ISCTE IUL
Instituto Universitário de Lisboa

Departamento of Science and Technology

Low-Cost, Lightweight IoT Platform with Custom LPWAN LoRa Integration

André Daniel Lopes dos Santos

Thesis submitted as partial requirement for the conferral of the degree of

Master in Open Source Software

Supervisor (a):
Professor João Ferreira, Assistant Professor ('Agregação'),

ISCTE-IUL

September, 2019

Acknowledgments

To my thesis supervisor, family, and friends.

Abstract

In this thesis, we present our IoT platform developed by using open source technologies, following a flexible component approach. The platform architecture is divided into multiple layers providing the flexibility required in order to allow the integration of custom gateways and application servers.

The proposed IoT platform supports Low Power Wide Area Networks (LPWAN) integration, such as LoRa, using a custom network server implementation. The network server supports uplink/downlink communications with a Message Queuing Telemetry Transport (MQTT) interface.

This work may be applied in remote places, for example, in rural areas in order to support the deployment of IoT solutions. To support multiple IoT enabled devices through different network types, a multi-protocol approach is intended for the proposed platform.

The main goal of our research is to provide a low-cost, lightweight, multi-protocol, as a custom LoRa implementation alternative, that supports, IoT applications, gateways, devices, and their respective sensors/actuators.

keywords: IoT, LoRa, MQTT, Open Source, Low Cost, Lightweight.

Resumo

Nesta tese, é apresentada uma plataforma IoT desenvolvida através da utilização de tecnologias open source, seguindo uma abordagem de componentes flexíveis. A arquitectura da plataforma está dividida em múltiplas camadas providenciando assim a flexibilidade necessária para permitir a integração de gateways e application servers personalizados.

A plataforma IoT proposta suporta a integração de tecnologias Low Power Wide Area Network (LPWAN), tal como LoRa, recorrendo a uma implementação adaptada no network server. O network server suporta comunicações uplink/downlink com uma interface Message Queing Telemetry Transport (MQTT).

Este trabalho pode ser aplicado em zonas remotas, como por exemplo, em áreas rurais com o objetivo de suportar o desenvolvimento de soluções IoT. Para que seja possível sustentar vários aparelhos IoT através de diferentes tipos de networks, é necessária uma abordagem multi-protocolar para a plataforma proposta.

O principal objetivo desta pesquisa foi providenciar uma alternativa de baixo custo, multi-protocolar e leve, com a implementação de tecnologias longo alcance que sustente aplicações IoT, gateways, aparelhos e os seus respetivos sensores/atuadores.

Palavras-chave: IoT, LoRa, MQTT, Open Source, Baixo Custo, Leve.

Contents

List of Tables

List of figures

Glossary

IoT – Internet of Things

MQTT – Message Queuing Telemetry Transport

LoRa – Long Range

OTAA – Over The Air Activation

ABP – Activation by Personalization

M2M – Machine to Machine

LPWAN – Low Power Wide Area Network

Chapter 1 – Introduction

1.1. Research Background

As the demand for Internet-of-Things (IoT) applications and the associated produced data rises, the number of devices available will continue to grow. According to IoT Analytics [1], by 2025, the prediction of device connections is of 21.5 billion from (IoT) devices only. Application domains like Smart Cities, Smart Grids, Smart Buildings, Agriculture, Internet of Vehicles (IoV), Industry 4.0 and others, require a large number of devices that are currently distributed among multiple types of networks, such as Wireless Personal Networks (WPAN), Wireless Local Area Networks (WLAN), Low-power Wide Area Networks (LPWAN) and many others. IoT platforms provide an answer to a fast-growing market, with tools that facilitate the IoT application development process. Each platform provides a set of services, such as data hosting, data routing, analytics, notifications, and respective data transfer/analysis between large amounts of devices and application servers.

By using an existing IoT platform, a given user is relieved of the end-to-end communication handling and the software maintenance process. All these advantages make IoT platforms indispensable when it comes to develop and maintain a solution with multiple applications, devices, users, and many other features. Following the IoT expansion, open-source hardware platforms, like Arduino [2], Beagleboard [3], Wiznet [4] and many others, provide open-source hardware for multiple use cases that are fundamental when creating customized solutions. When compared to proprietary IoT hardware prices, these hardware platforms gain incredible interest.

Existing communication protocols such as MQTT (a machine-to-machine (M2M) /IoT connectivity protocol, that was designed as an extremely lightweight publish/subscribe messaging transport [5]) provide a viable solution for IoT communications between devices and applications and the abstraction layer needed to support multiple application domains. MQTT requires a small code footprint and featuring scarce network bandwidth. The usage of high-level programming languages like Javascript, in combination with the MQTT protocol, decouples the low-level gateway/devices functionalities from the high-level post-processing features [6] in the network server, when routing communications between devices and applications.

1.2. Motivation

Recent LPWAN provides long-range connectivity in the IoT context for multiple application domains. The motivation for this research work is the possibility of exploring these new types of networks and their respective integration within an IoT platform built with fully open-source technologies.

1.3. Investigation objectives

The research objective of the following thesis, is oriented to, the integration of LPWAN technologies in a multi-protocol IoT platform. The investigation is segmented with the following objectives:

1. Develop an end-to-end multi-protocol IoT platform;
2. Use only available open-source technologies;
3. Perform an evaluation scenario with multiple application with different business domains;

1.4. Structure of the thesis

The first two investigation objectives will be accomplished in chapter four. We go through our proposed platform implementation. Within these sections, we develop a prototype IoT platform based on the proposed conceptual model (Chapter 3). In order to develop the proof of concept, multiple programming languages will be used together with the MQTT protocol and LPWAN LoRa technology.

The last two sections of the document (Chapter 5 - Evaluation scenario and results, and chapter 6- Conclusions) are dedicated to evaluation and results. With this section we intend to fulfill the last investigation objectives, evaluate and withdraw conclusions over the developed proof of concept IoT platform;

Chapter 2 – Literature Review

2.1 IoT platforms

An IoT platform should handle the end-to-end communications within a given IoT application. Besides the communication process, an IoT platform should provide a way of managing the integrated devices and respective applications.

As components, an IoT platform contemplates devices, gateways, network servers, and application servers. There are multiple ways of integrating devices within the platforms: i) centralized approach or, ii) using gateways serving as routing middleware for the devices. Gateways are commonly used to bridge network protocols not supported by the network server implementations.

The existing open-source platforms have similar provided services and very identical implementations, differing in implementation purposes and internal usage paradigm. When choosing an IoT platform, it all depends on the requirements of the IoT application to develop and the technical capabilities and limitations of the technology. In table 1, we can observe a list of the ten most-used open source IoT platforms and some of the architectural, technical details.

Table 1 – List of ten of the most used open-source IoT platforms

Name	protocols for data collection	LoRa	analytics	support for visualization	ref
Kaa	MQTT, CoAP, XMPP, TCP, HTTP	No*	Real-time IoT Data Analytics and Visualization with Kaa, Apache Cassandra, and Apache Zeppelin	Yes	[7]
SiteWhere	MQTT, AMQP, Stomp, WebSockets	No*	Real-time analytics (Apache Spark)	No	[8]
ThingSpeak	HTTP	No*	MATLAB Analytics	No	[9]
DeviceHive	REST API, WebSockets or MQTT	No*	MATLAB Analytics	Yes	[10]
Zetta	HTTP	No*	Using Splunk	No	[11]
IoT-DSA	HTTP	No*	No	No	[12]
Thingsboard.io	MQTT, CoAP, and HTTP	No*	Real time analytics(Apache Spark, Kafka)	No	[13]
Thinger.io	MQTT, CoAP, and HTTP	No*	Yes, Platform not confirmed	No	[14]
WSo2	HTTP, WSO2 ESB, MQTT	No*	WSO2 Data Analytics Server	Yes	[15]
Mainflux	HTTP, MQTT, WebSocket, CoAP	No*	(integrated) Platform not confirmed	Yes	[16]

* Through the use of the third-party platforms[17] or open-source libraries[18][19], it is possible to integrate LPWAN networks in this case LoRaWAN[23] (or LoRa with customized gateways[20]).

The network server component is responsible for the high-level logic within the IoT platform, managing the entire communications routing process. The application server component functions as user interface, data storage, data analytics tool, and many other services, providing the user with the analysis and interaction tools within the IoT platform.

2.2.1 LPWAN LoRa integration

When regarding LPWAN integration, in this case, LoRa, we can observe that none of the existing platforms in table 1 contemplates a native LoRa support. Instead, the platforms are relying on third-party platform services and libraries. LPWAN technologies for IoT, introduced by Sigfox (UNB solution) and Semtech (LoRa™) [21], triggered a new innovation cycle as they provide long-range connectivity answer for the IoT context. Most of these long-range technologies can achieve up to 20 km or higher range in line-of-site (LOS) condition and about 2km-4km in non-LOS conditions, just like dense urban/city environments. Initiatives based on LoRa such as TheThingsNetwork™ (TTN) [22] provide dense city environments solutions. It has, however, limited range coverage on remote areas [24], having a direct impact on IoT specific domains as beekeeping, agriculture, innovation, or sensor metering. In this article, our research aims to develop an IoT platform, proof of concept solution, that besides supporting the integration with third-party libraries and platforms for LPWAN, also contemplates a native custom LoRa integration.

2.1.2 Data collection protocols

Focussing on data collection, there are multiple available protocols. In table 1, we can notice that the existing platforms all rely on identical approaches. However, the MQTT protocol is a constant in almost all of the IoT platforms approaches. The MQTT is an established open protocol in machine-to-machine (M2M) communications, which was introduced in 1999 by Andy Stanford-Clark of IBM and Arlen Nipper of Arcom Control Systems Ltd (Eurotech). It has been used and supported by organizations such as IBM, Facebook, and Cisco and standardized by the OASIS Technical Committee. It is a publish/subscribe messaging protocol, where clients publish/subscribe to multiple addresses, known as topics. It was designed for lightweight M2M communications in constrained networks[25]. It supports three levels of Quality of service (QoS). Being an Oasis standard [26] makes this protocol easy to adopt for the wide variety of IoT devices, platforms, and operating systems. On the proposed platform, we focus on MQTT for its lightweight properties and scarce network bandwidth, ideal for remote places and low resource devices.

Chapter 3 – Platform proposal

When regarding LPWAN integration, in this case, LoRa, we can observe that none of the existing platforms in table 1 contemplates a native LoRa support. Instead, the platforms are relying on third-party platform services and libraries.

LPWAN technologies for IoT, introduced by Sigfox (UNB solution) and Semtech (LoRa™) [21], triggered a new innovation cycle as they provide long-range connectivity answer for the IoT context. Most of these long-range technologies can achieve up to 20 km or higher range in line-of-site (LOS) condition and about 2km-4km in non-LOS conditions, just like dense urban/city environments. Initiatives based on LoRa such as TheThingsNetwork™ (TTN) [22] provide dense city environments solutions.

It has, however, limited range coverage on remote areas [24], having a direct impact on IoT specific domains as beekeeping, agriculture, innovation, or sensor metering. In this thesis, our research aims to develop an IoT platform, proof of concept solution, that besides supporting the integration with third-party libraries and platforms for LPWAN, also contemplates a native custom LoRa integration.

As requirements for the proposed platform, it should:

- Provide connectivity to multiple devices and respective communications;
- Support medium and long-range connectivity;
- As any IoT platform, provide an abstraction over the system communication, turning its interaction intuitive, scalable and easy to use for the users.

Table 2 – Proposed platform architectural details

Protocols for data collection	**LoRa**	**Analytics**	**Support for visualizations**
MQTT, HTTP	Native custom implementation	No	Yes

Figure 1 shows our platform architecture, where a diverse range of sensors was installed over an array of devices to transmit collected data. These devices provide data to a cloud network where data is then transmitted through the associated applications.

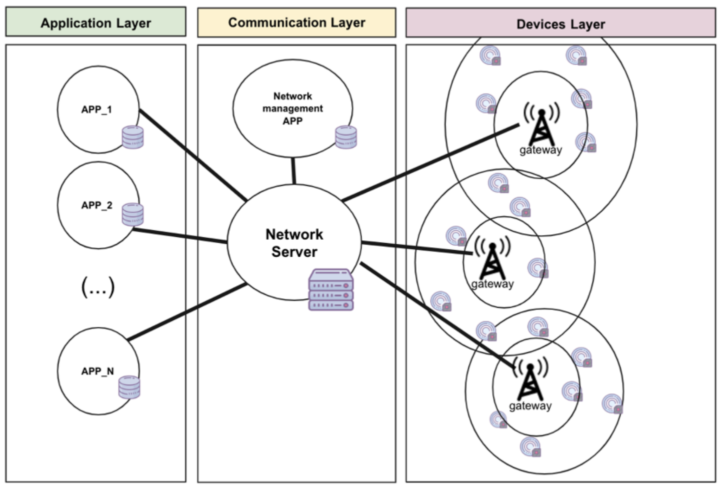


Figure 1 – Platform proposal architecture

The proposed IoT platform is divided into three major layers, the devices, the communication, and the application layer:

1. Devices layer - Sensor data acquisition, transmission, and gateways;
2. Communication Layer - Network management - Data exchange between app/device layer;
3. Applications layer - Where the developer can apply any business logic and data analytics;

Each layer, with its respective components, provides the full architecture for the end-to-end communication between devices and applications. Each layer is further described and analyzed through this document. The current approach resembles the LoRaWAN protocol architecture, organized in a star-of-stars topology, where the network server is also a bridge between gateways and applications [23], although LoRaWAN networks are specific for LoRa communications. With this star-of-stars topology in opposition to the centralized approach, it becomes very intuitive when migrating to a local deployment model, just by deploying the communication layer into a gateway. As the main features, the proposed platform aims to support:

- Protocol bridging (i.e. HTTP, MQTT, LoRa);
- Device management through a network management platform application;
- Access control through Basic Authentication using JSON Web Tokens (JWT), HTTPS, MQTTS;
- Default application server;

- Lightweight deployment for remote and local model approaches;

With our intention being the development of a low cost, lightweight, and hardware flexible platform for IoT that can be applied in multiple use cases. In order to support multiple types of network, customized gateways and application servers can be integrated to interact with the communication layer through the MQTT protocol, granting the flexibility to adapt to different application scenarios and respective needs.

Chapter 4 – Implementation

4.1 Communication layer

The bridge between the application and the device layers is the "Communication Layer." The goal for this layer is to have an optimized lightweight solution that can be deployed to less resourceful microprocessors, if needed, giving the developer the option to implement the complete platform using, for example, an embedded Linux system like Raspberry Pi. It was also developed keeping in mind the idea of saving the least amount of information as possible, making sure that the amount of storage used for this layer is optimized to the minimal required to normally perform its associated tasks. The communication layer is composed of two elements, network server, and management app.

4.1.1 Network server

The network server component is responsible for the network communication bridge between devices and application layer. Through the use of high-level languages, we were able to create an abstraction layer decoupled from the gateway communications and provide logic to manipulate the communication routing process.

4.1.1.1 Network server MQTT broker

For our network server implementation, the "mosca" MQTT broker [27] open-source library was used. Its CPU performance equates to that of the Mosquito, which is the one used as reference MQTT broker [28]. Every gateway and application is an MQTT client of the network server MQTT broker (Figure 2).

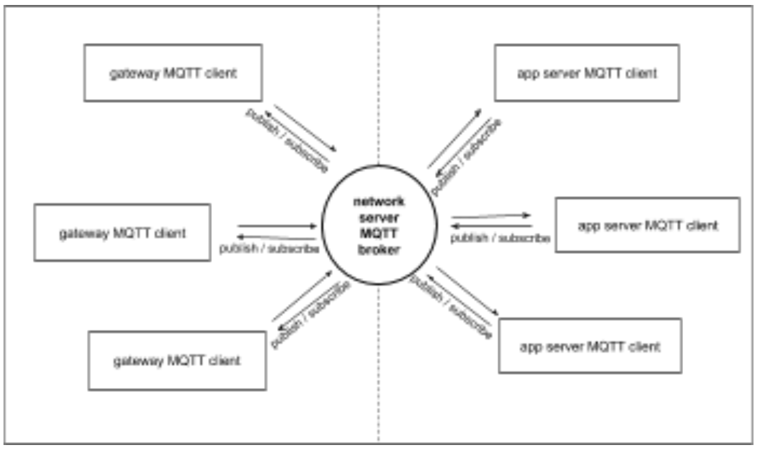


Figure 2 – Network server MQTT broker architecture overview.

The high-level logic, implemented into the network server MQTT broker, over the communication turn downlink and uplink message routing possible. In Figure 3, we demonstrate how we handled uplink messages using the high-level logic build with Javascript.

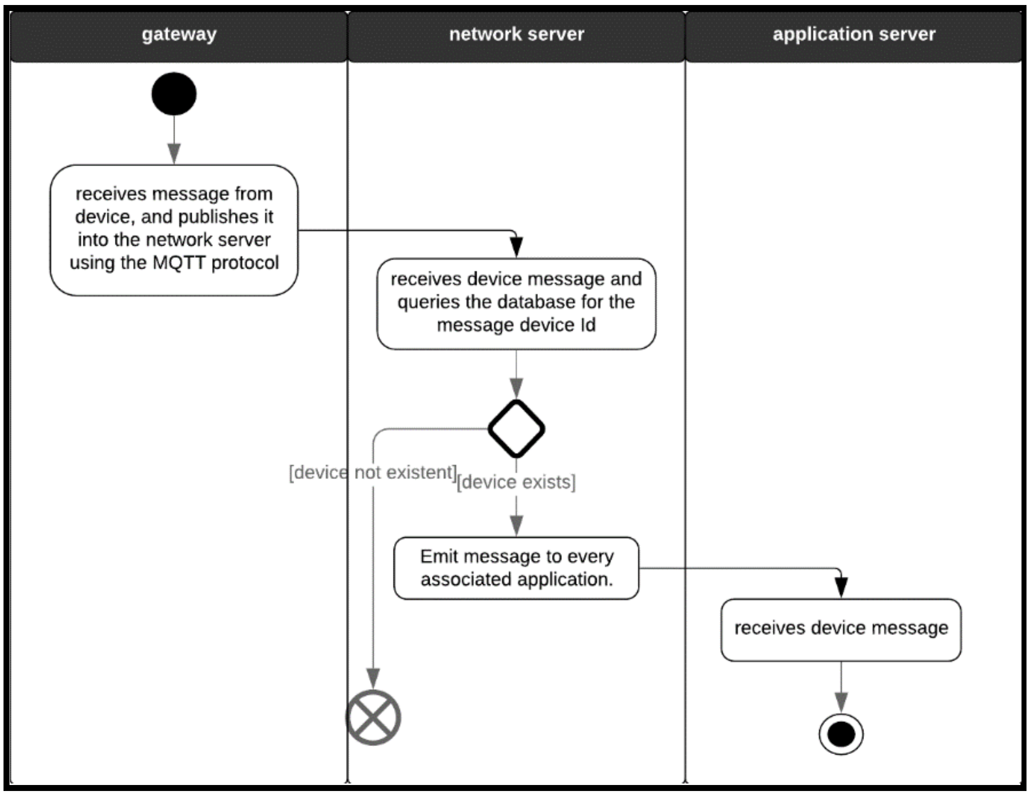
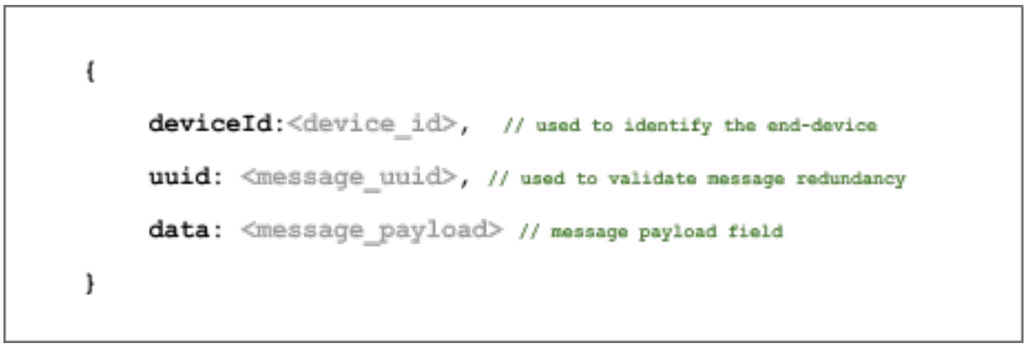

gateway
network server
application server
receives message from device, and publishes it into the network server using the MQTT protocol
receives device message and queries the database for the message device Id
[device not existent]
[device exists]
Emit message to every associated application.
receives device message

Figure 3 – Uplink flow UML activity diagram.

By the time the following flow occurs the IoT gateway is already authenticated within the network server MQTT broker. Taking into account the MQTT protocol QoS functionality, we are able to assure QoS between the multiple layers.

The accepted message format is as follows in Figure 4.



```
{
    deviceId:<device_id>,   // used to identify the end-device
    uuid: <message_uuid>, // used to validate message redundancy
    data: <message_payload> // message payload field
}
```

Figure 4 – Device uplink message format.

4.1.1.2 Custom LoRa implementation

One of the main objectives of this article investigation is the integration of LPWAN LoRa with IoT platforms. Within our state the art, we realized the usage of third party platforms and libraries, associated with the integration of a LoRa communication protocol LoRaWAN[23]. LoRaWAN builds on top of LoRA, complementing with addressing, encryption, and other additional layers. The implementation of the full LoRaWAN specification requires the gateway component to be able to listen on several channels and LoRa settings simultaneously, such gateways could increase the price of the solution and the complexity of a simple LoRa gateway. The usage of the LoRaWAN protocol also implies the existence of a LoRaWAN network server, even for a local platform deploy. In our approach, we implemented a custom LoRa integration. We didn't use the LoRaWAN protocol because the related network server complexity and specifications, would oppose this investigation purposes. Our approach does not imply any type of LoRa related software impositions, allowing us to maintain our flexible gateways approach and network server lightweight deployment. Therefore the network server does not handle retransmissions and many other LoRaWAN requirements [23]. Duty cycle regulations [29], when downlinking and uplinking, must also be prepared by the application developer. The regulation restrictions to uplink should be handled by the end-device and the downlink by the application layer. To assure some level of security within our exchanged messages, we used a similar approach to LoRaWAN authentication by personalization (ABP) method figure 5. We persist the device crypto key inside the communication layer and use it for every message decryption. This ABP approach is not as safe as Over-The-Air Activation (OTAA) but fits in our approach for its performance and flexibility of implementation.

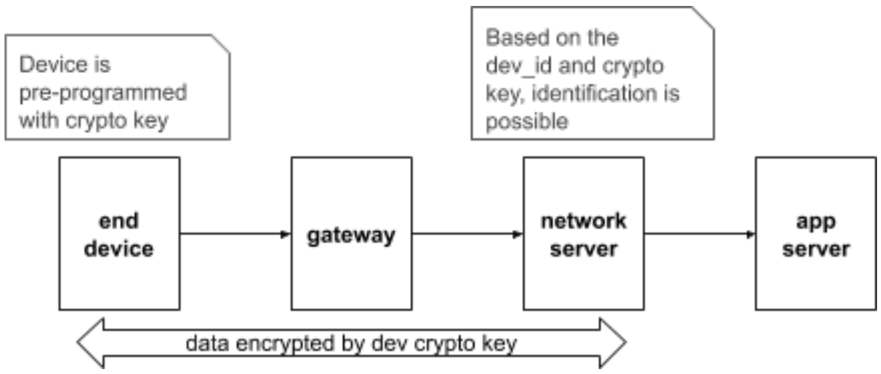
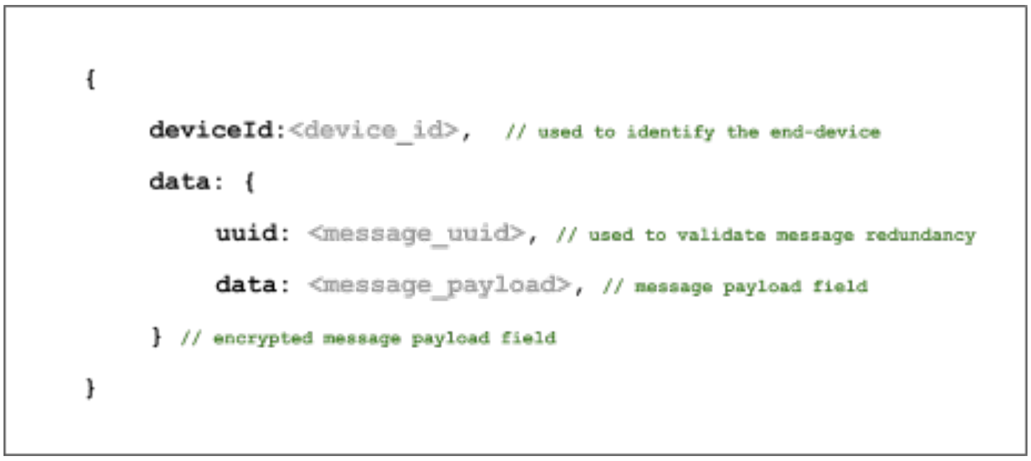

Device is pre-programmed with crypto key
Based on the dev_id and crypto key, identification is possible
end device
gateway
network server
app server
data encrypted by dev crypto key

Figure 5 – Encryption approach for LoRa packages.

When submitting a LoRa message to the network server broker, the message should be in the format presented in figure 6. Both message payload and uuid, must be encrypted inside the message data attribute.



```
{
    deviceId:<device_id>,   // used to identify the end-device
    data: {
        uuid: <message_uuid>, // used to validate message redundancy
        data: <message_payload>, // message payload field
    } // encrypted message payload field
}
```

Figure 6 – Device uplink message format.

Regarding LoRa packets that might be published from gateways, the network server predicts different rules (figure 7), in opposition to regular MQTT end-device published messages. It’s expecting an encrypted field from a LoRa published message. This is used to validate the authenticity of the incoming communication. It tries to decrypt the provided message with the device crypto encryption key. If successful, stores temporarily the message UUID (universally unique identifier) in order to avoid message redundancy between layers. This way, supporting gateway redundancy by validating if a message with a specific UUID was already routed. Application domain data encryption in published messages should be handled by the IoT application developer for uplink/downlink.

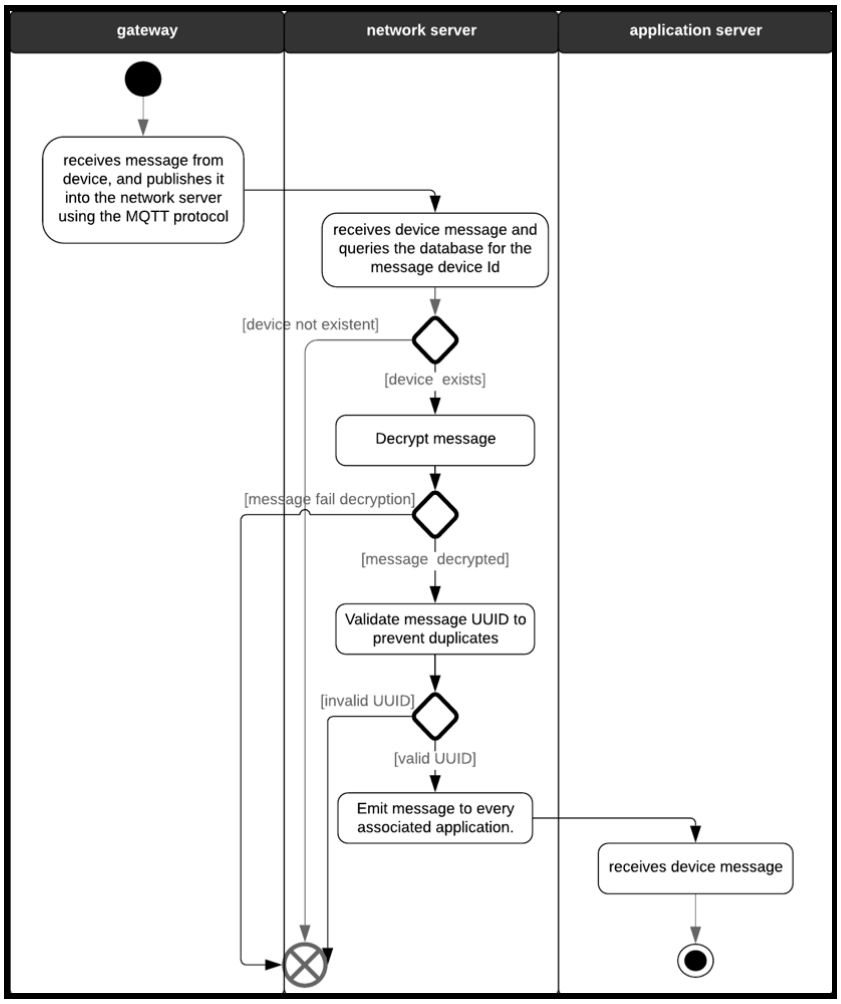

gateway
network server
application server
receives message from device, and publishes it into the network server using the MQTT protocol
receives device message and queries the database for the message device Id
[device not existent]
[device exists]
Decrypt message
[message fail decryption]
[message decrypted]
Validate message UUID to prevent duplicates
[invalid UUID]
[valid UUID]
Emit message to every associated application.
receives device message

Figure 7 – LoRa uplink message uplink flow.

4.1.1.3 Persistence

MongoDB is a cross-platform and open-source document-oriented database engine [30]. MongoDB, as a NoSQL database, prevents the relational database's table-based structure to adapt JSON-like documents with dynamic schemas BSON [31]. MongoDB was adopted for our platform as a consequence of its capability to store and analyze any type of data, in real-time and respective scalability [32]. MongoDB can support thousands of nodes, petabytes of data, and hundreds of thousands of operations per second. With the proper implementation, MongoDB can achieve a good performance on CRUD operations and low-power consumption metrics on embedded systems [33]. For the proposed platform persistence, the database schema present in Figure 8 was implemented in the Mongo database engine, making use of the available data models and patterns[34]. Entities, such as "applications", "users," "devices," and "gateways", define the data model.

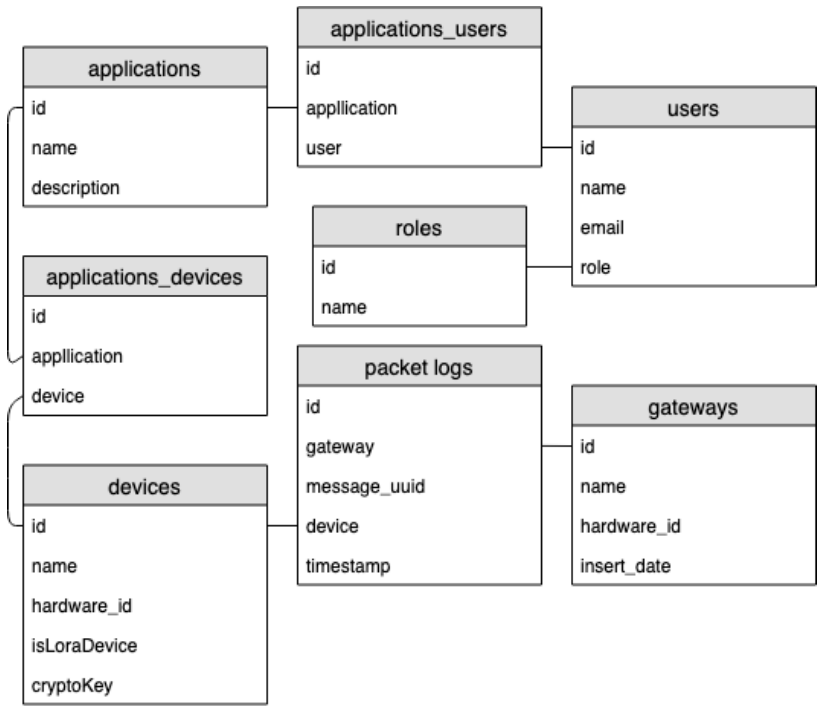


Figure 8 – Platform persistence architecture schema.

For every received, uplink message, a packet log is created. It's through these logs that we store the devices routing information, like a gateway, message UUID and timestamp. In order to emit a downlink message to a specific end-device, it must have sent at least one uplink message. Regarding the relation between devices and applications, each application has multiple devices, and one device might belong to multiple applications. Each application has a user relation, one user might have multiple applications, and an application can have multiple associated users.

Users relate to devices through applications, for every application, a set of devices can be associated that users can interact with. Users are also divided into multiple roles. These roles are used to manage the network permissions over certain actions (section 4.1.1.5).

4.1.1.4 RESTful API

A RESTful API is an application program interface (API) that uses HTTP requests to GET, PUT, POST, and DELETE data. For a given IoT application, the real-time devices data stream might not be sufficient to implement some functionalities. Obtaining the list of devices associated to a specific platform application, or getting information about a specific device or application, are some examples of basic features that justify the need for an API implementation for the proposed platform. Besides routing, the network server has an available API that provides the application owners endpoints (Table 3) to obtain end-devices and applications' related information that is designed for this end, in opposition to the MQTT broker functionality within the proposed platform.

Table 3 – Network server public API endpoints.

Endpoint	Description
device	returns all devices associated with a specific application with the respective static information.
device/:id	returns device static information for the provided Id.
application	returns all applications associated with a specific app owner login.
application/:id	returns application static information for the provided Id.

4.1.1.5 Network Management App

In order to manage the network traffic logic and network settings, a management app associated with the network server was developed. The network management application, grants the user access to real-time message tracking, static information about users, devices, applications, and gateways. Figure 9 presents a use case UML diagram for multiple actors, such as network admin, managers, and app owners. These actors have the ability to influence the platform by creating users, devices, gateways, manage applications, and track routed data. The goal of this application is to have an abstraction layer over the network server that allows intuitive network management by different user roles.

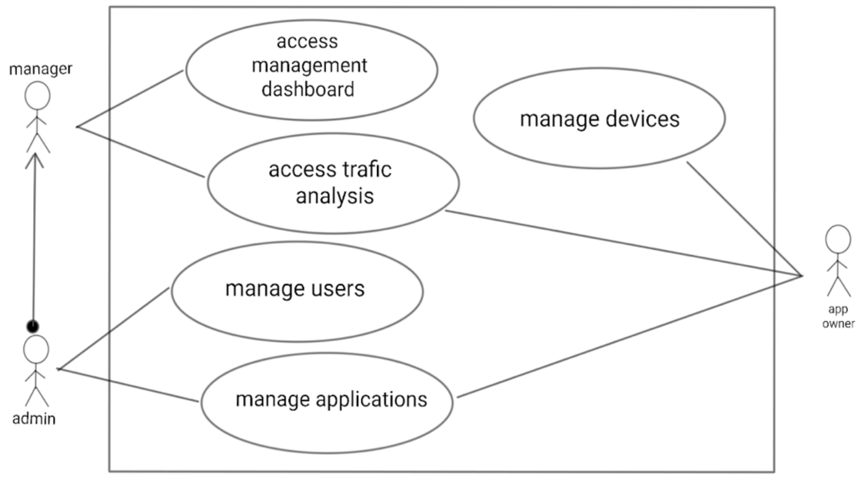


Figure 9 – Network management app, UML use case diagram.

For the proposed platform, three types of roles were created for the management app. The “manager”, “admin” and “app owner”.

- For the “manager” role, it was given permission to access the management dashboard and monitoring the device’s traffic. The management dashboard allows the users to have a general view of the network (how many devices, gateways, users), as well as monitoring the effectiveness of the communication.
- As for the “admin” role, it was given access to the same features as the manager. The “admin” also has access to users’ and applications’ management , enabling him to create users (new “app owners”) and to register new applications on the management app.
- The “app owner”, has the possibility to access the device’s traffic analysis, allowing him to comprehend the gateway range regarding the devices. It also can manage the applications that the “admin” gave him permission to. As for the management of the devices, the app owner is the only entity with access to the devices.

4.2 Devices layer

4.2.1 Gateways

IoT Gateways are emerging as a key element on the legacy and next-gen devices support. Gateways are flexible with different hardware implementations, networking protocols, storage and facilitate data flow securely between end-devices and the network server. The proposed platform isn't restricted to any gateway specific implementation pattern or software. The gateway objective is providing the end-devices the support for the applications data flow, with the necessary networking requirements(e.g., support LoRa or WiFi communications) and respective communication exchanges (e.g., downlink/uplink).

In remote locations where connectivity is limited or non-existent and many other scenarios, the gateway assumes a highly important role in the platform deployment model. In a local deployment model, the gateway element should be able to host the network server and respective application servers, that would demand an embedded system environment like Raspberry Pi, to ensure the proposed platform communication layer deployment. In Figure 10, we can observe multiple deployment models for the gateway and the entire platform.

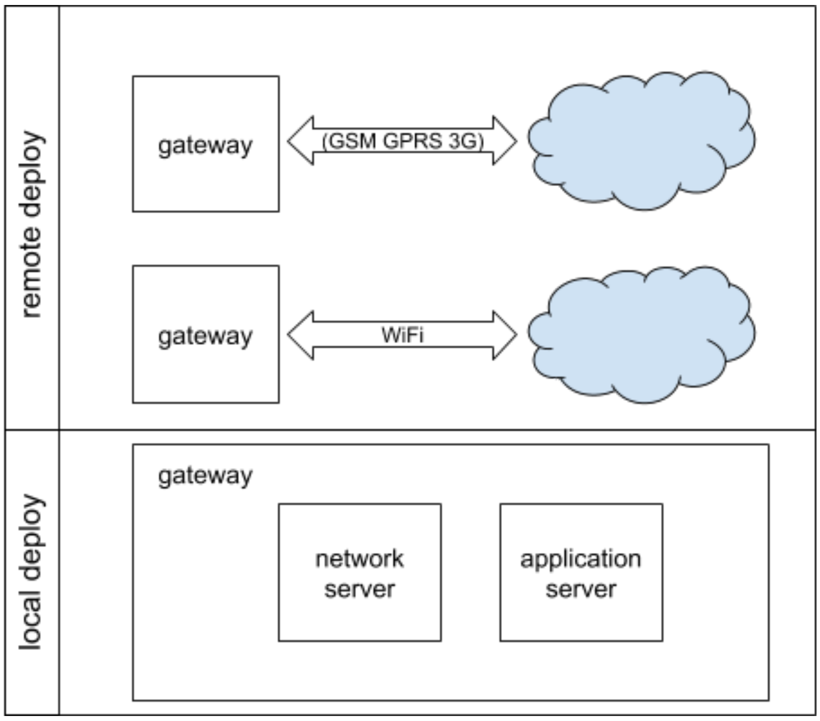


Figure 10 – Platform deployment models.

In a remote deployment model, the gateway needs to ensure connectivity to the Internet with ideal fallbacks (e.g., WiFi, 3G), this redundancy would allow for a more stable gateway/network. For certain implementation scenarios, the gateway could even be deployed in a microcontroller.

The network server, integrates gateways into the IoT platform, through the use of the available MQTT broker. The network server MQTT broker predicts the following topics that gateways are allowed to publish (Table 4).

To receive downlink messages, the gateway should subscribe to the "GATEWAY/<gateway_id>/device/downlink" topic. This will allow the network server to reach the gateway when downlinking through the MQTT protocol.

Table 4 – Network server MQTT broker - Gateways topics.

Topic	Description
device/connected	topic to publish when Wifi end-device connects to gateway MQTT broker.
device/disconnected	topic to publish when Wifi end-device disconnects from the gateway MQTT broker.
device/LoRa	Topic to publish when sending uplink LoRa communication.
device/WiFi	Topic to publish when sending uplink WiFi communication.

4.2.2 End Devices

End-devices are a key component of the proposed platform. They provide the real-world collected data, that is further stored and analyzed in multiple contexts. Platforms such as Arduino, have a wide range of available board models with different features from large and powerful to smaller with less energy consumption. With alternatives like Arduino [35], nodeMCU [36] and many others, it becomes possible, and it's clearly an opportunity to integrate the proposed platform with a variety of end-devices and sensor data.

When building an IoT application, a variety of end-devices and respective sensors can be used. It's up to the developer the implementation of the application domain logic specifics. As an end-device in order to connect to an application, it must be in the range of a supported gateway network. The communication layer is abstracted from the "low-level" device/gateway communications, and therefore, the end-devices communication requirements depend directly on the gateway implementation.

4.3 Application layer

The application layer is the bridge between the business logic, user interface, and the communication layer. This layer is responsible for storing and managing the routed raw data. The proposed platform provides a default application server with limited data visualization capabilities. It’s up to the developer to manage how to receive the devices data flow and respective operations over the exchanged data, such as analytics and others. The application layer interacts with the network server through the MQTT protocol and also with an available API (Figure 11). A specific application receives the associated device's data and is able to interact with that application(s) related devices.

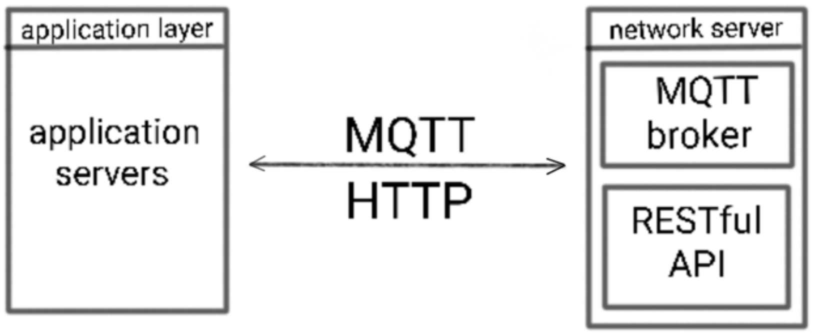


Figure 11 – Communication pattern overview between application servers and communication layers.

The application server specifications depend directly on the IoT Application requirements to be developed. In a local deployment model, the application(s) server(s) must be able to reach, the communication layer, network server component, assuring that all functionalities work in a local environment. Therefore it is very important to have in consideration the required resources associated with the IoT application(s) server(s).

4.3.1 Default application server

In order to facilitate the IoT application integration, we developed a default application server. This application server is prepared only for data visualization. In Figure 12 we can find a use case UML diagram for the default application server.

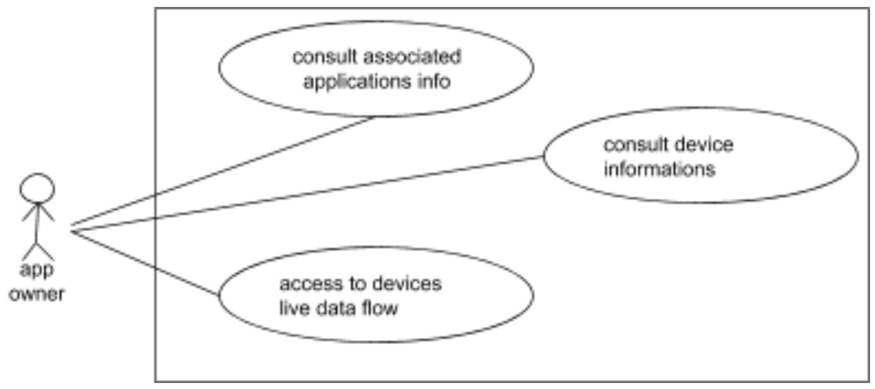

consult associated applications info
consult device informations
access to devices live data flow
app owner

Figure 12 – Default application server, UML use case diagram.

Chapter 5 – Evaluation scenario and tests

Taking into account, the developed IoT platform we wanted a test scenario that holds multiple IoT application with different requirements. We want to test the developed platform in different rural scenarios creating two dedicated application to validate, identified in Table 5.

Table 5 – Evaluation scenario testing applications.

Name	Description	Network requirements
monitBee	monitors unwanted intrusion detection in beehives	long-range network
monitPal	monitors temperature and humidity in the farm's warehouse	wifi network

Our farm is located with a distance of approximately 120 km from Lisbon, where internet coverage is not available, the only cellular. On Figure 13 we can see a satellite photo of the place. The beehives area is represented in green and the warehouse in blue. The beehives are separated in multiple areas all distant from the warehouse.

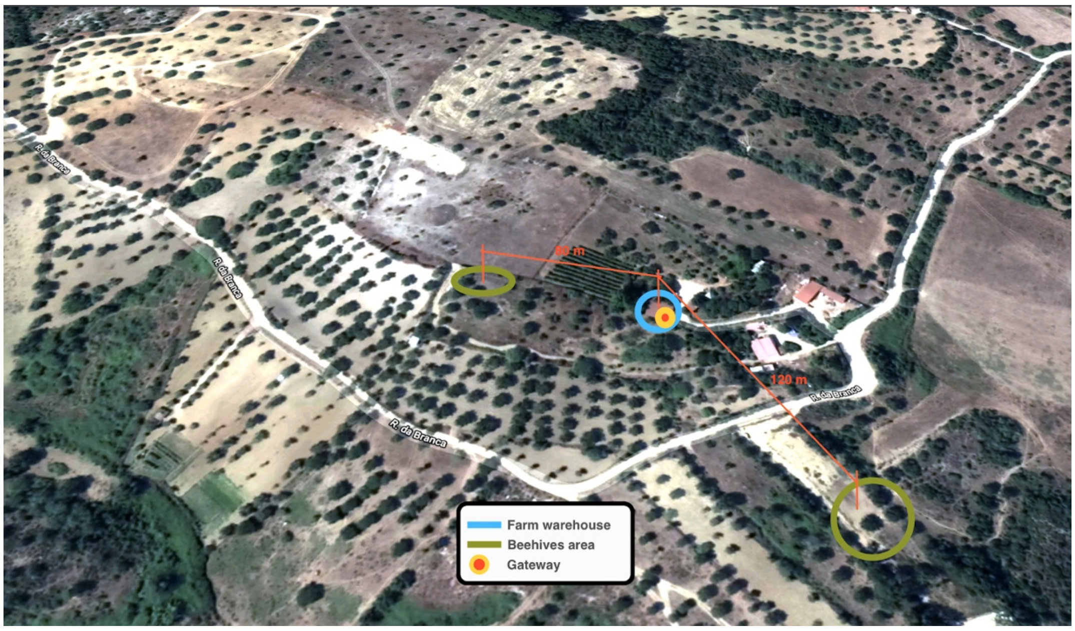


Figure 13 – Test scenario over Google maps image.

The farm's warehouse is where many of the farm's products are stored, and It's important for the farmer to control the temperature and humidity of its rooms.

Our test was performed in multiple steps:

1. Communication layer deployment;
2. build a custom gateway (supporting WiFi and LoRa communications);
3. acquire end-devices that comply with the gateway custom implementation;
4. develop and deploy custom application servers;

5.1 Communication layer deployment

To achieve a remote distributed deployment environment, we deployed our communication layer into a cloud service [37], that provides, among other services, ARMv8 servers built for developers. They include a server, a volume, and an IPv4 and its dedicated to development, prototyping and running services. For our communication layer deployment, we used a server with the characteristics in Table 6.

Table 6 – Communication layer host service.

Spec	Description
vCPUs	2 vCPUs
Memory	2 GB
SSD Storage	20 GB
Bandwidth	100 Mbits/s

In this virtual server instance, we used a Linux Debian distribution as our operating system. The hosting costs are less than five euros, a very comfortable price for our tests. Besides the server, we also acquired a domain to easily reach our remote network server instance.

5.2 Custom Gateway implementation

The provided gateway (Figure 14) supports LoRa/WiFi communications, is able to connect to the web and has a power supply. It provides a WiFi network using the raspberry pi and with the "HelTec WiFi LoRa 32" we were able to bridge the LoRa packages into MQTT.

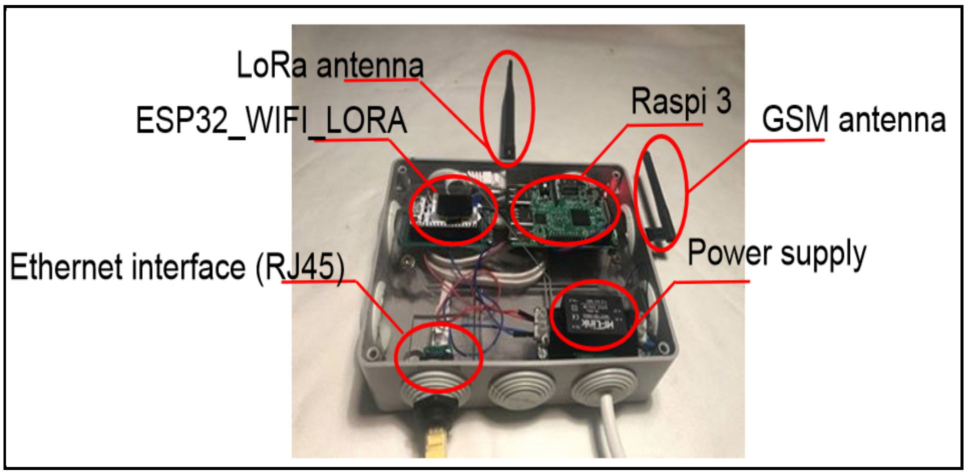


Figure 14 – Gateway and respective components.

Table 7 – Gateway components.

Component	Description
Radio antenna	Connects to the HelTec WiFi LoRa 32 System-on-Chip (SoC) as a LoRa radio antenna
HelTec WiFi LoRa 32	SoC is responsible for mapping LoRa messages into MQTT over WiFi when uplinking and MQTT to LoRa when downlinking.
Raspberry Pi 3	SoC used for WiFi LAN creation, GSM integration, network server communication, and Ethernet interface support.
GSM antenna	In remote areas where connectivity issues arise, this gateway supports a GSM connection as an alternative.
Ethernet interface (RJ45)	external ethernet interface connects to a microprocessor (raspberry pi) grating the gateway with an internet connection it's possible to connect the gateway to the internet with GSM or to use the ethernet interface.
Power supply	5v, 1 amp power supply.

5.2.1 LPWAN LoRa implementation in custom gateway

For the proposed platform test gateway, LoRa implementation was made through the use of an open-source library that exposes the LoRa radio directly [38] and allows sending data to any radios in range with the same radio parameters. All data is broadcasted, and there is no addressing. To manage the addressing issue, this library offers the possibility of using radios with different Sync Words, and InvertIQ function to create a simple Gateway/Node logic by inverting the LoRa I and Q signals. In the current platform context, the addressing layer was created based on the end-device identifier and an asynchronous encryption key. It's possible to develop a low-cost LoRa single and multiple channel architectures independent of the network server architecture as long as the communications through MQTT remain between the gateway and the network server. The developed gateway is single-channel and its highly susceptible to package collisions since we didn't develop any time-slot reservation algorithm.

Providing more radio channels, developing gateway-driven frequency hopping scheme and collision-free access for higher priority traffic, are some mechanisms that improve the gateway's performance and still keep a low-cost approach [39].

5.3 End-devices

5.3.1 LoRa End devices

When sending a message through LoRa Figure 15, the gateway receives the message and routes it to the network server through MQTT (Figure 15). The LoRa end-device must be in the range of the developed gateway to be able to reach the IoT platform data flow.

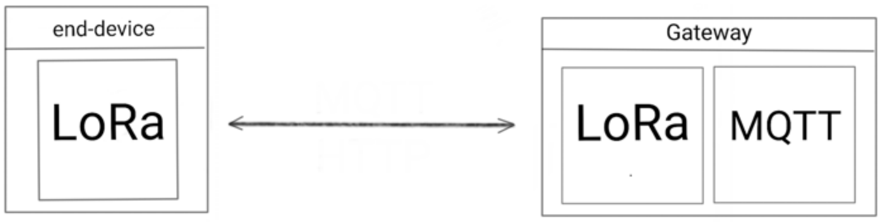


Figure 15 – LoRa communication between end-device and gateway.

In order to accomplish the application objective ("beehive human intrusion detection, monitoring"), we integrated into the end-device a tilting sensor(SW-420) that would report every time the beehive was tilted (Fig. 16). Two long-range end-devices were placed on each of the beehives areas. Our LoRa supporting end-device is a development board by BSFrance "LoRa32u4" that has a battery circuit built-in. With our end-device, we included a 1000mAh battery with 3.7v.

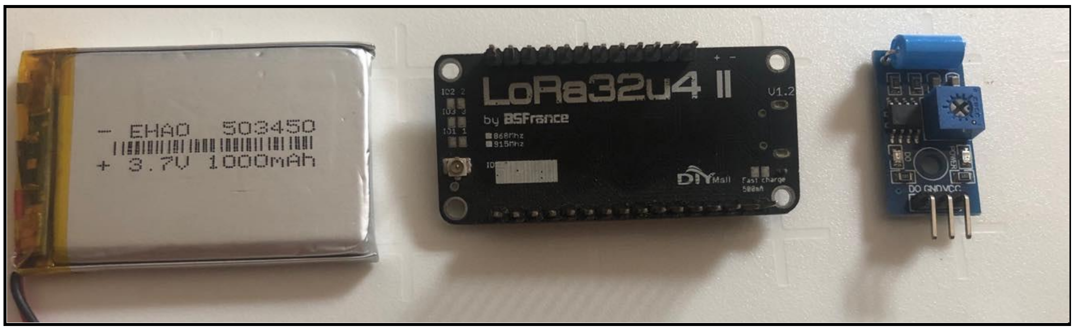


Figure 16 – LoRa end-device used by MonitBee (testing app)with tilt sensor and 1000 mAh battery.

Duty cycle regulations for long-range communications were calculated based on parameters of Table 8.

Table 8 – LoRa configuration parameters used in Time – on – Air calculations.

Parameter	Value	Description
Payload size	40 bytes	Total data payload
Spread factor	SF8	Higher means more range and better reception, and consequent increase in airtime
Explicit header	yes	This is the low-level header that indicates coding rate, payload length, and payload CRC presence
Low DR optimize	no	intended to correct for clock drift at SF11 and SF12
Coding rate	4 / 5	This is the error correction coding. Higher values mean more overhead.
Preamble symbols	8	8 for all regions defined in LoRaWAN 1.0, can be different using plain LoRa
Bandwidth	125kHz	Usually 125, sometimes 250 or 500

According to the parameters in Table 8, and using a LoRa ToA Time-on-Air online calculator tool, we determined that the time between t subsequent packet starts would be 15.41 seconds. With this parameter configuration, and according to the duty cycle regulations, we would be allowed to emit 5712 messages/day. This is not very important in this test scenario because our long-range hive devices can only emit if the tilt sensor is triggered, so a given end-device, could go several days without sending a single message.

5.3.2 End devices supporting MQTT protocol over WiFi interface

Any end-device that has a WiFi/Ethernet interface and supports MQTT protocol is suitable to integrate the provided testing gateway. In figure 17, we have an overview of the device/gateway communication flow.

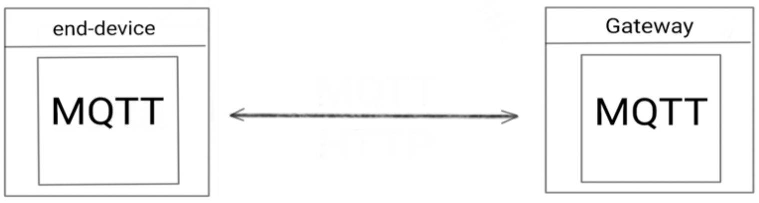


Figure 17 – MQTT communication between end – device and gateway.

For the short-range devices, we added a temperature/humidity sensor that allows the end-device to collect that data (Fig. 18). These end-devices connect to the gateway using the WiFi network generated by the raspberry pi. In our wifi end-devices "NodeMCU" we also included the same battery as the LoRa devices.

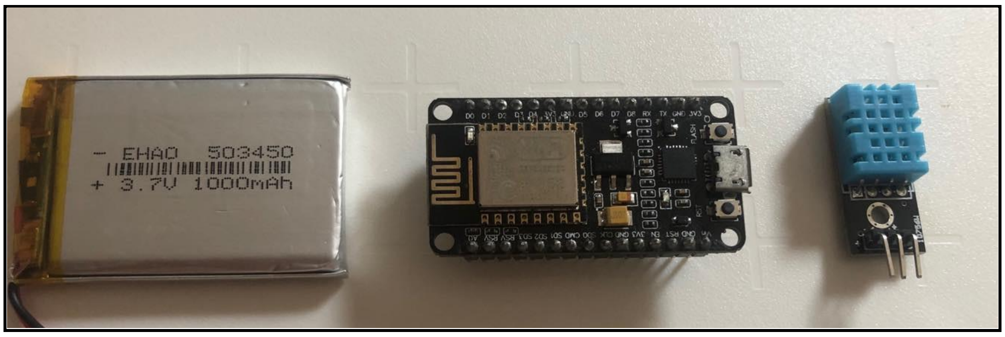


Figure 18 – WiFi end-device used in MonitPal(testing app) with Hum/Temp sensor and 1000mAh battery.

By using the deep sleep function, we're able to sleep the device while not transmitting, this way, saving power and extending the battery range.

5.4 Application servers

Within our test scenario, we have two distinct applications that require different application server instances. We developed two sample application servers, based on the nodeJS technology, that included a FrontEnd component built with the Angular framework. We persisted every package received from the network server in the same MongoDB engine instance used for the communication layer.

We implemented some basic functionalities for both applications. The beehive app provides the user with an interface to monitor the beehives movements and an alarm system based on web sockets that warn the user of the beehive tampering. For the warehouse temp/hum monitoring app, we store every package received by the end-devices and display them in graphics, showing the temp/hum variations along a defined timeline.

Since we've deployed both our applications in the same remote environment of the network server, they're now reachable using a mobile phone or any other device with an internet browser and internet access. This makes it possible for the beehives keepers to be warned anywhere as long as an internet connection is established.

5.5 Results

We conducted our prototype for three months at the farm site in Figure 13. In this section, we present the results associated with the collected data for both testing apps.

5.5.1 MonitBee

This application's objective was to warn the beekeeper of possible beehive tampering. We used long-range devices that supported LPWAN in this case, LoRa (section 7.3.1). To sense the beehive movements, we use a tilt sensor, that would activate the device in sleep mode, every time it tilts. The device would send a message to the gateway and respective application, where an alarm would be generated. In Table 9, we can see some of the metrics associated with the communication for the three months experience.

Table 9 – MonitBee application results.

Name	**Nº end-dev**	**Nº Exchanged messages**	**Message size**
monitBee	**4**	**17**	**40 byte**

None of the received messages where real alarms with the beehives. All messages match the beekeeper visits to the beehive sites, for maintenance purposes. The total hardware costs, for MonitBee are described in Table 10. It accounts for end-devices and respective sensors and batteries.

Table 10 – MonitBee applications hardware costs.

Component	**unit**	**Unit cost**	**Total cost**
LoRa32u4 II Lora	4	≈ 13.00 €	≈ 50.00 €
Tilt sensor	4	≈ 1.00 €	≈ 4.00 €
Battery	4	≈ 3.00 €	≈ 12.00 €
Wiring and casing	4	≈ 1.00 €	≈ 4.00 €
Total	4	≈ 18.00 €	**≈ 72.00 €**

The total hardware costs for the MonitBee app were off, approximately seventy-two euros to have 4 devices, 2 on each beehive area according to the evaluation scenario in Figure 13.

5.5.2 MonitPal

The MonitPal IoT application has the objective of providing the farm with metrics associated with humidity and temperature in the farm's warehouse, where all the farm's goods are stored. For this application, WiFi end-devices were used with a humidity/temperature sensor (section 7.3.2). In Table 11, we can see some of the metrics associated with application communication.

Table 11 – MonitPal applications results.

Name	Nº end-dev	Nº Exchanged messages	Message size
monitPal	4	≈ 25 000	22 byte

Warehouse rooms with windows vary in temperature during the day and keep ideal humidity values (≈50%). However, more isolated rooms revealed more stable temperatures with higher values of humidity (80%). We can observe this by looking at the data collected (stored on the application server of section 7.4) for two days of metering in Figure 12. In the present chart with the violet color, we have a room with solar exposition and in blue a more isolated room of the warehouse.

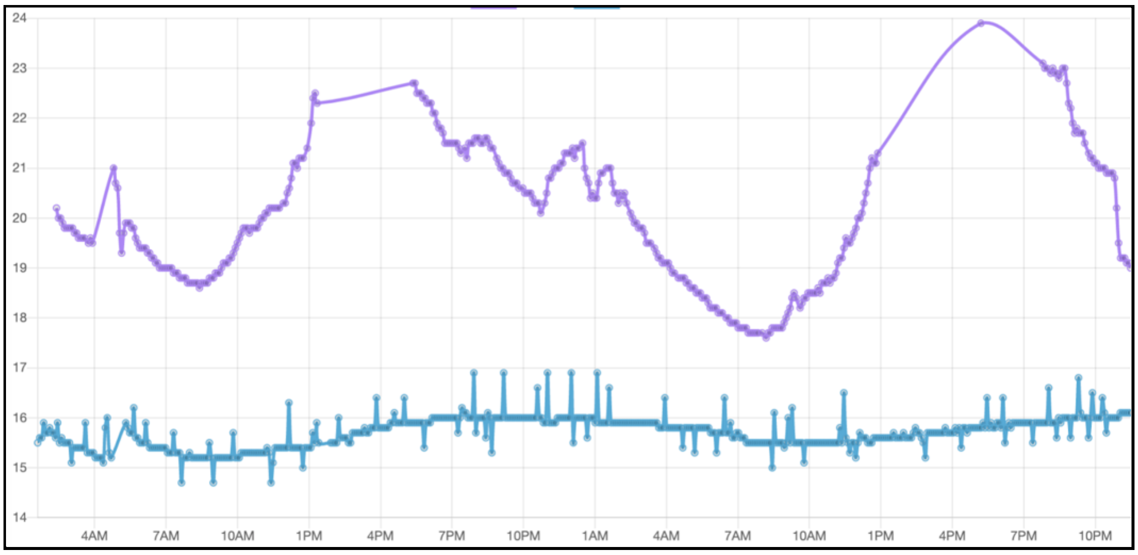


Figure 19 – Temperature over two days chart in two of the warehouse rooms.

With the collected data from the application server (section 7.4), we can now use it to improve the warehouse conditions to store the farm's goods. The total hardware costs, for the MonitPal application, are described in Table 12. It accounts for end-devices and respective sensors and batteries.

Table 12 – MonitPal applications hardware costs.

Component	unit	Unit cost	Total cost
Node MCU Amica	4	≈ 5.00 €	≈ 20.00 €
Hum/Temp sensor	4	≈ 1.00 €	≈ 4.00 €
Battery	4	≈ 3.00 €	≈ 12.00 €
Wiring and casing	4	≈ 1.00 €	≈ 4.00 €
Total	4	≈ 10.00 €	**≈ 40.00 €**

The total hardware costs for the MonitPal app were off, approximately forty euros to have four devices, one on each warehouse room according to the evaluation scenario in Figure 13.

Chapter 6 – Conclusion

By incorporating a custom LoRa implementation, we were able to reach remote areas where connectivity is an issue and gather data for multiple ends. Anyone can develop a LoRa network custom implementation. The only must is to comply with regulatory rules. In our tests, we calculated the ToA associated with our LoRa configuration parameters (table 8) and determined that our end devices could send as much as 5712 messages/day. Besides being vulnerable to many issues when regarding the custom within our network server LoRa implementation serves the needs of the proposed testing apps.

With the proposed platform, we were able to integrate two different applications with different business domains and technology requirements. We integrated a custom gateway that we developed, supporting LPWAN LoRa and Wifi. Both hardware and software, open-source goals were achieved, having a direct impact on the implementation costs, we implemented the described solution for less than two hundred euros. This was the developed platform fulfills the requirements to support long/ medium communications, have a low-cost approach and a lightweight design.

6.1 Future work

As future developments, for the proposed platfoms, we would like to implement a gateway driven QoS system that would improve downlink communications. Apply an asynchronous security pattern for the communications in the devices layer, improving security. Also very important would be to develop a tool, that allows for basic data analysis, incorporated into the default application server.

References

1. *Lueth, K. (2018). State of the IoT 2018: Number of IoT devices now at 7B – Market accelerating. Available online: https://iot-analytics.com/state-of-the-iot-update-q1-q2-2018-number-of-iot-devices-now-7b/ (accessed on 08-Dec-2018).*
2. *Arduino official organization site. Available online: https://www.arduino.org/ (accessed on 8-Dec-2018).*
3. *Beagleboard About page. Available online: http://beagleboard.org/about/(accessed on 6-Jan-2018).*
4. *Wiznet Homepage. Available online: http://wiznet.io/(accessed on 12-Jan-2018).*
5. *MQTT home page . Available online: http://mqtt.org/(accessed on 08-Jan-2018).*
6. *Congduc Pham, (Univ. Pau, LIUPPA Laboratory); Abdur Rahim , (CREATE-NET); Philippe Cousin, (Easy Global Market). Low-cost LoRa IoT platforms. "Low.cost, Long-range Open IoT for Smarter Rural African Villages".*
7. *KAA documentation. Available online: http://kaaproject.github.io/kaa/docs/v0.10.0/Welcome/ (accessed on 5-Feb-2018).*
8. *Sitewhere documentation. Available online: https://sitewhere1.sitewhere.io/ (accessed on 5-Feb-2018).*
9. *Thingspeak documentation. Available online: https://www.mathworks.com/help/thingspeak/ (accessed on 5-Feb-2018).*
10. *Devicehive documentation. Available online: https://docs.devicehive.com/docs (accessed on 5-Feb-2018).*
11. *Zetta documentation. Available online: https://github.com/zettajs/zetta/wiki (accessed on 5-Feb-2018).*
12. *IoT-DSA documentation. Available online: https://github.com/IOT-DSA/docs/wiki (accessed on 5-Feb-2018).*
13. *Thingsboard.io documentation. Available online: https://thingsboard.io/docs/ (accessed on 5-Feb-2018).*
14. *Thinger.io documentation. Available online: http://docs.thinger.io/ (accessed on 5-Feb-2018).*
15. *WSo2 documentation. Available online: https://wso2.com/wso2-documentation (accessed on 5-Feb-2018).*

16. *Mainflux documentation. Available online: https://mainflux.readthedocs.io/ (accessed on 5-Feb-2018).*
17. *TheThingsNetwork architecture page. Available online: https://www.thethingsnetwork.org/docs/network/architecture.html (accessed on 5-Feb-2018).*
18. *Lora-server github page. Available online: https://github.com/brocaar/loraserver (accessed on 5-Feb-2018).*
19. *Lora-adapter github page. Available online: https://github.com/mainflux/mainflux/ (accessed on 5-Feb-2018).*
20. *KAA gateway support. Available online: https://www.kaaproject.org/platform (accessed on 5-Feb-2018).*
21. *SEMTECH, A., & Basics, M. (2015). AN1200. 22. LoRa Modulation Basics, 46.*
22. *TheThingsNetwork homepage. Available online: https://thethingsnetwork.org (accessed on 12-Feb-2018).*
23. *LoRaAlliance, "LoRaWAN specification v1.1" 2017.*
24. *Congduc Pham, (Univ. Pau, LIUPPA Laboratory); Abdur Rahim , (CREATE-NET); Philippe Cousin, (Easy Global Market). Introduction - Limit dependency to proprietary infrastructures and provide local interaction model . "Low.cost, Long-range Open IoT for Smarter Rural African Villages".*
25. *N. Naik; "Choice of effective messaging protocols for IoT systems: MQTT, CoAP, AMQP and HTTP," 2017 IEEE International Systems Engineering Symposium (ISSE), Vienna, 2017, pp. 1-7. doi: 10.1109/SysEng.2017.8088251.*
26. *Standard, O. A. S. I. S. (2014). MQTT version 3.1. 1. Available online: http://docs. oasis-open. org/mqtt/mqtt/v3, 1 (accessed on 12-Feb-2019).*
27. *MQTT broker as a module. Available online: https://www.npmjs.com/package/mosca (accessed on 15-Feb-2019).*
28. *Andrei B. B. Torres; Atslands R. Rocha; José Neuman de Souza; Grupo de Redes de Computadores, Engenharia de Software e Sistemas (GREat) Universidade Federal do Ceará (UFC) – Fortaleza – CE – Brazil. Results - "Análise de Desempenho de Brokers MQTT em Sistema de Baixo Custo".*
29. *The Things Network TTN - Duty Cycle for LoRaWAN Devices. Available online: https://www.thethingsnetwork.org/docs/lorawan/duty-cycle.html (accessed on 18-Dec-2018).*

30. *MongoDB Homepage. Available online: https://www.mongodb.com/(accessed on 10-Jan-2019).*
31. *BSON. Available online: http://bsonspec.org/ (accessed on 4-Mar-2019).*
32. *MongoDB documentation. Available online: https://docs.mongodb.com/ (accessed on 11-Nov-2018).*
33. *Paethong Pornpat, (Tokyo University of Agriculture and Technology); Mitaro Namiki, (Tokyo University of Agriculture and Technology). MongoDB Database, "Low-power Distributed NoSQL Database with Message Queue Protocol on Embedded System".*
34. *MongoDB - Model Relationships Between Documents. Available online: https://docs.mongodb.com/manual/applications/data-models/ (accessed on 18-Mar-2018).*
35. *Arduino official organization site. Available online: https://www.arduino.org/ (accessed on 19-Mar-2018).*
36. *NodeMCU official organization site. Available online: https://www.nodemcu.com (accessed on 22-Mar-2018).*
37. *Scaleway Cloud Homepage. Available online: https://www.scaleway.com/en/(accessed on 22-Mar-2018).*
38. *Arduino-Lora. Available online: https://github.com/sandeepmistry/arduino-LoRa (accessed on 16-Mar-2018).*
39. *Nuno Gil Polónia Manita Nico (Instituto Superior Técnico); Development of the LoRa Gateway. "Development of Low-cost LoRaWAN Gateway for Private Deployments".*