeReadingInterval-3"' in the payload. However, this strategy would have required the parsing of the value field and a harder work for the bridge, and might have been misleading for values containing the symbol '-'. Considering that the number of commands associated to a node is usually small, we opted for creating the 'fake' command sensors.

this purpose the bridge has a resources map that stores all the IDs of the registered sensors and their associated providers, along with the components' names.

Data

The MQTT topic reserved to sensor data and the URI used by Sentilo are, respectively:

<MQTTprefix>/<AS>/<NS>/data/<nodeAddr>/<sensorStr>
PUT <HTTPprefix>/data/<provId>/<sensorId>

To build the URI it is necessary to have the provider ID and the sensor ID; the latter is easily obtainable from the MQTT topic as <NS>-<nodeAddr>-<sensorStr>, as explained previously. The corresponding provider is instead found in the resources map that the bridge keeps updated and that contains each provider/sensor association. For what concerns the payload, Sentilo data payload requires the value field and considers as optional the fields of timestamp and location. The sensors actually used by Patavina do not have a GPS module and hence they are incapable of tracking their position; so the location field is not used. Instead, the timestamp is usually sent and, if not present, the timestamp at which the bridge receives the message is added. Timestamps are converted from the UNIX format used by LoRa nodes to Sentilo special format: the former indicates the number of seconds passed from the 1^{st} January 1970 at UTC, the latter has the pattern `dd/MM/yyyyTHH:mm:ss`. So, the date 28^{th} August 1991 at 14:00:00 is represented as 683388000 in the Unix timestamp and as 28/08/1991 T 14:00:00 in Sentilo format.

Commands and Responses

The orders sent from Sentilo and from the MQTT broker have the following URI and topic:

PUT <HTTPprefix>/order/<provId>/<sensorId>
<MQTTprefix>/<AS>/<NS>/command/<nodeAddr>/<commandStr>

Commands, or orders, are sent from Sentilo (on behalf of some external user) to a command sensor, whose ID allows the bridge to identify the command name and the node to which forward the order. If the sensor is not found in the bridge resources map, the request is simply not handled.

For what concerns responses, remember that this service is not contemplated by Sentilo, so MQTT response messages are forwarded to Sentilo as data messages that target the sensors specifically created for handling the commands. The bridge has a command/response map that contains the associations between each command name and the corresponding response name for the all nodes. The responses in MQTT and REST follow this pattern, respectively:

<MQTTprefix>/<AS>/<NS>/response/<nodeAddr>/<responseStr>
PUT <HTTPprefix>/data/<provId>/<sensorId>

Once the bridge has identified the correct sensor ID, it is straightforward to retrieve the corresponding provider from the sensor-provider map that the bridge keeps updated. Responses can be either 0 or 1, according to the order having been respected or not. Thus, the bridge needs to save both the value used before sending the order and the value sent in the order; in this way, the *data* message sent to Sentilo carries the correct value, whether the command has been succesful or not.

The alarm service provided by Sentilo is not contemplated by the LoRa architecture for the moment. This way the bridge does not handle alert and alarm messages.

Operation not supported by the REST API

The REST API provided by Sentilo has some limitations for what concerns the *catalog* service and does not allow to perform all the operations that on the contrary are available when using the administration console. More precisely, the creation of new providers and of new sensor types is not supported by the REST API. The reason for this restriction is the belief of the platform developers that only the system administrator should be in charge of performing such operations, without contemplating the idea that providers and sensor types could be not fixed a priori. Nonetheless, we deemed the provider creation to be central in the Patavina Technologies project, as the strategy for assigning providers to sensors may change from customer to customer, thereby disabling the possibility of manually adding each provider in Sentilo. Furthermore, the use of a single and predetermined provider for all the registered sensors is to discourage, as providers are central in the user permissions system and therefore play a non negligible role in the security framework. Luckily, being the bridge agent dependent on Sentilo, it can take

advantage of its Java code and use the functions that register new entities in the platform.

Anyway, a problem still arises: as soon as a provider (or a sensor type) is created, it may not be 'visible' to Sentilo, which means that a sensor registration for that provider would receive a *403 Forbidden* response, thus resulting in an error. This happens because when a new sensor message is received, Sentilo first verifies that the petitioner of the request[7] has got administration permissions over the involved provider but, to perform this check, Sentilo looks up a permission map of the entities registered in the platform, which is updated every 5 minutes (according to the information stored in the database, which is always updated). Therefore, when a new provider is created, Sentilo will recognize it as existing only after 2.5 minutes in mean and after 5 minutes in the worst case. It is not possible to force the update of the permission map in the code to notify Sentilo about the existence of the new provider because the map is accessible in no way (non public visibility of the necessary methods). The workaround I implemented to solve the problem consists in starting a new thread every time there is the need to create a new provider, This thread is responsible for successfully registering the new nodes under the just created provider. The thread tries doing it and, if a 403 error response is received, it *sleeps* for one minute and retries it. Within 5 minutes the procedure will be completed and the thread dies. Data messages from the involved sensors are temporarily stored by the bridge and sent later with the correct timestamp.

Similarly to providers, new sensor types are supposed to be created only by using the administration console. The same process used for providers may therefore be used, but we deemed sensor types to be ancillary and the process described above to be computational heavy. Thus, the sensor types that are predicted to be used are injected in Sentilo during the installation phase together with a generic sensor type to which all 'unknown' types are mapped. It is quite likely to predict all the sensor types that will be used to classify the nodes, as users know which sensors they are going to use and what they measure. Notice, however, that it is always possible to add new types by using the admin console interface.

4.2.4 Java project structure

Sentilo has been written in Java and has a modular structure, with all modules being related to each other through Maven dependencies. Maven is a build automation tool that allows to declare the structure of a Java project,

[7] Identified by a header field in the HTTP request, as explained in Section 5.2.

the external libraries it uses and the dependencies among the different elements of the project. It simplifies the extension of Sentilo, allowing the creation of own modules (called *agents* by Sentilo developers) that are not necessary to Sentilo to work but that rely on Sentilo and broaden its functionalities.

To start the bridge, the user just needs to run a bash script, which activates both the MQTT and the HTTP component establishing the necessary connections with the Sentilo platform and the MQTT broker. Afterwards, every message received by the MQTT client is redirected to the bridge core, which identifies the type of message (*newNode, data* or *response*) and sends an HTTP request to the Sentilo REST interface accordingly. Similarly, when an order is sent to Sentilo, since the HTTP component of the bridge is subscribed to the order service of all providers, it receives the order and publishes a command message to the broker, after proper handling. An external file contains the configuration data for initialising the bridge: the IP addresses of Sentilo server and MQTT broker, the identity token to insert in the REST requests[8], and some configuration options for the bridge, such as the SSL enabling, and the credentials if the broker supports client authentication.

The Java project includes the following classes:

- a starter class, whose main method reads configuration data from an external file and starts the bridge. Thanks to the *@PostConstruct* notation, the unique method of this class is invoked as soon as the bridge module is started. If Sentilo is not running at the address given in the configuration file, the bridge keeps waiting for it;

- the core class of the bridge, *BridgeClient*, which provides methods for identifying the types of messaging received, properly converting MQTT messages in REST messages and viceversa, and forwarding the corresponding converted message to the specific element;

- if the *BridgeClient* class represents the 'mind' of the bridge, then *JsonBridge* is its arm, the class that really processes the messages. It is responsible for interpretating the payloads of the messages, keeping the volatile databases updated and building the corresponding payload with the other syntax;

[8] As explained in next chapter, requests to Sentilo must contain an authentication key in the header to identify who is performing the requests and the permissions he owns upon the sensor. There exists an all-powerful role which is assigned administration permissions upon all items, by default. The bridge uses it to be sure that all requests have a successful response.

- a class that directly makes use of the methods provided in the original Sentilo project in order to create new providers, operation forbidden by the REST API (see Section 4.2.3, subparagraph *Operation not supported by the REST API*). Another class implements a Thread responsible for ensuring the correct delivery of the REST messages immediately after the creation of a new provider (as the Sentilo server detects the presence of a new provider after a maximum time of five minutes and prior to that moment no message for creating sensors under that provider is correctly processed);

- a class that implements an MQTT client for handling all message exchanges with the broker and which is used by the real and proper bridge element;

- a class responsible for performing the HTTP requests to Sentilo server, also used by the proper bridge;

- a package of classes that implement a HTTP server, to which Sentilo forwards the commands addressed to the sensors. It runs on localhost as it is part of the bridge and it only has to redirect the messages of the bridge core for their processing prior to transmitting it to the broker;

- other ancillary classes, e.g., for representing elements used by the bridge, for payload parsing and static classes containing the tokens of the MQTT and REST message syntax.

As previously mentioned, the sensor-provider association is up to the user. Nevertheless, knowing the provider assigned to each sensor is essential for building up the URIs. Furthermore, as already explained, distinguishing different providers is central to exploit the possibility to give different read and write permissions to the users, as a user is granted the same privileges over all the sensors belonging to the same provider. Under these considerations, we wanted to realize the bridge in such a way to be flexible with respect to the context of use. For this reason we decided to take advantage of the concept of *abstract* classes in Java, which allows to define but not implement abstract methods that will be implemented when extending the class. They can be thought as in between classes and interfaces. In this respect, the core class of the bridge (`JsonBridge.java`) Java project contains the abstract method `assignProvider` which returns the provider associated to the sensor. So, for each scenario the client would like to have, it is sufficient to create a new class that extends `JsonBridge.java` and implements such method. Taking into consideration the context of smart cities, I thought about possible strategies of assigning providers. I implemented two classes:

- the first one gathers together sensors of the same type. So, all command sensors are under a single provider, and the same happens for all sensors in charge of receiving data about LoRa statistics. Then, all sensors that measure the same thing belong to the same provider. The reason behind this choice relies on the idea of having different persons responsible for different measures;

- a more specific role differentiation may be having people responsible for areas of sensors, e.g., a user role should have the same privileges over temperature and humidity sensors as they are both related to the meteorological area. So, I developed a class that assigns a provider to a sensor according to the area to which it belongs. A configuration file in the Json format tells the bridge which are the areas and which are the sensor types referring to each area.

Another interesting strategy would be to introduce a geospatial division in addition to the areas division, i.e., grouping together sensors that measure related quantites and belong to the same city district. In fact, there could be people specialized for monitoring the CO_2 concentration or other factors for each district. Although fitting the smart cities scenario, this strategy does not find support in the physical technology used by Patavina Technologies, which lacks a GPS module. Anyway, when using different sensors, this strategy is implementable in Java by exploiting the abstract method concept.

The Java implementation of the bridge is quite robust and capable of handling the most probable and common errors, such as problems in reading the payloads of the message arrived or the reception of error messages from Sentilo, without interrupting or altering the bridge's functionalities. It is also highly and easily configurable: the sockets of Sentilo and of the MQTT broker are inserted in a configuration file read by the bridge at the very beginning and also LoRa key words, e.g. the topic levels fixed names and the MQTT prefix, are read by the core classes from a static class. So, if changes in the MQTT syntax happen, it is sufficient to modify the values of the variables of this class, in a very intuitive way. The same happens for Sentilo syntax. The security aspects of Sentilo are treated in the next chapter, after a more general distention about security in IoT.

4.3 User interface

The bridge forwards the nodes information and their data to Sentilo. The final user, however, is not supposed to access Sentilo platform but a specific website showing data in a more friendly.

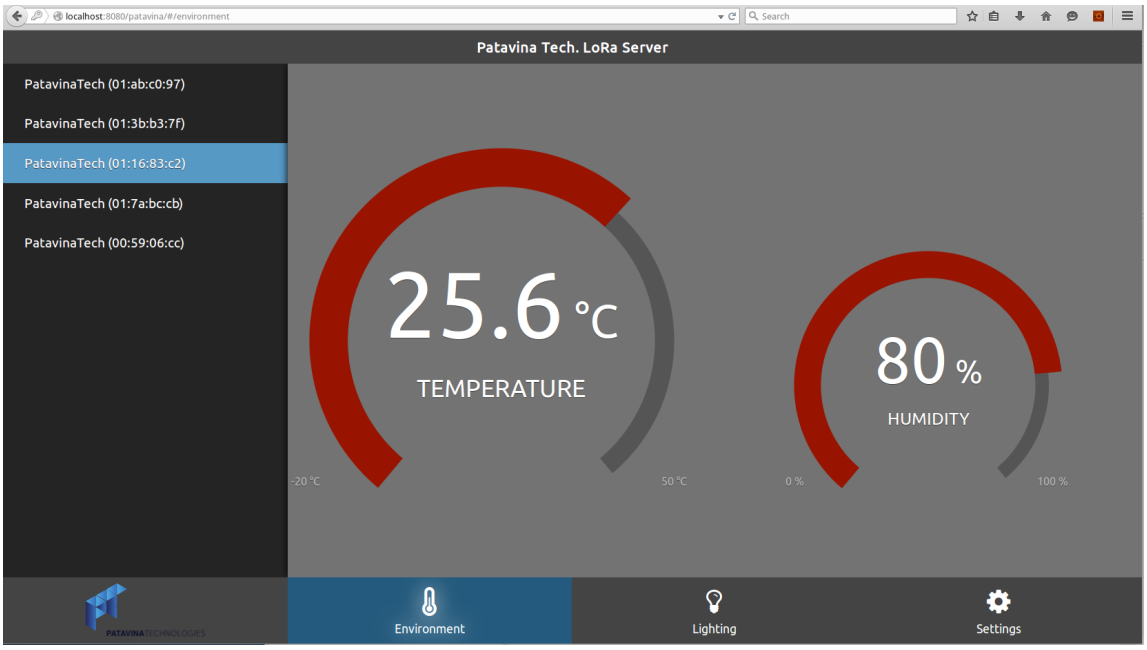


Figure 4.1:

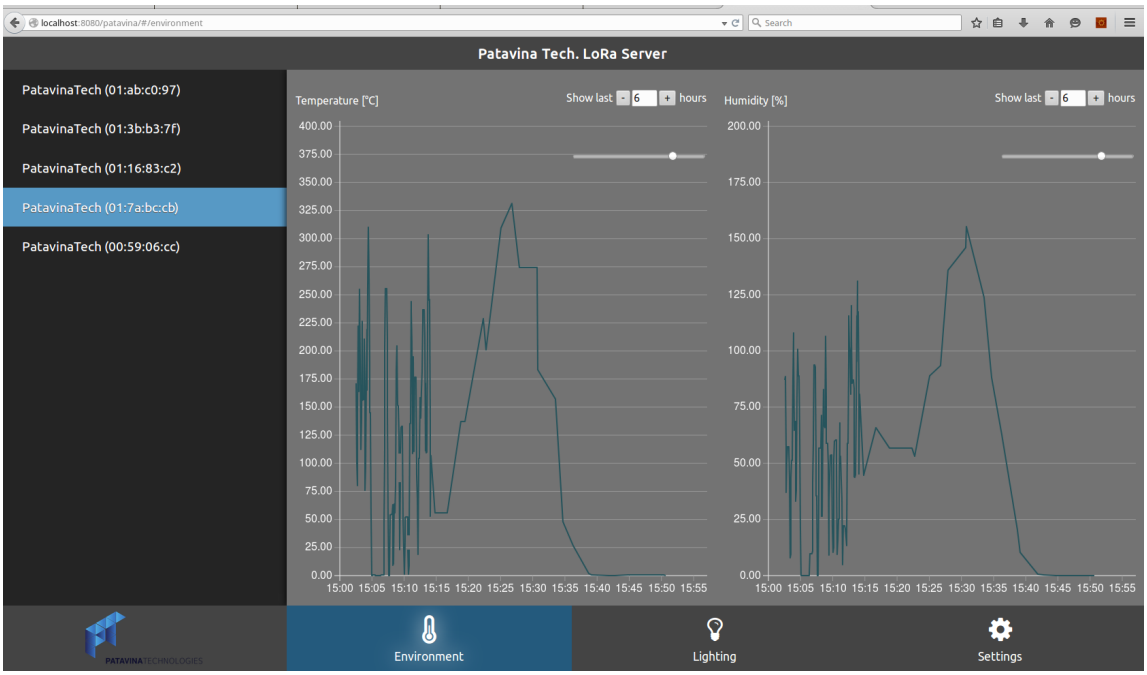


Figure 4.2:

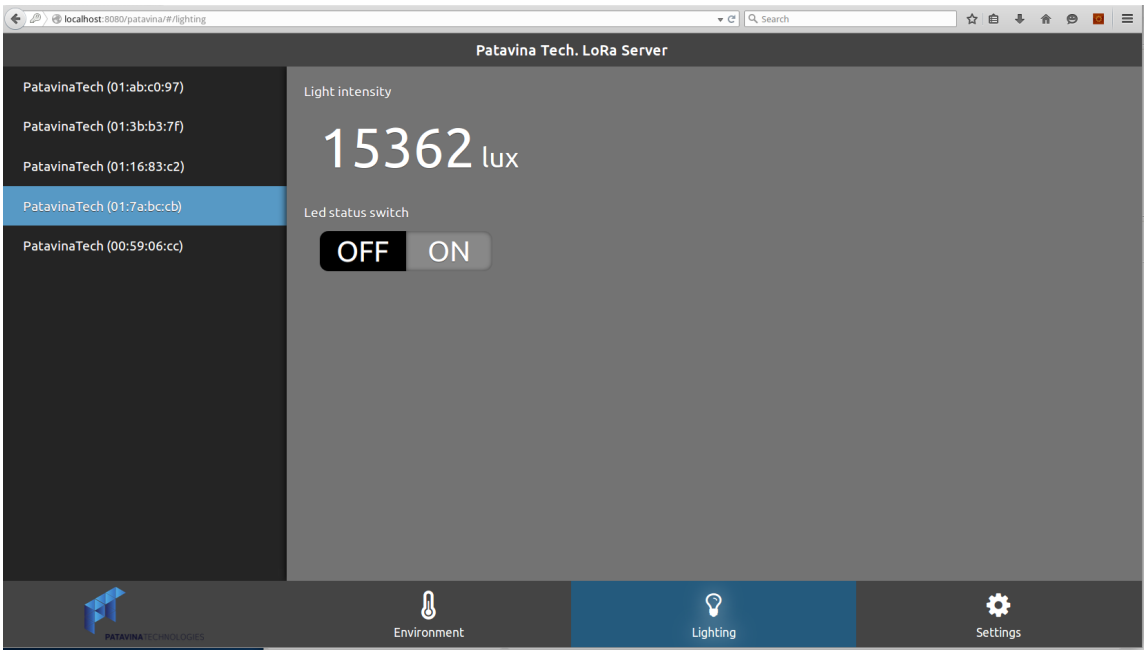


Figure 4.3:

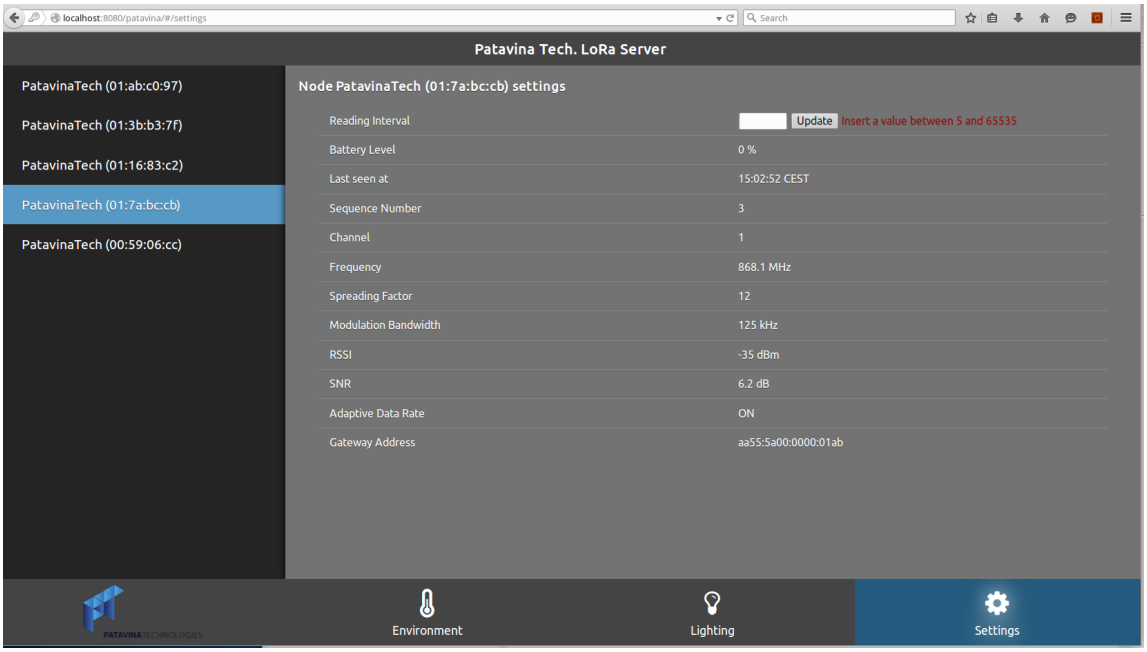


Figure 4.4:

The data shown depend on the permissions owned by the user, as explained in Section 5.3. The actual user interface is customised for the nodes used by Patavina Technologies, which contain sensors of temperature, humidity, light intensity and battery level; moreover, the gateways add some statistics mainly regarding the channel conditions. Figures 4.1, 4.2, 4.3 and 4.4 show some screens of the web interface.

The user can see the the active nodes (over which he owns at least reading permissions). For each node, the current values of temperature, humidity, light intensity and of the statistics are shown. It is also possible to visualize the graphs of the historical data and send some orders to the sensors, such as switching on or off the led (Figure 4.3) and changing the reading interval of the sensors (Figure 4.4). HTTP is not a push protocol, so that it is necessary to continuously ask Sentilo for the needed information as there is no mechanism for the server to independently send data to the client without the client first making a request. Anyway, there exists a technique called 'long polling' which emulates a server push feature: the client makes a request to the server, which holds it open until new data is available and once available, the server responds and sends the new information. When the client receives the new information, it immediately sends another request, and the operation is repeated. In this way it is possible to dynamically update the web interface and show the user the current values almost in real time.

5. Security issues

Security represents a central issue in IoT and revolves about the properties of identification, confidentiality, integrity and undeniability [46]. Since the IoT is built to broadly execute unverified user-implemented applications from different users, both applications and users can be sources of security threats to the IoT. Moreover the heterogeneous and distributed nature of IoT architectures greatly affects the degree of infrastructure protection. Thus, security needs to be granted at every element of an IoT architecture and to meet the new requirements implied by the pervasive presence of the Internet in any aspect of daily life [47]. This chapter sketches an overview of security in IoT scenarios and then focuses on the security measures concerning the interaction with the final user through the Internet.

5.1 Security in IoT

The Internet is under continual attack and this does not bode well for the IoT which relies on it and also incorporates many constrained devices for which it is hard to apply security mechanisms such as frequency hopping comunication and public key encryption [48]. But as IoT also touches many sensitive areas such as medical services, intelligent transportation and national economies, security represents a challenge that cannot be neglected: attacks and malfunctionings would just outweigh any of the IoT benefits. Security experts are currently investigating whether actual protocols can be integrated in IoT or new designs are required for accomplishing security goals. What mainly introduces new threats is the distributed nature of IoT architectures and the use of fragile technologies, such as limited-function embedded devices in public areas where they are accessible by anyone and may be physical harmed [46]. As sensors are typically simple devices with low power, they cannot even support ordinary security measures: network firewalls and protocols can manage the high-level traffic flowing through the Internet, but the protection of the endpoint devices with limited resources available to accomplish it raises new challenges and demands for revolutionary solutions [49].

Key features for gaining security are the following [50]:

- *identification*: the 'things' must be uniquely identified indipendently of their underlying mechanisms, e.g., the IP address they are associated to. Assigning a unique ID to devices is the basis for the authentication step and the consequent authorization [51];

- *confidentiality*: it is roughly equivalent to privacy and can be described as the property that information is not made available or disclosed to unauthorized individuals, entities, or processes [52]. Privacy is fundamental in an IoT scenario, in which a plethora of devices transmit messages leading to an explosion of data. Access to these data must be controlled mainly by means of cryptographic mechanisms and users access lists;

- *integrity*: for maintaining the consistency, accuracy, and trustworthiness of data over its entire life cycle. Data must not be changed in transit or altered by unauthorized people;

- *availability*: for any information system to serve its purpose, the information must be available when it is needed. Availability may be hindered by legitimate users too, if tampering with shared multimedia data or exhausting network resources to interrupt services available to other legitimate users.

Clearly, the IoT is prone to be more susceptible to attacks than the Internet, since billions more devices will be producing and consuming a large number of different services. From a network perspective, the sensors should open a secure communication channel with more powerful devices exploiting cryptographic algorithms and using an adequate system for exchanging the keys [46]. Such procedure is for example used in LoRa, where the nodes securely communicate with the Network Server and the keys for the AES algorithm are exchanged during the join procedure (see Section 2.3). A safe transmission over TCP/IP connections can be achieved by enabling TLS, which asks the parties to authenticate themselves and encrypts messages. At the application level security needs for different application environments are different, although data privacy, access control and disclosure of information are likely common requirements. In [53] the authors stress the crucial role of security and privacy and highlight how the public acceptance of the IoT will happen only when strong security and privacy solutions will be in place; as the Internet first appeared, no security infrastructure had been built and when the first security problems came out, the solutions proposed were just

patches. But in IoT security is intrinsic and the scientific community must find new solution for addressing this challenge.

5.2 Security aspects in Sentilo

Sentilo creators developed a quite complex security framework, which considers both a secure transmission of the messages and the existence of different users roles, features that made the platform gain the role of middleware for the project (see Chapter 3).

All the HTTP requests received by Sentilo are validated according to the AAA architecture: Authentication, Authorization and Accounting. This means that the platform first identifies the petitioner of the request, it then checks the permission for that user to perform the requested action over the requested resource, and it finally traces the request by auditing the action and who performed it. Notice also that it is possible to use the secure HTTPS instead of HTTP. The authentication part is enabled by the mandatory use of an identification field in the HTTP headers, resulting in the so-called token-based authentication [54]. The general concept behind a token-based authentication system is to allow users to enter their login credentials in order to obtain a token; this token guarantees access to a specific resource to the remote site for a finite period of time without using username and password. This approach introduces many advantages, for example the user avoids authenticating with username and password for each single request within a time-limited session, AJAX calls to any server or domain do not meet cross-domain problems[1] and it also enforces the server side scalability, as there is no need to keep track of the session. The token must be included in the header of any HTTP request with key *IDENTITY_KEY*. The very first thing that Sentilo server verifies is the presence of a valid authentication token, otherwise it does not even analyze the request and immediately sends a response with error code *401 (Unauthorized)*. When the token is valid, Sentilo checks whether the permissions associated to the token allow the petitioner to perform the request. When the user is not allowed to perform the requested operation, an HTTP response with error code *403 (Forbidden)* is sent. Finally, every information about the request is properly registered.

[1] Cross-origin resource sharing (CORS) [55] is a mechanism that allows only restricted resources on a web page to be requested from another domain outside the domain from which the resource originated. Hence, authentications with credentials may encounter some obstacles when the CORS mechanism is activated. Instead, by using a token, the user information is transmitted in the HTTP header, thus avoiding cross-domain complication. See Section 5.4.

The tokens are stored in a Mongo database, accessible only through authentication. A mechanism to distribute the tokens outside the platform has not been realized yet, so it is up to the platform user to create one for sending the tokens among the different users in a secure way.
In Sentilo, entities are associated to tokens made of 26 hexadecimal numbers; these tokens are built by using a fixed numerical prefix, the ID of the entity and the time (with milliseconds precision) of the token generation itself. These three elements are concatenated and the SHA-256 digest[2] of this final string is computed and finally converted to the hexadecimal format. The fact that the token is not based on a random string creates a significative security leak, as shown further in Section 6.2.

For what concerns the administrator console, it is accessible only via authentication and the administrator credentials are declared in a configuration file editable before installing Sentilo. In turn, the administrator may add other users with administration priviliges.

Adaptations

Sentilo is an open-source project, so the whole code is easily accessible via GitHub. It is straightforward that all passwords related to secure access must be changed in order to sell a safe and privacy-caring product. Therefore, all the credentials needed to access the database used (mongoDB for the *catalog* data, Redis for observations data and orders, and mySQL if the user also wants to store data in a relational database) should be modified. The administrator's username and password and the token associated to the all-powerful user (see next section) should be changed, too. To grant a more secure token-based authentication, the tokens generation function must be revised: I will prove the weakness of the one implemented by Sentilo developers. For this reason I traded the timestamp, which can be known or somehow derived, for a completely random string. In this way the tokens are still hexadecimal strings of length 26 and cryptographically encoded, but they are randomly generated and therefore more secure against brute-force attacks.

[2] The *Secure Hash Algorithm* [56] is a family of cryptographic hash functions which produce a message digest of fixed length from a fixed-length string and whose security relies in their not being reversible. SHA-256 produces digests of 256 bits.

5.3 User permissions

Security and data privacy represent a key point in Internet of Things applications. Nonetheless, they are not sufficient to ensure a fair and safe system, as not all users are equal and have the same privileges. Just think about a building that adopts a home automation architecture: the owners of a specific apartment would certainly be granted administration permissions over everything concerning their own flat, whilst other extraneous people should not have visibility about their sensors data and information. Now think about the individual in charge of checking the gas data of all flats: he should be able to read such values but not to modify them, hence to have reading but not writing permissions over the considered entities. So, basically, different roles should be assigned to different users, with each role being characterized by permissions over items, i.e., sensors and providers in the case of Sentilo. User permissions specify what tasks users can perform and what features users can access. In the Smart Cities context, it is fundamental to differentiate among users: common citiziens could be able to read the data of some sensors, but not to send them orders, whereas the individual responsible for managing a particular area should be granted also writing permissions over the concerned sensors. A quite intersting feature in this scope concerns the capability of temporally triggering stronger privileges when some conditions are met. For example, when fires occur, the sensors responsible for measuring the levels of CO_2 and CO in the air register very high and anomalous values, indicating danger. In that case any citizen should have the possibility of activating some water actuators, if present, albeit in normal situations these actuators may be activated only by a restricted number of people. Anyway, this feature has not been implemented yet.

For what concerns Sentilo, its developers have implemented a structure for managing users permissions. It is in fact possible to define *applications*, identified by a unique ID, and each application is associated to specific permissions over the entities registered in the platform. The permissions are associated to providers (the logical entities that group components and hence sensors) and to applications. So, every provider and application has its own identity token, which must be used in each HTTP request and allows Sentilo to identify the requester and to ensure that who makes a request is authorized to do it. The permission can be: only reading, reading and writing, administration. A read permission allows to retrieve information about the item and its historical data, a write permission allows also to send observation data and orders and to update or delete items in the platform. The admin permission over a provider lets the user create new sensors belonging to it.

By default, every provider owns administration privileges over its sensors.

There exists an application called *sentilo-catalog* which owns admin permissions over all entities, by default. Hence, when using its token in the HTTP header, the request cannot be blocked because of invalid authorization. The bridge uses its token for all requests it performs, as it must be able to fully manage everything concerning the sensors.

Being the permissions structure already defined in Sentilo, it was much easier to implement role differentiation. Instead such architecture is still missing in openHAB and the development of a permission infrastructure from scratch would have required a lot of time and effort. In light of this consideration, we opted for letting the final customer to choose between the two platforms, according to his needs. Anyway, we deem Sentilo to be more suitable in the Smart Cities context thanks to the users roles system, whereas we are more likely to use openHAB for home automation. In fact, when used for making houses 'smart', all people that can access the management platform are likely to have the same read and write rights, so a simple authentication via login is sufficient and there is no need to make differentiations.

The token-based authentication of Sentilo implies two things:

- the need for implementing a mechanism to safely distribue the tokens to the users;
- as the token is long and made up of random characters, making its mnemonic learning difficult, it is preferable to hide the tokens to the users and let them only perform the login through username and password insertion, whereas another entity is in charge of automatically inserting the corresponding token in each HTTP request.

For these purposes, I developed a login server that handles the authentication to Sentilo and filters the requests in order to insert the correct identity token into the header. It also provides an administration console for managing the user-token association and the insertion, updating or removal of Sentilo users.

5.4 The login server

The login server has the main objective of authenticating the final user and filtering its request, so it logically stays between the final user backend and the Sentilo platform, as shown in Figure 5.1.

In the development of the Login Server we had to consider the fact that Cross-site HTTP requests initiated from within scripts are subjected to restrictions by browsers because of security reasons. Authorization cookies are

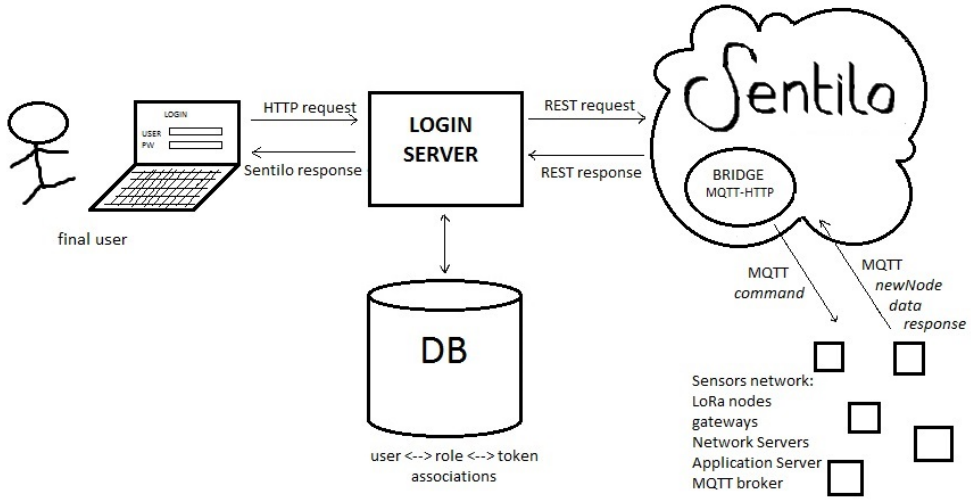


Figure 5.1: System architecture

in fact transparent among different windows and tabs of the same browser and this may lead to unwanted and unsafe behaviours. CORS, *Cross-Origin Resource Sharing* [55] is a mechanism that allows restricted resources on a web page to be requested from another domain outside the domain from which the resource originated. Actually, most browsers currently support CORS. It defines a way in which a browser and a server can interact to safely determine whether or not to allow the cross-origin request, bypassing the blocking mechanism that exists among different domains. The CORS mechanism works by adding some custom HTTP headers to the requests. When a cross-origin HTTP request to a certain server originates, it is not immediately sent by the browser that supports CORS. Instead, it sends a HEAD request[3] to the target server, including the headers defined in the CORS standard. If the server does not support the CORS technology, it is incapable of recognizing the additional headers and simply ignores them, responding as if the request were a normal request. On the other hand, if the server identifies those header fields, it includes some additional CORS headers in the response, too. In this way the browser understands that CORS is supported and that it can send the original request even if it is a cross-site

[3] A HEAD request asks for the same response that would correspond to a GET request, but without the response body. Its purpose is to retrieve meta-information contained in the response headers, without having to transport the entire content.

request. Anyway, the CORS standard only allows some requests not to be blocked and they normally are 'simple' ones, such as GET requests or POST requests with a content type among a restricted group. Requests containing custom headers or dedicated to authentication are not supported.

For the above explained reasons, we decided to deploy the login server on Tomcat, i.e., the same domain of the website used by the final user (Sentilo is in fact deployed on the Tomcat web container). It is actually implemented by exploiting the concept of Java servlet. Java servlets are small Java programs that run within a Web server and receive and respond to requests from Web clients, usually across HTTP. They can be thought of as applets that run on the server side. It is possible to develop more than one servlet within the same project, making each servlet responsible for handling requests to a specific set of URIs. To deploy and run a servlet, it is essential to use a web container, i.e., the component of a web server that interacts with the servlets, such as Tomcat. Web containers simplify the implementation of HTTP servers, as they are responsible for managing the lifecycle of servlets, mapping a URI to a particular servlet and ensuring that the URI requester has the correct access rights.

The Login Server project uses three different servlets for handling the following tasks:

- the *user servlet* is in charge of handling all the operations performed by a user, hence the login phase and the filtering of all the HTTP requests in order to redirect them to Sentilo. The same holds for the HTTP responses coming from Sentilo. If the user has not logged in, the servlet repels it and blocks its requests. This service has been thought to be a middle step between the user interface and the Sentilo platform and supports no GUI, but it is rather used through AJAX requests;

- the *login servlet* is responsible for the login of the administrators, done by inserting the username and the password;

- finally, when an administrator correctly authenticates himself, he can make use of the administration console, handled by the *admin servlet*. As explained further, this console esssentially allows the admin to insert, update or remove normal users and to insert or remove administrators.

Each servlet is assigned a set of URIs. If no authentication has been made, the forwarding to Sentilo is not allowed; similarly, only the administrator can access the user management console, while all other users are blocked. I exploited the concept of HTTP session in the implementation of the services

provided by the servlets. An HTTP session is a sequence of network request-response transactions set up at a certain point in time and then torn down when one of the parts involved wants to close it or when a certain amount of inoperative time has passed. HTTP/1.1 supports persistent connection, whilst the former versions required to open a new connection for each request, so session parameters were useless (normally there is one session on each connection). Anyway, HTTP/1.1 is the default HTTP version nowadays.

5.4.1 The user authentication

The user authentication contemplates the insertion of a unique username and a password. The login server relies on Mongo (the same non-SQL database used by Sentilo) for storing information about the users. In fact one of the Mongo *collections* of the database dedicated to Sentilo stores the list of *applications* along with their corresponding tokens. Tokens are unfortunately saved as they are[4], without encoding or hashing, but at least the database is accessible only through authentication. I created another database in Mongo dedicated to the login server, which stores the usernames together with their hashed passwords and roles, i.e., the Sentilo *applications*, they have. In this way, when a user tries to log in, the *user servlet* looks up the Mongo table: in case of missing username or of incorrect password, the users' requests are blocked, otherwise after the log in the user is allowed to communicate with Sentilo and hence with the 'Things'; moreover, session parameters are exploited to identify the user and to store its authentication token. In this way the user can dialog with Sentilo during the whole HTTP session without needing to authenticate himself at every request. The *user servlet* is then responsible for handling the HTTP requests and basically acts as a forwarder: it does not modify the requests received, but it just inserts the *IDENTITY_KEY* field retrieved from the session parameters in the HTTP header. Similarly, any response received by Sentilo is forwarded to the final user, without alterations. There exists only one case in which the filtering action of the *user servlet* does not reduce to a mere forwarding: when the user tries to publish sensors observations. Sentilo is in fact unaware of who is actually performing a request, e.g., a physical node able to produce data or a human being that just exploits the REST API. The only thing that identifies a petitioner is the authentication token, which carries all the information about the permissions owned by who is performing the request. Therefore, it is important to check that only real sensors send data to Sentilo

[4] This is how Sentilo developers conceived the storing mechanism. We thought about hashing tokens before their insertion in the database, but it would have represented an alteration of the existing code, and we wanted to avoid it.

in order to avoid a misfunctioning of the whole architecture. The *user servlet* intercepts the HTTP requests aimed at publishing observations[5] and blocks them, informing the final user that such service is unavailable for him. At every request, the presence of the correct session parameters is checked.

The *user servlet* has been thought to be interrogated through AJAX requests. AJAX, *Asynchronous JavaScript and XML*, is a technique for creating fast and dynamic web pages [57]. In fact it allows web pages to be updated asynchronously[6] by exchanging small amounts of data with a server behind the scenes, thus avoiding the reload of the whole page. The website developed by Patavina provides a graphical user interface that intuitively makes the final user interact with Sentilo (or openHAB). In the backstage, the connection with Sentilo is handled through AJAX requests made to the *user servlet*.

5.4.2 The administration console

The Login Server also provides a GUI for administration operations, which can be carried out by any user who has been granted administration privileges. An admin needs to authenticate himself through a login form, using a mechanism analogous to the users authentication. Credentials are in fact stored in Mongo (in a collection entirely dedicated to the administrators) and passwords are hashed with the same algorithm used for normal users' passwords. If the credentials inserted in the login page are incorrect, the user is redirected to the failed login page[7] and he is given the possibility to try the login again. When the authentication is done, the admin can access the administration console, which displays the information about the users and their roles (although passwords are not shown) and it allows him to delete a user, to modify its password or role or both of them. All the operations directly modify the data stored in the Mongo database, so the new information is immediately displayed. Any admin can also add new administrators or delete existing ones and edit his own password.

For what concerns Sentilo roles assigned to administrators, we opted for always assigning them the all-powerful role. In fact any admin may just add

[5] Hence, all PUT requests to an URI containing the label *data*.

[6] An asynchronous event is an event that runs outside the application's main thread. Asynchronous actions are actions executed in a non-blocking scheme, allowing the main program flow to continue processing. In synchronous operations, the main program flow is interrupted till the completion of the operation.

[7] A message warns the user that something is wrong with his credentials. Anyway, no hint about what is wrong -the username or the password- is given, as no clue should be given to potential attackers. In fact, if an attacker knows for sure that a username is valid, he may perform a brute force attack to retrieve the associated password.

a new user with such role and then use its credentials for communicating with Sentilo. So it was simpler to directly grant them all the privileges.

5.4.3 Password storing

As previously said, users' and administrators' passwords are stored in Mongo. Although the access to the databases requires authorization, it is always a good habit not to store passwords as they are, as a malicious attacker may find a way to access the database. A very simple system would just store the passwords themselves, and validation would be a simple comparison. But if a hostile outsider were to gain a simple glimpse at the contents of the database table which contains the passwords, then that attacker would learn a lot [58]. Unfortunately, such partial, read-only breaches do occur in practice: a mislaid backup tape, a decommissioned but not wiped-out hard disk, an aftermath of a SQL injection attack. This is why passwords should never be stored in plain text. However, remember that Sentilo stores the identity tokens in clear, so anybody who accesses the database can read and use them, invalidating the security precautions I implemented: a system is as safe as its least secure part. Hopefully, Sentilo developers may change the mechanism to store sensitive data.

How to securely store passwords is a hot theme in the literature, there are plenty of articles and blogs comparing the different existing schemes. Passwords may be encrypted or hashed. Encryption is a process to render the password unreadable and that uses an extra piece of secret data, the key; it is reversible by using the same (symmetric encryption) or a distinct but somehow mathematically related (asymmetric encryption) key. In this way the password is kept secret, but it is retrievable if the decryption key is known. On the contrary, the hashing mechanism uses no key and is not reversible. Well-known hashing algorithms are MD-5 and the family of the SHA algorithms [56]. A hash function is deterministic, so you can always rehash a putative password and see if the result is equal to a given hash value. Thus, a hashed password is sufficient to verify whether a given password is correct or not. This is why the most used strategies for attacking hashed passwords are typically guessing techniques [59]:

- *brute force* tecnique consists of systematically enumerating all possible candidates for the solution and checking whether each candidate satisfies the problem's statement. Being an exaustive search, it is always succesful. The key point in the protection against brute force attacks is making the number of candidates prohibitively large, leading to exceedingly large computational times;

- *dictionary attacks* is another guessing attack which uses a precompiled list of options. Rather then trying every option as in bruteforce, it restricts the candidates to a dictionary of the options which are deemed as more likely to work;

- *rainbow tables* are precomputed tables that allow to perform a simple look up for each hash. In the web there are many rainbow tables already built for the most common hashing algorithms and for some hashing lengths.

The longer the password, the hardest for these strategies to be effective. Anyway, hashing prevents an attacker from retrieving the plain text password in the event he gets read access to the database. Thus, hashing passwords will not make a site any more secure, but it will perform damage containment in the event of a breach.

An expedient to enforce the hashing function is to use a salt [60], i.e., a non-secret value which is typically appended to the password before getting hashed. It is used to prevent rainbow table lookups and other parallel attacks. Among the advantages of the attacker over the defender is parallelism: the attacker usually grabs a whole list of hashed passwords, and is interested in breaking as many of them as possible, so he may try to attack several in parallel. Salting is about using not one hash function, but a lot of distinct hash functions: ideally, each instance of password hashing should use its own hash function. Hence, a salt is like a way to select a specific hash function among a big family of hash functions. Properly applied salts will completely thwart parallel attacks. It is extremely important for the salt associated to a password to be unique and unrelated to any other information given by the user, to prevent two databases from returning the same hash for a given password. Uniqueness makes it more difficult to perform bruteforce and dictionary attacks: when attacking a whole database, a different salt must be taken into account for each password, making the computational times longer.

Another requirement that good hash functions should fulfill is slowness, to counter the always increasing computational speed of computers. Hashing should be inherently slow and this is obtainable by defining the hash function to use a lot of internal iterations. The process should be slow enough to make the hashing of many passwords very time-consuming but still fast enough not to affect the hashing of a single password, at least for what concerns the human perception of time: password hashing function must not be intolerably slow for the honest system.

Cryptographic hash functions[8] which are deemed to be good are PBKDF2 [61], bcrypt [62] and scrypt [63]. PBKDF2 is a key derivation function that applies a pseudo-random function, such as a cryptographic hash, to the input password along with a salt, and then derives a key by repeating the process as many times as specified. Although it is a key derivation function, it uses the principle of key strengthening at its core. Bcrypt is a password hashing function which aims at being slow. It is derived from the Blowfish block cipher which uses look up tables to generate the hash. This means that a certain amount of memory space needs to be used before a hash can be generated. This can be done on CPU, but when using the power of GPU it will become a lot more cumbersome due to memory restrictions. Scrypt instead moves the problem away to another level and instead of doing a lot of hash function invocations, it concentrates on an operation which is hard for anything else than a PC, e.g., random memory accesses. Scrypt has been designed to be far more secure against hardware brute-force attacks than PBKDF2 or bcrypt. Anyway, it is still relatively new (it has been designed in 2009) and I prefered to go for the more vetted and tested bcrypt.

In conclusion, I chose the bcrypt hashing mechanism to safely store passwords. More precisely, I made use of the already built-in Java library, *Bcrypt*, which provides methods for hashing a string and also for generating the salt. Before storing the password in the database I hash it, and for checking whether the password introduced in the login form is correct, I compare the stored hashed password and the hashing of the inserted password.

[8] A cryptographic hash function is a one-way hash function, practically impossible to invert.

6. Performance analysis

My work is just a part of a bigger project, so it was not straightforward to define a way to measure its goodness. These kinds of works may be evaluated from a functionality point of view and for their performance intended as efficiency.

The choice of the platform is kind of unrelated to the other two tasks and there is no easy or objective way to tell whether the final decision was the best choice or not, but we can agree that Sentilo does what we expected it to do. For what concerns the bridge implementation and the login server, their functioning can be tested only empirically. We tested them many times, detecting errors or misfunctionings, and their actual versions are considered to be stable, although when dealing with software nothing can be granted and unexpected behaviours can always appear. Anyway, they both comply with their tasks, so I can affirm that the functionality evaluation is positive. But I also wanted to somehow measure the efficiency of what I implemented. In the next sections I will illustrate the experiments I carried out for determining the goodness of the bridge and of the login server.

6.1 The bridge performance

Finding a way to test the efficiency of the protocol bridge has been one of the most challenging tasks of my thesis. Performance measures of such systems typically concern scalability, reliability, robustness and energy efficiency, but all these metrics usually ask for a deployment of the system on a large scale and for a long observation period; none of these requirements was reachable as the project is still under development and I did not have enough time. Therefore, all I could do was a performance evaluation of the REST interface over MQTT in order to estimate the 'cost' of the bridge. It is not easy to find similar evaluations in the literature: the topic is not common and IoT systems are relatively recent, so we had to conceive our own experiment. We decided to measure the delay (for both transmission and processing) and the traffic intensity between the HTTP client of the bridge and Sentilo server in

comparison with the same metrics from the observations generation until the MQTT messages are received at the MQTT client of the bridge.

6.1.1 Simulation scenario

The whole system network is represented in Figure 2.1. I had to define a use case, i.e. a realistic network setup for evaluating the performance. For the sake of the simulation, I made use of virtual devices because the real elements were used by Patavina Technologies meanwhile. Anyway, they provided me software tools to mimic the behaviour of LoRa nodes and gateways, of the Network Server, Application Server and Authentication Server. Although the virtual nodes, GWs and NS emulate the behaviour of their real counterparts at upper levels, the LoRaWAN standard defines its own specification for the MAC layer; thus, the evaluation of metrics such as delay and traffic intensity cannot be realistic if done with the virtual nodes. For this reason I decided to put all the 'LoRa' virtual elements on the same machine as the traffing among them does not correspond to the real one and performance would have not been realistic. The Authentication and the Application Servers are typically installed in the same hardware, and the MQTT broker may likely stay alone, whereas the bridge has been though to be an adjucnt to Sentilo, naturally belonging to the same hardware.

I did not have the possibility to use many physical machines for running each element of the network setup, so I decided to create virtual machines (VMs) with the tool *Oracle VM VirtualBox* [64]. With virtual machines it is possible to emulate the behaviour of a specific computer system by executing a complete operating system, and this is very useful when real hardware is not available. The chosen scenario then requires four machines: one for the virtual LoRa elements, one for the MQTT broker, another for the AS, and a last one for Sentilo and the bridge. Moreover, there exists a tool provided by the Linux Foundation called *Netem* [65], which provides network emulation functionalities for testing protocols by emulating the properties of wide area networks. The current version emulates variable delay, loss, duplication and re-ordering, and allows to specify a transmission rate for a connection, too. I exploited such tool for mimicking real connections among virtual machines setting delay and transmission rate. A very likely configuration sees the AS (together with the AUS), the broker and Sentilo on a cloud and the NS close to the GW and connected to the cloud by means of an ADSL or a GSM connection, thus making the link between the NS and the broker the bottleneck of the whole connection. The reason of not having the NS on the cloud is the high latency that the LoRa network would experience: after some experiments Patavina Technologies established that having the

NS close (within 1 km) to the GWs outperforms the case of having the NS in the cloud. Reasonable values for the average transmission rates may be 1 Gbps for the links in the cloud and 10 Mbps for the connection between broker and NS, whereas the average delays experienced by the packets could be 10 ms in the former case and 100 ms in the latter. Notice that these values may be highly influenced by the network setup: if the NS is connected to the broker by means of a cellular connection such as GSM, the latency is much higher than when an ADSL connection is used, especially in case of fiber-optic.

Figure 6.1 shows the network setup, highlighting the division in different physical machines.

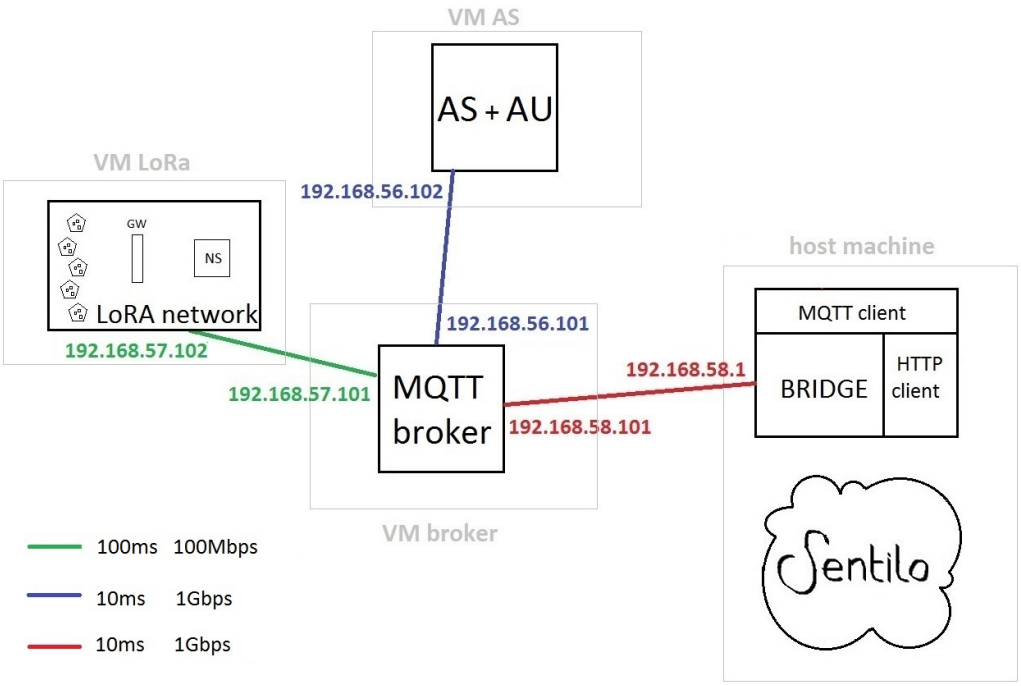


Figure 6.1: Simulation setup

For obtaining measures about the traffic, I made use of *tcpdump*, a packet analyzer that runs under the command line and allows the user to display TCP/IP and other packets being transmitted or received over a network to which the computer is attached. It allows to extract statistics over the captured packets; however, after running a complete simulation I realized the impossibilty of extracting the delays measures from the capture files of *tcpdump*. The problem arises from the fact that the packets arriving at the MQTT client of Sentilo bridge need to be matched to the corresponding

packets originated from the nodes in order to compute the time it took each packet to be transmitted and processed by the various elements. My original idea consisted in hashing the packet headers by using the CRC-32 algorithm. This technique has been succesfully used in [66] and allows to compare arriving and departing packets in an easy and fast way. The authors of [66] measure the single-hop delay on an IP backbone network, so that the IP header of a packet remains unmodified till it arrives at the destination host, with the exception of some fields such as the Time To Live (TTL) which may be modified by routers. However, the messages coming from the nodes and those published to the MQTT client of Sentilo are completely different (although they provide the same information) as the AS processes the messages published by the NS and rearrange them in a human-readable fashion. We may identify two different tranmission moments: in the former the nodes send their data to the NS, which publishes the corresponding MQTT messages to the broker, which in turn forwards them to the AS; the latter involves the messages elaborated by the AS which are published to the bridge. Two main problems arise: first of all, even if it were possible to measure the two delays from the NS to the AS and then from the AS to the bridge, the time needed by the AS for the elaboration process would be omitted; secondly, MQTT publish messages to and from the broker appear to be the same at the application level, but look completely different at the network and transport levels: the IP and TCP headers are totally uncorrelated and this makes it hard to find the correct matching. Moreover, long messages are likely to be rearranged in separated packets by the broker, so it is very difficult to make packets correspond. For all these reasons, I had to find another way to compute delays.
The delay experienced by a packet from its departure from the NS till its arrival at Sentilo can be written as:

$$\begin{aligned} delay \;=\; & t_{TX,NS\to brk} + t_{brk} + t_{TX,brk\to AS} + t_{AS} \\ +\; & t_{TX,AS\to brk} + t_{brk} + t_{TX,brk\to bridge} + t_{bridge} \\ & + t_{TX,bridge\to Sentilo} + t_{bridge} + t_{TX,Sentilo\to bridge} \end{aligned} \tag{6.1}$$

The processing times of the AS and of the bridge, namely t_{AS} and t_{bridge}, can be computed by tracking each packet directly in the sofware code. Similarly, the average time needed by the broker for redirecting a message to the clients subscribed to the message topic can be retrieved from the *tcpdump* capture files. Finally, the propagation times are highly dependent on the network configuration itself, the setup of Figure 6.1 is just one of many possibilities. Notice that since the bridge and Sentilo are meant to run on the

same machine, the transmission times $t_{TX,bridge \rightarrow Sentilo}$ and $t_{TX,Sentilo \rightarrow bridge}$ are supposed to be negligible.

The time elapsed from the arrival of a packet in the broker and its tranmission to the subscribed clients, namely t_{brk}, highly depends on the QoS used. We always set the QoS level to 2, which means that the packet is ensured to be received exactly once[1], and it is handled by the MQTT specification with a three-way handshake message exchange: the party that received the publish (PUB) message sends a PUBREC to mean that the publish has been received, the other party sends then a PUBREL to say that the publish has been released and finally the first party sends a PUBCOMP message to tell the publish has been completed. It takes a negligible time for the broker to look up the list of subscribed clients to which sending the PUB message received, but actually it needs to wait for the reception of the PUBREL message. Hence, t_{brok} is expressable in a valid way as the propagation time of two short messages (they only have a 4-bytes header from an application point of view, plus the headers of the lower layers).

Simulation technical parameters

I ran the simulations on a Dell computer with an Intel® Core™ i7-2600 Quad core unlocked with Hyper-Treading (which means that there are 2 threads per core). Since I needed 3 virtual machines plus the host machine, there was one core per machine. The OS of the host machine is Xubuntu, which is based on Linux Ubuntu-Trusty, while I installed Linux Mint on the VMs.

6.1.2 Results

I executed two independent simulations: the former considers non-secure connections, the latter examines the network behaviour when TLS is enabled. The final deployment of the system will very likely make use of TLS, however the analysis of the former scenario is for the sake of a comparison with the purpose of determining the additional load of TLS in traffic intensity and transmission delay. The other settings were the same for both simulations in order to guarantee a fair comparison. Hence, both experiments used the same network configuration with one single gateway and 8 nodes running, each of them publishing 17 observations every 10 seconds and with the observations of two nodes spaced by 1 s. Based on the fact that the predominant traffic is in uplink - i.e., with many more messages generated by the nodes rather

[1] QoS 1 corresponds to receving a packet at least once, whereas QoS 0 gives no guarantee about the reception of the packet, which is known to be received at most once.

than commands sent from the users to the sensors - I decided to run the simulations considering only uplink traffic.

The analysis of the *tcpdump* logs let me identify the type of messages exchanged between the parties in each connection, which correspond to the expected flows scheduled by the MQTT and HTTP protocols. The following descriptions refer to a steady-state flow and thus neglect the extra messages exchanged at the very begininning of the network setup: the MQTT clients need to connect to the broker and make the proper subscriptions, whilst the HTTP client of the bridge must send some *catalog* and *subscribe* requests to the Sentilo server. Anyway, these messages are supposed to happen only once, and therefore should be excluded by the 'normal' flows.

1. Network Server ↔ broker

   The message exchange between the NS and the broker regards the MQTT messages containing the sensors' observations that the NS publishes to the broker. The pattern of the transaction is the following. The NS publishes a message coming from a sensor with QoS set to 2, so fundamentally, for each message generated by a LoRa node, 4 messages are exchanged between the NS and the broker: one for the data delivery and 3 more for confirming correct data reception. It is also possibile to pinpoint a TCP flow with ACK messages in the capture logs. Notice that streams of MQTT keep alive messages (i.e. PING and PONG messages) are missing since the NS sends PUB messages at a fast pace.

2. Application Server ↔ broker

   The exchanges between the AS and the MQTT broker are more complex than those of the previous case as the AS is both subscribed to the messages coming from the NS and publishes its own messages. Looking at the capture files we can identify the following pattern. The MQTT broker sends a PUB message to the AS which is followed by the PUBREC, PUBREL and PUBCOMP as described previously. Meanwhile, the AS processes the information received from the NS and sends some PUB messages containing the elaborated data, each of them followed by the acknowledging messages envisaged by QoS equal to 2. Notice that in MQTT it is possible to send multiple messages in the same packet, reducing the number of headers -and therefore bytes- sent.

3. broker ↔ bridge

   The data flow is toward the bridge, which is subscribed to the messages coming from the AS. Similarly in the connection between broker and

NS, it is possible to identify the MQTT messages scheduled by the QoS 2 and an underlying TCP stream.

4. BRIDGE ↔ SENTILO

   The traffic flowing between the bridge and Sentilo is of a completely different nature. Then a HTTP request is sent for each *data* message received and the corresponding response is sent back from Sentilo, after the server has correctly processed it. Every node data message includes observations of multiple sensors, for a total numer of 15 with the nodes used. Thus, for each node message, the bridge performs at least 15 different HTTP requests to Sentilo -there might be retranmissions or unexpected responses from Sentilo causing the sending of multiple requests.

The simulations ran for about 15 hours, which correspond to more than $2.5 \cdot 10^6$ data observations from the peripheral nodes. Table 6.1 shows the average number of Bytes exchanged at every connection for each data observation sent by a node.

	NS → broker	*AS → broker*	*broker → bridge*	*bridge ↔ Sentilo*
NO TLS	0.6kByte	3.9kByte	3.4kByte	7.7kByte
TLS	0.8kByte	6.7kByte	6.2kByte	7.7kByte

Table 6.1: Bytes exchanged at each link for one node observation

Notice that the MQTT traffic between the AS and the broker or between the broker and the bridge is considerably higher than the traffic between the NS and the broker. This is due to the fact the the NS sends a single packet for every node message, publishes it to a single topic and moreover the data payload is extremely compressed as the sensors information is sent as a hexadecimal string. The AS interpretes such message and breaks it up in many messages published at different topics, as outlined in Section 4.2.1. Whereas the MQTT messages can be sent in the same packet, mimicking the piggybacking technique[2], the bridge must send separate HTTP requests to Sentilo, as the URI typically change for each data message (they include the ID of the provider associated in Sentilo to that specific sensor). The

[2] Bi-directional data transmission technique in the network layer in which the acknowledgement appended to a data frame in the reverse direction rather than being sent in an individual frame.

use of TLS increases the size of the exchanged messages, as it requires and additional header and the data encryption also affects the packet size. Although the overhead for a single packet may not be significant, it becomes a considerable weight when thousands of messages are transmitted; however, this is the price to pay for having a secure tranmsission.

For what concerns the delay metric, Equation 6.1 gives an expression of the delay experienced by a packet from the Network Server till the reception of Sentilo response to its corresponding HTTP message. The delay indicated as t_{brk} is the time elapsed since the reception of a data message from the broker until the departure of that message[3] and highly depends on the QoS level: if it is set to 2, the broker has to wait till it receives the PUBREL message from the other party before sending the message to the subscribed client. Thus, t_{brk} actually is the propagation time of two short messages - PUBREC and PUBREL - plus the time the broker needs to identify the clients to which to redirect the message; the latter is almost negligible, so t_{brk} can be calculated as twice the time needed for sending packets whose length is given by 4 bytes of the MQTT application layer plus the headers of the lower layers.
In order to evaluate the cost of the bridge in terms of delay, expression 6.1 must be compared with the time elapsed since the NS publishes a messages till the moment in which the MQTT client of the bridge receives the corresponding message. Table 6.2 shows the average values of each factor of Equation 6.1, considering about $2.5 \cdot 10^6$ elements. Remember that the transmission times are those of a specific network configuration and notice that the processing times are highly influenced by the features of the hardware machine, so they may be lower if more powerful hardware were used, which will likely happen in a real deployment of the system.

DELAY [ms]	$t_{tx,all}$	$t_{broker,NS}$	t_{AS}	$t_{broker,AS}$	t_{bridge}	$t_{Sentilo}$
NO TLS	130	200	4	20	8.5	120
TLS	130	200	3.7	20	8	120

Table 6.2: Delays

It is clear that the use of TLS does not increase the average propagation and processing delays in a significant way. As expected, the poor connection between the NS and the broker affects the overall delay, as it influences not

[3] From an application level point of view the message is the same, altough the headers of the lower layers are not correlated at all.

only $t_{TX,NS \to brk}$ but also $t_{brk,NS}$ for the reasons already explained. Another huge contribute is given by the processing time of Sentilo, which performs a lot of operations upon receiving a message.

Summing up, the conversion from MQTT to HTTP and the successive processing of the HTTP requests highly influence the communication between the user and the 'things'. In the used network scenario, the 'Internet connection' represents about 24.9% of the total traffic intensity in both cases, and the time needed for properly converting an MQTT message and then handling it constitutes 26.6% of the total delay. It is evident that the cost of the REST interface over MQTT is quite high.

6.2 Security improvements

Security is a big concern in the design of a system, mainly because there is no way to tell how secure the system is. The developer should always think as an attacker if he wants to find the leaks and weak points of the architecture in order to take measure against them. The login server I implemented allows only authenticated users to dialog with Sentilo; after the user logs in, the server keeps track of the session and identifies the user by exploiting the session parameters. Anyway, malicious users could by-pass this measure by exploiting a valid computer session: this attack strategy is called *session hijacking* and it is made possible by the use of cookies and session parameters, which on the other hand are necessary for avoiding that the final user has to log in at every request to the server [67]. There exist some techniques for preventing session hijacking, such as the use of a long random number or string as the session key, the regeneration of the session ID after a successful login or at each request, or the encryption of the data traffic between client and server by using TLS, which is the method we opted for.

Anyway, what the login server wants to keep secret is the authentication token associated to the role the user has been assigned. The orginal Sentilo implementation builds tokens by hashing some non random values: a prefix retrievable in the source code, the easily known name of the entity for which the token is being generated, and the creation time of this entity, with a millisecond accuracy. The hashing process produces a hexadecimal string made of 26 characters but still guessable as ground for a dictionary attack based on the knowledge of the creation date. I myself tried to perform a brute-force attack to retrieve the token associated to a particular Sentilo role, an attack that ended with success. The code I wrote for retrieving the token associated to the particular Sentilo entity, considers a given timestamp in

the Unix format[4], calculates the token associated to that role for that timestamp, makes a request to Sentilo inserting the token in the HTTP header and, in case of a response with HTTP code *403 Unauthorized*, increments by 1 the timestamp and the process is repeated until the correct token is found or a given number of tries has been made. In this way, on the same Intel® Core™ i7-2600 processor previously described, the mean computing time for an attempt is 4.003 ms. This number has been computed as the average time out of 10^6 consecutives attempts and is associated to a standard deviation of 8.762 ms. Thus, approssimating such mean time to 4 ms per attempt and considering that every time second corresponds to 1000 possible timestamps, if the attacker knows the creation time with an accuracy of one unit, it will take him 4 units to guess it in the worst case. So, if he knows the day in which the entity has been created, he just needs four days to derive the corresponding token. And this is an over-estimation of the time needed, as the code I used for performing the brute-force attack might be improved for speeding up the process and parallel computing could be exploited, too. Moreover, it is well-known that running code on a GPU (Graphics Processor Unit) rather than on the CPU (Central Processing Unit) considerably accelerates the process [68].

The original token generation is clearly unsafe and represents a big leak in the security of Sentilo, so we decided to change it for a more secure function. The essence of the algorithm does not change, we just traded the creation timestamp for a completely random hexadecimal number. If a brute-force attack is perpetrated by trying all possible strings of 26 hexadecimal characters, the average time for determining the correct token increases considerably. There are $16^{26} \simeq 2.03 \cdot 10^{31}$ possible combinations but the number of tries before a success cannot be represented as a geometric random variable as the attempts, despite being independent of each other, are not identically distributed: after one (wrong attempt), $16^{26} - 1$ combinations remain and in general $16^{26} - k$ are still to be considered after k attempts. Let $\mathcal{X}$ be the random variable that models the number of failures before getting the token and let p_j be the probability of success at the j^{th} attempt. It holds:

$$\begin{cases} p_j = 0 & for \;\; j < 0 \\ p_j = \dfrac{1}{1/p - j} & for \;\; 0 \leq j < 16^{26} \\ p_j = 1 & for \;\; j \geq 16^{26} \end{cases}$$

where $p = 16^{-26}$ is the probability of a single combination of 26 hexadec-

[4] Reminder: the Unix timestamps are represented as the number of seconds passed since January 1^{st}, 1970.

imal characters to be the correct one. The probability of making exactly k attempts is $P(X = k) = \prod_{j=0}^{k-1}(1 - p_j) \cdot p_k$, and the cumulative distribution function (c.d.f.) of $\mathcal{X}$ is:

$$P(X \leq k) = 1 - P(X \geq k + 1) = 1 - \prod_{j=0}^{k}(1 - p_j) \tag{6.2}$$

It is quite complex and difficult to manage, so I computed lower and upper bounds to it. Considering that $p_{k+1} \leq p_k \ \ \forall k \geq 0$, it holds:

$$1 - \prod_{j=0}^{k}(1 - p_0) \ \leq \ 1 - \prod_{j=0}^{k}(1 - p_j) \ \leq \ 1 - \prod_{j=0}^{k}(1 - p_k) \tag{6.3}$$

$$\prod_{j=0}^{k}(1 - p_k) \ \leq \ \prod_{j=0}^{k}(1 - p_j) \ \leq \ \prod_{j=0}^{k}(1 - p_0) \tag{6.4}$$

$$(1 - p_k)^{k+1} \ \leq \ \prod_{j=0}^{k}(1 - p_j) \ \leq \ (1 - p_0)^{k+1} \tag{6.5}$$

Bernoulli inequality states that $(1 + x)^r \geq 1 + r\,x \ \ \forall$ integer $r \geq 0$ and real number $x \geq 1$ [69]. Therefore,

$$(1 - p_k)^{k+1} \ \geq \ 1 - p_k\,(k + 1) \tag{6.6}$$

Another useful bound states that $(1 + r/k)^k \leq e^r$[69], from which

$$(1 - p_0)^{k+1} \ = \ (1 + \frac{-p_0\,(k + 1)}{k + 1})^{k+1} \ \leq \ e^{-p_0\,(k+1)} \tag{6.7}$$

So, in the end:

$$1 - p_k\,(k + 1) \quad \leq \quad \prod_{j=0}^{k}(1 - p_j) \quad \leq \quad e^{-p_0\,(k+1)} \tag{6.8}$$

$$1 - e^{-p_0\,(k+1)} \quad \leq \quad P(X \leq k) \quad \leq \quad p_k\,(k + 1) \tag{6.9}$$

Figure 6.2 shows the upper and lower bounds to the c.d.f of $\mathcal{X}$, i.e. to the probability of needing at least k attempts before succesfully finding the authorization token, for $10^{29} \leq k \leq 2 \cdot 10^{31} \simeq \#$ possible combinations. For $k = 2 \cdot 10^{30}$ the probability is still low, about 0.1. With an average computing time of 4 ms per attempt, about 10^{20} years would be needed for trying $k = 2 \cdot 10^{30}$ combinations. It is evident that using a random token certainly improves security against brute force attacks.

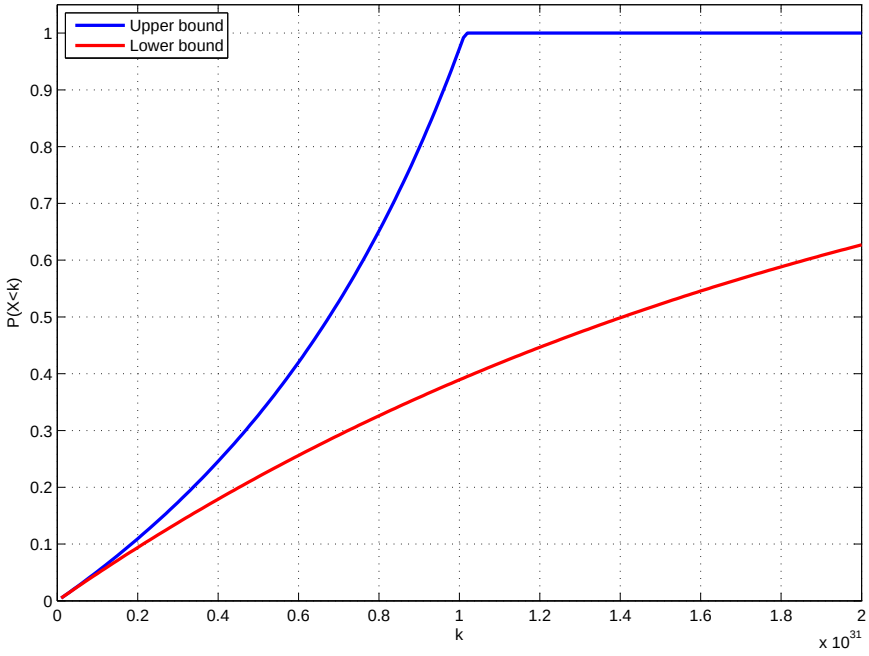


Figure 6.2: Bounds to the cumulative function of $\mathcal{X}$

Actually, when using a random string, no hashing is needed. So probably Sentilo creators decided for the described solution in order to be able to regenerate the identity token a second time. We deem they make use of an application firewall in the Sentilo deployment in Barcelona for making it hard to perpetrate brute force attacks: application firewalls are filters that apply a set of rules to an HTTP conversation for covering security attacks, such as cross-site scripting (XSS) and SQL injection. Anyway, we think it is safer to use random strings as we do not see any advantage in being capable of generating the same token again.

Of course, developing strategies against brute-force reduces the risk of succesful attacks, but this is not enough: it is extremely important to provide protection to Mongo, too. Access control has already been enabled, nonetheless somebody may try to intrude. This is why we deem safer to hash user passwords before inserting them in the database, and this should be done with the identity tokens, too.

7. Conclusions

The Internet of Things is a new paradigm thus no consolidated solution has been established yet. Certainly many issues need to be addressed and albeit the scientific world is already investigating the validity of different approaches, an integrated solution that embraces all the challenges described in Chapter 1 is still missing. The architecture developed at Patavina Technologies is complete and even if it has not enough scalability for covering wider scenarios, it fits very well Smart Cities schemes. It is quite complex and is based on many elements - LoRa, MQTT, REST, ... - making them fitting together. Security is granted at all levels and parts of the connection, and user access differentiation is also provided. Thanks to LoRa the energy consumption of the sensors is minimized when not fast reacting nodes are needed, and Class B and class C nodes respond faster to stirrings coming from the NS in spite of a higher energy consumption.

My thesis work was just a part of the whole architecture. It is threefold:

- I firstly analyzed and compared three different platforms with the objective of detecting the one better fitting Patavina Technologies requirements, finally choosing Sentilo;
- I adapted Sentilo in order to communicate with the existing architecture by developing a protocol bridge;
- I implemented a system for the users authentication by exploiting the users privileges infrastructure provided in Sentilo.

Sentilo proved to satisfy all requirements of the system, and thanks to the bridge it is able to convert MQTT messages into HTTP messages, whose information is displayed to the human user through a friendly interface and according to the permissions he owns. Permissions allow to efficiently manage the secure access to the platform and to treat users accordingly.

Future work

The Patavina Technologies project represents a proprietary solution for IoT application that is the proof of the feasibility of IoT systems being simultaneously secure, robust and energy efficient. Anyway it targets specific nodes features and specific application requirements thus it is not a fundamental contribution in the more general research of a world-wide MTC vision outlined in Section 1.1. Future work may focus at making the bridge faster in computing and in enhancing the role management of the different users priviliges. This thesis project is intended to be the prologue of my work in the Internet of Things world: I would like to research about more general solutions capable of addressing the issues arising in the IoT.

Bibliography

[1] K. Ashton, "That 'Internet of things' thing. In the real world, things matter more than ideas." http://www.rfidjournal.com/articles/view?4986, Jun 2009.

[2] F. Xia, L. Yang, L. Wang, and A. Vinel, "Internet of Things," *International Journal of Communication System*, vol. 25, 2012.

[3] A. Zanella, N. Bui, A. Castellani, L. Vangelista, and M. Zorzi, "Internet of Things for Smart Cities," *IEEE Internet of Things Journal*, vol. 1, Feb 2014.

[4] M. Weiser, "The computer for the 21st century." http://www.ubiq.com/hypertext/weiser/SciAmDraft3.html, Feb 1991.

[5] E. Schmidt, " World Economic Forum in Davos, Switzerland." http://www.hollywoodreporter.com/news/google-chairman-eric-schmidt-internet-765989, Jan 2015.

[6] "Juniper Research, 'Internet of Things' connected devices to almost triple to over 38 billion units by 2020," Jul 2015.

[7] "IEEE Standard for Local and Metropolitan Area Networks-Part 15.4: Low-Rate Wireless Personal Area Networks (LR-WPANs)," *IEEE Standard 802.15.4*, 2011.

[8] "IEEE Standard for Local and Metropolitan Area Networks-Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications (Includes IEEE Std 802.11, 1999 Edition; IEEE Std 802.11A.-1999; IEEE Std 802.11B.-1999; IEEE Std 802.11B.-1999/Cor 1-2001; and IEEE Std 802.11D.-2001)," *IEEE Standard 802.11*, 2005.

[9] G. Mulligan, "The 6LoWPAN architecture," *Proceedings of the 4th workshop on Embedded networked sensors, pages 78-82*, 2007.

[10] Z. Shelby, K. Hartke, C. Bormann, and B. Frank, "Constrained application protocol (CoAP)," *IETF 2013*, 2013.

[11] `http://www.semtech.com/wireless-rf/lora.html`.

[12] Carnegie Mellon University, "The 'Only' Coke Machine on the Internet." `https://www.cs.cmu.edu/~coke/history_long.txt`.

[13] "The Internet of Things: Dr. John Barrett at TEDxCIT." `http://tedxtalks.ted.com/video/The-Internet-of-Things-Dr-John`, 2012.

[14] H. Kopetz, *Internet of Things*. in "Real-Time Systems", Springer New York, Feb 2011.

[15] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of Things (IoT): A vision, architectural elements, and future directions," *Future Generation Computer Systems*, vol. 29, 2013.

[16] A. Biral, M. Centenaro, A. Zanella, L. Vangelista, and M. Zorzi, "The challenges of M2M massive access in wireless cellular networks," *Digital Communications and Networks*, vol. 1, 2015.

[17] G. Madueno, C. Stefanovic, and P. Popovski, "How many smartmeters can be deployed in a GSM cell?," *2013 IEEE International Conference on Communications Workshops (ICC)*, Jun 2003.

[18] P. Jain, P. Hedman, and H. Zisimopoulos, "Machine type communications in 3GPP systems," *IEEE Communication Magazine*, vol. 50, Nov 2012.

[19] A. Lo, Y. Law, M. Jacobsson, and M. Kucharzak, "Enhanced LTE-Advanced Random-Access Mechanism for Massive Machine-to-Machine (M2M) Communications," *27th World Wireless Research Forum (WWRF) Meeting*, 2011.

[20] A. Laya, L. Alonso, and J. Alonso-Zarate, "Is the Random Access Channel of LTE and LTEA Suitable for M2M Communications? A Survey of Alternative," *IEEE Communications Surveys & Tutorials*, vol. 16, 2014.

[21] J. Kim, J. Lee, J. Kim, and J. Yun, "M2M Service Platforms: Survey, Issues, and Enabling Technologies," *IEEE Communications Surveys & Tutorials*, vol. 16, 2014.

[22] P. Kinney, "Zigbee technology: Wireless control thatsimply work," *Communications Design Conference* , Oct 2003.

[23] "3GPP Specification, Feasibility Study for Proximity Services (ProSe)." `http://www.3gpp.org/DynaReport/22803.htm`, 2003.

[24] S. Tozlu, "Feasibility of wi-fi enabled sensors for internet of things," *Wireless Communications and Mobile Computing Conference (IWCMC), 2011 7th International*, 2011.

[25] "IEEE Computer Society, Advanced Message Queuing Protocol," *IEEE Internet Computing*, vol. 10, 2006.

[26] T. Nam and T. Pardo, "Conceptualizing smart city with dimensions of technology, people, and institutions," *Conceptualizing smart city with dimensions of technology, people, and institutions*, 2011.

[27] NavigantResearch, "Smart Cities." `http://www.navigantresearch.com/research/smart-cities`.

[28] V. Guitérrez, J. Galache, J. Santana, P. Sotres, L. Sánchez, and L. Muñoz, "The Smart City Innovation Ecosystem: A Practical Approach," *Multimedia Communications Technical Committee IEEE COmmunications Society*, vol. 9, Sep 2014.

[29] `http://www.padovasoftcity.it/`.

[30] A. Cenedese, A. Zanella, A. Vangelista, and M. Zorzi, "Padova Smart City: an Urban Internet of Things Experimentation," *2014 IEEE 15th International Symposium on A World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, 2014.

[31] `http://www.smartsantander.eu/`.

[32] R. Want, "An introduction to RFID technology," *IEEE Pervasive Computing*, vol. 5, 2006.

[33] LoRaAlliance, "LoRa Specification," 2015.

[34] J. Daemen and V. Rijmen, *The Design of Rijndael: AES - The Advanced Encryption Standard*. Springer-Verlag, 2002.

[35] MicrosoftTechnet, "Virtual Private Networking: An Overview," Sep 2001.

[36] RFC5246, "The Transport Layer Security (TLS) Protocol Version 1.2." `https://tools.ietf.org/html/rfc5246`, Aug 2008.

[37] G. Fersi, "Middleware for Internet of Things: a study," *International Journal of Communication System*, 2015.

[38] S. Bandyopadhyay, M. Sengputa, S. Maiti, and S. Dutta, "Role of Middleware for Internet of Things," *International Journal of Computer Science & Engineering Sutvey*, vol. 2, no. 3, 2011.

[39] S. Bandyopadhyay, M. Sengputa, S. Maiti, and S. Dutta, "Middleware for the Internet of Things, design goals and challenges," *IEEE Journal on Selected Areas in Communications*, vol. 21, Aug 2003.

[40] `http://www.openhab.org/`.

[41] `http://www.sentilo.io`.

[42] `https://parse.com`.

[43] M. Collina, G. Schiele, A. Vanelli Coralli, and G. Corazza, "The ponte project: Platform architecture, primitives, and data formats for interoperability in the internet of things," *IEEE Internet of Things Journal*, 2014.

[44] "OASIS standard: MQTT Version 3.1.1." `http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html`, Apr 2014.

[45] R. Fielding and R. Taylor, "Principled Design of the Modern Web Architecture," *ACM Transactions on Internet Technology (TOIT)*, vol. 2, May 2002.

[46] R. Roman, P. Najera, and J. Lopez, "Securing the Internet of Things," *IEEE Network*, Sep 2011.

[47] R. Weber, "Securing the Internet of Things," *Computer Law & Security Review*, vol. 26, 2010.

[48] H. Suo, J. Wan, C. Zou, and J. Liu, "Security in the Internet of Things: A Review," *2012 International Conference on Computer Science and Electronics Engineering*, 2012.

[49] K. Zhao and L. Ge, "A Survey on the Internet of Things Security," *2013 Ninth International Conference on Computational Intelligence and Security*, 2013.

[50] A. Riahi, Y. Challal, E. Natalizio, Z. Chtourou, and A. Bouabdallah, "A systemic approach for IoT security," *2013 IEEE International Conference on Distributed Computing in Sensor Systems*, 2013.

[51] C. Medaglia and A. Serbanati, *An Overview of Privacy and Security Issues in the Internet of Things.* in "The Internet of Things", Springer New York, 2010.

[52] "ISO Standards: ISO/IEC 27000." `http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=63411`.

[53] L. Tan and N. Wang, "Future Internet: The Internet of Things ," *2010 3rd International Conference on Advanced Computer Theory and Engineering(ICACTE)*, 2010.

[54] L. Holmquist, J. Redström, and P. Ljungstrand, *Token-Based Access to Digital Information.* in "Handheld and Ubiquitous Computing", Springer Berlin Heidelberg, 1999.

[55] W. Recommendation, "Cross-Origin Resource Sharing." `http://www.w3.org/TR/cors/`, 2014.

[56] B. Forouzan, *Cryptography and Network Security.* Mc Graw-Hill Education, 2011.

[57] J. Garrett, "Ajax: A New Approach to Web Applications." `http://adaptivepath.org/ideas/ajax-new-approach-web-applications/`, 2005.

[58] O. De Coi and D. Olmedilla, *A review of trust management, security and privacy policy languages.* International Conference on Security and Cryptography, 2008.

[59] S. Marechal, "Advances in password cracking," *Journal in Computer Virology*, vol. 4, 2008.

[60] D. Klein, "'Foiling the Cracker': A Survey of, and Improvements to, Password Security," *Proceedings of the 2nd USENIX Security Workshop*, 1990.

[61] B. Kaliski, "PKCS #5: Password-Based Cryptography Specification Version 2.0,." `http://tools.ietf.org/html/rfc2898`, Sep 2000.

[62] N. Provos and D. MaziÃ¨res, "A Future-Adaptable Password Scheme," *USENIX Annual Technical Conference*, 1999.

[63] C. Percival, "Stronger Key Derivation via Sequential Memory-Hard Functions." `http://www.tarsnap.com/scrypt/scrypt.pdf`, 2009.

[64] `https://www.virtualbox.org/`.

[65] `http://www.linuxfoundation.org/collaborate/workgroups/networking/netem`.

[66] K. Papagiannaki, S. Moon, C. Fraleigh, P. Thiran, and C. Diot, "Measurement and Analysis of Single-Hop Delay on an IP Backbone Network," *International DisCoTec Workshop on Context-aware Adaptation Mechanisms for Pervasive and Ubiquitous Services (CAMPUS 2010)*, vol. 28, 2010.

[67] M. Kolšek, "Session Fixation Vulnerability in Web-based Applications." `http://www.acrossecurity.com/papers/session_fixation.pdf`, Dec 2002.

[68] J. Owens, M. Houston, D. Luebke, S. Green, J. Stone, and J. Phillips, "GPU computing," *Proceedings of the IEEE*, vol. 96, May 2008.

[69] J. Gubner, *Probability and Random Processes for Electrical and Computer Engineering*. Cambridge University Press, 2006.