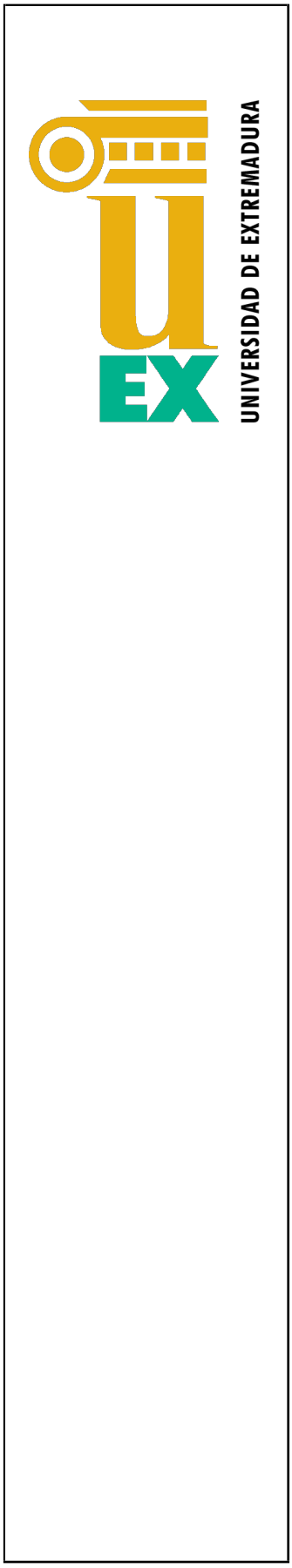

University of Extremadura

Faculty of Science

Physics degree

Degree Final Project

Developement of a FIWARE-based application

for tree species monitoring (dendrometry)

Javier Fernández Aparicio

jfernandil@alumnos.unex.es

July 2020

Abstract

This document gives a detailed description of this project, which is focused on researching possibles low-cost alternatives for wireless dendrometry systems. Currently there exist a lot of expensive and professional systems in the market, that's because 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 over free software such as FIWARE[1] or free hardware such as Arduino[2] and Raspberry Pi[3].

1 Introduction

This project arises itself from a direct interaction with professionals inside forestal sector. The original idea was to give technical coverage for particular necessities which professionals in this sector had to face off with. At this point is easy to notice this solution will need to be a distributed solution, due high samples dispersion. As indicated, there are even remote techniques to predict this sample density/dispersion using remote methods which predicts **between** 157-170 **indviduals per hectare**[4] (depending on the used model). So according to this and sample size determination theories, to get a great resolution could be necessary a big size for samples and the necessity of a big wireless network of distributed devices, since each device will correspond with an individual.

This is more or less, the definition of the IoT (Internet of Things) concept; according to the abstract in [5] IoT concept comes from an earlier concept called M2M (Machine-to-Machine) communications. However, also according to [5, p. 1(71)] there is not an official definition for IoT concept, but according to [6, p. 2(920)]

> "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 little part what this project is intended to do: "Interconnect ordinary physical objects with the identifiable adresses" to provide intelligent services. These physical objects are in this case ordinary dendrometers.

Over the years there have existed analog and manual dendrometers, thus data acquisition had to follow a manual process in the same way. This could turn out bothering because the big size for this statistical population, as previously stated. So it was traditionally necessary to go there and as part of the field work, take individual by individual the whole sample data.

1.1 Dendrometry, a formal definition

The GEMET (General Multilingual Environmental Thesaurus) adopts the definition for *dendrometry* from [7]:

> "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 could be interesting to extend this project to include also a sensor to heights measurement.[1]

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

There are quite a few types of dendrometers but according to [8] "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 for this project is being developed a "contact dendrometer" based on a linear potentiometer.

[1] This project is already considered extensive enough.

1.2 Physical Layer

1.2.1 Arduino, multipurpose microcontroller

It's not difficult to justify the use of such an interesting platform as Arduino. 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.

Due circumstances exposed in introduction section is needed an accessible and multipurpose platform to be the basis for the device design itself, this is to say 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 because it consist only of three parts,

- **Linear potentiometer:** which is the sensor itself due 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 for 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 its complexity these kind of interfaces are integrated circuits which are mounted on a PCB in order to obtain a pluggable card/shield.

1.2.2 LoRa® (Long Range)

LoRa is a "long-range, low-power, low-bitrate, wireless telecommunications system"[9]. This is because 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 necessity 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 [9]

> "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.

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

In comparison with other common telecommunication technologies, LoRa uses lower bandwidth, but it reaches longer distances:

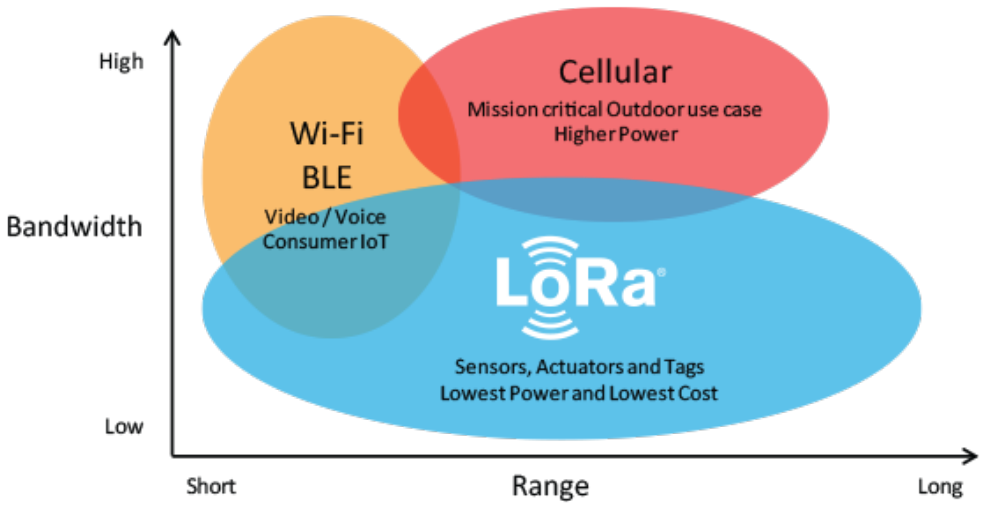

Figure 1: Bandwidth vs range plot for two of the most used telecommunication systems and LoRa.

1.2.3 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.[2] From the hardware point of view, Raspberry Pi implements a better one than Arduino, this also has an impact on a higher cost; however, this hardware allows a Raspberry Pi to support a whole operating system; that's because this 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.[10]

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 embebed (as in Arduino) in a separate pluggable shield/card/hat.

1.2.4 Dragino shields/hats

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

This project is using the following models:

- *LoRa GPS HAT for Raspberry Pi*,[11] which makes use of the extended 40-pin GPIO header to be plugged[3]

- *LoRa Shield for Arduino*[13], which is plugged through analog and digital pins.

[2] There are a lot of 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.

[3] It is important to note 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"[12], **(I'm not sure about this cite (It's a forum))** so this means this hat is designed to be a node-class radio, not a gateway.

1.3 Abstract layer

1.3.1 FIWARE

It is defined as “The Open Source Platform for Our Smart Digital Future”[1]; in a deeper sense is “an open source initiative defending an universal set of standards for context data management”[1], which basically means that *FIWARE* is an open source platform where develop IoT and smart solutions. 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*[14].[4]

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

2 Technical description

Like in the Introduction, the most proper way to present this technical description is to difference between software and hardware parts. However, is still interesting to present a simple and general diagram for whole designed solution. So as the reader can see in the figure 2 is shown the general diagram.

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 of provide any kind of graphical user interface to easily setup 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.

2.1 Hardware

2.1.1 Nodes

Nodes are the parts of the system in direct contact with trees themselves (one node per each tree). Most important features in those devices are: low-cost, low-powered, wireless communication and small size. The figure 3 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, however, by circumstances is not possible to use it and in order to perform a kind of proof of concept, **this project will implement a regular rotary potentiometer.**
- **A signal conditioning stage, which is needed in order to improve the quality of the voltage signal produced by the potentiometer.**

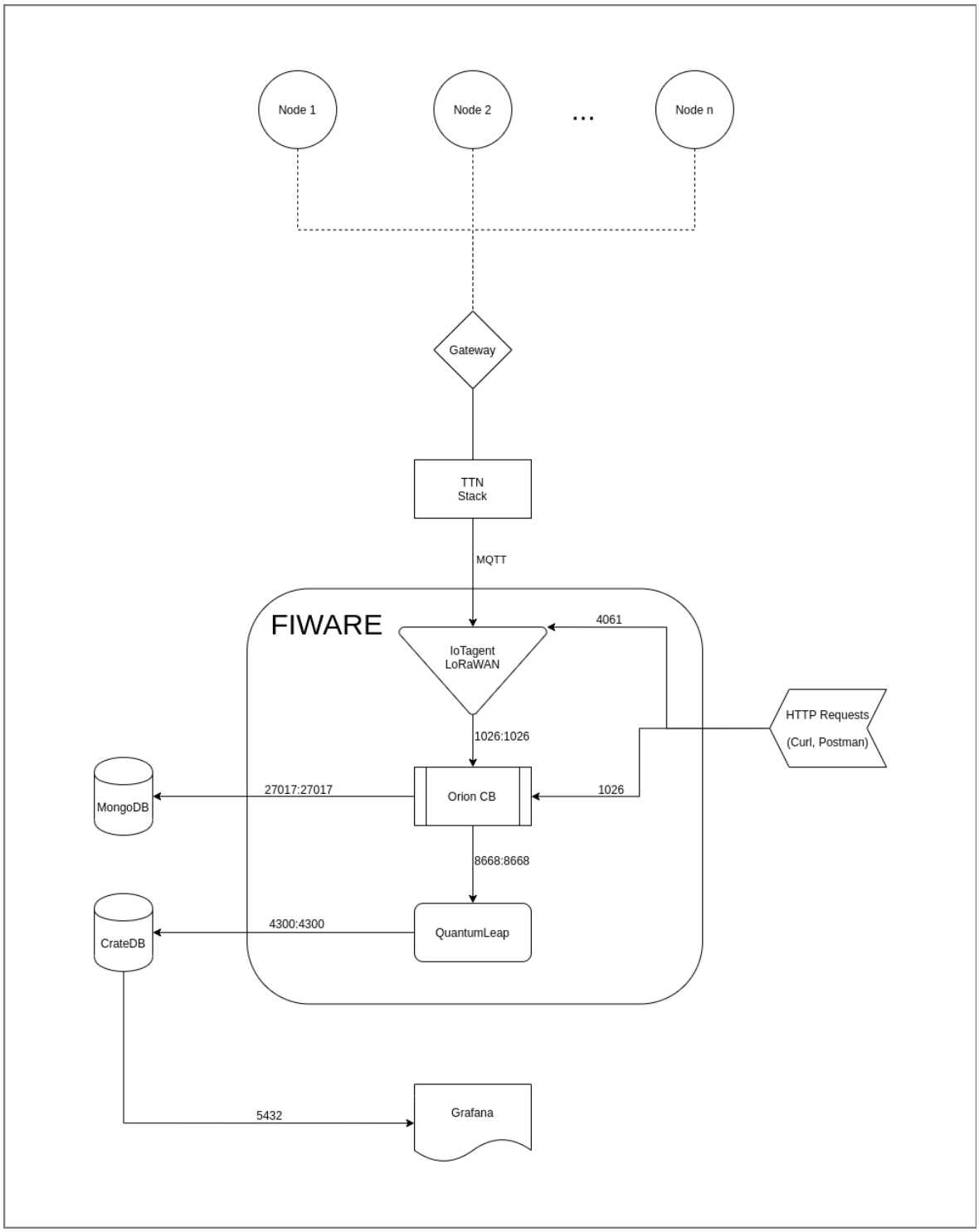

Figure 2: General view of this solution.

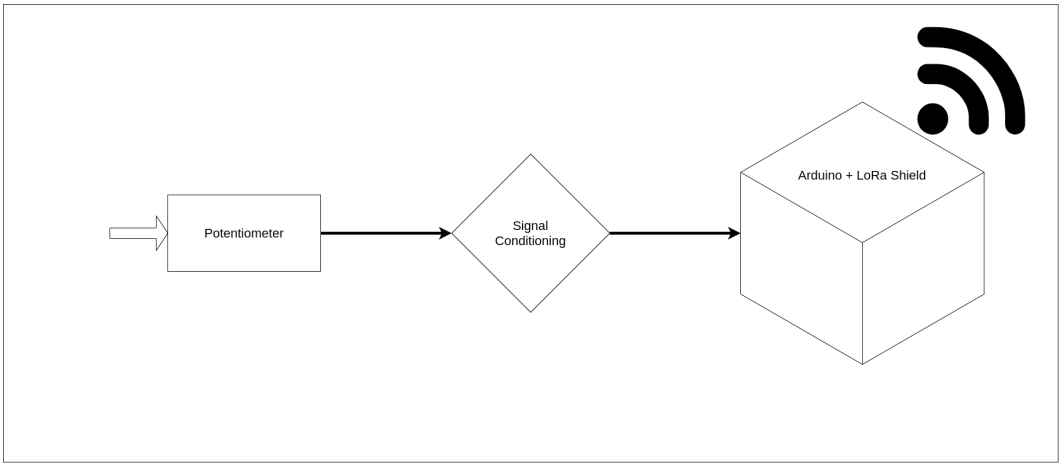

Figure 3: Diagram for a detailed view of a node.

- Arduino + Dragino LoRa Shield; of course it is necessary to schedule data transmission and acquisition, as well as transmit them using the LoRa physical layer and LoRaWAN protocol. For this purpose an Arduino-like microcontroller is perfect.

2.1.1.1 Linear Potentiometer

As said, ideally this potentiometer should be a linear potentiometer with $5\ \text{k}\Omega$ 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 because the figure 3 includes a signal conditioning stage, which will be described below.

2.1.1.2 Signal conditioning

This stage is focused on improve 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 4 shows a possible schematics for this stage.**

It is also recommended by the manufacturer to use the potentiometer as a voltage divider instead a variable resistor, this is because taking advantage of the Arduino power source, particularly the 5 V and GND pins.

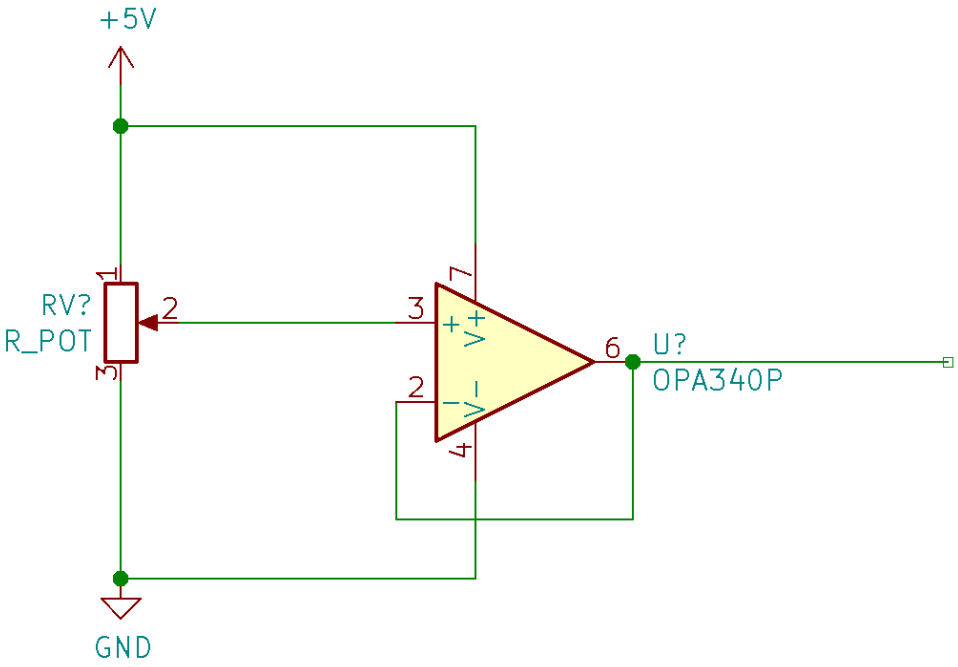

Figure 4: Signal conditioning stage.

2.1.1.3 Arduino sketch

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.

By other hand, in order to this sketch takes advantage of the plugged LoRa Shield, is required to base it on the LoRaMAC-in-C library [15] (`arduino-lmic`)[5] implemented by MCCI, whose technical documentation can be found in [16]. 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" [15]. 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

[5] 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. [17]

As commented lines 30 and 31 point, the most important thing before start working on the sketch is to configure the `arduino-lmic` library, so after install the library with the library manager —in the Arduino IDE or any other chosen IDE—, the reader will need to go 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 looks like in te 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 has 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 set 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 5 —which can also be found in Dragino documentation.

The reader can see the LoRa Chip reset pin is `D9` and as DIO pins —Digital Input/Output— the board

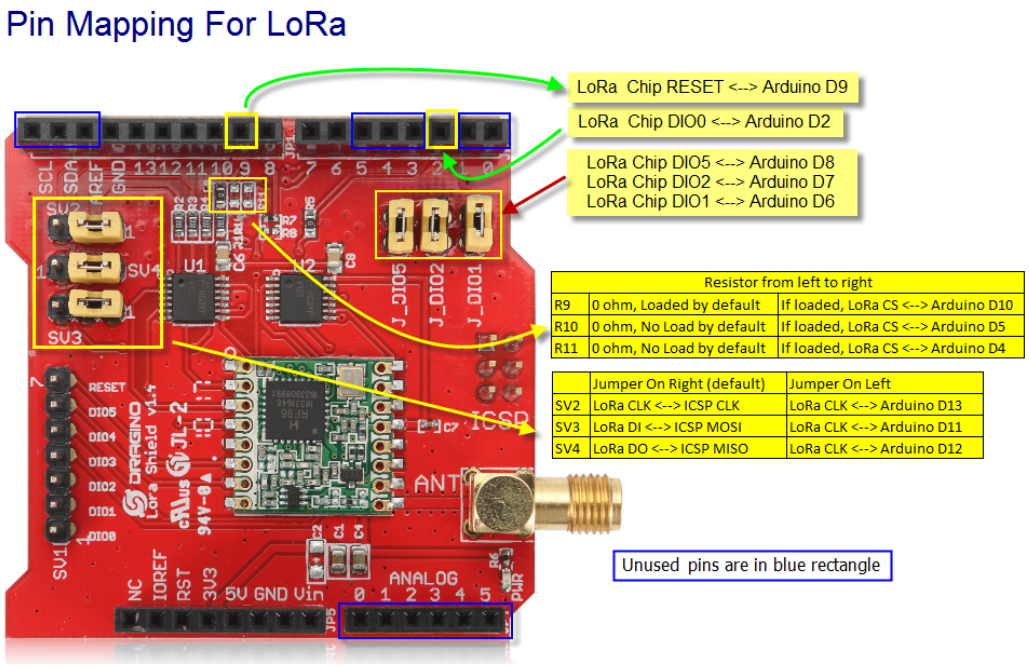

Figure 5: Arduino LoRa Shield pin mapping.

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  };
```

By other hand “Any pins that are not needed should be specified as `LMIC_UNUSED_PIN`” as 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.

Regarding to LoRaWAN and TTN security, as previously indicated, the chosen method is ABP, so are necessary the `NWKSKEY` (Network Session Key), `APPSKEY` (Application Session Key) and `DEVADDR` (End-Device Address); these constants placed at lines 48, 52 and 57 must be obtained from TTN console **AS WILL BE EXPLAINED IN NEXT SECTIONS**. The sketch also sets analog pin `A2` to acquire the sensor signal as 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: Prpareing the payload.

```
181 // Prepare upstream data transmission at the next possible time.
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 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” [18]; 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 [19] —which is also dependent of

ArduinoJson library [20]; 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 is possible to represent numbers from -32768 until 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 will be stated in 2.1.2.2 Dragino GPS and LoRa HAT subsection, the SX1276 is a single channel chip and only can listen in one channel, so despite inside `void setup()` function the channels are setup as shows the example 5, the gateway only will be able to listen the channel 0 (868.1 MHz), which is the frequency set in global_conf.json at [21]

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 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 the channel 0, the microcontroller is transmitting at this channel once in 9—

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.

2.1.2 Gateway

2.1.2.1 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, [22] is in fact a Debian port to ARM architecture. After a little researching, it is possible to conclude that there is not other operative system which improves the performance of Raspberry Pi OS in a Raspberry Pi. Due this former argument this project is going to use Raspberry Pi OS.

One of the most interesting features of Raspberry Pi is precisely that 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.[6] 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), meanwhile Full version is the same than Recommended version but with a few extra applications.

This project is using the Recommended version for Raspberry Pi OS, nevertheless, the project does not require absolutely the desktop environment, so to maximize the available free space in the SD card, 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 [23].

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 tu plug the cable) or using the WiFi interface, which can be configured using also `raspi-config` [24]. Due to its versatility, there are multiple setup that could fulfil the requirements of this projects:

- **PoE** (Power-over-Ethernet); this is not actually a suitable option because Raspberry Pi does not

[6] 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.

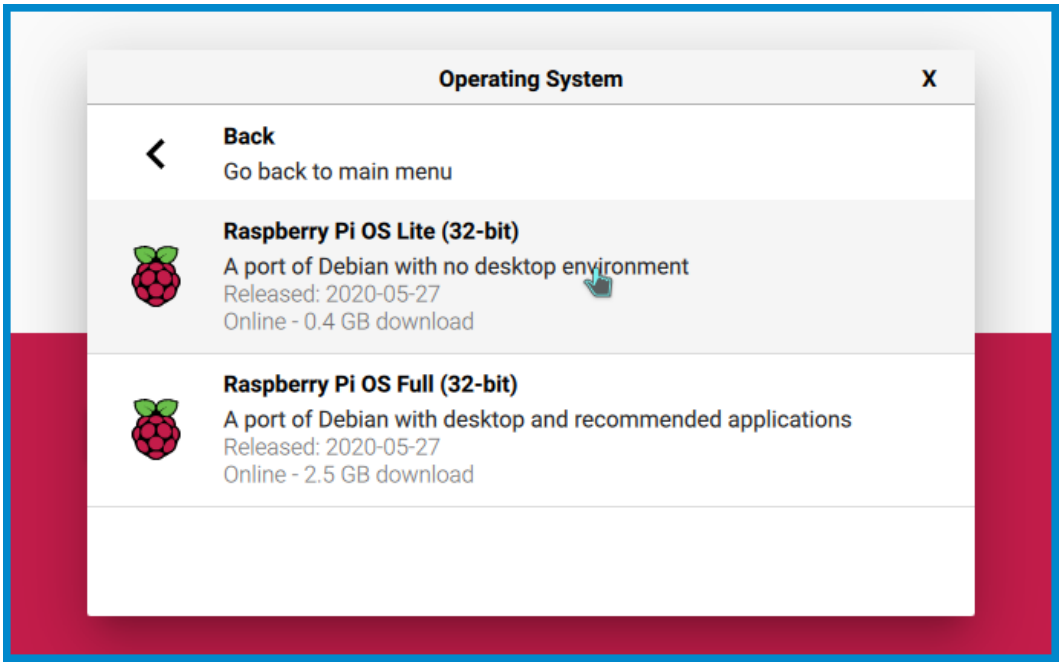

Figure 6: Raspberry Pi Imager showing different options to install Raspberry Pi OS in a microSD card.

support PoE by itself, it needs a separate HAT which is using the GPIO connector, so this makes 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 [25]. 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.[7]

- **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 consider this the ideal setup. This setup doesn't require a conventional access point because the internet access is granted through a GSM module, in a similar way than a mobile phone. This would be the ideal way because

[7] The LoRasPI HAT is more suitable for this option because does not cover whole GPIO connector, leaving free the GPIO pins used in [25].

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 though GPIO pins comes through this does mean the HAT supports stacking, and even if that does, 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 also will to increment the size of the whole device.

It is important to note the necessity of 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` [26].

So generating a pair of keys and configuring properly the ssh access;[8] 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.

```
$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/wyre/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/wyre/.ssh/id_rsa
Your public key has been saved in /home/wyre/.ssh/id_rsa.pub
The key fingerprint is:
SHA256:vkNRk/Fo8iGGo0pYlwb2L3vf3TgOfm11MmZW+BipXgQ wyre@DESKTOP-AFG84JJ
The key's randomart image is:
```

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

```
+---[RSA 3072]----+
|  o       .o     |
| . o . .  +oE    |
|  . = o +.+... o |
| o o o o.= .  = .|
|. . o . S..  o = |
| . . o ..   . O +|
|  . . ... .. * +.|
|     . ..+ o+oo  |
|        o.oo+o.  |
+----[SHA256]-----+
```

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

At this point and depending on the type of access point used to provide internet service to the Raspberry, the reader will be able even to remote control the Raspberry over internet. For this purpose 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.[9]

2.1.2.2 Dragino GPS and LoRa HAT

Gateways doesn't need actually a diagram or detailed description because there are multiple devices which could play this role. These usually are generic devices due 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 by multiple gateways. Each gateway will forward the received packets from the end-node to de cloud-

[9] 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.

based network server. Besides, this project is using a Raspberry Pi 3B+ with a Dragino Hat which mounts a SX1276 LoRa **transceiver**[27]; this is so important because means, 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"[28, p. 12]. In fact, for Europe, and again as reported by [28, 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 exist the possibility of use it as a gateway, because it is technically possible.

Dragino foresees this and provides a dual channel controller [21]. 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 shows the figure 7

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 [21] and compile it performing the command shown in example 8 —Like will be explained below, it is also important to note where `git` will clone the repository and where the reader will compile it, because

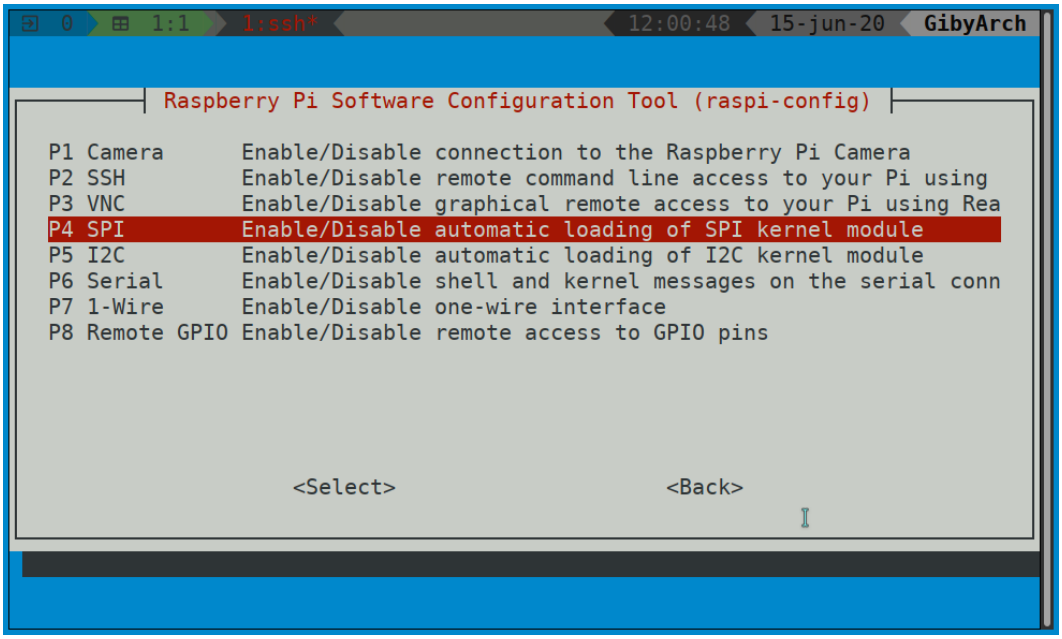

Figure 7: SSH session to enable the SPI kernel module in Raspberry Pi using `raspi-config`. Inside "5 Interfacing options"

the location of the `dual_chan_pkt_fwd` binary is so critical for that 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

```
cd ~
git clone https://github.com/dragino/dual_chan_pkt_fwd
cd dual_chan_pkt_fwd
make
```

Once the controller is compiled, the reader must check the contents of `global_conf.json`, for this projects, due it is using the LoRa GPS HAT Single Channel LoRa [11], the pins configuration in `global_conf.json` must be the following as stated by [21]

Example 9: defined pins in `global_config.json`

```
"pin_nss": 6,
"pin_dio0": 7,
"pin_rst": 0
```

I 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 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 to have 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 perform `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.[10]

The reader must be aware of the path where is located the `dual_chan_pkt_fwd` binary, 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 point have been checked, the reader will be able to manage the service to

[10] 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 8 it can be seen the service active (running)

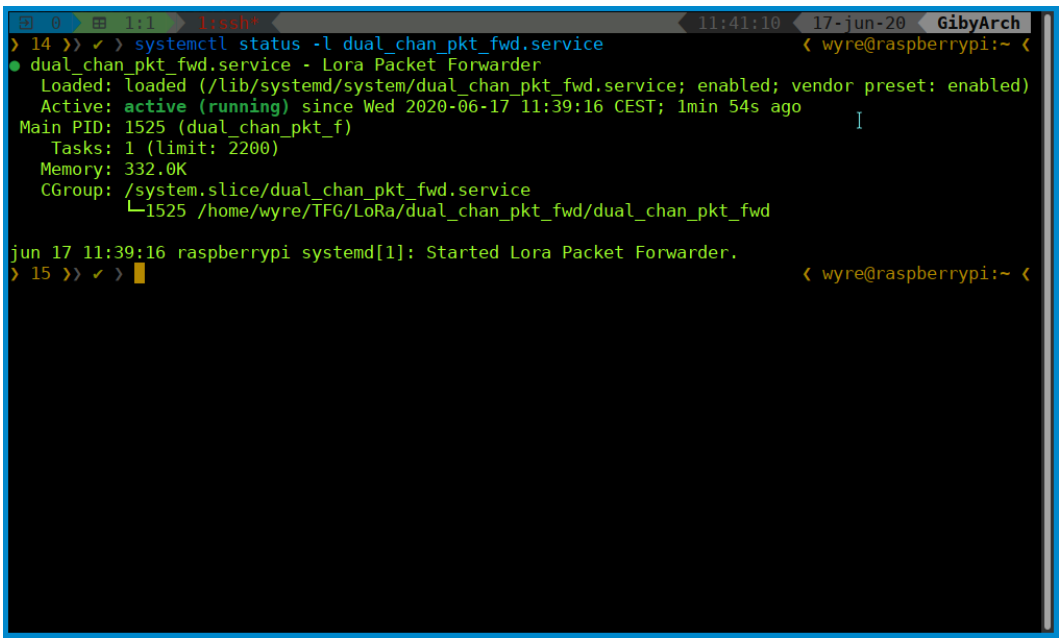

Figure 8: systemd showing the `dual_chan_pkt_fwd.service` status; it is active and running.

As 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`)[11]

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

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

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

2.2 Software

The core of the software part is deployed using Docker containers [29]. These docker containers are deploying essentially the following components

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

- **Orion Context Broker**, this is the core generic enabler, in fact as indicated, this is the only one required component to the solution can be considered as “powered by FIWARE”. Besides this generic enabler requires a **Mongo database**.

- **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 information 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.

References

[1] FIWARE Foundation, e.V. (2020). “FIWARE Home,”
[Online]. Available: https://www.fiware.org/ (visited on 05/2020).

[2] Arduino Company. (2020). “Arduino,”
[Online]. Available: https://www.arduino.cc/ (visited on 05/2020).

[3] Raspberry Pi Foundation. (2020). “Raspberry Pi,”
[Online]. Available: https://www.raspberrypi.org/ (visited on 05/2020).

[4] 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.

[5] A. Abdul-Qawy, E. Magesh, and S. Tadisetty. (Dec. 2015). “The Internet of Things (IoT): An Overview,”
[Online]. Available: https://www.researchgate.net/publication/323834996_The_Internet_of_Things_IoT_An_Overview.

[6] 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.

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

[8] 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.

[9] 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.

[10] 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).

[11] 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).

[12] 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).

[13] 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).

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

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

[16] ——, (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).

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

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

[19] myDevices, Inc. (2020). “CayenneLPP arduino library,” [Online]. Available: https://github.com/ElectronicCats/CayenneLPP (visited on 05/2020).

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

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

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

[23] ——, (2020). “Raspberry Pi OS,” [Online]. Available: https://www.raspberrypi.org/documentation/configuration/raspi-config.md (visited on 05/2020).

[24] ——, (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).

[25] Albert David’s blog. (2020). “Poor man’s PoE for Raspberry Pi−3/4 under ∼$2,” [Online]. Available: http://albert-david.blogspot.com/2019/09/poor-mans-poe-for-raspberry-pi-3-under-2.html (visited on 05/2020).

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

[27] 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).

[28] 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).

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