April, 2018

I-GREENHOUSE
Aquaponics connected greenhouse

Project carried out by
SURIER GAROFALO Aurélien
FERREIRA Joffrey
OZENDA Thomas

Tutored by
PALIX Nicolas

Summary

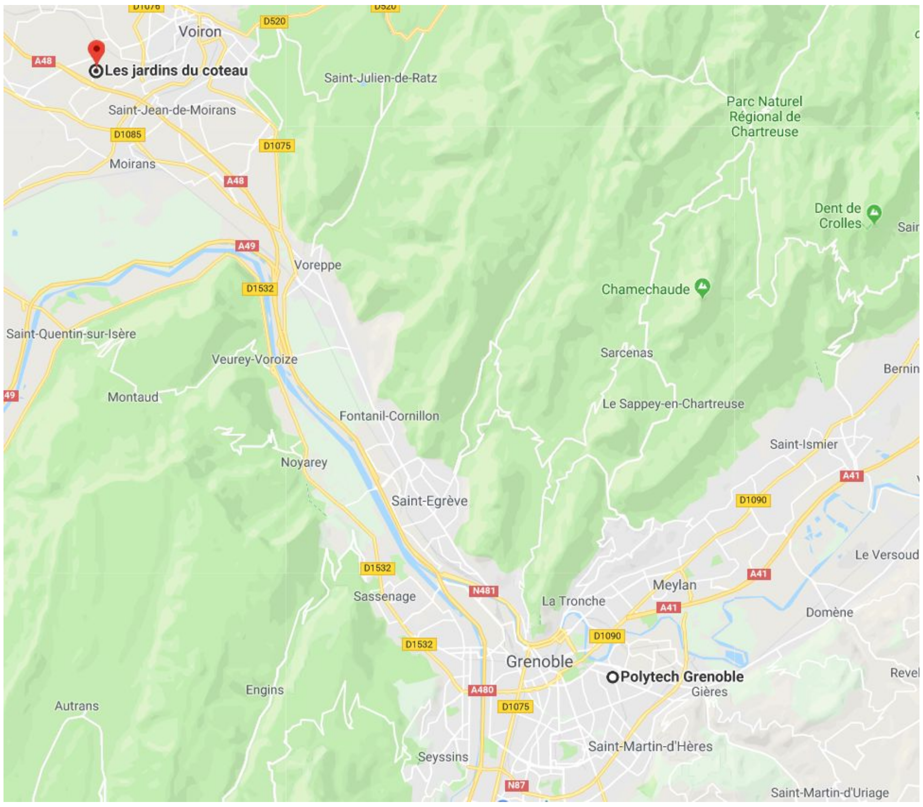
Introduction

Aquaponics farming combine aquaculture and hydroponics, the aim is to automate the greenhouse with sensors which will allow the farmer to monitor its proper functioning.
It has been done in collaboration with IESE, who will program sensors, and us who has deployed a system which will manage reception, processing and display datas.
Moreover, one purpose of this project was to manipulate LoRa communication technology
To link LoRa and internet, a gateway is needed and It is shared with 2 others project so 8 people have worked on this part of the project and informations have been shared for the part beyond the gateway because our work was similar.

I - Project bases

1 - LoRa

LoRa is a long range wireless data communication protocol (up to 10 km). A 7xxx card is placed in the greenhouse with a nucleo..
There are 3 reasons why LoRa has been chosen :
_ Its long range capacity : As written above, the gateway is used in 3 projects. One of this project is a connected greenhouse at “Les jardins du couteau” which is far from the 2 other projects. LoRa allow us to centralize all communications toward one server and use less energy.



_ Low consumption use :In an ecological interest, IESE students have programmed the card to be in low power mode.

_ LoRaWAN : In the event where several aquaponics greenhouse will be connected, a protocol layer above the physical one would be necessary. LoRa already has one : LoRaWAN which is efficient and easy to implement.

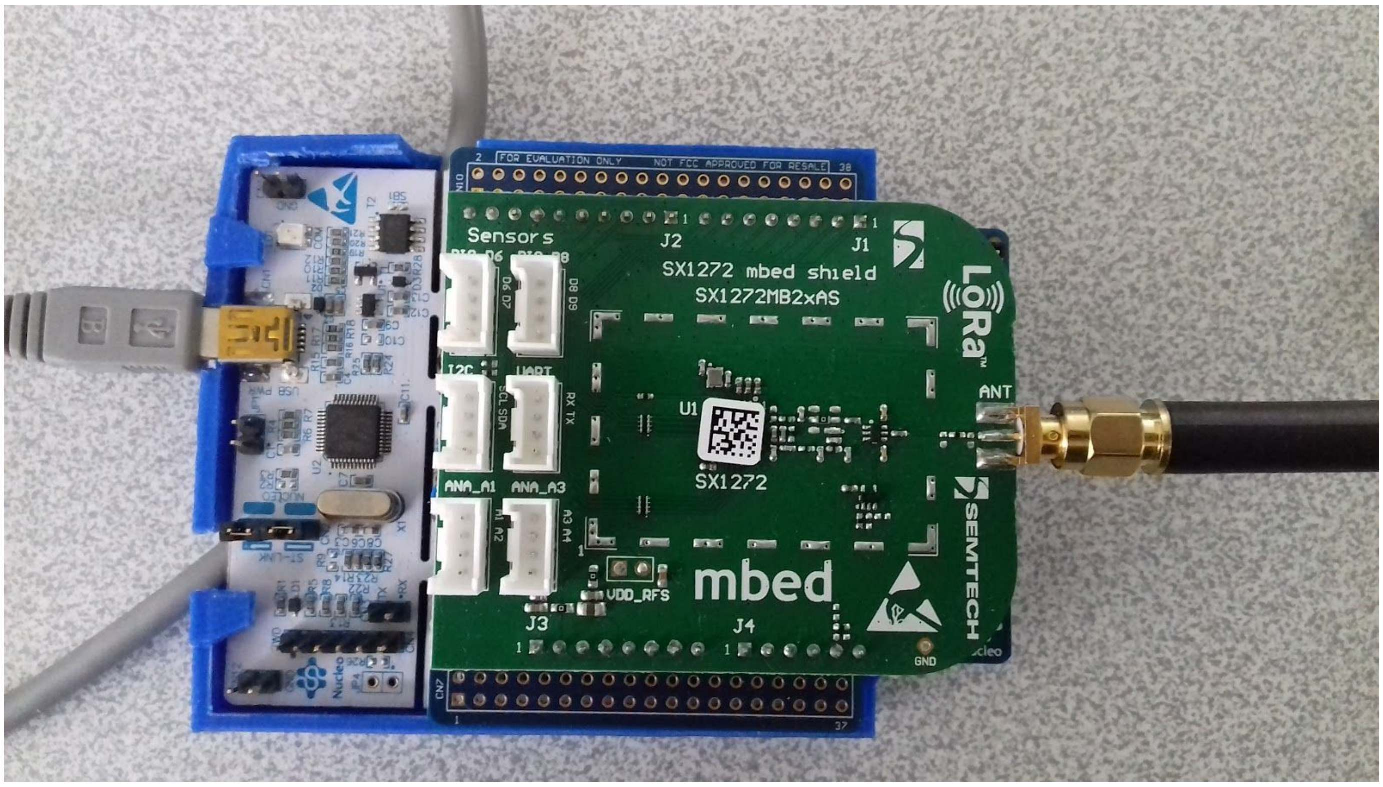
In the first step, we tested the LoRa nodes between them (using the ping-pong demo program from Semtech), and with the gateway in “raw” LoRa, meaning that the gateway was listening to the environment, without any “intelligence”.

The nodes radio parameters for our experimentations are configured as following :
- Region Europe, 8683 MHz
- SF7 (payload up to 130 bytes)

The packet forwarder is configured to match these parameters.

2 - Data framing

There are 5 differents sensors in the greenhouse which will measure temperature, humidity and water level. They will ensure the smooth running of the greenhouse. IESE students have taken care of their configurations and the programmation of LoRa card.



A protocol has been decided between IESE students and us for LoRa packet :

sensor1_id	measure_value	sensor2_id	measure2_value	...
1 byte	4 bytes (1 float)	1 byte	4 bytes (1 float)	

All values (sensors IDs and measurements values) are concatenated, without separators, to save bandwidth.
It does not complexify the parsing on the server, as the parser will consume bytes from the payload one-by-one, to fill typed variables.

Adding an ID to each measure (instead of just concatenating measurements as in the old protocol) allows us to transmit only data that has to be sent. For instance, some values are not going to change between two sensors readings, so they can be omitted to save bandwidth.

For the moment, we have chosen to use a 1-byte encoded ID, but it lacks of optimization, as 4 bits are sufficient to monitor up to 16 sensors.

3 - Gateway and switch

A LoRa PicoCell gateway will be setted up at Polytech to receive all LoRa packet delivered by the 3 projects, to serialize them into JSON objects and then transmit with a packet forwarder program.
It will be connected to a Raspberry Pi where a node-red program will be deployed.

This node will listen on UDP port 1080 (as configured in the JSON configuration file used by the packet forwarder), and receive raw datas which will be redirected using a mqtt broker (mosquitto) to our node-red program parsing the packet.

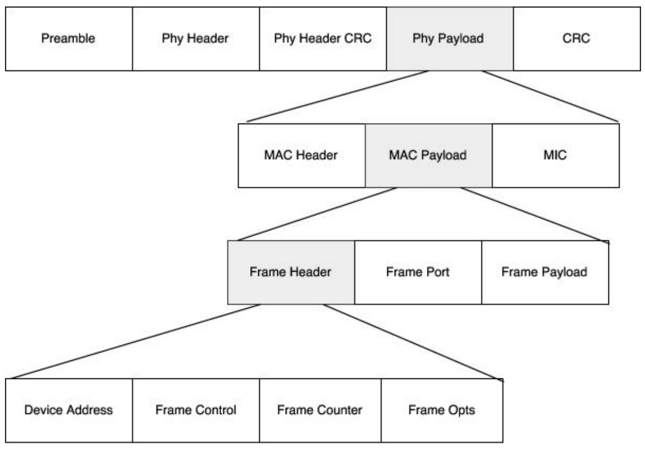
II - Server

1 - LoRaWAN

The Lora Server

With the LoRaWAN server, we don't need to use the node-red program to switch the data between the projects (so gaining 1 byte). A LoRaWAN server has been installed to ensure QoS, security, low-power running, and to manage frequency channel and data rate.

When the server receive a raw LoRaWAN packet from the packet forwarder, It checks if It comes from our greenhouse's card, else discards it, making our system a private network.
This LoRaWAN packet looks like a LoRa one, but the MAC mechanism is embedded in the payload field, so adding some header, as described in this figure :



LoRaWAN packet structure (https://www.researchgate.net/figure/LoraWAN-Packet-Structure_fig2_318575428)

In our project, we have decided to use the Activation By Personalization (ABP) method. In this case, the MAC parameters are hard-coded in the device. For instance, we have defined three mandatory parameters in the node (defined in the Commissioning.h file on the LoRa node) :

- the devAddr, the unique logical address of the node within the private network, which is 12345678 here
- the AppSKey, used in encryption and decryption of the payload
- the NwkSKey, to check the validity of the message.

Its seems to be the best method to use when prototyping a LoRaWAN solution, as it is more easy to debug. In fact, the other method, Over-The-Air-Activation, needs a dialogue with the LoRa server to negotiate its address and the keys, making it more difficult to debug, but making it more secure as the keys are not fixed.

We didn't have time to compare the main (most popular) open-source solutions, which are :

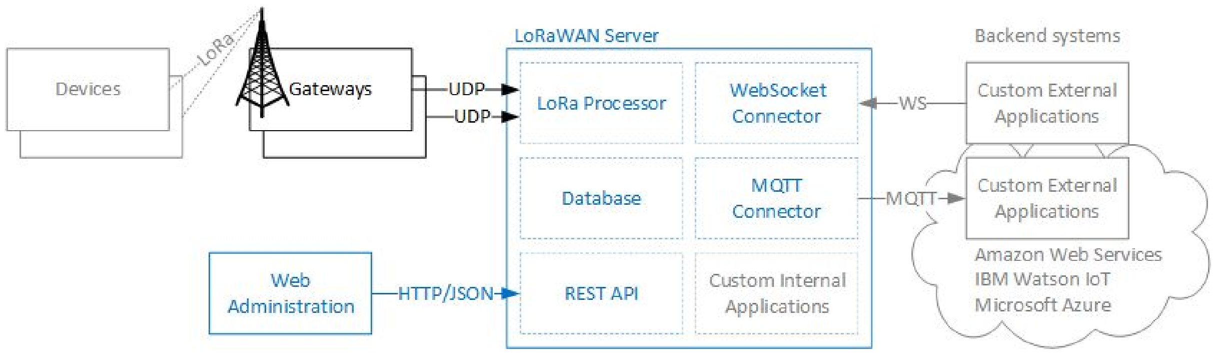
- the The Things Network (TTN) stack, deployed on one of the well-known global networks,
- Loraserver.io, a direct competitor, often advertised on the TTN forum,
- Lorawan-server (https://github.com/gotthardp/lorawan-server), from Petr Gotthard, written in Erlang, licensed under the MIT licence, which we are using in the project.

The latter is easy to install. After installing the requirements such as the Erlang interpreter, you need to install a deb file. Moreover, it comes with a dense documentation, which was heavily used during the server settings, as LoRaWAN comes with lots of parameters to configure.

Even light, it comes with features such as :

- a web frontend to manage it
- an API
- interfaces to other protocols to send processed data

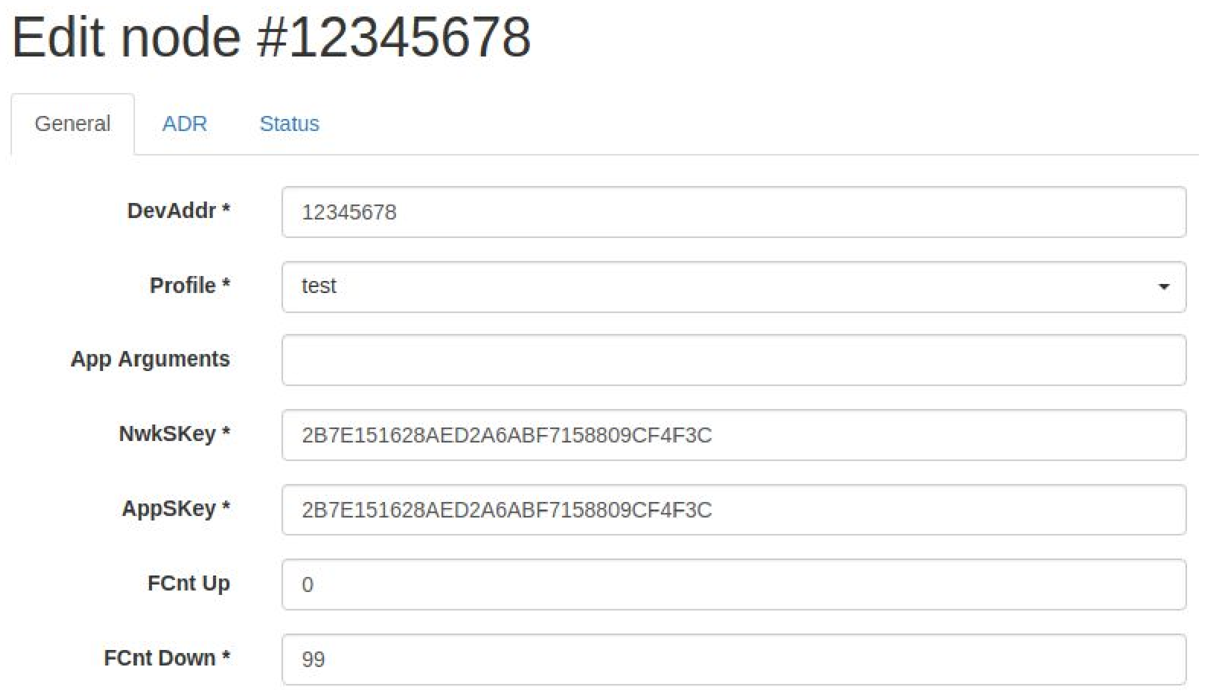
This figure below describes the software's architecture :



The Lorawan-server architecture. We will use the web administration frontend to configure the LoRa processor, and a MQTT connector to publish the decrypted useful data. (https://github.com/gotthardp/lorawan-server).

Although this document is not intended to be a tutorial of how to install and configure this LoRaWAN server, we will briefly discuss on the main steps to achieve a running private network :

- register a gateway. It has to match the packet forwarder's configuration (such as the gateway ID and the UDP port the server has to listen to)
- create a network profile, defining the radio parameters, such as the band used, the coding rate, the Adaptive Bit Rate, the transmission powers
- create a profile for nodes that will be connected to this network
- create a node, with its address and the differents keys as seen above.

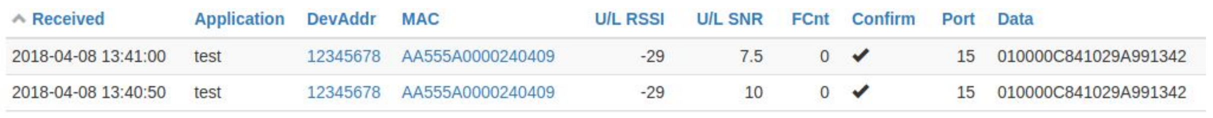

Edit node #12345678

General | ADR | Status

DevAddr *	12345678
Profile *	test
App Arguments	
NwkSKey *	2B7E151628AED2A6ABF7158809CF4F3C
AppSKey *	2B7E151628AED2A6ABF7158809CF4F3C
FCnt Up	0
FCnt Down *	99

Node edition on the web interface with its parameters.

At this point, the server is capable of receiving, decrypting and extracting our applicative data :



Received	Application	DevAddr	MAC	U/L RSSI	U/L SNR	FCnt	Confirm	Port	Data
2018-04-08 13:41:00	test	12345678	AA555A0000240409	-29	7.5	0	✓	15	010000C841029A991342
2018-04-08 13:40:50	test	12345678	AA555A0000240409	-29	10	0	✓	15	010000C841029A991342

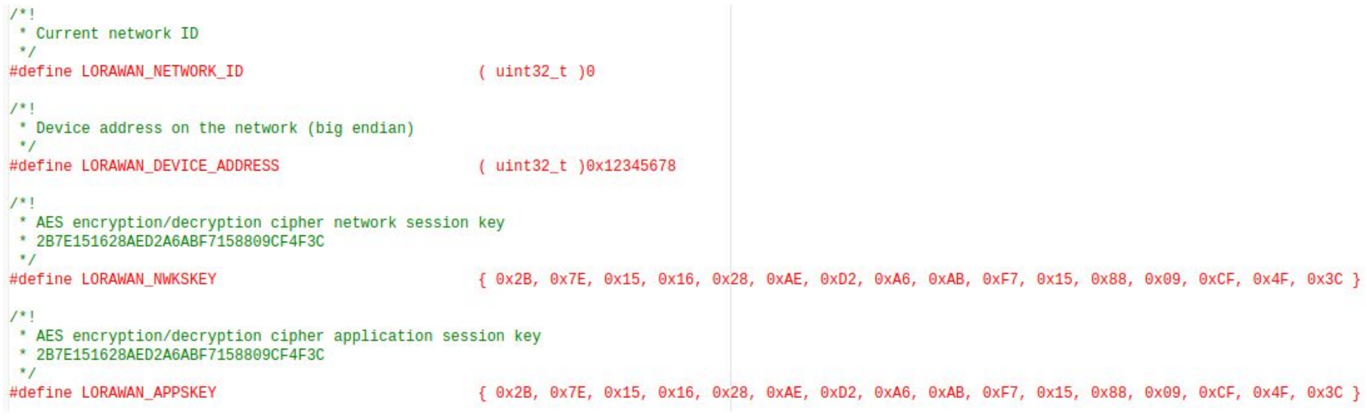
Here are some dumped frames. We can see that packets are coming from node with address 12345678, and are receiving by the gateway with address AA...09 (MAC field). We can also see useful information about the radio transmission quality, which can be used to properly configure the radio parameters. Note that the Port (15) corresponds to the application logic (at each application a port is designated, for instance the reserved port 224 is used for control). The data, displayed as hex, corresponds of two measures sent with the new protocol (see "01" and "02" IDs).

To publish application data for next processing (i.e the parsing and the storage to the database), we use MQTT as a connector.

The nodes

In the node side, it exists numerous librairies for mbed-powered devices to implement the LoRaWAN protocol, such as IBM LMIC. Here, we will use the LoRaWAN library, developed by Semtech.
So, we have forked one of their programs, to analyse and test it.

First, we have to define the MAC parameters (defined in the Commissioning.h file) :



```
/*!
 * Current network ID
 */
#define LORAWAN_NETWORK_ID                          ( uint32_t )0

/*!
 * Device address on the network (big endian)
 */
#define LORAWAN_DEVICE_ADDRESS                      ( uint32_t )0x12345678

/*!
 * AES encryption/decryption cipher network session key
 * 2B7E151628AED2A6ABF7158809CF4F3C
 */
#define LORAWAN_NWKSKEY                             { 0x2B, 0x7E, 0x15, 0x16, 0x28, 0xAE, 0xD2, 0xA6, 0xAB, 0xF7, 0x15, 0x88, 0x09, 0xCF, 0x4F, 0x3C }

/*!
 * AES encryption/decryption cipher application session key
 * 2B7E151628AED2A6ABF7158809CF4F3C
 */
#define LORAWAN_APPSKEY                             { 0x2B, 0x7E, 0x15, 0x16, 0x28, 0xAE, 0xD2, 0xA6, 0xAB, 0xF7, 0x15, 0x88, 0x09, 0xCF, 0x4F, 0x3C }
```

Then other application and radio parameters need to be defined to connect to a private network (they are defined on the main.cpp file), such as :

- the port : `LORAWAN_APP_PORT`
- the payload size (useful data only) : `LORAWAN_APP_DATA_SIZE`
- the join method : `OVER_THE_AIR_ACTIVATION` (defined to 0 for ABP)
- the ability to connect to public network : `LORAWAN_PUBLIC_NETWORK`
- the ADR : `LORAWAN_ADR_ON`
- the datarate, corresponding to a certain spread factor and bandwidth : `LORAWAN_DEFAULT_DATARATE` (here DR_5 corresponds to SF7 125MHz).

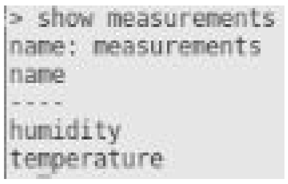
Finally, we added all the sensor data collecting and the framing code parts (in the `PrepareTxFrame` function) we have developed and tested earlier to complete the program, which can now send data through LoRaWAN.

2 - Data storing

Then, measures are parsed and put in an influxDB database.
InfluxDB has been chosen because It is an open-source time series database very suitable for IoT.

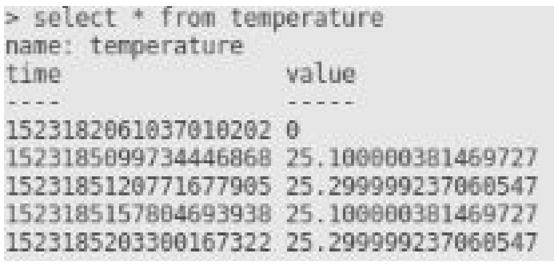
A database with one table is created with several keys : the time first then there will be all the measurements.

The (simple) schema of the database is the following.



```
> show measurements
name: measurements
name
----
humidity
temperature
```

For each “table” containing data, we can see that each records comes with a timestamp and a sensor measure :



```
> select * from temperature
name: temperature
time                value
----                -----
1523182061037010202 0
1523185099734446868 25.100000381469727
1523185120771677905 25.299999237060547
1523185157804693938 25.100000381469727
1523185203300167322 25.299999237060547
```

3 - Data visualization

At the beginning, Grafana was chose to interface with the end user. It provides custom dashboards, containing graphs, to monitor data in real-time.

However, in order to not rely on a third party application, we have decided to create a website that will be able to :

- See the datas
- Show location of the sensors
- Add new sensors.

The webpage was written with the Express framework to do the front and back in JavaScript, the npm package body-parser to encode and decode JSON in JS and plotlyjs as graphic library.

4 - Alerting

We added an alert system that notify the end user via HTML5 notifications if a new value gets above a threshold. For the moment it is only an HTML5 notification, so the webpage has to be opened in a web browser to see this notification, but in the near future we hope to generate notification from the server side using a third party system that allows us to send notifications using any means necessary such as SMS, Email, popup notification on a smartphone for example.

Currently, every time new values are added, the front end script checks that none of the new values get above the threshold. If that is not the case, then the script generate a notification.

III - Project future

1 - Automatisation

Something which could be very interesting to implement is a 2 way communication between the farmer and the greenhouse. If there is a problem in the crops, the farmer can change a parameter or run a device remotely. It has not been been implemented because the material and time necessary was unavailable.

2 - Mobile application

Considering that there are 2 student in multimedia speciality, It has been thought to develop a mobile application which could notify the farmer when there is a problem and allow him to watch out his greenhouse all the time without requiring him to have a computer.
To achieve this, an application querying data directly in the influx database would have been our way to do It..

3 - Docker

One of the objective at the start of the project was a docker to make It easier to install and run our project. Yet, the implementation of a LoRaWAN server has been privileged.

4 - Easy node registration

Thanks to the LoRaWAN server, the registration of new nodes will be easier.

The end-user could add new nodes on the web application, with known parameters, and will call the server API to register it.
Finally, the LoRaWAN parameters could be installed on the node with a script, retrieving parameters from the server and compiling the embedded program with these definitions.

Conclusion

The objectives fixed have been nearly all accomplished except for the project’s docker.
This project teach us to work with another team (IESE) while working in a team of 8 persons and a lot of new technologies of IoT word like node-red, mosquitto and influxDB.
Our biggest difficulty have been to master LoRa technology : C++ was a language never seen before and radio communications technologies was a foreign technology to us.