UNIVERSIDAD DE EXTREMADURA

FACULTAD DE CIENCIAS

Grado en Física

TRABAJO DE FIN DE GRADO

Developement of a FIWARE-based application for tree species monitoring (dendrometry)

Javier Fernández Aparicio

jfernandil@alumnos.unex.es

July 2020

Fernando Javier Álvarez Franco, profesor del Departamento de Ingeniería Eléctrica, Electrónica y Automática de la Universidad de Extremadura.

Informa:

Que D. Javier Fernández Aparicio ha realizado bajo su dirección el Trabajo de Fin de Grado. Considera que la memoria reúne los requisitos necesarios para su evaluación.

Badajoz, 8 de julio de 2020

Fdo. Fernando J. Álvarez Franco.

Resumen

Este documento proporciona una descripción detallada del proyecto, el cual está centrado en la investigación de posibles alternativas de bajo coste para sistemas de dendrometría inalámbricos. Actualmente existen una amplia gama de sistemas profesionales en el mercado, sin embargo debido a su elevado coste, este proyecto pretende abordar la reducción en el mismo, incrementando la versatilidad, accesibilidad y escalabilidad de este tipo de sistemas.

Para conseguir estos objetivos el proyecto se apoyará en tecnologías de hardware libre como Arduino [1] o Raspberry Pi [2], y en sistemas de software libre o código abierto, como FIWARE [3], The Things Network [4] y Python [5].

Abstract

This document gives a detailed description of this project, which is focused on researching possible low-cost alternatives for wireless dendrometry systems. Currently there are a lot of expensive and professional systems in the market, that's why this project is intended to reduce costs and increase the versatility, scalability and accessibility.

In order to reach these objetives the project will be supported with free software technologies such as FIWARE[3], The Things Network[4] and Python[5] or free hardware systems such as Arduino[1] and Raspberry Pi[2].

1 Introduction

This project arises itself from a direct interaction with professionals inside forestry sector. The original idea was to give technical coverage for particular needs that professionals in this sector had to face off with. At this point it is easy to notice this solution will need to be a distributed one, due to high samples dispersion. As we can see, there are even remote techniques to predict this sample density/dispersion using remote methods which predicts between 157-170 individuals per hectare[6] (depending on the model used).

So according to these sample size determination theories, a big size for samples and the need for a big wireless network of distributed devices is essential for a great resolution, since each device will correspond with an individual.

1.1 Internet of Things

This big wireless network of distributed devices, more or less, the definition of the IoT (Internet of Things) concept; according to the abstract in [7] IoT concept comes from an earlier one called M2M (Machine-to-Machine) communications. Although according to [7, p. 1(71)] there is not an official definition for IoT concept yet, [8, p. 2(920)] defined it as:

> "based on the traditional information carriers including the Internet, telecommunication network and so on, Internet of Things (IoT) is a network that interconnects ordinary physical objects with the identifiable addresses so that provides intelligent services."

This, at least, covers a small part of what this project is intended to do: "Interconnect ordinary physical objects with the identifiable addresses" to provide intelligent services. These physical objects are in this case ordinary dendrometers.

Over the years there have been analog and handled by hand dendrometers, thus data acquisition had to follow a manual process in the same way. This could become a time-intensive task because of the big size for this statistical population, as previously stated. Traditionally, one would need to go there and as part of the field work, take the whole sample data individual by individual.

1.2   Dendrometry, a formal definition

The GEMET (General Multilingual Environmental Thesaurus) adopts the definition for *dendrometry* from [9]:

> "The measuring of the diameter of standing trees from the ground with a dendrometer that can also be used to measure tree heights."

This one is a bit wide definition because nowadays most dendrometry researches are focused on stem diameter; however, at this point extending this project could be interesting to include also a heights measurement sensor; however, this could be an interesting research premise for future projects.

Many comercial dendrometry systems are available in the market, nevertheless more than single and manual dendrometers those are complex and professional distributed systems. Consequently, one of the most important objectives in this project is to research about the possibility to get lower the costs of the whole system, since those professional systems are still expensive. This research is intended to get a cheaper system and make it accessible to everyone who wants to monitor the growth of one or more trees.

There are quite a few types of dendrometers but according to [10] "It is possible to define two broad categories of dendrometer: those that contact the stem and those that do not". This project is focused on the former kind, so we are developing a "contact dendrometer" based on a linear potentiometer.

1.3   Objectives

As previously stated in the Abstract, the most important objective for this project is to research about the possibility to make this dendrometry system cheaper, more scalable and accessible.

Forests are remote and large surface areas which will require long-range communication technologies to be controlled. That's why LoRa is the perfect wireless technology to achieve this, and these kind of automatic systems seem to be the proper solution to this hard and slow process. Hence IoT has much to contribute to this task. Forestry professionals could take advantage of these technologies to make this in an easier, cheaper and automated way. Hence this is the main purpose of this project.

2 System Architecture

A simple block diagram is shown in figure 1, this diagram intends to emphasize the difference between two big layers, a physical layer that should be placed —IoT dendrometers at least— close to the sample, and another one called abstract layer which can be deployed locally —TTN stack is a cloud server, though.
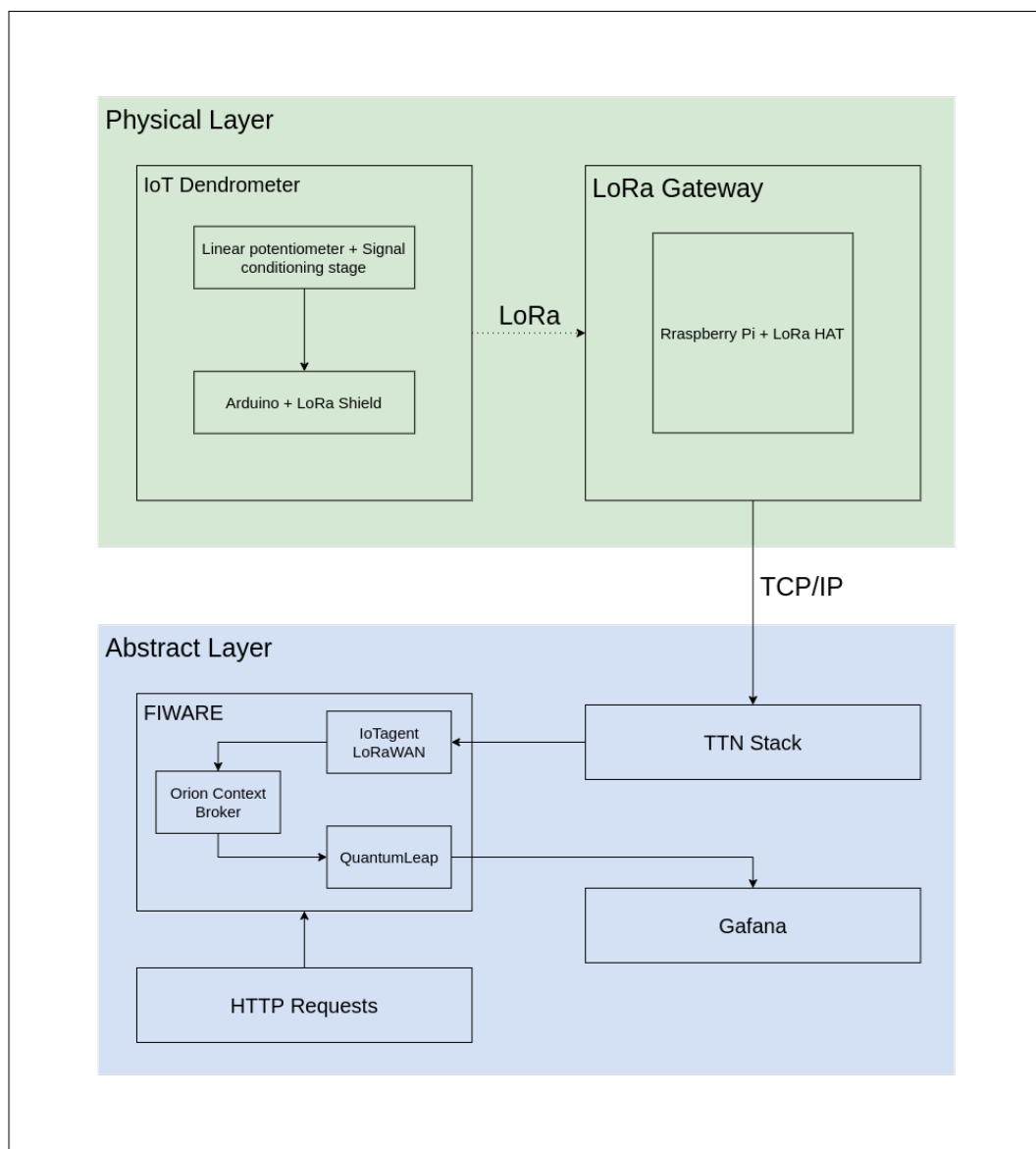

Figure 1: Block diagram overview

The following subsections will detail each of one. As we can see in figure 1, the physical layer is made up of two main modules: the first one is an Arduino microcontroller with its LoRa interface and a linear potentiometer acting as a sensor —this whole set is the IoT dendrometer— and a second one

consisting of a Raspberry Pi 3b+ with its LoRa interface to act as a gateway.

Meanwhile, the abstract layer is made up of three software modules. TTN stack is a cloud server to retrieve LoRa packets from all existent gateways, FIWARE is a local application server to gather all this context data; and finally Grafana to plot the context data.

2.1   Physical Layer

2.1.1   Arduino, multipurpose microcontroller

Justifying the use of such an interesting platform as Arduino is not difficult. The adaptability is one of its strengths, therefore; it is able to acquire and process certain data coming from a set of sensors and manage it to send this via any plugged wireless network interface.

With these constraints, an accessible and multipurpose platform to be the basis for the device design itself is needed; i.e. the core part of the dendrometer is an Arduino microcontroller.

Since the idea is to produce a low-cost device in order to distribute a high number of them, it must be a simple design; that's why it consist only of three parts,

- Linear potentiometer: which is the sensor itself due to it is directly in contact with the stem. In order to improve data acquisition it will be necessary to use a High Input Impedance Amplifier.

- Arduino microcontroller: this is the core part for the device, it will be responsible to acquire linear potentiometer data and send it to a gateway through LoRa interface.

- LoRa interface: similar to other existing wireless interfaces, it is necessary to forward the sensor data to a concentrator (gateway). Usually and due to its complexity these kind of interfaces are integrated circuits which are mounted on a PCB in order to obtain a pluggable card/shield.

2.1.2   LoRa® (Long Range)

LoRa is a "long-range, low-power, low-bitrate, wireless telecommunications system"[11]. This is why some devices inside IoT paradigm tend to be economical and low-resources devices, in order to get them distributed/scattered, as it has been pointed. So this low availability (of resources) along with their tendency to be distributed/scattered causes the need for a low-power consumption and a

long-range telecommunication.

In a more general sense, there is a wider concept to include all these kind of technologies which fullfil the IoT communication requirements, this is the "Low-Power Wide Area Networks" (LPWAN), so as claimed by [11]

> "Colloquially speaking, an LPWAN is supposed to be to the IoT what WiFi was to consumer networking: offering radio coverage over a (very) large area by way of base stations and adapting transmission rates, transmission power, modulation, duty cycles, etc., such that end-devices incur a very low energy consumption due to their being connected."

It is important to note that when talking about "low-power consumption", in many cases it is actually meaning battery-powered devices.

On the other hand, LoRa can commonly refer to two distinct layers; a physical layer (LoRa itself) and a MAC layer protocol (LoRaWAN). The physical layer, is a proprietary technology developed by Semtech, so it is not fully open. LoRaWAN, however, is a protocol built to use LoRa physical layer, it is intended for sensor networks, wherein those sensors exchange packets with some server with a "low data rate and relatively long time intervals (one transmission per hour or even days)."[11, p. 9]. This particularly means that LoRaWAN protocol is perfect for the purpose of this project.
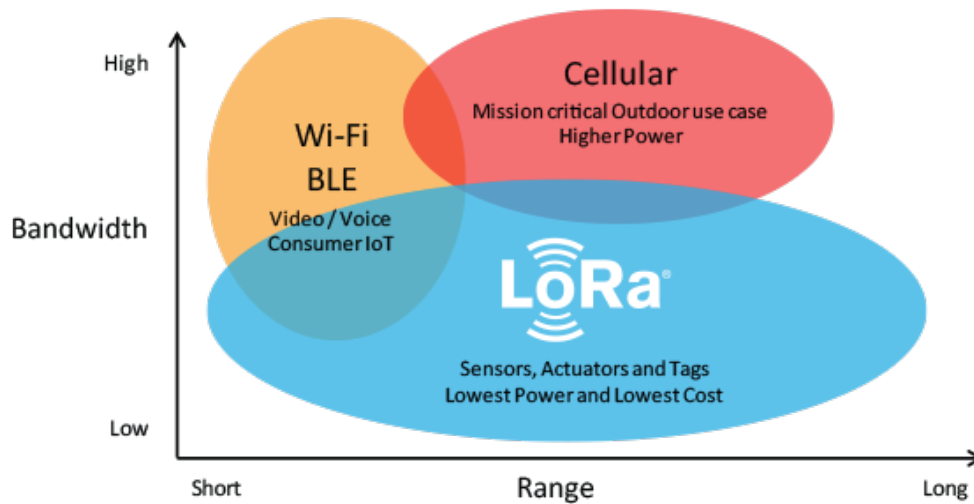

Figure 2: Bandwidth vs range plot for two of the most used telecommunication systems and LoRa.

In comparison with other common telecommunication technologies, LoRa uses lower bandwidth, but

it reaches longer distances; a comparative diagram shows different wireless technologies in figure 2.

2.1.3   Raspberry Pi, a powerful microcomputer

As in the case of Arduino, Raspberry Pi provides a powerful platform, however in the case of a Raspberry Pi this platform is slightly more complex than for an Arduino.[1] From the hardware perspective, Raspberry Pi implements a better one than Arduino, this also results on a higher cost; however, this hardware allows a Raspberry Pi to support a whole operating system; that's why these devices are usually called *single-board computers*.

A few of interesting hardware specs are for instance, a *Broadcom BCM2837B0, Cortex-A53 (ARMv8) 64-bit SoC @ 1.4GHz* as its CPU, *1GB LPDDR2 SDRAM, Gigabit Ethernet over USB 2.0 (maximum throughput 300 Mbps)* or even an *Extended 40-pin GPIO header* among others.[12]

The role for Raspberry Pi in this project is to act like a gateway, receiving all the data from different nodes. So in order to perform this task correctly it is necessary to provide a LoRa interface, which is also embedded (as in Arduino) in a separate pluggable shield/card/hat.

2.1.4   Dragino shields/hats

These shields/expansion cards are necessary because those devices (Arduino and Raspberry Pi) do not have the ability to comunicate through LoRa physical layer, that's why they need a physical interface in order to manage those LoRa packets. Both shields are based on the SX1276/SX1278 transceiver. However the Raspberry Pi hat also has a L80 GPS interface (Base on MTK MT3339), while Arduino does not.

This project is using the following models:

* *LoRa GPS HAT for Raspberry Pi,*[13] which makes use of the extended 40-pin GPIO header to be plugged[2]

---

[1] There are various Arduino models, and maybe some of them could be able to support, for example, multiple kind of Real Time Operative Systems, but it is not the case of this project, wherein the Arduino role is just to act as a core for the nodes.

[2] It is important to note that this LoRa HAT is not actually designed to play a gateway role, in fact, this is considered a "Hack where a node-class radio tries to impersonate a gateway"[14], so this means this hat is designed to be a node-class radio, not a gateway.

- *LoRa Shield for Arduino*[15], which is plugged through analog and digital pins.

2.2   Abstract layer

2.2.1   The Things Network

TTN provides a backend system to route the traffic between devices and applications. It is basically a network server placed between gateways and applications servers. It is necessary because LoRaWAN is a non-IP protocol, but a MAC layer protocol and does need some form of routing and processing messages before they can reach the application side. TTN takes care of these steps.

Moreover, this project is based on FIWARE (described in the next subsection) and the LoRaWAN agent in FIWARE is developed to interact with TTN infrastructure,
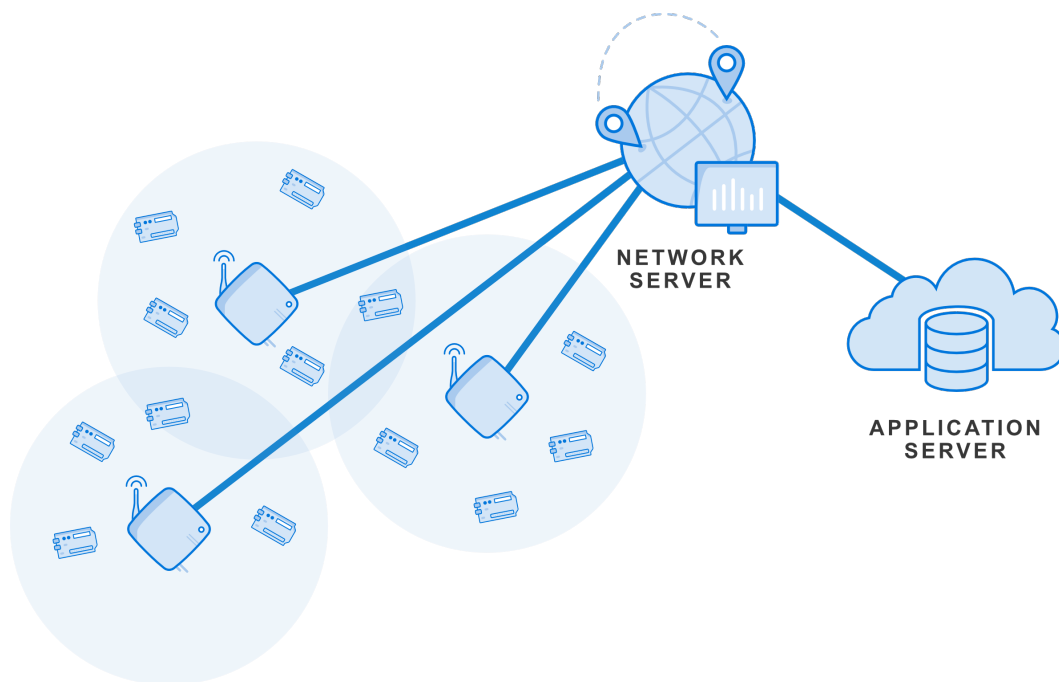

Figure 3: TTN basic diagram.

End-node devices broadcast LoRaWAN packets over LoRa radio protocol to the gateways, these gateways will forward these packets to the TTN backend, where a software module called Router will manage the gateway status and schedule transmissions. These routers are connected to other software modules called Brokers, that are responsible for mapping a device to an application —into TTN environment— allowing downlink and uplink transmissions. Once these uplink transmissions are inside TTN environment, the FIWARE LoRaWAN agent will retrieve them, however this is actually the

application server shown in figure 3. "The goal of The Things Network is to be very flexible in terms of deployment options." [16]. As explained in the docs, TTN infrastructure provides different "core functions" divided in four main groups:

- Gateway-related functions, in order to schedule and manage the gateways.

- Device-related functions, to manage the state and received context data from specific devices.

- Application-related functions, to group different devices used for similar purposes.

- Service discovery functionality, which helps components to determine where traffic should be routed to.

After this, the traffic is routed to FIWARE side, which is deployed locally and the user has total control over it.

2.2.2   FIWARE

FIWARE is defined as "The Open Source Platform for Our Smart Digital Future"[3]; in a deeper sense it is "an open source initiative defending an universal set of standards for context data management"[3], which basically means that *FIWARE* is an open source platform where IoT and smart solutions can be developed. The platform provides a number of software modules called *Generic Enablers*; among all those *Generic Enablers* there is one particularly important called *Orion Context Broker*. In fact, for a solution to be considered as *Powered by FIWARE* it must to use the *Orion Context Broker* at least.

This *Context data* is just a way to name any kind of data coming from any kind of sensor. *Orion Context Broker* is designed to manage this context data through concepts such as *subscriptions* or *entities*, to name but a few; however, all these *Generics Enablers* communicate with each other using the *FIWARE NGSI RESTful API*[17].[3]

---

[3] *RESTful* term comes from the software architectural style *REST*, which stands for *Representational State Transfer*

2.2.3   Simple Graphical Interface

Despite FIWARE is a powerful platform, we cannot consider it to be user-friendly. This is why at the end of this document —3.2.4 subsection— a simple graphical interface developed to manage end-nodes in FIWARE side is described. As it will be indicated, this graphical interface has been developed using python and its `tkinter` module intended to build graphical interfaces.

There are a lot of HTTP clients available, however the reader would still be forced to make these HTTP requests manually, and these could add complexity for a non advanced user, hence the need to develop a graphical interface.

It's not perfect, it could be improved in a future, but is by the moment an appropriate way to manage devices, entities and subscriptions in locally deployed FIWARE instance. The code of this graphical interface is available in the project repository at GitHub.

3 Design

Like in the Introduction, the most proper way to present this technical description is to differentiate between software and hardware parts. However, it is still interesting to present a simple and general diagram for the designed solution. This can be seen at figure 4.

Firstly the TTN stack send through MQTT protocol the payload to FIWARE IoTagent-LoRaWAN generic enabler. Secondly, FIWARE generic enablers process that payload and store the contained information (also called *context data*) along with a timestamp in the persisting data base (CrateDB). After this, the context data can be visualized in a web view provided by Grafena.

At this moment this solution requires to send manually the HTTP requests in order to communicate with FIWARE generic enablers manually. The ideal situation would be to develop a desktop or web application capable to provide any kind of graphical user interface to easily set up the two most important requirements to get the solution working as intended —create a new device using the IoTagent-LoRaWAN, create a subscription in Orion Context Broker which sends notifications to QuantumLeap.

3.1 Hardware

3.1.1 Nodes

Nodes are the parts of the system in direct contact with trees themselves (one node per tree). Some essential features in those devices are: low-cost, low-powered, wireless communication and small size. Figure 5 shows a diagram of its most relevant parts:

This diagram shows three different parts:

- A linear potentiometer, ideally an RS Pro Conductive Polymer Potentiometer for Automotive Applications

- A signal conditioning stage, which is needed in order to improve the quality of the voltage signal produced by the potentiometer.

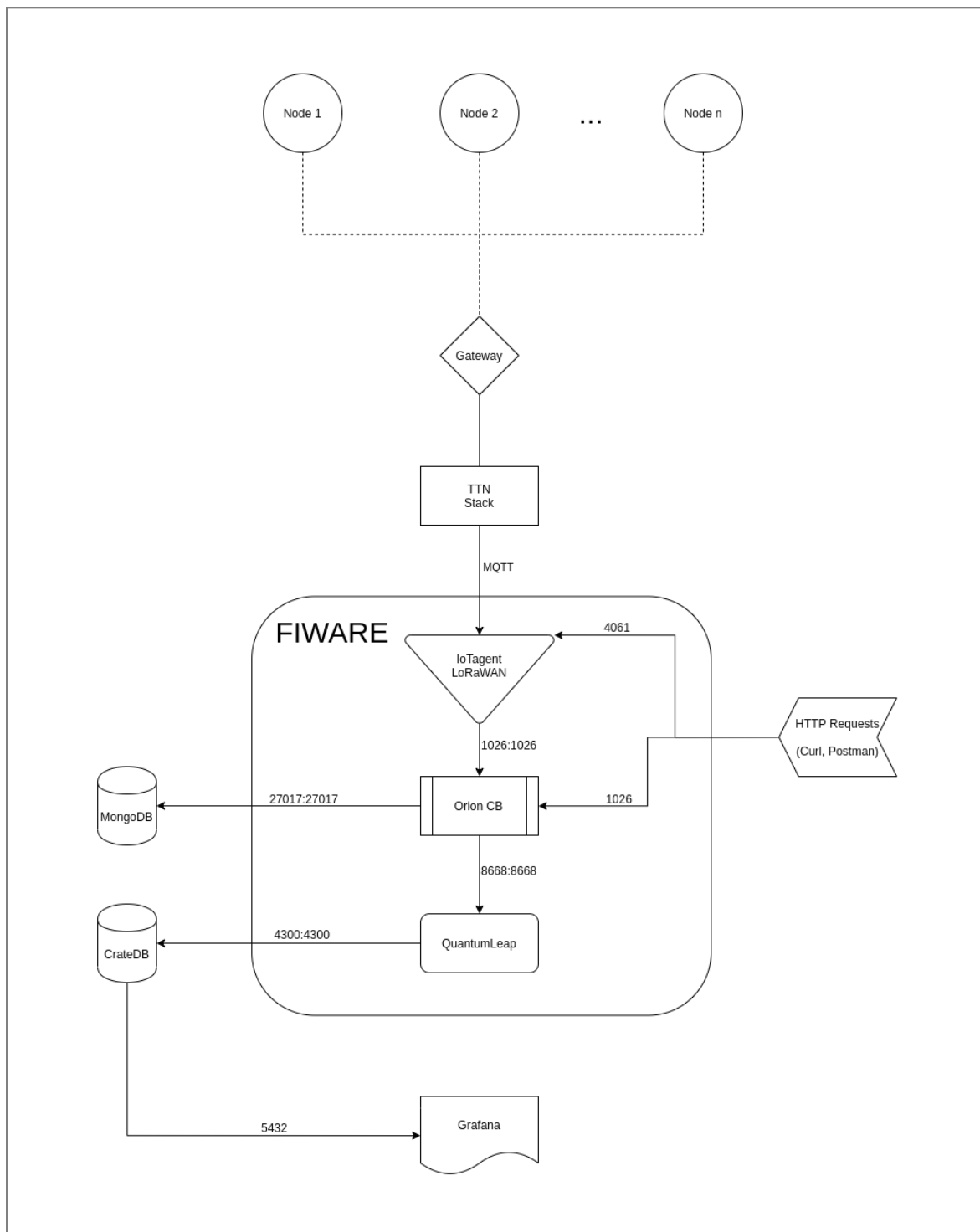- Arduino + Dragino LoRa Shield; of course it is necessary to schedule data transmission and
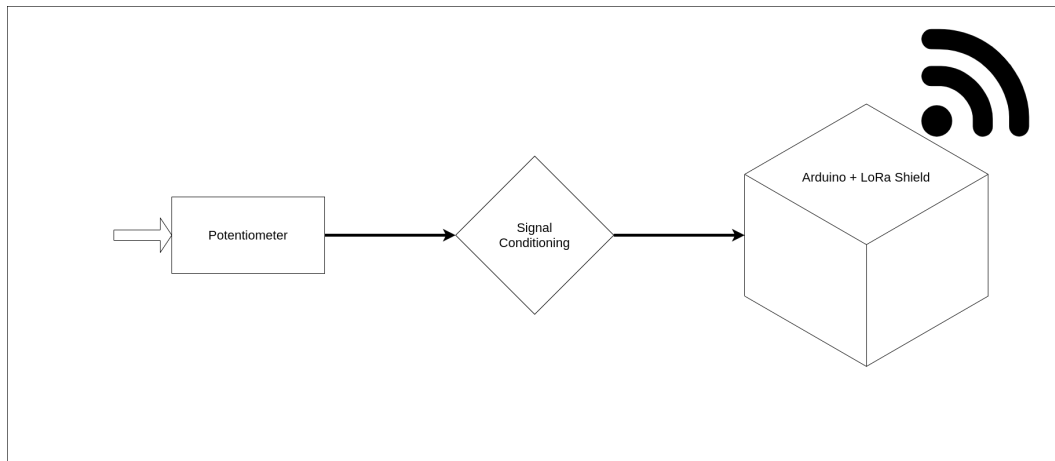
Figure 4: Solution overview.

Figure 5: Diagram for a detailed view of a node.

acquisition, as well as transmitting them using the LoRa physical layer and LoRaWAN protocol. An Arduino-like microcontroller is perfect for this purpose.

3.1.1.1   Linear Potentiometer

As said, ideally this potentiometer should be linear with 5 kΩ of maximum resistance, however, in order to improve its performance, it's recommended to use it as a voltage divider; also it would be convenient to buffer the resulting output with a high impedance amplifier, that's why the figure 5 includes a signal conditioning stage, which will be described below.

3.1.1.2   Signal conditioning

This stage is focused on improving the signal acquisition. According to this potentiometer specifications, the best way to achieve this objetive is using a high input amplifier, in fact, a single-supply, rail-to-rail operational amplifier is the best option. **Figure 6 shows a possible schematics for this stage.**

It is also recommended by the manufacturer to use the potentiometer as a voltage divider instead of a variable resistor, this is we use the Arduino power source, particularly the 5 V and GND pins.
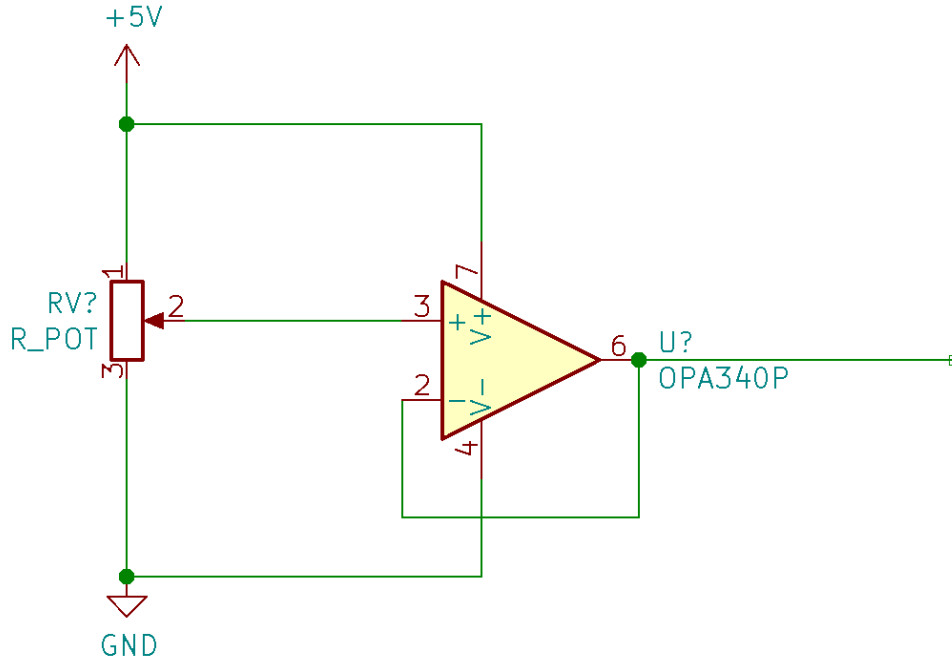
Figure 6: Signal conditioning stage.

3.1.1.3 Arduino

In a first place, the sketch can be found in `src/dendro/dendro.ino` path. This is the sketch that must be uploaded to Arduino microcontroller. MEGA 2560 in this case.

On the other hand, for this sketch to use the plugged LoRa Shield, it is required to base it on the LoRaMAC-in-C library [18] (`arduino-lmic`)[4] implemented by MCCI, whose technical documentation can be found in [19]. The `arduino-lmic` library is, as its description in the README.md says, a port from the IBM LoRaWAN MAC in C, but "slightly modified to run in the Arduino environment allowing using the SX1272, SX1276 transceivers and compatible modules" [18]. The original IBM LoRaWAN MAC in C library was developed for microcontrollers in general, their examples and HAL —Hardware Abstraction Layer or rather Hardware Annotation Library— were only for STM32 MCUs (but bare-metal, so no Arduino on top).

There is an *Arduino-like wrapper* library called `arduino-lorawan` also developed by MCCI; this library provides apparently a higher abstraction level and it could be helpful in a future to keep growing

---

[4] Dragino has its own fork of this library but has been considered less upgraded than the MCCI version, which is a fork itself from the first port of LMIC library by Matthijs Kooijman.

this project, however, for now this project is implemented using the mentioned `arduino-lmic`.

Basically the sketch is based on the ttn-abp example, as the reader can see in the first comment block; that example sends a LoRaWAN valid package with the payload "Hello World!". Besides it uses the ABP (Activation-by-personalization) method —in opposition to OTAA (Over-the-air-activation) method—, depending on the used method it is necessary to include different security keys provided by The Things Network stack (TTN stack) in the code [20]. **It is important to note that frame counters in TTN console must be reset every time the device is reset, powered off or flashed again** [21].

As commented lines 29 and 30 point, the most important thing before start working on the sketch is to configure the `arduino-lmic` library, so after installing the library with the library manager —in the Arduino IDE or any other chosen IDE, the reader will need go to the library path and locate the `<lib_path>/project_config/lmic_project_config.h` file to ensure the line 2 (`#define CFG_eu868 1`) is **uncommented** and the line 3 (`#define CFG_us915 1`) is **commented**. This is to enable specific functions in the library for Europe region (where the radio regulations are slightly different than in other regions); so this `lmic_project_config.h` must look like in the example 1

Example 1: Configuring lmic library radio.

```
1    // project-specific definitions
2    #define CFG_eu868 1
3    //#define CFG_us915 1
4    //#define CFG_au915 1
5    //#define CFG_as923 1
6    // #define LMIC_COUNTRY_CODE LMIC_COUNTRY_CODE_JP    /* for as923-JP */
7    //#define CFG_kr920 1
8    //#define CFG_in866 1
9    #define CFG_sx1276_radio 1
10   //#define LMIC_USE_INTERRUPTS
```

After this, the sketch may use specific functions to perform radio transmissions in Europe region. This in fact, will have an impact in the `void setup(){}` function —in the example sketch ttn-abp— so lines between line 232 and line 254 will be executed when the library is configured for Europe region.

Once the library has been properly configured, the second most important thing is setting the pin mapping for the chip. According to the Pin Mapping section in the `README.md`, "a variety of configurations are possible". Then, the most appropriate way to make the pin mapping is following the Dragino indications, so following the figure 7 —which can also be found in Dragino documentation.
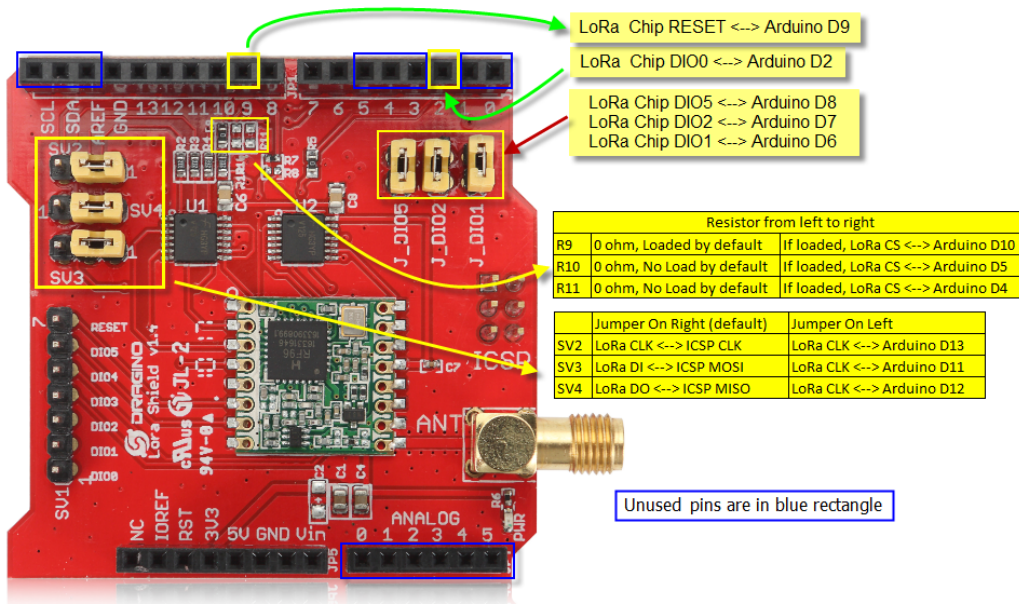
Figure 7: Arduino LoRa Shield pin mapping.

The reader can see the LoRa Chip reset pin is `D9` and as DIO pins —Digital Input/Output— the board has assigned `D2` to DIO0, `D6` to DIO1, `D7` to DIO2. Besides, a brand new board will only have loaded the `R9` resistor, this means the LoRa CS pin will be the `D10`. This leads to set the pin mapping shown in example 2

Example 2: Pin mapping for this Dragino LoRa Shield.

```
77    // Pin mapping
78    const lmic_pinmap lmic_pins = {
79      .nss = 10,
80      .rxtx = LMIC_UNUSED_PIN,
81      .rst = 9,
82      .dio = {2, 6, 7},
83    };
```

On the other hand "Any pins that are not needed should be specified as `LMIC_UNUSED_PIN`" as it can be also read in the Pin Mapping section of the library `README.md`. As the reader can see, the example 2 starts at line 76, that is because this is the line where the pin mapping is placed in the written sketch.

With regards to LoRaWAN and TTN security, as previously indicated, the chosen method is ABP, so the `NWKSKEY` (Network Session Key), `APPSKEY` (Application Session Key) and `DEVADDR` (End-Device Address) are necessary; these constants placed at lines 48, 52 and 57 must be obtained from TTN console as will be explained in The Things Network Stack subsection, so **please, refer to that**

subsection for further details. The sketch also sets analog pin `A2` to acquire the sensor signal as it can be seen in line 70.

Example 3: Variables for the potentiometer signal acquisition.

```
69    // Potentiometer pins
70    int potPin = A2;              // Potmeter pin
71    double potVal = 0;           // Potmeter value
```

Finally, the reader can find the core of this sketch inside `void do_send(osjob_t* j){}` function, at line 176; here the reader can find the following lines of code

Example 4: Preparing the payload.

```
181   // Prepare upstream data transmission.
182
183   // Read the analog value of the potmeter (0-1023)
184   potVal = analogRead(potPin);
185   potVal = 100.0 * potVal / 1023;
186
187   // Write the value to the serial monitor
188   Serial.println(potVal);
189
190   // Reset CayenneLPP buffer
191   lpp.reset();
192
193   // Add the measured value to CayenneLPP buffer
194   lpp.addAnalogInput(1, potVal);
195
196   // prepare upstream data transmission at the next possible time.
197   // transmit on port 1 (the first parameter); you can use any value from 1 to 223 (others
          are reserved).
198   // don't request an ack (the last parameter, if not zero, requests an ack from the
          network).
199   // Remember, acks consume a lot of network resources; don't ask for an ack unless you
          really need it.
200   LMIC_setTxData2(1, lpp.getBuffer(), lpp.getSize(), 0);
201   Serial.println(F("Packet queued"));
```

So, as it can be seen, in order to prepare the payload, firstly the sketch reads the analog pin and stores the read value in `potVal` variable —which is a `double` type and this makes no difference with `float` type "On the Uno and other ATMEGA based boards" but it makes "On the Arduino Due" [22]; after this, the function `lpp.reset()` is used to reset the CayenneLPP buffer, then the measure `potVal` is added to the CayenneLPP buffer in the first channel using the function `lpp.addAnalogInput()`. Finally the buffer content is prepared to be transmitted at the next possible time

using the function `LMIC_setTxData2()`.

These `lpp.*()` functions are part of the CayenneLPP library [23] —which is also dependent of ArduinoJson library [24]; in particular, the `lpp.addAnalogInput()` function encodes the raw value as a number encoded in 2 bytes (16 bits signed) with a unit of 0.01. In 16 bits signed it is possible to represent numbers from -32768 to 32767, so with a unit of 0.01, that amounts to a range from -327.68 to +327.67; that is because the sketch includes the line 185, where the raw value `potVal` is scaled to a [0, 100.0] range. This range will fit in the range that can be encoded in CayenneLPP.

CayenneLPP is so important because FIWARE will work with this data model, so any other third party encoding will not be admitted/decoded by the FIWARE LoRaWAN agent.

It is also important to note that, as we will see in 2.1.2.2 Dragino GPS and LoRa HAT subsection, the SX1276 is a single channel chip and can only listen in one channel, so despite the channels inside `void setup()` function being set up as shown in example 5, the gateway only will be able to listen to channel 0 (868.1 MHz), which is the frequency set in global_conf.json at [25]

Example 5: Channels setup.

```
251    LMIC_setupChannel(0, 868100000, DR_RANGE_MAP(DR_SF12, DR_SF7),  BAND_CENTI);  // g-band
252    LMIC_setupChannel(1, 868300000, DR_RANGE_MAP(DR_SF12, DR_SF7B), BAND_CENTI);  // g-band
253    LMIC_setupChannel(2, 868500000, DR_RANGE_MAP(DR_SF12, DR_SF7),  BAND_CENTI);  // g-band
254    LMIC_setupChannel(3, 867100000, DR_RANGE_MAP(DR_SF12, DR_SF7),  BAND_CENTI);  // g-band
255    LMIC_setupChannel(4, 867300000, DR_RANGE_MAP(DR_SF12, DR_SF7),  BAND_CENTI);  // g-band
256    LMIC_setupChannel(5, 867500000, DR_RANGE_MAP(DR_SF12, DR_SF7),  BAND_CENTI);  // g-band
257    LMIC_setupChannel(6, 867700000, DR_RANGE_MAP(DR_SF12, DR_SF7),  BAND_CENTI);  // g-band
258    LMIC_setupChannel(7, 867900000, DR_RANGE_MAP(DR_SF12, DR_SF7),  BAND_CENTI);  // g-band
259    LMIC_setupChannel(8, 868800000, DR_RANGE_MAP(DR_FSK,  DR_FSK),  BAND_MILLI);  // g2-band
```

Because of what's been explained above, the gateway will just receive one package per each ~10 minutes —at line 74 in the sketch the transmission interval is set at 60 seconds, so as there are 9 channels but the gateway is just listening to channel 0, the microcontroller is transmitting at this channel once in 9 times—

Example 6: Transmission interval.

```
73    // Schedule TX every this many seconds (might become longer due to duty
74    // cycle limitations).
75    const unsigned TX_INTERVAL = 60;
```

Of course this parameter can be adjusted, but for the final purpose of this project 60 seconds are even too many, in order to monitor changes in stem diameter of a tree, a more appropriate value would be

3600 seconds (1 hour) or even greater.

3.1.2   Gateway

3.1.2.1   Raspberry Pi 3 B+

This model of Raspberry Pi is able to run almost every GNU/Linux distribution ported to ARM architecture, Raspberry Pi OS, formerly known as Raspbian, [26] is in fact a Debian port to ARM architecture. After a little research, it is possible to conclude that there is no other operative system that improves the performance of Raspberry Pi OS in a Raspberry Pi. Due to this former argument this project is going to use Raspberry Pi OS.

One of the most interesting features of Raspberry Pi is precisely that the OS is installed and run from a microSD card, so the hardware is loading the operative system from this microSD card to the RAM directly, which improves notably the system load times, and increments its portability.

Raspberry Pi foundation provides the *Raspberry Pi Imager* to perform the installation in a microSD card.[5] This tool allows the reader to choose between three different versions of Raspberry Pi OS: Recommended, Lite and Full version. Lite version is the same than Recommended version but without graphical user interface (GUI or Desktop Environment), while Full version includes a few extra applications.

This project is using the Recommended version for Raspberry Pi OS, nevertheless, the project does not require the desktop environment at all, so to maximize the available free space in the SD card, it is better to install the Lite version instead of the Recommended version.

Even so, independently of the installed version, the reader will be able to disable the graphical environment using the console based `raspi-config` application [27].

One of the first and most important configurations is the internet access; this can be done via two different ways, using the ethernet port (which does not require any extra configuration, just to plug the cable) or using the WiFi interface, which can be configured using also `raspi-config` [28]. Due to its versatility, there are multiple setup that could fulfil the requirements of this projects:

---

[5] There are many ways to perform the Raspberry Pi OS installation in a microSD card, this document leaves it to readers to use their preferred method. However, this document also considers the *Raspberry Pi Imager* method as the best one.
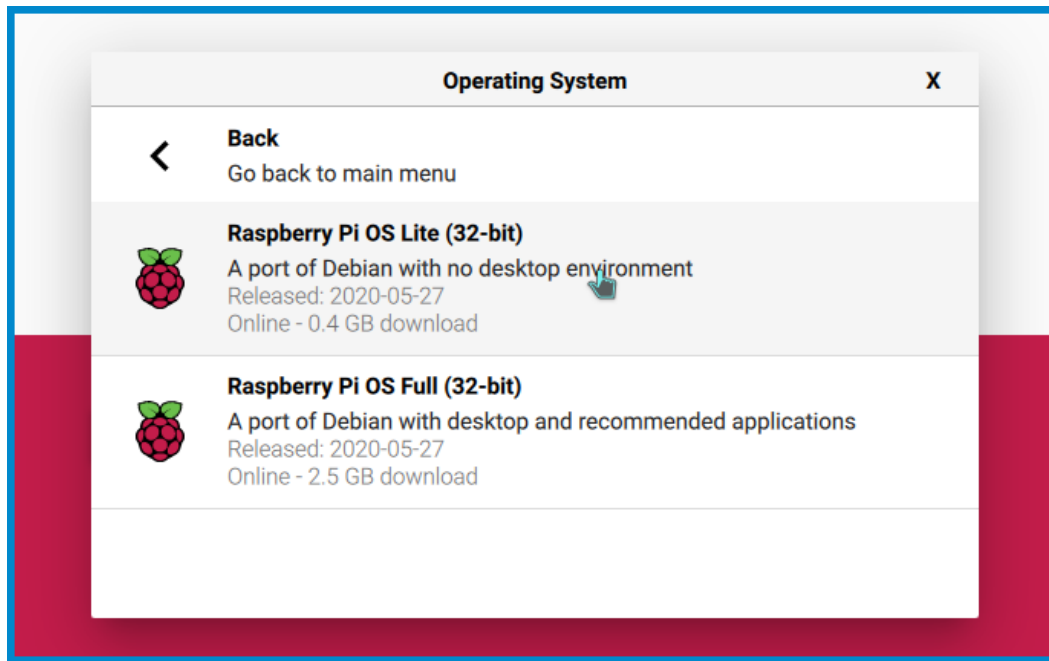
Figure 8: Raspberry Pi Imager showing different options to install
Raspberry Pi OS in a microSD card.

- **PoE** (Power-over-Ethernet); this is not actually a suitable option because Raspberry Pi does not support PoE by itself, it needs a separate HAT using the GPIO connector, so this makes it impossible to use it along the LoRa HAT. Apparently there are hacks to reduce this HAT size, but they also use pins in the GPIO connector [29]. Besides, this method only works with up to 100m cables, so it would be necessary to have a power source and internet access point within a 100m radius.[6]

- **WiFi** and battery powered; this setup implies also the existence of a relatively close power source, because the access point used to provide that wireless network connection should work also with mains power. Probably the WiFi radio is not enough to make it the difference between this setup and the next one suggested.

- Simple **ethernet** and mains power, this setup is similar to the previous one exposed, but omitting the WiFi limitations/configurations, the access point and the Raspberry Pi should be practically both at the same location.

- **GSM module** and battery or mains powered; this document considers this as the ideal setup. This setup doesn't require a conventional access point because the internet access is granted

---

[6] The LoRasPI HAT is more suitable for this option because does not cover whole GPIO connector, leaving free the GPIO pins used in [29].

through a GSM module, in a similar way than a mobile phone. This would be the ideal way because it minimizes the resources needed, which is crucial in an environment where those probably won't be available —in the middle of the forest.

Furthermore, this is apparently possible through two different options:

– Itead Raspberry Pi GSM Board (SIM800). This is not the most interesting way, because although GPIO pins comes through, this does not mean the HAT supports stacking, and even if that does, it doesn't mean every other HAT would.

– USB GSM module. This is apparently the most promising option. These kind of GSM modules are in general compliant with GNU/Linux systems and using a regular SIM card, they would be eventually able to provide an internet access point. Of course, this will also increment the size of the whole device.

It is important to note the need for a relatively stable internet connection; those gateways will forward the traffic to the cloud server, so they are receiving LoRa packets through its LoRa interface and then forwarding them to the specified server.

Another configuration which must be done is the Secure Shell (`ssh`) service; which according to Raspberry Pi documentation can be done from terminal using `systemd` [30].

So, generating a pair of keys and configuring properly the ssh access;[7] it will probably be necessary to paste the `*.pub` key content into `/.ssh/authorized_keys` file (located in the local folder in the Raspberry Pi); and give it the right permissions (`700`), after this the reader should be able to connect through ssh service.

Example 7: Creating a pair of ssh keys.

```
1    $ ssh-keygen
2    Generating public/private rsa key pair.
3    Enter file in which to save the key (/home/wyre/.ssh/id_rsa):
4    Enter passphrase (empty for no passphrase):
5    Enter same passphrase again:
6    Your identification has been saved in /home/wyre/.ssh/id_rsa
7    Your public key has been saved in /home/wyre/.ssh/id_rsa.pub
8    The key fingerprint is:
9    SHA256:vkNRk/Fo8iGGo0pYlwb2L3vf3TgOfm11MmZW+BipXgQ wyre@DESKTOP-AFG84JJ
```

---

[7] The reader will probably need to figure out the Raspberry IP in their local network, otherwise, inside a local network it will not be necessary specify any port for ssh, though it is usually the 22 by default.

```
10    The key's randomart image is:
11    +---[RSA 3072]----+
12    |  o       .o     |
13    | . o . .  +oE    |
14    |  . = o +.+... o  |
15    | o o o o.= .  = .|
16    |. . o . S..  o = |
17    | . . o ..    . O +|
18    |  . . ... .. * +.|
19    |     . ..+ o+oo  |
20    |         o.oo+o.  |
21    +----[SHA256]-----+
```

Once the remote control for Raspberry Pi through ssh is available, is possible to manage it completely from the command line, even upgrading the system and compiling the required controller to get the LoRa transceiver working properly.

At this point and depending on the type of access point used to provide internet service to the Raspberry, the reader will be able to even remote control the Raspberry over internet. For this purpose it could be useful to read the access point documentation in order to perform port forwarding or alternatively a reverse tunnel over ssh —in the case the ISP does not allow ssh when using a GSM module, also the reader may find that IPv6 works but IPv4 does not. Anyway, as said the internet access point could be variable and it is not determined by this project.[8]

3.1.2.2   Dragino GPS and LoRa HAT

Gateways don't actually need a diagram or detailed description because there are multiple devices which could play this role. These usually are generic devices due to LoRaWAN protocol versatility. LoRaWAN is a cloud-based medium access control (MAC) layer protocol, but actually acts as a network layer protocol for managing communication between gateways and nodes, similar to a routing protocol. So it is possible for any device which implements hardware for a LoRa physical layer, to act as a gateway.

Nevertheless, there are important considerations about the said above, for example, nodes are not actually associated with an specific gateway. Instead, data transmitted by a node is typically received

---

[8] The most important thing is to work comfortably with the Raspberry Pi with no HDMI, keyboard and mouse plugged, and this can be done locally in a LAN to configure it in a first place.

by multiple gateways. Each gateway will forward the received packets from the end-node to the cloud-based network server. Besides, this project is using a Raspberry Pi 3B+ with a Dragino Hat which mounts a SX1276 LoRa **transceiver**[31]; this is so important because it means that, according to *Semtech*, this transceiver is not intended to play a gateway role, but a end-node role.

A transceiver, by definition, is a device that is able to both, transmit and receive (in fact, the word itself is a mix between both, **trans**miter and re**ceiver**) that is what a node must be able to do; i.e. transmit the sensor data (context) and receive data to perform operations with its actuators.

Nevertheless, LoRaWAN specification varies from region to region "based on the different regional spectrum allocations and regulatory requirements"[32, p. 12]. In fact, for Europe, and again as reported by [32, p. 13]

> "LoRaWAN defines ten channels, eight of which are multi data rate from 250bps to 5.5 kbps, a single high data rate LoRa channel at 11kbps, and a single FSK channel at 50kbps."

Here is the important point. This Dragino HAT for the Raspberry Pi, mounts an SX1276 transceiver, which is known as node-class transceiver, so in conclusion, **this Dragino LoRa GPS HAT is LoRaWAN compatible but isn't LoRaWAN compliant**. The main reason the SX1276 transceiver is not suitable to work as a gateway is that **it is actually a single channel transceiver**. Despite all of this, there still exists the possibility of using it as a gateway, because it is technically possible.

Dragino foresees this and provides a dual channel controller [25]. The most important thing about this dual channel controller, is not the possibility to use the transceiver in a dual channel mode, but to use it as a gateway in the physical sense.

In order to get the HAT operative, the reader will need to enable the SPI (Serial Peripheral Interface). This can be done also with `raspi-config` like shown in the figure 9

Then it would be necessary to install the GPIO access library from Raspberry repositories, so performing `sudo apt install wiringpi` the package manager should install `wiringpi` providing all necessary libraries.

After this, the reader will be in position to clone the controller from the github repository [25] and compiling it, performing the command shown in example 8 —As it will be explained below, it is also
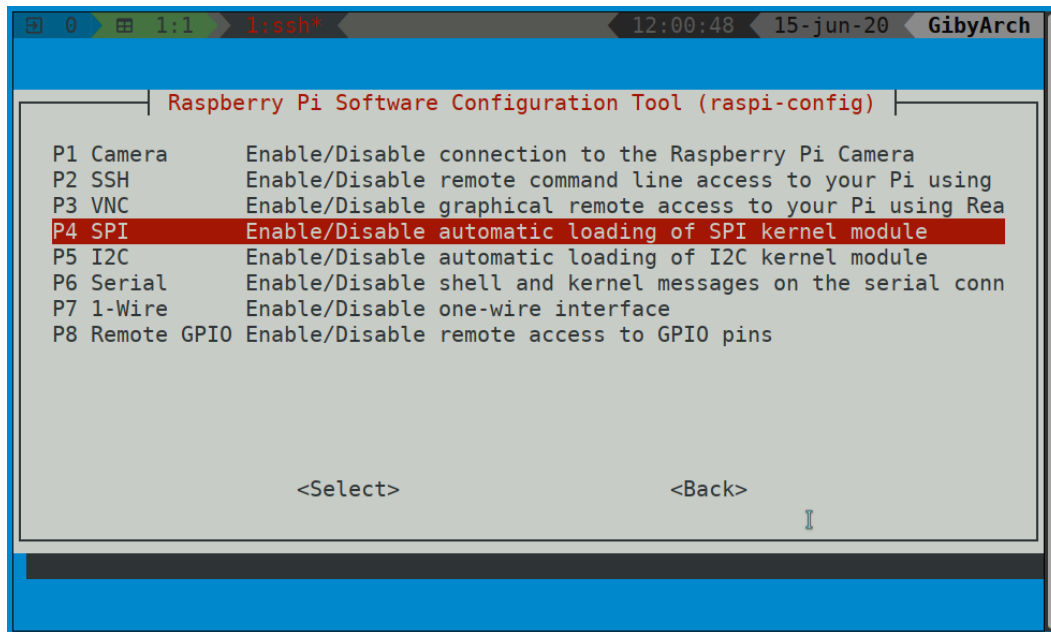
Figure 9: SSH session to enable the SPI kernel module in Raspberry Pi using `raspi-config`. Inside "5 Interfacing options"

important to note where `git` will clone the repository and where the reader will compile it, because the location of the `dual_chan_pkt_fwd` binary is critical so that the systemd service (also included in the GitHub repository) `dual_chan_pkt_fwd.service` works properly.

Example 8: Instructions to compile and install as a system service

```
1   cd ~
2   git clone https://github.com/dragino/dual_chan_pkt_fwd
3   cd dual_chan_pkt_fwd
4   make
```

Once the controller is compiled, the reader must check the contents of `global_conf.json`, because it is using the LoRa GPS HAT Single Channel LoRa [13], pins setup in `global_conf.json` must be the following as stated by [25]

Example 9: defined pins in `global_config.json`

```
1   "pin_nss": 6,
2   "pin_dio0": 7,
3   "pin_rst": 0
```

In order to obtain the gateway ID the reader will need to perform the following command to run for the first time the `dual_chan_pkt_fwd` binary, i.e. once the compilation process has produced the `dual_chan_pkt_fwd` binary it is necessary to run it directly to check the terminal output and find for the gateway ID.

Example 10: Running for the first time the LoRa HAT controller.

```
1    $ sudo ./dual_chan_pkt_fwd
2    server: .address = router.eu.staging.thethings.network; .port = 1700; .enable = 1
3    server: .address = router.eu.thethings.network; .port = 1700; .enable = 0
4    Gateway Configuration
5      your name (a@b.c)
6      Dual channel pkt forwarder
7      Latitude=0.00000000
8      Longitude=0.00000000
9      Altitude=10
10     Interface: eth0
11   Trying to detect module CE0 with NSS=6 DIO0=7 Reset=3 Led1=unused
12   SX1276 detected on CE0, starting.
13   Trying to detect module CE1 with NSS=6 DIO0=7 Reset=3 Led1=unused
14   SX1276 detected on CE1, starting.
15   Gateway ID: b8:27:eb:ff:ff:1b:14:9b
16   Listening at SF7 on 868.100000 Mhz.
17   Listening at SF7 on 868.100000 Mhz.
18   ----------------------------------
19   stat update: 2020-06-15 10:53:04 GMT no packet received yet
```

So at the line 15 in the previous example 10, the reader can see the Gateway ID, which wil be needed to connect this gateway to The Things Network stack. This ID is unique and it depends on the hardware.

After having obtained the gateway ID (`b8:27:eb:ff:ff:1b:14:9b` for this Dragino HAT), the reader can proceed then to install the system service by performing `sudo make install`. As the content of `Makefile` shows at the line 22, `sudo make install` will copy the system service `dual_chan_pkt_fwd.service` to the path `/lib/systemd/system/` and it will enable it in order to start the service —the service will start the controller as the reader can see at line 9 in `dual_chan_pkt_fwd.service`— at every system startup.[9]
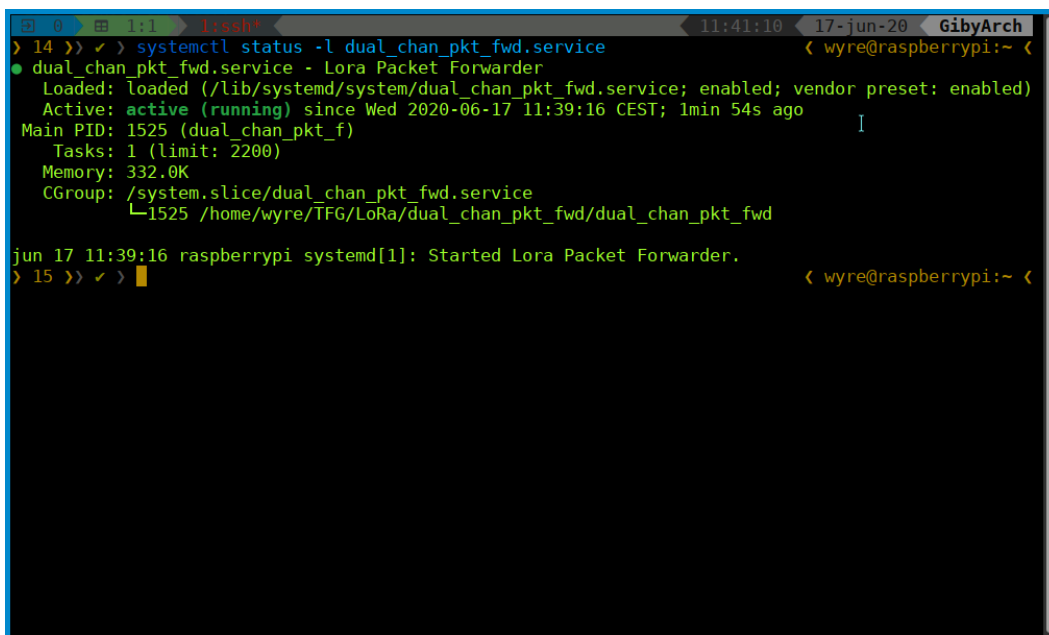
The reader must be aware of the path where the `dual_chan_pkt_fwd` binary is located, this is so important because the service `dual_chan_pkt_fwd.service` is using the absolute path to run the `dual_chan_pkt_fwd` binary; So this could result in an error if the binary is not properly located or service is not properly modified.

Once the service is installed and all these points have been checked, the reader will be able to manage the service to

---

[9] These kind of services are the way in which `systemd` manages the daemons running in background.

- start it with `sudo systemctl start dual_chan_pkt_fwd.service`

- stop it with `sudo systemctl stop dual_chan_pkt_fwd.service`

- check its status with `systemctl status -l dual_chan_pkt_fwd.service`

- check the journal with `journalctl -u dual_chan_pkt_fwd.service`

These last two instructions show information about service status and `dual_chan_pkt_fwd` status; in the picture 10 the service can be seen active (running)

Figure 10: systemd showing the `dual_chan_pkt_fwd.service` status; it is active and running.

As it can be seen, this service was modified in order to reach properly the `dual_chan_pkt_fwd` binary, which is not located in `/home/pi/dual_chan_pkt_fwd/dual_chan_pkt_fwd` (the default location defined for the binary in the GitHub repository); mainly because the user is not called `pi` anymore but `wyre`, the path needs to be modified to that path containing the binary (`/home/wyre/TFG/LoRa/dual_chan_pkt_fwd/dual_chan_pkt_fwd`)[10]

After this the Raspberry Pi is ready to forward LoRa packets to the specified server at line 33 in `global_conf.json`. Besides, the reader will be able to manage the Raspberry Pi through ssh

---

[10] This is a personal setup and is completely optional, the reader can choose their own path or even set up as suggested with `pi` user.

service, so the Raspberry only precises any kind of internet access point and electrical power supply.

3.2   Software

The core of the software part is deployed using Docker containers [33]. These docker containers are deploying essentially the following components

- **IoTagent-LoRaWAN**, which is in charge of forwarding the traffic from TTN stack to Orion Context Broker.

- **Orion Context Broker**, this is the <u>core generic enabler,</u> in fact as indicated, this is the only one required component so the solution can be considered as "powered by FIWARE". Besides, this generic enabler requires a **Mongo database** to store entities and subscriptions among other things.

- **QuantumLeap**, in order to make the context data persistent, this generic enabler is needed. Orion Context Broker works mainly with two elements; entities and subscriptions, however, for any created entity the context data does not persist in Mongo database, instead the context data will be replaced/updated; so creating a subscription which notifies to QuantumLeap, this generic enabler will store the context data in a **Crate database** to make it persistent. Also a **Redis database** is used for geocoding purposes, however this project will not deal with it —this is why of this database is not even being included in the figure 4.

- **Grafana**, this container is intended to provide a web interface to visualize and manage the retrieved context data. It stores its configuration and customization values in an embebed sqlite3 database; so there is not need for a separate container.

The terms above highlighted in bold are the deployed containers that will be detailed later, many of them are not even part of FIWARE platform, though. First this document will expose how to set up the gateway, create an application and add devices inside The Things Network stack.

3.2.1   The Things Network Stack

This is a platform for the LoRaWAN protocol where it is possible to manage gateways, devices and applications. It provides a centralized way to manage these kind of things regarding security too, generating several keys to tie the end-devices to the chosen server cloud solution (FIWARE in this

case). There are three fundamental processes the reader must follow to include nodes and gateways
in the platform, those are luckily pretty well documented.

- **Register a gateway**; this is the process where `Gateway ID` shown in example 10 is needed;
  this process is, as indicated, well documented in [34].

- **Add an application**; this application will provide a set of security keys as well as a zone where
  to manage different devices (these are basically the end-nodes i.e. the arduino microcontrollers).
  It is also documented in [35].

- **Register a device** within added application; these devices will be ultimately the microcon-
  trollers. Again the reader can find the relative documents in [36].

Once these steps are completed, the TTN web interface will provide an overview with all data regard-
ing the device like figure 11 shows.
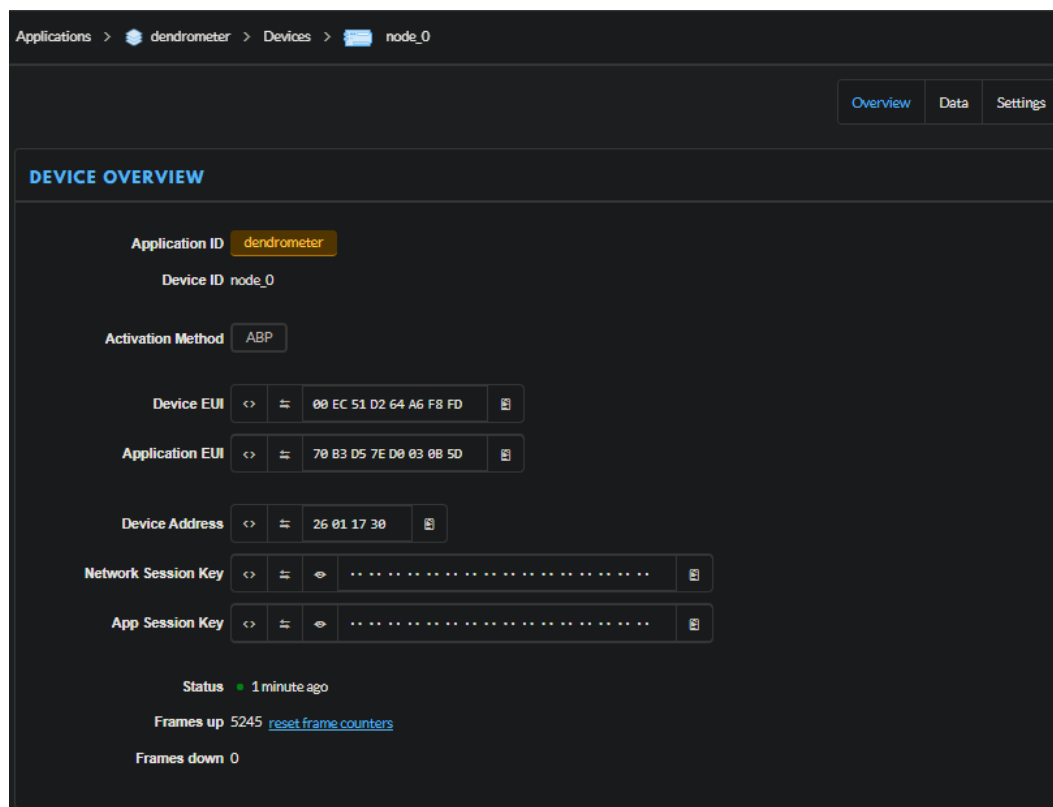
Figure 11: A device overview.

Another important consideration inside TTN platform is to check which decoder is chosen for the
received payloads, this can be checked by clicking on `Payload Formats` tab inside Application

Overview. As it can be seen in figure 12, it is so important to choose the `CayenneLPP` decoder, because as indicated, <u>the payload is being encoded in the end-nodes using the CayenneLPP standard</u>.
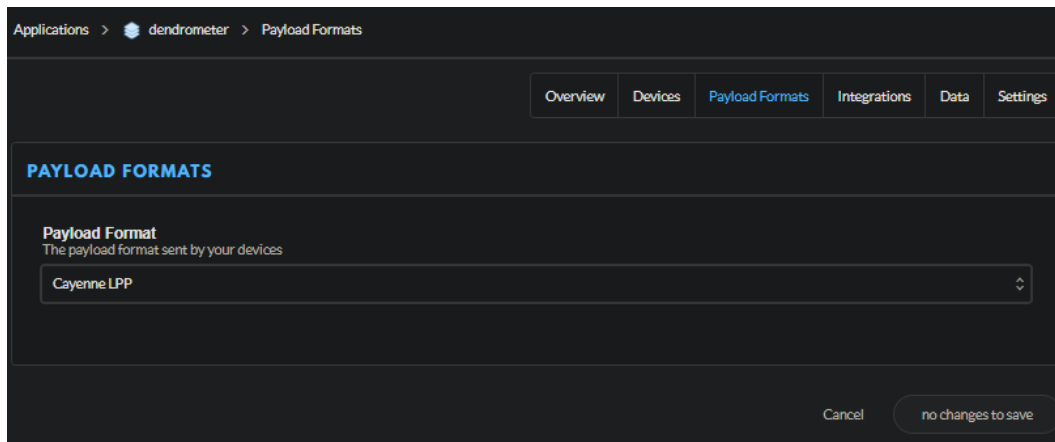
Figure 12: Choosing CayenneLPP as payload decoder

Once this has been done, the microcontrollers have also been programmed and the gateway configured, the reader should be able to read the sensor data in the `Data` tab like figure 13

Figure 13: Following sensors data in TTN.

Then it should be possible to retrieve this data using the FIWARE IoTagent-LoRaWAN. The setup of this different FIWARE generic enablers will be explained in the next subsection, but essentially this IoTagent will use, as indicated previously, all those application and devices identifiers in order to manage the incoming context data from different microcontrollers (also called devices or end-nodes).

3.2.2 FIWARE

As shown in figure 4, this solution is being implemented using particularly three FIWARE generic enablers, the most important one, of course, **Orion Context Broker**, also a LoRaWAN agent called **IoTagent-LoRaWAN** and finally to make data persistent, **QuantumLeap**.

The workflow is like picture 4 shows, first the IoTagent receives the context data from TTN, this IoTagent sends this context data to the Orion Context Broker and this latter notifies QuantumLeap the variations on the context data, thus QuantumLeap can proceed to store them into Crate database.

However this process does not happen out of the box, i.e. it is necessary to properly set up these generic enablers to get this workflow. This can be done following the FIWARE API documentation. These configuration can be donde using HTTP requests due to FIWARE essentially provides a RESTful API [37], these HTTP requests can be performed using multiple clients like `curl` or `Postman` among others. This project in fact also provides a `Postman` collection and environment that can be imported to manage them using a confortable graphical interface.

Anyway and after of this little introduction, the most important thing is to deploy all FIWARE generic enablers and its dependencies in a secure and stable way. This is why this project uses `docker` and `docker-compose` in order to ensure these modules deployment and execution. Again this project provides the `docker-compose.yml` file to ensure a correct deployment.

Figure 14: `tmux` showing docker containers status and logs.

So cloning the project and navigating to `src/fiware/` folder in the terminal the reader will must

perform `docker-compose up -d` to deploy all containers and keep detached from the logs output. Once this has been done, performing `docker ps -a` will show the status of deployed containers, as figure 14 shows. In a similar way, to check the logs the reader could perform `docker-compose logs -f <NAME>` where the argument `<NAME>` is optional and when is not provided `docker-compose` will show logs for all containers; this is also shown in picture 14.

When these containers are properly deployed, it should be possible to send the HTTP requests, however, these must follow a precise order;

- In a first place the reader will need to provision a new device for the **IoTagent-LoRaWAN**, this can be done performing a `POST` HTTP request whose body will include the necessary data to retrieve the TTN stack context data.

Example 11: Device provisioning for IoTagent-LoRaWAN

```
1    $ curl localhost:4061/iot/devices -s -S -H 'Content-Type: application/json' -H '
         Accept: application/json' -H 'fiware-service: atosioe' -H 'fiware-servicepath:
         /lorattn' -d @- <<EOF
2    {
3      "devices": [
4        {
5          "device_id": "node_0",
6          "entity_name": "LORA-N-0",
7          "entity_type": "LoraDevice",
8          "timezone": "Europe/Madrid",
9          "attributes": [
10           {
11             "object_id": "analog_in_1",
12             "name": "analog_in_1",
13             "type": "Number"
14           }
15         ],
16         "internal_attributes": {
17           "lorawan": {
18             "application_server": {
19               "host": "eu.thethings.network",
20               "username": "dendrometer",
21               "password": "ttn-account-v2.173lH8wwDiIRC8E2JgM9ScXyuNRlPpMefpazS0
                   TIhnU",
22               "provider": "TTN"
23             },
24             "dev_eui": "00EC51D264A6F8FD",
25             "app_eui": "70B3D57ED0030B5D",
26             "application_id": "dendrometer",
27             "application_key": "3736B21218C4443F639FAF4114B723A1",
28             "data_model": "cayennelpp"
```

```
29              }
30            }
31          }
32        ]
33      }
34      EOF
```

As example 11 shows, is mandatory to provide a `"device_id"`, which would be desirable to match with de TTN device ID, an `"entity_name"` in order to make Orion Context Broker create that entity and also an `"username"` and `"application_id"` that must match with the application ID in TTN. Besides It will be needed the `"password"`, which can be obtained from the Application Overview in TTN website and the `"dev_eui"`, `"app_eui"` and `"application_key"`, all of these —relative to the created device— are available into TTN dashboard. This `POST` request is included in the `Postman` collection provided by this project repository; of course the reader will need to replace the keys in the environment for their own.

When a new device is created, **IoTagent-LoRaWAN** causes that **Orion Context Broker** creates a new entity. In fact this can be checked by performing a `GET` HTTP request,

Example 12: Querying OrionCB for entities list.

```
1    $ curl localhost:1026/v2/entities \
2      -H 'fiware-service: atosioe' \
3      -H 'fiware-servicepath: /lorattn' | python -mjson.tool
4     % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
5                                     Dload  Upload   Total   Spent    Left  Speed
6   100   248  100   248    0     0   4275      0 --:--:-- --:--:-- --:--:--  4275
7   [
8       {
9           "id": "LORA-N-0",
10          "type": "LoraDevice",
11          "TimeInstant": {
12              "type": "DateTime",
13              "value": "2020-06-27T15:58:48.00Z",
14              "metadata": {}
15          },
16          "analog_in_1": {
17              "type": "Number",
18              "value": 55.03,
19              "metadata": {
20                  "TimeInstant": {
21                      "type": "DateTime",
22                      "value": "2020-06-27T15:58:48.00Z"
23                  }
24              }
```

```
25              }
26          }
27      ]
```

Example 12 shows the response of that `GET` request when a new entity is created by Orion Context Broker.

- Secondly, after have created the device and therefore the entity, the next thing to do is to create a subscription in order to Orion Context Broker notifies somewhere —QuantumLeap in this case. In a similar way, to create this subscription the reader can perform a `POST` request against Orion Context Broker, so as example 13 shows,

Example 13: Creating a subscription to notify QuantumLeap.

```
1   $ curl localhost:1026/v2/subscriptions -s -S -H 'Content-Type: application/json' -
        H 'fiware-service: atosioe' -H 'fiware-servicepath: /lorattn' -d @- <<EOF
2   {
3     "description": "A subscription to get info about LORA-N-0",
4     "subject": {
5       "entities": [
6         {
7           "id": "LORA-N-0",
8           "type": "LoraDevice"
9         }
10      ],
11      "condition": {
12        "attrs": [
13          "analog_in_1"
14        ]
15      }
16    },
17    "notification": {
18      "http": {
19        "url": "http://quantumleap:8668/v2/notify"
20      },
21      "attrs": [
22        "analog_in_1"
23      ],
24      "metadata": ["dateCreated", "dateModified"]
25    },
26    "throttling": 5
27  }
28  EOF
```

As the reader can see, for this subscription has been set up the `"url": "http://quantum`

`leap:8668/v2/notify"` i.e. QuantumLeap will be notified at any change in the entity with `"id": "LORA-N-0"`; also it's a good practice to check that all went as expected, so sending a `GET` request the response should be like example 14

Example 14: Querying for all existent subscriptions in OrionCB.

```
1   $ curl localhost:1026/v2/subscriptions \
2   -H 'fiware-service: atosioe' \
3   -H 'fiware-servicepath: /lorattn' | python -mjson.tool
4     % Total     % Received % Xferd  Average Speed   Time    Time     Time  Current
5                                     Dload  Upload   Total   Spent    Left  Speed
6   100   535  100   535    0     0   9067       0 --:--:-- --:--:-- --:--:--  9067
7   [
8       {
9           "id": "5ef772be8153f85ef485b5fd",
10          "description": "A subscription to get info about LORA-N-0",
11          "status": "active",
12          "subject": {
13              "entities": [
14                  {
15                      "id": "LORA-N-0",
16                      "type": "LoraDevice"
17                  }
18              ],
19              "condition": {
20                  "attrs": [
21                      "analog_in_1"
22                  ]
23              }
24          },
25          "notification": {
26              "timesSent": 17,
27              "lastNotification": "2020-06-27T16:40:56.00Z",
28              "attrs": [
29                  "analog_in_1"
30              ],
31              "onlyChangedAttrs": false,
32              "attrsFormat": "normalized",
33              "http": {
34                  "url": "http://quantumleap:8668/v2/notify"
35              },
36              "metadata": [
37                  "dateCreated",
38                  "dateModified"
39              ],
40              "lastSuccess": "2020-06-27T16:40:56.00Z",
41              "lastSuccessCode": 200
42          },
43          "throttling": 5
```

```
44          }
45      ]
```

Once these two steps have been performed successfully, the context data should be routed to the Crate database where will be persistent.

3.2.3 Grafana

Grafana is a powerful dashboard used at many scenarios in the software industry, it allows to monitor almost everything the reader can imagine; due to it is actually capable to acquire the data from multiple sources. In this project will be used in order to monitor the context data along the time.

This web application can be accessed via `http://localhost:3000` in the web browser —the file `docker-compose.yml` is already setup to expose that port and allow the user to connect Grafana in that way. The default credentials for the first access are `username: admin` and `password: admin`. Once the reader has accessed for a first time is possible to modify those credentials or even create new users with different privileges.
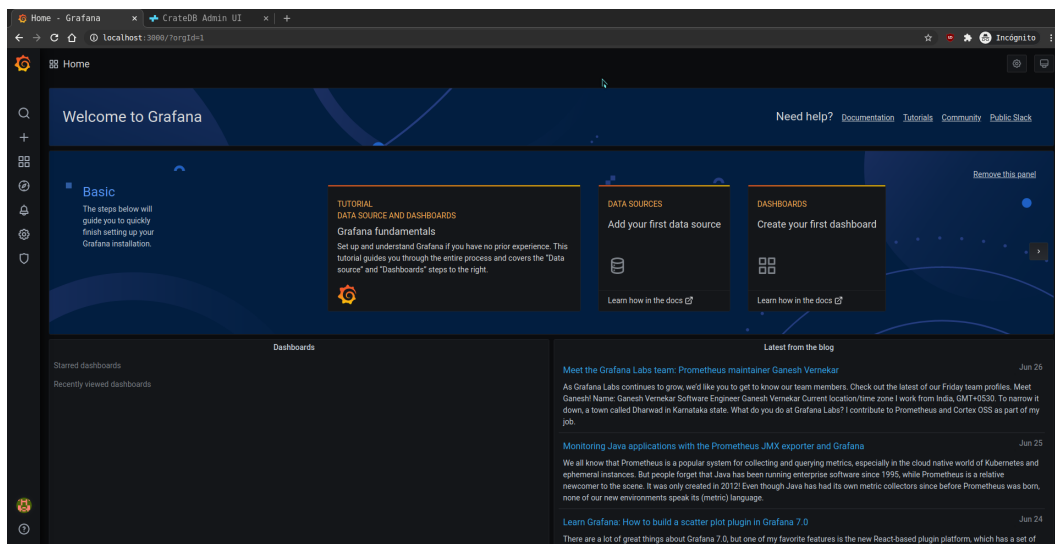

Figure 15: Grafana home view.

AS the central panel in the home view indicates, there are two fundamental task that must be performed to get a detailed view of our context data; these are to add a data source and set up a new dashboard.

To add a new data source the reader can use the left sidebar or click where says "Add your first dada source" in the home view. CrateDB is PostgreSQL compatible, so it must be chosen a PostgreSQL

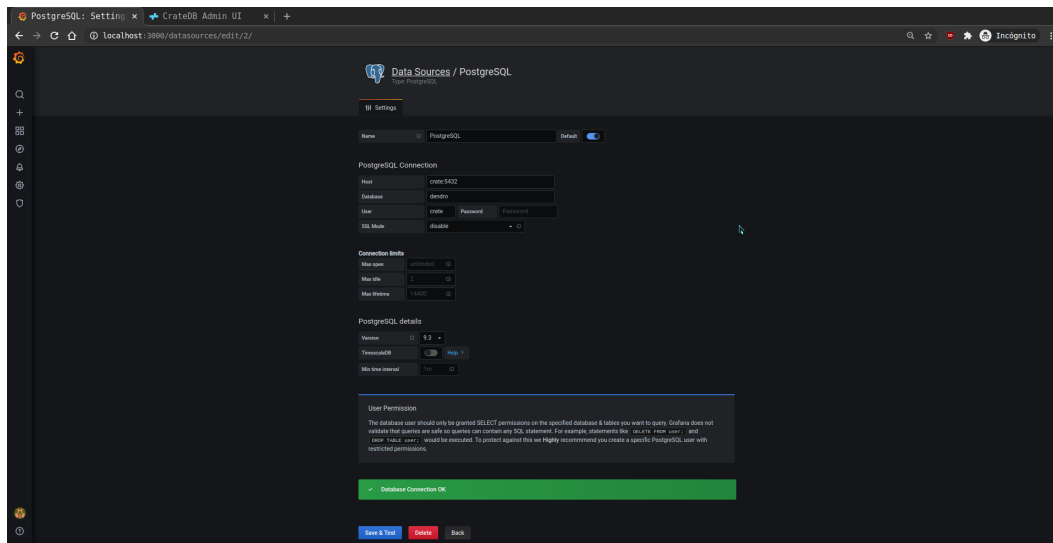database and fill the form with the data shown in figure 16

Figure 16: Adding CrateDB as the data source

"Host" must be `crate:5432` because is the port which grafana uses to queries under postgres protocol, this can be seen also in figure 14 another important credential is the "User" that must be `crate` with no password because CrateDB does not set anyone by default, "Database" can be set arbitrarily, and finally the "SSL" has to be disabled.
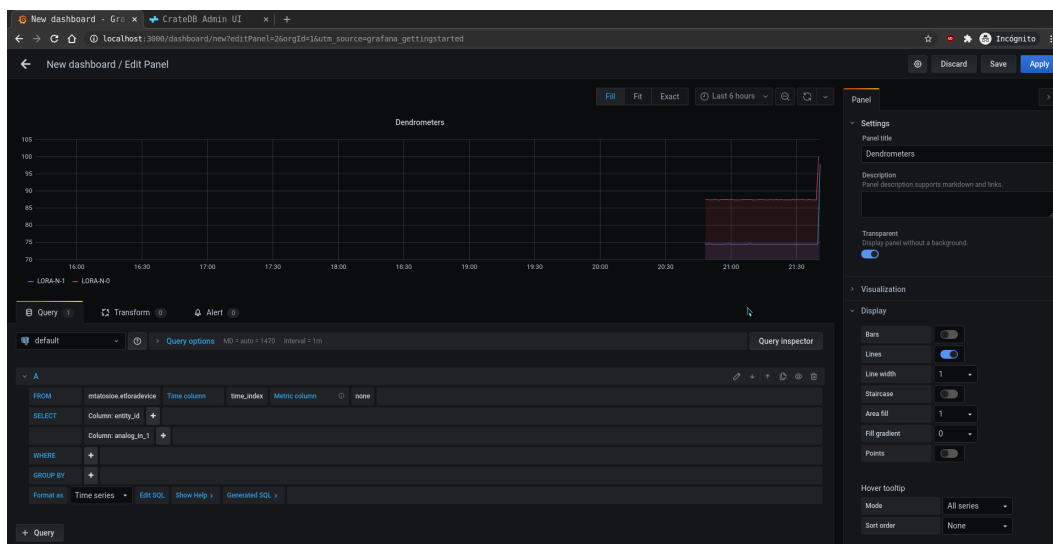

Figure 17: Setting up a panel.

To get the panel drawing the CrateDB stored context data, the reader must specify a query to retrieve the data from the database. This can be done in the "Query" tab below the graph shown in picture 17, here the user could use the graphical interface or write the query manually by clicking on the pencil

button at the right, so essentially the query should looks as follows in example 15

Example 15: Query to retrieve the stored context data.

```
1    SELECT
2    time_index AS "time",
3    entity_id,
4    analog_in_1
5    FROM mtatosioe.etloradevice
6    ORDER BY 1
```

The table name `mtatosioe.etloradevice` can be found inside CrateDB admin panel, which can be accessed in `http://localhost:4200/#!/tables` from the web browser. So after clicking on "Apply" button on top right corner the reader should have something similar to figure 18

Figure 18: Dendrometers panel

Grafana has a wide set of options intended to monitor different kind of systems, it can represent data in different type of graphics and even allows export to csv format, this can be useful for a more deep analysis using other technologies like, for instance, R, a well-known language widely used for statistical analysis.

Of course, this context data could be acquired directly from CrateDB by means of some custom application developed specifically to this task, however this is beyond of the purpose of this project. Maybe in a future a feature like this could be implemented.

3.2.4   FIWARE API Graphical User Interface

Finally, to make more friendly the FIWARE interaction exposed in FIWARE subsection, this project also provide a simple and light graphical user interface. This graphical interface has been developed using the well-known tkinter library in python, it requires also a few python modules that are specified in `/src/api-gui/pip_requirements.txt`, in fact the user should be able to install them by performing in a terminal the following command `pip install -r pip_requirements.txt`.

When the requirements are satisfied and all docker containers are running properly[11] the GUI can be run performing `./api-gui.py`; this should open the window shown by figure 19
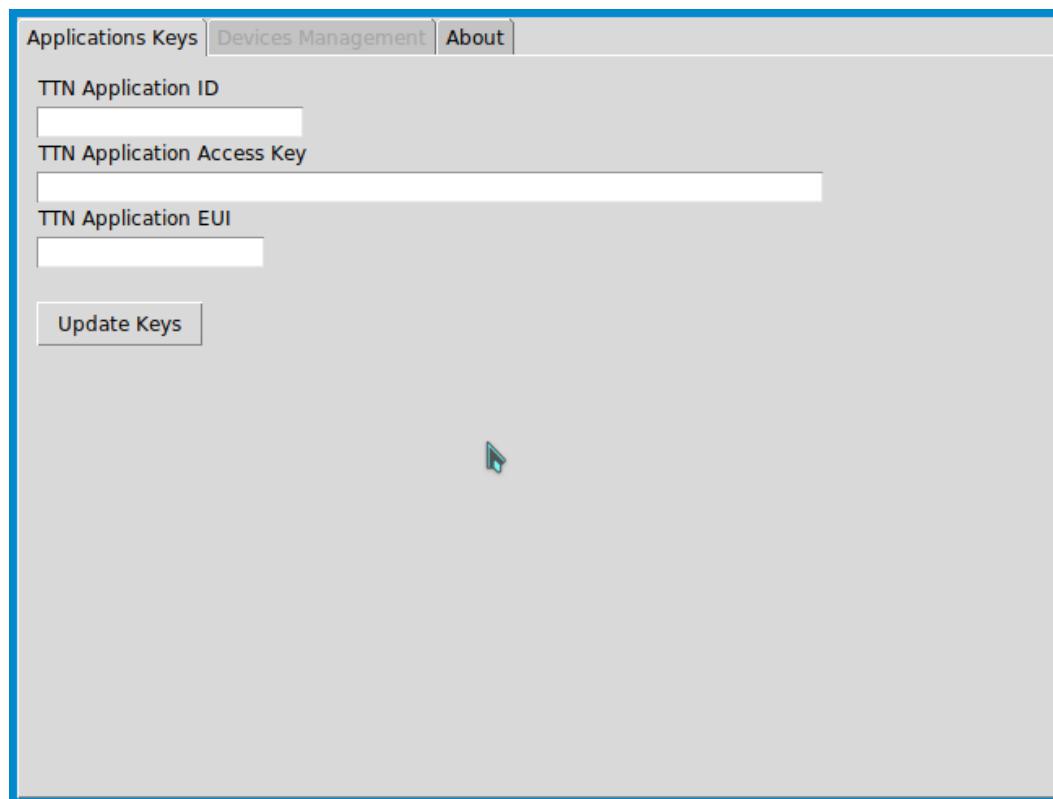

Figure 19: First view of `api-gui.py`
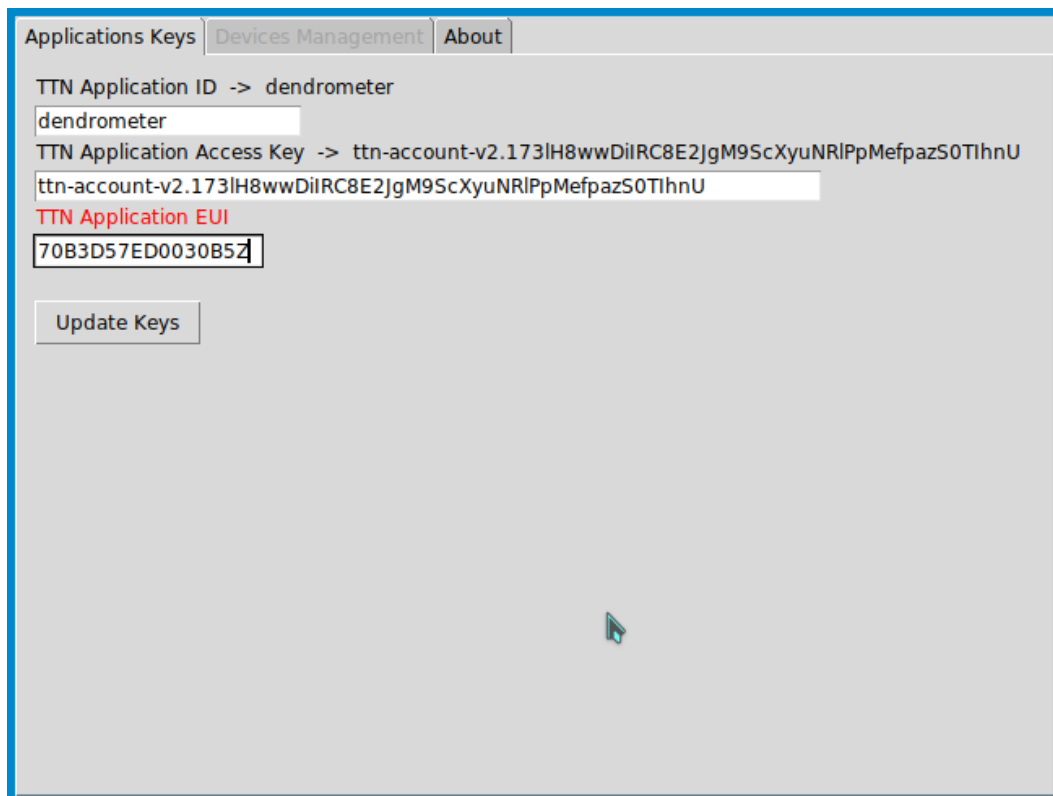
To check Orion Context Broker, IoTagent-LoRaWAN and QuantumLeap versions running in the docker containers the reader has to click on "About" tab, however, this first shown tab —"Application Keys"— is the most important one because without complete all the required info won't be enabled the "Devices Management" tab. Actually this tab performs a series of checks regarding TTN

---

[11] In a first place the GUI will retrieve the versions for Orion Context Broker, IoTagent-LoRaWAN and QuantumLeap, so if they are not running the GUI wont open.

keys. Following the same TTN requirements,

- TTN Application ID must be greater than 2 characters.

- TTN Application Access Key must starts with `ttn-account` string.

- TTN Application EUI must be 16 characters long and be hexadecimal.

When any of this requirements are not fulfilled the related label will turn red as figure 20 shows

Figure 20: TTN Application EUI must be hexadecimal

This information of course must match exactly with information provided by TTN, actually the "TTN Application Acess Key" and "TTN Application EUI" are provided by TTN, however the "TTN Application ID" must be chosen by the user when a new application is created using the TTN web interface, but this ID must be unique and it must be at least 3 characters long, that is why the developed GUI follows the same criteria.

When the keys are updated properly the labels over the text fields are also updated with the chosen values and the "Devices Management" tab is enabled. This tab provides also a more friendly interface to manage devices. In a similar way this info will must match with the info provided by TTN interface,

so that as figure 21

Figure 21: Device Management tab

This view allows the user to create and delete devices —along with its entities and subscriptions— nevertheless, the "Delete device and entities" button remains disabled until a device is chosen in the right list. In a similar way than "Applications Keys", to add a new device to FIWARE deployed instance by docker, all keys must fullfil a set of minimum requirements; as it can be seen in picture 22

- Device entity and ID must be also unique and it has to be at least 3 characters long. Moreover, this ID must match exactly with the ID chosen in TTN interface when the device is created,

- Application Session Key must be 32 characters long and also an hexadecimal string.

- Device EUI has to be 16 characters long and again an hexadecimal string.

These keys can be obtained from TTN as it can be seen in screenshot 11 —this screenshot shows info about `node_0`, though. We can see in picture 22, that there was a problem with "Device EUI", this was caused because of the final `Z` letter in the string which doesn't belong to hexadecimal characters set.

Applications Keys | Devices Management | About

Device and entity ID -> node_1

node_1

Application Session Key -> 3736B21218C4443F639FAF4114B723A1

3736B21218C4443F639FAF4114B723A1

Device EUI

00784C34E4081CDZ

Create device and entity

node_0

Delete device and entities

DEVICES   ENTITIES   SUBSCRIPTIONS
node_0   node_0   5ef8c3d02d873140493dae26
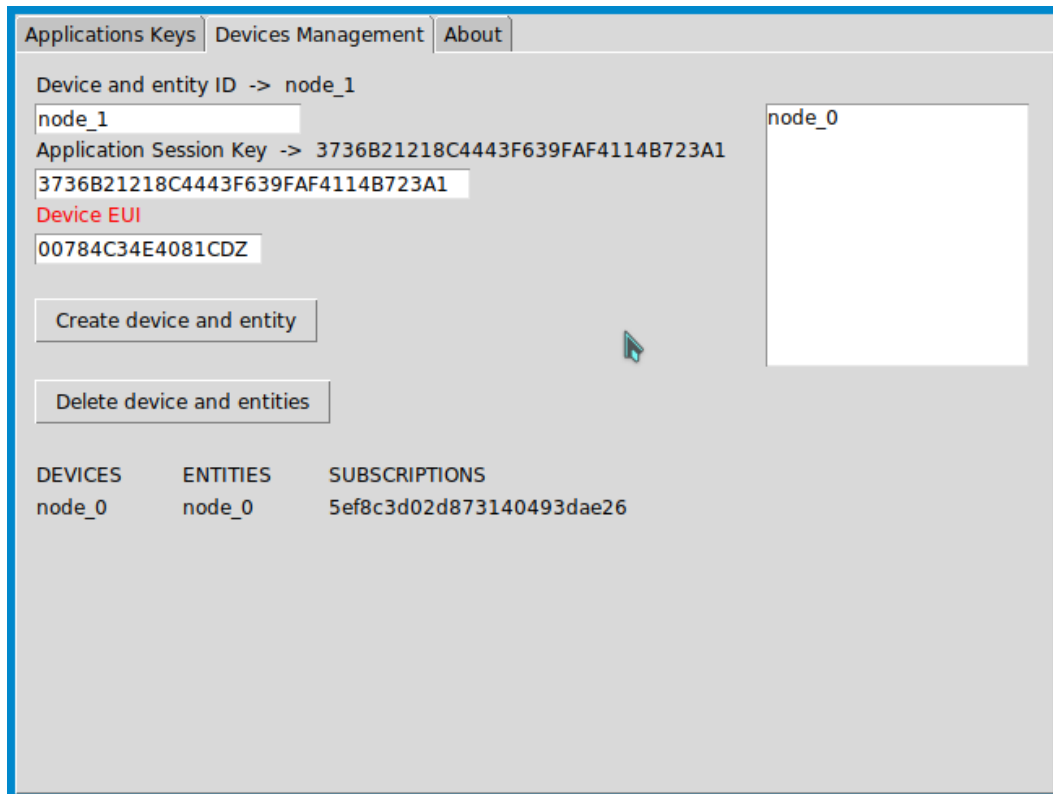
Figure 22: Trying to add a new device

Applications Keys | Devices Management | About

Orion version: 2.4.0-next
IoT version: 2.10.0-next
QuantumLeap version: 0.7.5
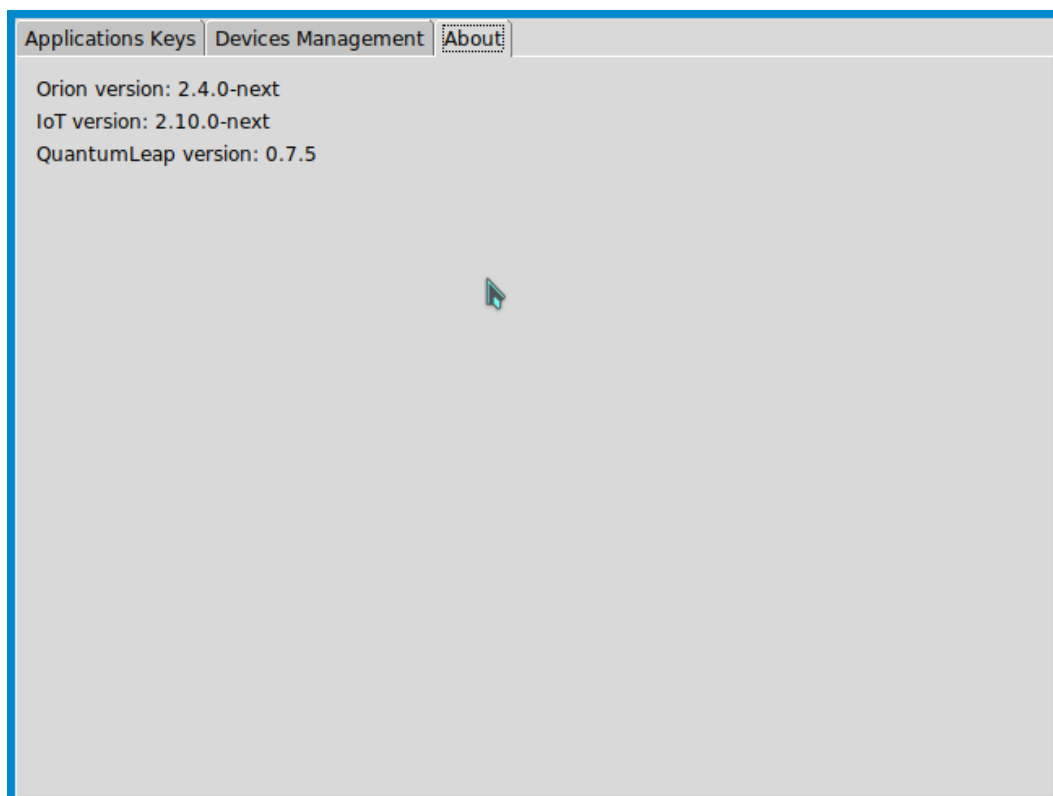
Figure 23: Running versions in About tab.

Finally, as indicated, the "About" tab shows the versions info as it can be seen in 23. The QuantumLeap version is specified in `docker-compose.yml` file as well as the CrateDB version, this is to ensure compatibility between QuantumLeap and CrateDB; however, Orion Context Broker and IoTagent-LoRaWAN versions are not specified so they should be always the latests, this is why could be helpful to retrieve these.
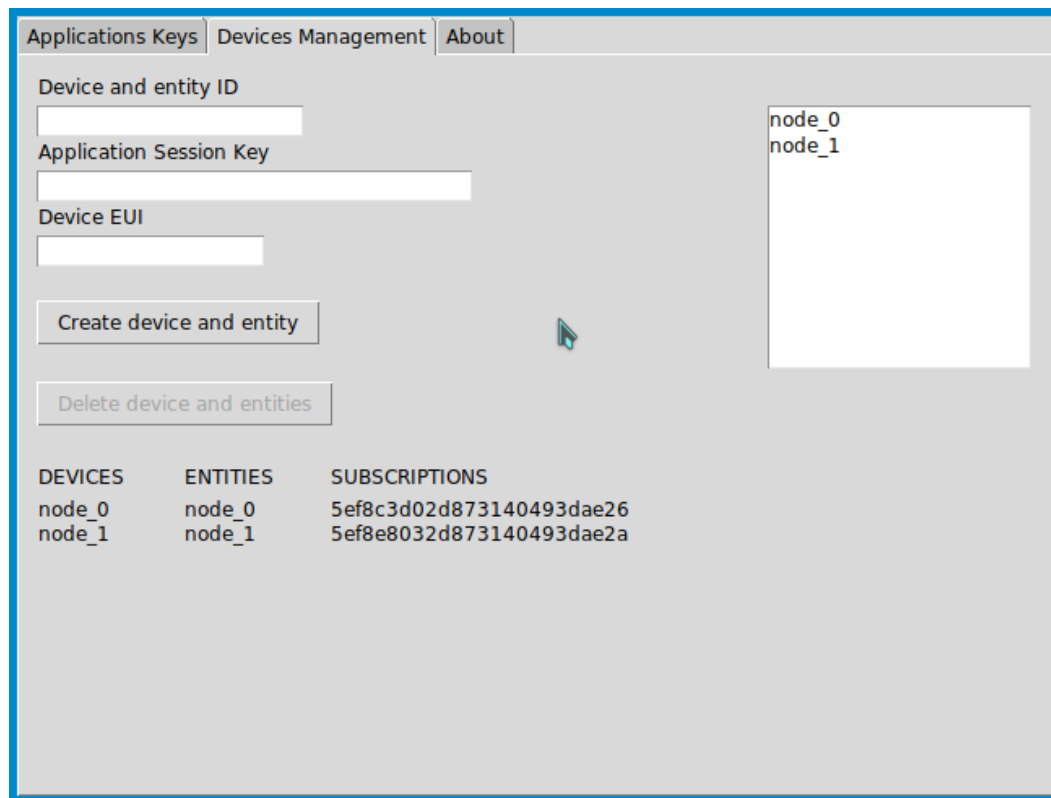
Figure 24: Graphical interface showing 2 created nodes

As figure 24 shows, once a node is created is added to the list, subscription and entity ID are useful to remove manually just a subscription or an entity by performing a manual `DELETE` request, in fact have been also included several `DELETE` requests in the attached Postman collection, however, when a node is deleted using this graphical interface button is also being removed its entity and its subscription.

4 Proof of concept, deployment

Due to circumstances at early 2020, this section has been reduced, instead of test the system in a real production environment, this section will focus on summarize needed steps/stages to deploy the system properly. Although these steps are already detailed separately, this section tries to put them together to give a consistent and practical point of view.
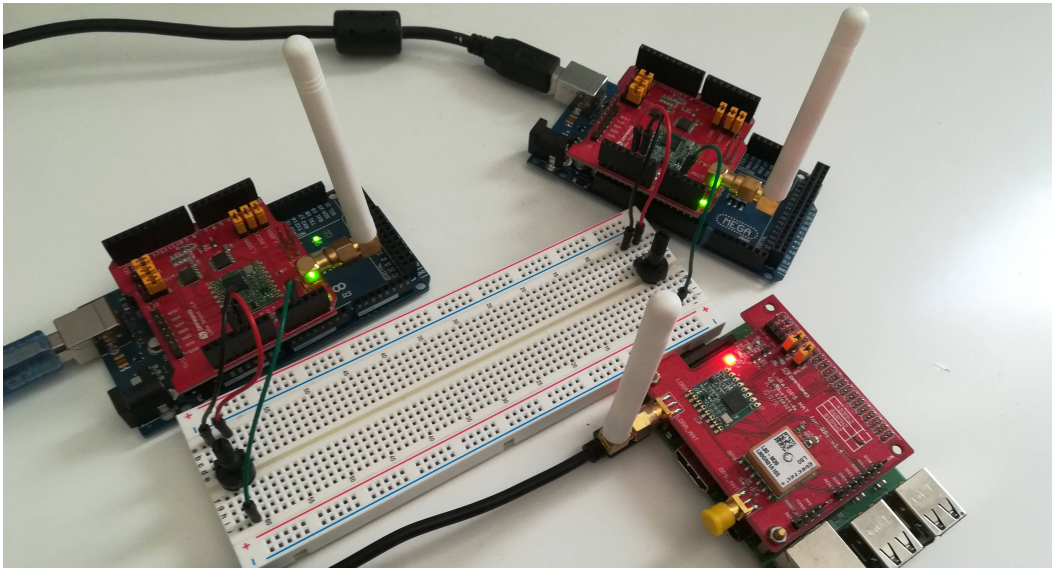
Figure 25: Two prototypes transmitting through the gateway

4.1 Set up the gateway

This process has been widely detailed in 3.1.2 subsection. The most important point on this stage is to ensure the systemd service is working properly and to retrieve the Gateway ID, which will be needed in the next stage.

4.2 TTN Stack

The first thing that must be done is to follow the three steps described in subsection 3.2.1 —Register a Gateway using its "Gateway ID", Add an application and register a device for that application; this is the first step because the reader will need the security keys to two following stages:

- Embebe these keys in Arduino sketch. This sketch will require the "Application Session Key", "Network Session Key" and "Device Address".

- Use them to create devices, and subscriptions in FIWARE. To create the device —and also its corresponding entity and subscription when using the designed graphical interface shown in 3.2.4 subsection— the IoTagent will require the "Application ID", "Application Access Key", "Device ID", "Application EUI" and again "Application Session Key".

We must remember also it is so important to choose the CayenneLPP as the payload decoder. Once these keys are available in TTN platform, the process can continue.

4.3   Customize the Arduino sketch for a specific device

As indicated, this sketch will need the "Application Session Key", the "Network Session Key" and **"Device Address"**, this latter is the most important key, because several nodes chan use the same "Application Session Key" and "Network Session Key", but "Device Address" must be unique —it is in fact also provided by TTN and the reader can check those are different for every node created in TTN platform.

Of course, to be able to compile the sketch some dependencies must be satisfied, such as CayenneLPP ArduinoJson and lmic-arduino libraries. Moreover, the **linear potentiometer has been replaced for a rotary potentiometer**, again due to circumstances in the moment when the project is developed.
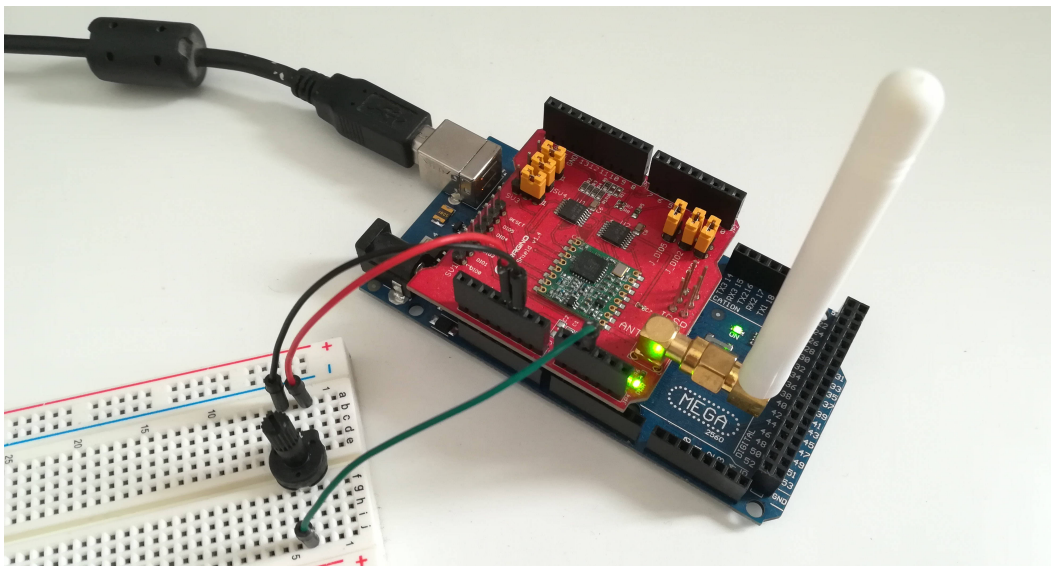

Figure 26: Prototype broadcasting LoRa packets.

It's important to note that has been chosen the analog pin 2 for analog signal acquisition. Besides the reader can check the status of the device in serial monitor, however, the reader must also note that

serial monitor is setup at 115200 bits per second.

4.4   Deploy Docker containers

Once the hardware has been properly setup, the next step is deploy locally all Docker containers, so as indicated in 3.2.2 subsection, using `docker-compose` is possible to do this easily.

This will deploy essentially the IoTagent-LoRaWAN, Orion CB, QuantumLeap and Grafana. The status of these containers can be checked by performing `docker ps -a`.

4.5   Devices, entities and subscriptions

When Docker containers are deployed, it should be possible to send HTTP requests to create a device (with its entity) and a subscription to notify QuantumLeap at any change for a particular entity. This process can be don by two different ways as indicated previously,

- Using developed graphical user interface (recommended, see 3.2.4 subsection)

- Using `curl` or Postman to perform directly the HTTP requests

4.6   Grafana

Once any device (with an entity) and particularly its subscription are properly created, CrateDB should start journaling the context data, therefore next step is to add CrateDB as a data source in Grafana.

So if the data source is properly added, it should be possible to create a new panel into a new dashboard, setting up the proper query for that panel, Grafana should start to plot the context data. (see 3.2.3 subsection for further details)

Finally the system it would be completely deployed and journaling the context data, as it can be seen in figure 18.

5 Conclusions

Finally there are a few considerations that could be done,

- A rotary potentiometer does work well enough as a proof of concept, but would have been more appropriate to have the expected linear potentiometer to try its sensibility, which is a good point in order to measure such little variations as produced in the stem diameter in a tree.

- The fact that the Raspberry Pi HAT uses a single channel chip (SX1276) has two important implications:

    - In a first place, as indicated in its corresponding section, these kind of gateways are Lo-RaWAN compatible but not compliant. So it would be desirable to replace this HAT by another one, or simple to acquire a proper gateway for a production environment.

    - Secondly, Nodes cannot comunicate simultaneously with the gateway because it is listening only one channel, so this is causing a time gap in data acquisition as it can be seen in the picture 27

Figure 27: Little gap between times caused by the single channel chip.

- End-nodes are being activated using ABP method, this method could not be as effective as OTAA method in the case that downlinks packets were transmitted, this is why due to some security implications frame counters must be reset for ABP devices every time they are shutdown

for any circumstance; in particular when downlinks are transmitted, could be necessary to reset the frame counter in the devices, so in this situation would be interesting to consider swap to OTAA method. In fact, TTN offers the possibility to reset the frame counter in TTN console but this could be removed in a future.

- Maybe the developed GUI could have handle also the context data plots, however this project couldn't have covered in anyway this development with the available time, so this is not considered as part of the project because Grafana is complete enough to this task for now and perhaps for ever due to it is extremely powerful handling databases.

- For a production environment it seems clear that arduino devices should be encapsulated and the sensor (potentiometer) part must be improved; however this is not hard because of the wide offer for cases and power supply systems, this would lead a very compact device which could be also improved adding more sensor due to versatility of CayenneLPP encoding standard which allows to send multiple measurements of different sensors.

- Finally this project is a good candidate to replace expensive commercial and proprietary systems; so it likely keeps developing, depending on the needs of the researchers.

References

[1]   Arduino Company. (2020). "Arduino,"
      [Online]. Available: `https://www.arduino.cc/` (visited on 05/2020).

[2]   Raspberry Pi Foundation. (2020). "Raspberry Pi,"
      [Online]. Available: `https://www.raspberrypi.org/` (visited on 05/2020).

[3]   FIWARE Foundation, e.V. (2020). "FIWARE Home,"
      [Online]. Available: `https://www.fiware.org/` (visited on 05/2020).

[4]   The Things Industries. (2020). "The Things Network,"
      [Online]. Available: `https://www.thethingsnetwork.org/` (visited on 05/2020).

[5]   Python Software Foundation. (2020). "Python,"
      [Online]. Available: `https://www.python.org/` (visited on 05/2020).

[6]   J. Mohammadi, S. Shataee, and M. Babanezhad, "Estimation of forest stand volume, tree density and biodiversity using Landsat ETM + Data, comparison of linear and regression tree analyses,"
      Biophysical Chemistry - BIOPHYS CHEM, vol. 7, pp. 299–304, Dec. 2011.
      DOI: `10.1016/j.proenv.2011.07.052`.

[7]   A. Abdul-Qawy, E. Magesh, and S. Tadisetty. (Dec. 2015). "The Internet of Things (IoT): An Overview,"
      [Online]. Available: `https://www.researchgate.net/publication/32383499` `6_The_Internet_of_Things_IoT_An_Overview`.

[8]   H.-D. Ma, "Internet of things: Objectives and scientific challenges,"
      J. Comput. Sci. Technol., vol. 26, pp. 919–924, Nov. 2011.
      DOI: `10.1007/s11390-011-1189-5`.

[9]   J. A. Dunster,
      Dictionary of natural resource management.
      1996,
      ISBN: 9780851991481.

[10] N. Clark, R. Wynne, and D. Schmoldt, "A review of past research on dendrometers,"
      Forest Science, vol. 46, pp. 570–576, Nov. 1999.
      DOI: `10.1016/j.dendro.2009.06.008`.

[11] Augustin, Aloÿs and Yi, Jiazi and Clausen, Thomas Heide and Townsley, William, "A Study of LoRa: Long Range & Low Power Networks for the Internet of Things,"
Sensors, vol. 16, p. 1466, Oct. 2016.
DOI: `10.3390/s16091466`.

[12] Raspberry Pi Foundation. (2020). "Raspberry Pi 3 B+ Specs,"
[Online]. Available: `https://www.raspberrypi.org/products/raspberry-pi-3-model-b-plus/` (visited on 05/2020).

[13] Dragino Technology Co., LTD. (2020). "LoRa GPS HAT for Raspberry Pi,"
[Online]. Available: `https://www.dragino.com/products/lora/item/106-lora-gps-hat.html` (visited on 06/2020).

[14] The Things Industries. (2020). "The Things Network Forum,"
[Online]. Available: `https://www.thethingsnetwork.org/forum/t/application-is-not-showing-any-data-but-gateway-shows-traffic/36855/2` (visited on 06/2020).

[15] Dragino Technology Co., LTD. (2020). "LoRa Shield for Arduino,"
[Online]. Available: `https://www.dragino.com/products/lora/item/102-lora-shield.html` (visited on 06/2020).

[16] The Things Industries. (2020). "Network Architecture,"
[Online]. Available: `https://www.thethingsnetwork.org/docs/network/architecture.html` (visited on 05/2020).

[17] FIWARE Foundation, e.V. (2020). "FIWARE NGSI RESTful API,"
[Online]. Available: `http://fiware.github.io/specifications/ngsiv2/stable/` (visited on 06/2020).

[18] MCCI Corporation. (2020). "Arduino LoRaWAN MAC in C,"
[Online]. Available: `https://github.com/mcci-catena/arduino-lmic` (visited on 05/2020).

[19] ——, (2020). "Arduino LoRaWAN MAC in C,"
[Online]. Available: `https://github.com/mcci-catena/arduino-lmic/blob/master/doc/LMIC-v3.0.99.pdf` (visited on 05/2020).

[20] The Things Industries. (2020). "LoRaWAN Security,"
[Online]. Available: `https://www.thethingsnetwork.org/docs/lorawan/security.html` (visited on 05/2020).

[21] ——, (2020). "LoRaWAN Security, Frame Counters,"

[Online]. Available: `https://www.thethingsnetwork.org/docs/lorawan/security.html#frame-counters` (visited on 05/2020).

[22] Arduino Company. (2020). "Double (Data types),"
[Online]. Available: `https://www.arduino.cc/reference/en/language/variables/data-types/double/` (visited on 05/2020).

[23] myDevices, Inc. (2020). "CayenneLPP arduino library,"
[Online]. Available: `https://github.com/ElectronicCats/CayenneLPP` (visited on 05/2020).

[24] Benoît Blanchon, among others. (2020). "ArduinoJson library,"
[Online]. Available: `https://github.com/bblanchon/ArduinoJson` (visited on 05/2020).

[25] Dragino Technology Co., LTD. (2020). "Dual Channel LoRaWAN Gateway,"
[Online]. Available: `https://github.com/dragino/dual_chan_pkt_fwd` (visited on 05/2020).

[26] Raspberry Pi Foundation. (2020). "Raspberry Pi OS,"
[Online]. Available: `https://www.raspberrypi.org/downloads/raspberry-pi-os/` (visited on 05/2020).

[27] ——, (2020). "Raspberry Pi OS,"
[Online]. Available: `https://www.raspberrypi.org/documentation/configuration/raspi-config.md` (visited on 05/2020).

[28] ——, (2020). "Setting up a wireless LAN via the command line,"
[Online]. Available: `https://www.raspberrypi.org/documentation/configuration/wireless/wireless-cli.md` (visited on 05/2020).

[29] Albert David's blog. (2020). "Poor man's PoE for Raspberry Pi$-$3/4 under $\sim$\$2,"
[Online]. Available: `http://albert-david.blogspot.com/2019/09/poor-mans-poe-for-raspberry-pi-3-under-2.html` (visited on 05/2020).

[30] Raspberry Pi Foundation. (2020). "Raspberry Pi Remote Access, Documentation,"
[Online]. Available: `https://www.raspberrypi.org/documentation/remote-access/ssh/` (visited on 05/2020).

[31] S. Corporation. (2020). "Semtech SX1276, 137 MHz to 1020 MHz Long Range Low Power Transceiver,"
[Online]. Available: `https://www.semtech.com/products/wireless-rf/lora-transceivers/sx1276` (visited on 06/2020).

[32] LoRa Alliance. (2020). "LoRaWAN™, What is it?"
      [Online]. Available: `https://lora-alliance.org/sites/default/files/2018-04/what-is-lorawan.pdf` (visited on 06/2020).

[33] Docker, Inc. (2020). "What is a Docker container?"
      [Online]. Available: `https://www.docker.com/resources/what-container` (visited on 05/2020).

[34] The Things Industries. (2020). "Gateway Registration,"
      [Online]. Available: `https://www.thethingsnetwork.org/docs/gateways/registration.html` (visited on 05/2020).

[35] ——, (2020). "Add an Application,"
      [Online]. Available: `https://www.thethingsnetwork.org/docs/gateways/registration.html` (visited on 05/2020).

[36] ——, (2020). "Device Registration,"
      [Online]. Available: `https://www.thethingsnetwork.org/docs/devices/registration.html` (visited on 05/2020).

[37] FIWARE Foundation e.V. (2020). "FIWARE-NGSI v2 Specification,"
      [Online]. Available: `http://fiware.github.io/specifications/ngsiv2/stable/` (visited on 05/2020).