



INTERNSHIP REPORT

Design of a communication platform for emergency messages

Directed by
Marc PINET



Under the supervision of
Nhan LE-THANH, Tran VAN LIC

In order to obtain the **DUT Informatique**
Academic year **2021-2022**

Acknowledgements

Before starting this report, I would like to thank Mr. Nhan Le Thanh who offered us the opportunity to do an internship at the DNIIT as well as his assistant Mrs. Thanh Thi Thu Nguyen, who allowed us to stay in an excellent residence not far from the institute and for her investment in obtaining our Vietnamese visas.

Furthermore, I would like to thank Mr. Tran Van Lic for providing us the necessary equipments for the realization of this project and Mr. Souébou Bouro — a PhD student in 6th year of electronics — for teaching us the programming of embedded systems and for bringing us his knowledge on the subject.

I would also like to thank Mr. Nicolas Ferry for taking over the responsibility of international relations for the computer science department and his help with the many administrative procedures, Mr. Fabien Ferrero for letting us borrow his equipment and for his help as well as Mrs. Dijana Bojović for being our internship tutor.

I thank the anonymous Vietnam Airlines employee for his energy and patience that saved us from spending a whole night outside in Ho Chi Minh.

Finally, I thank Marcus Aas Jensen, Lyne El dada and Loïc Pantano - the 3 other students of the IUT with whom I realized this internship - for the good atmosphere they created within the team as well as for the work they provided throughout the project.

Table of contents

Introduction	7
I. Specifications.....	8
1. Forewords.....	8
2. Description of the institution	8
3. My hierarchical position	9
4. Needs analysis	9
5. Study of the existing.....	9
5.1. In the company.....	9
5.2. In terms of technology.....	10
6. Distribution of the tasks to be carried out	11
II. Justification of technological choices.....	12
1. Why LoRaWAN?.....	12
1.1. A wise choice.....	12
1.2. How it works in detail.....	12
1.2.1. Radio waves and gateways.....	13
1.2.2. LoRa server and The Things Network.....	15
2. Arduino electronic boards, Bluetooth modules and Raspberry Pi.....	16
2.1. Initial problems encountered.....	16
2.1.1. Meshtastic and bad cards	16
2.1.2. Incompatible code and Bluetooth sending.....	17
2.2. Adaptation to new equipment.....	17
2.2.1. Connections and external module.....	17
2.2.2. Setting up the Arduino software environment	18
2.3. Setting up the Raspberry Pi server.....	20
2.3.1. Installation of a database	20
2.3.3 Listening to uplinks via The Things Network API	22
III. Project implementation and in-depth analysis	24
1. Bluetooth Low Energy 4.0 (BLE)	24
1.1. Module HM-10	24
1.2. Bluetooth communication.....	26
1.2.1. Connections and configuration at the code level.....	26

1.2.2. How the receiving works	29
1.2.3. Sending issue.....	30
2. Sending to The Things Network from a board.....	30
2.1. First adaptation of the code	30
2.1.1. Adaptation methodology	30
2.1.2. Results obtained.....	32
2.2. Second adaptation of the code.....	33
2.2.1. Merge the program with the Bluetooth program	33
2.2.2. Results obtained.....	34
3. Python script and Raspberry Pi server.....	35
3.2. Python Script.....	35
3.1.1. Uplink listening	36
3.1.2. Database.....	38
3.1.3. Sending downlinks	38
3.2. Setting up the Raspberry Pi server.....	40
3.2.1. Installation	40
4. Adaptation with the mobile application.....	41
4.1. Limited messages and firmware.....	41
4.2. Adaptation of the Python script.....	41
5. Return on the remaining features to be implemented.....	41
IV. Final results	42
V. Conclusion	42
References.....	43
Technical appendices.....	44
Abstract.....	48

Table of figures

Figure 1 - UCA Education Board.....	10
Figure 2 - Raspberry Pi 4 Model-B	10
Figure 4 - LoRaWAN protocol logo	12
Figure 3 - Modèle OSI avec LoRa.....	12
Figure 5 - Descriptive diagram of the LoRaWAN operation.....	13
Figure 6 - Layout of the gateways in Da Nang city	14
Figure 7 - Topology « star of stars ».....	14
Figure 8 - The Things Stack et The Things Network logos.....	15
Figure 9 - LilyGO TTGO T-Beam V1.1 ESP32 LoRa	16
Figure 10 - Logo de Meshtastic.....	16
Figure 11 - Module HM-10 Bluetooth (BLE 4.0).....	18
Figure 12 - Arduino and Arduino IDE logo.....	18
Figure 13 – Main Arduino IDE interface.....	19
Figure 14 - Card specific parameters.....	20
Figure 15 – Debian logo	21
Figure 16 – MySQL logo.....	21
Figure 17 – MariaDB logo.....	21
Figure 18 – Description of our database.....	22
Figure 19 – MQTT logo.....	23
Figure 20 - Visual Studio Code logo	23
Figure 21 – Python logo	23
Figure 22 - Diagram of the complete communication between all devices.....	24
Figure 23 - Comparative sheet between BLE and Bluetooth	25
Figure 24 - Connecting the Bluetooth module to the UCA Education Board.....	26
Figure 25 - UCA Education Board Datasheet	26
Figure 26 - Minimalist code for Bluetooth data reception.....	27
Figure 27 - Serial Monitor and Arduino IDE with a bad Baud Rate	28
Figure 28 - Lines of code associated with Vietnamese radio frequencies.....	31
Figure 29 - Hexadecimal values to fill in the code	31
Figure 30 - First results obtained in the console of the TTN application by node1	32
Figure 31 - Final code running continuously on a board.....	33
Figure 32 - Payload received on the TTN application console by node1	34
Figure 33 - .env folder created with the activation file of the latter	35
Figure 34 - Selection of the environment interpreter	36
Figure 35 - Minimalist code for listening to uplinks in Python.....	37
Figure 37 - Minimum code to perform a downlink.....	39
Figure 38 - Downlink display in the application console for node2	39
Figure 39 - Logo of Git, decentralized version management tool	40
Figure 40 - Vim logo, text editor on console	40
Figure 41 - Complete picture of the communications made so far	42

Glossary

Gateway: Equipment for moving data from one environment to another (relay).

LoRaWAN: Low power WAN protocol based on radio communications and using gateways as intermediaries, which uses LoRa.

LoRa: Modulation technique* and physical layer to which LoRaWAN belongs, allowing to connect objects in a volume and with a reduced cost.

Modulation technique: Variation (of amplitude, intensity, frequency) of a physical phenomenon (e.g.: emission in frequency modulation).

IoT: The IoT (Internet of Things) is the interconnection between connected objects equipped with sensors, software, or any other technology likely to communicate in network.

End device: A device that can send and receive data from and to a network, which does not connect directly to the transmission line.

ISM: Radio band used in industrial, scientific, and medical fields.

Firmware: Program integrated into a computer hardware.

Mesh: A mesh network is a LAN topology in which infrastructure nodes (i.e., gateways, switches, etc.) connect directly, dynamically, and non-hierarchically to as many other nodes as possible and cooperate with each other to efficiently route data to and from clients.

API: An API (Application Programming Interface) is a software interface that allows one software or service to be connected to another software or service to exchange data and functionality.

Board: Electronic board. I will use this word several times because this term is explicitly used by Arduino IDE, to define an electronic board.

MTU Negotiation: In a computer data transmission, MTU (Maximum Transmission Unit) defines the maximum size (in bytes) of the packet that can be transmitted at one time. The MTU is used as the basis for negotiating the MSS (Maximum Segment Size) in the establishment of a TCP session to avoid packet fragmentation.

Pin: Small metal rod usually used to connect two electronic devices together using cables.

Sampling period: This is the time between two samples. It is fundamental in the acquisition of a signal because the slightest disturbance or irregularity will result in erroneous signals.

Introduction

This report is about the internship I did from May 10th to July 20th at the DNIIT in Da Nang, Vietnam.

The project we had to work on was called « Lo-Texto ». The objective was to design a communication platform to send and receive emergency messages.

Indeed, Vietnam has many risky areas: from the ocean to the mountains far from the city center, many people have no way to warn the population when they find themselves in a dangerous situation. Thus, our application will allow these

people to communicate with the city and its services, without the benefit of an Internet connection.

It is through a multiple choice of situations that the user will be able to choose and send his current state to the adapted services of the region, which will then be able to act accordingly.

It is with 3 other trainees that we must carry out this project, on which potentially human lives will depend.

At first, the institute that proposed us the project that we were given to work on will be presented. Then, a global study of this project and a more in-depth analysis of my part in this project which will lead to the obtained results then a final assessment.

I. Specifications

1. Forewords

Marcus Aas Jensen, Loïc Pantano and I could not start the internship at the same time as Lyne El Dada because of the UQAC exams in Quebec. Despite this delay, we quickly adapted thanks to her who was in France and who took care to keep us informed of her progress and of the information she received every day.

2. Description of the institution

The Da Nang International Institute of Technology (DNIIT) is a collaborative project between Da Nang University and the Université Nice Côte d'Azur to strengthen international relations in the development of various applications (data processing, management, pollution analysis, services, etc.). One of the main objectives of the institute is to improve academic exchanges between teachers,

researchers, and students, and to develop interdisciplinary projects of real value to the Vietnamese economy and society.

3. My hierarchical position

As an intern at DNIIT, I am required to follow the guidelines of Mr. Nhan Le Thanh and Mr. Tran Van Lic. We also had the chance to have a PhD student from an electronic master in the same working room as us, and sometimes some Vietnamese students.

4. Needs analysis

Our project aims to serve people in dangerous conditions or special cases such as fishermen, mountain dwellers, etc. They will be able to send emergency messages without the availability of an internet network.

Potential customers would be people far from the city center who cannot communicate using Wi-Fi or mobile networks. An alternative to these networks would have to be found that would allow them to send and receive messages over long distances without the need of Internet access.

5. Study of the existing

5.1. In the company

The Lo-Texto project has not yet been developed. So, we had to realize the whole project from the instructions we were given, especially on the network protocol to use: the LoRaWAN*. We will come back in detail on its functioning in a later part dedicated to it.

5.2. In terms of technology

The city of Da Nang is already equipped with LoRa* gateways*, which allowed us to test our firmware without having to install the hardware ourselves. The messages will then be retrieved via these gateways placed at different locations in the city and will relay the message to the appropriate services.

For our experiments, we used 2x Android smartphones for the mobile application part as well as 2x UCA Education Board which allowed us to realize LoRa communications.

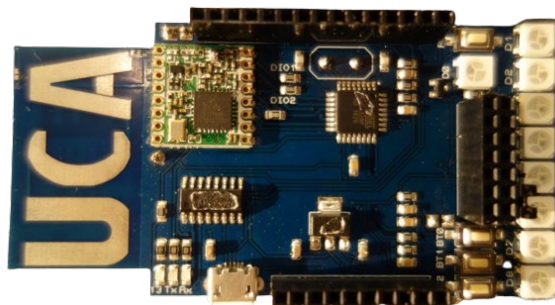


Figure 1 - UCA Education Board

Finally, Mr. Tran Van Lic provided us with a Raspberry Pi mini-server so that we could retrieve and store the sent and received messages in a database.

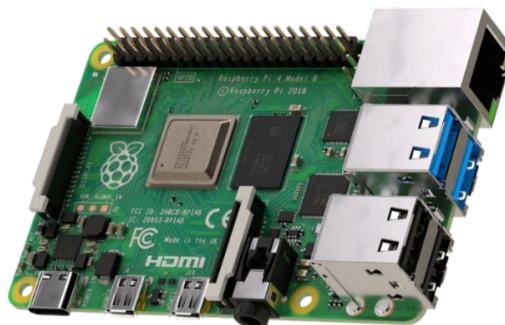


Figure 2 - Raspberry Pi 4 Model-B

In addition to the electronic boards, the server, and the gateways, we also had at our disposal small snippets of code rather simplistic, but which served us as documentation of the functions and methods to be used in our programs. These small pieces of code can be found on the GitHub platform and are therefore obviously open source.

6. Distribution of the tasks to be carried out

To divide the work fairly, we chose to divide our project into two categories: the development of the mobile application and the communication between the different electronic devices (phone, electronic cards, gateway, and server).

For the mobile application part, it was divided into two parts: the frontend and the backend. Marcus Aas Jensen oversaw the frontend (interface, user experience, displays) while Loïc Pantano oversaw the Bluetooth communication between the phone and the board*.

Finally, for the second part, as this was a brand-new field for us, we decided to work on it together. So, Lyne El Dada and I chose to work on the same files until we were both comfortable with the field. Once we understood the Arduino programming and the use of APIs*, we divided the tasks equally: Lyne took care of the boards wiring, the configuration of the network application for IoT* that was proposed to us by Mr. Tran Van Lic and the setting up of the database. I took care of the firmware of the Arduino boards, the installation of the Raspberry Pi server and the script that allowed us to listen and save the received and the redirection of these messages.

II. Justification of technological choices

1. Why LoRaWAN?



Figure 3 - LoRaWAN protocol logo

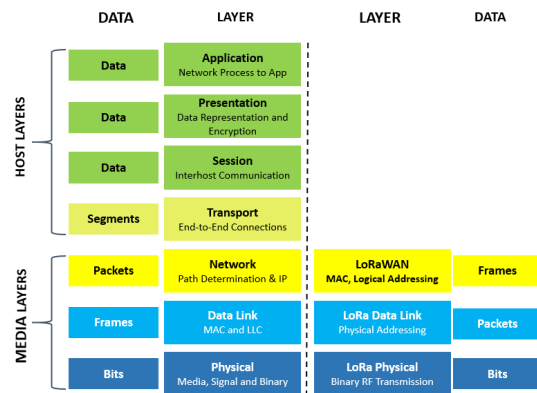


Figure 4 - Modèle OSI avec LoRa

LoRaWAN is a communication protocol based on radio waves belonging to the LoRa network. LoRa belongs to the physical and data link layer whereas LoRaWAN belongs to the network layer.

1.1. A wise choice

LoRa technology is inexpensive both in terms of the resources needed to set it up and in terms of power consumption, which makes it very favorable for large-scale installation.

Although it does not have a high bandwidth, this protocol targets long-range communications thanks to low-power and generally inexpensive end-devices* (from 1€ to 5€) which, through gateways, communicate with application servers. It is therefore very interesting to use such a protocol for people far from the city center who do not have access to the Internet.

1.2. How it works in detail

We will see how LoRaWAN works in detail, what technologies we have chosen to use for its implementation.

1.2.1. Radio waves and gateways

As mentioned above, LoRaWAN is based on radio communications. The frequency used for LoRa communication depends on the continent: for example, in Europe, the radio bands used are ISM* 868 MHz while in North America, it will be ISM 915 MHz. In this project, we will use the ISM 921 MHz, specific frequencies to the Asian continent.

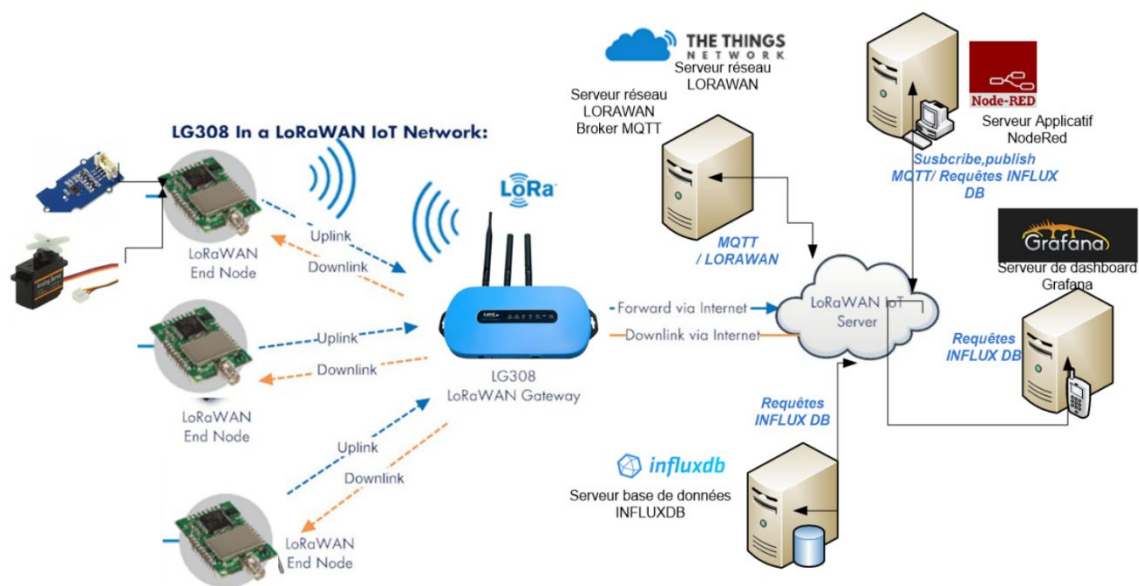


Figure 5 - Descriptive diagram of the LoRaWAN operation

There are 3 types of LoRaWAN classes: class A, class B and class C. Class A can only receive data when it sends data, class B can only receive on time intervals and class C is continuously listening to the messages sent to it. All these classes have different power consumption because listening to downlinks consumes energy. We decided to use class B because it's the most balanced.

The embedded systems we use are natively equipped with a LoRa micro-antenna. These will, with the help of the firmware* that we have programmed, send information to the nearest gateway.

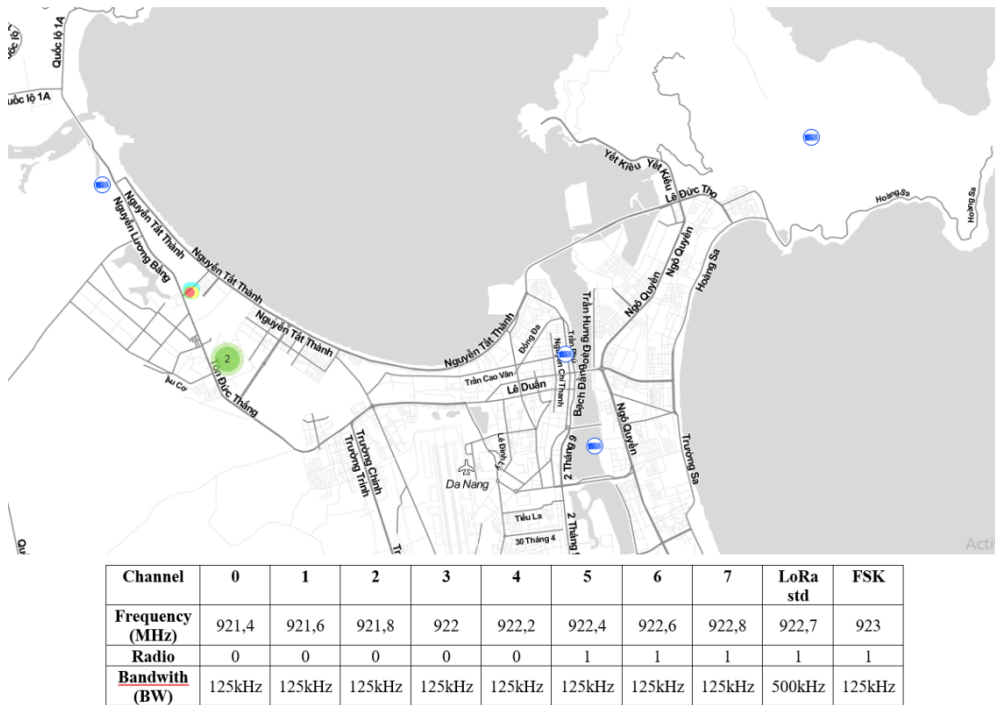


Figure 6 - Layout of the gateways in Da Nang city

The gateway will then send the information to a server dedicated to LoRa communications, allowing the passage of the packet from the LoRa layer to the application network layer.

The LoRaWAN network topology is called star-of-stars because an application server is connected to a multitude of gateways which are in turn connected to a multitude of end devices.

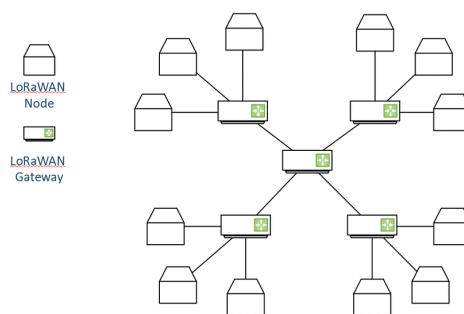


Figure 7 - Topology « star of stars »

1.2.2. LoRa server and The Things Network

To temporarily store the information sent by the gateway and to turn the LoRa packet into an online-usable packet (application layer), we chose to use The Things Network (TTN).

TTN is a community and open-source network for the Internet of Things using LoRa technology. It is possible to use this network freely, but it is also possible to help extending the network by deploying gateways, as the city of Da Nang has done. The Things Network manages The Things Stack Community Edition, which is a participatory, open, and decentralized LoRaWAN network. We use their API to do uplink and downlink communications (see figure 4).



Figure 8 - The Things Stack et The Things Network logos

Their website also allows to see live data transmission of messages that passes through our TTN application.

The choice of the various communication intermediaries having now been made, let's move on to the next part.

2. Arduino electronic boards, Bluetooth modules and Raspberry Pi

2.1. Initial problems encountered

2.1.1. Meshtastic and bad cards

At the beginning of the internship, some boards of type LilyGO TTGO T-Beam V1.1 ESP32 LoRa were lent by Mr. Fabien Ferrero to Lyne in order to start from an already existing software solution: the Meshtastic project.

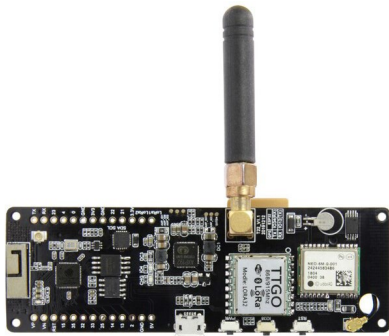


Figure 9 - LilyGO TTGO T-Beam V1.1 ESP32 LoRa



Figure 10 - Logo de Meshtastic

Meshtastic is a project allowing communication via the Mesh* protocol using GPS communicators for a reduced price. Our initial task was to adapt this existing project by adding a LoRa layer.

The documentation was great, and Lyne worked many weeks to understand the whole project and how we should do to put the LoRa layer into their software. Unfortunately, not everything went as planned.

2.1.2. Incompatible code and Bluetooth sending

Our first tests were conclusive. We had succeeded in making two boards communicate over short distances thanks to the LoRa antenna. Nevertheless, these cards had very little (if any) online documentation, which made it impossible to use them apart from the few snippets of code we could find on the Internet.

Our tutor having noticed that our boards would prevent us from progressing properly, he decided to ask Mr. Souébou Bouro to give us hardware that could support the pieces of code available on Mr. Fabien Ferrero's GitHub repository. This is how we got our new UCA Education Board. One of the particularities of this board, just like the old ones, is to have a very high lifetime when their firmware is well optimized (about several months).

However, these cards were not natively equipped with Bluetooth antennas. We had to buy an external module and opted for the HM-10 BLE 4.0 module.

2.2. Adaptation to new equipment

2.2.1. Connections and external module

We connected on each board an external Bluetooth module (an Arduino HM-10 BLE 4.0) which we used to realize the communication between the phones and the electronic boards. Furthermore, this module allows connections with Android and iOS devices (cross-platform) which is what we need to achieve this project. As the main goal is to make this application accessible by anyone, it's important to make it cross-platform as the mobile application is also.

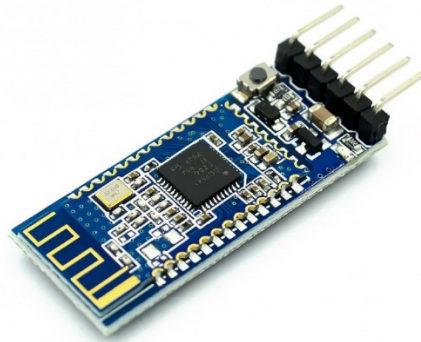


Figure 11 - Module HM-10 Bluetooth (BLE 4.0)

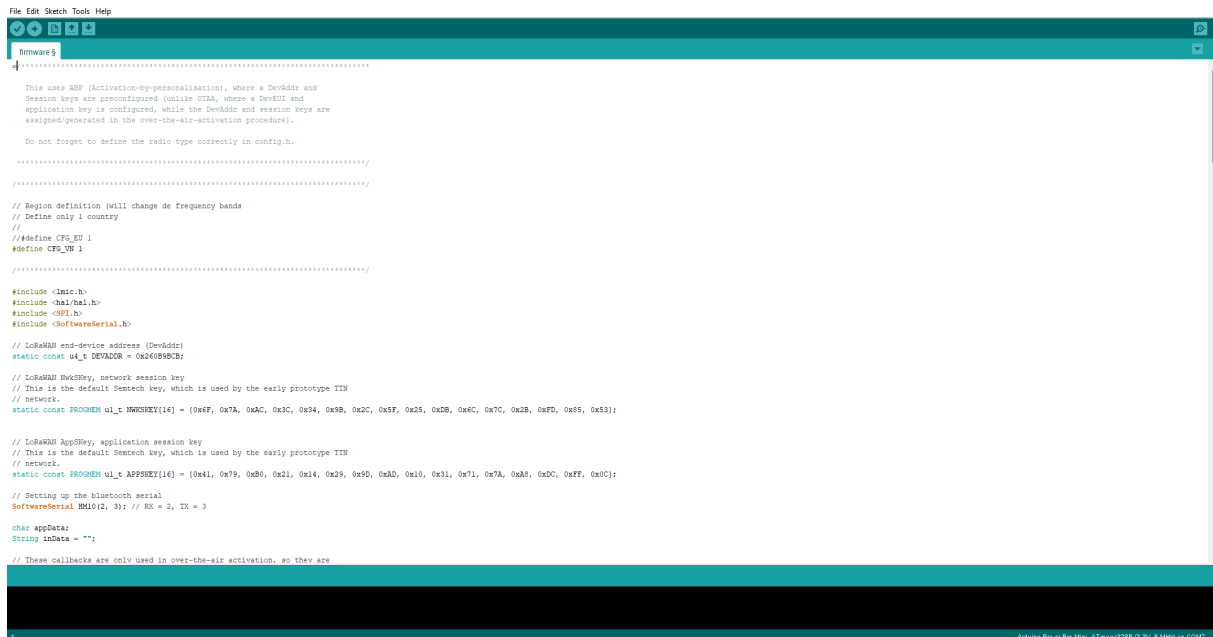
By following a tutorial on the Internet, we managed to make the connection and turn on the module once the card is connected to a computer via Micro-USB. We will come back to the hardware wiring in detail in a later part.

2.2.2. Setting up the Arduino software environment



Figure 12 - Arduino and Arduino IDE logo

To realize our code injections in the electronic board, we used Arduino IDE. Before anything, it should be noted that Arduino is also a programming language derived from C/C++ and that we will use it to realize the firmware of our boards.



```

File Edit Sketch Tools Help
-----
simu@re$
-----
This uses ABF (Activation-by-Fragmentation), where a DevAddr and
Session Keys are preconfigured (unique IDs), where a DevAddr and
application key is configured, while the DevAddr and session keys are
assigned/generated in the over-the-air-activation procedure).

Do not forget to define the radio type correctly in config.h.

-----
// *****
// Region definition (will change de frequency bands
// Define only 1 country
//
// #define CFG_EU 1
// #define CFG_VN 1
// *****

#include <lmic.h>
#include <hal/hal.h>
#include <SPI.h>
#include <SoftwareSerial.h>

// LoRaWAN end-device address (DevAddr)
static const u1_t DEVADDR = 0x0188E83b;

// LoRaWAN AppSKey, network session key
// This is the default-Samech key, which is used by the early prototype TTN
// network.
static const PROGMEM u1_t APPSKEY[16] = {0x6F, 0x7A, 0xA6, 0x3C, 0x34, 0x9B, 0x2C, 0x5F, 0x25, 0x0B, 0x6C, 0x7C, 0x2B, 0xF0, 0x65, 0x53};

// LoRaWAN AppSKey, application session key
// This is the default-Samech key, which is used by the early prototype TTN
// network.
static const PROGMEM u1_t APPSKEY[16] = {0x41, 0x79, 0xB0, 0x21, 0x14, 0x29, 0x50, 0x3D, 0x10, 0x31, 0x71, 0x7A, 0xA5, 0xDC, 0x6F, 0x0C};

// Setting up the bluetooth serial
SoftwareSerial BM10(2, 3); // RX = 2, TX = 3

char appData;
String inData = "";

// These callbacks are only used in over-the-air activation, so they are

```

Figure 13 – Main Arduino IDE interface

This software allows us, once configured, to inject our code into the electronic board with a single click. In addition of having a search tool for external libraries, it is very easy to use for beginners in embedded systems programming such as us. In order for the card to be detected by our computer, we had to install a driver for the CH340C chip support (which our card has).

Once this was done, we only had to set up the software for the card specifically and select the serial port so that the software could « listen » to the standard inputs and outputs.

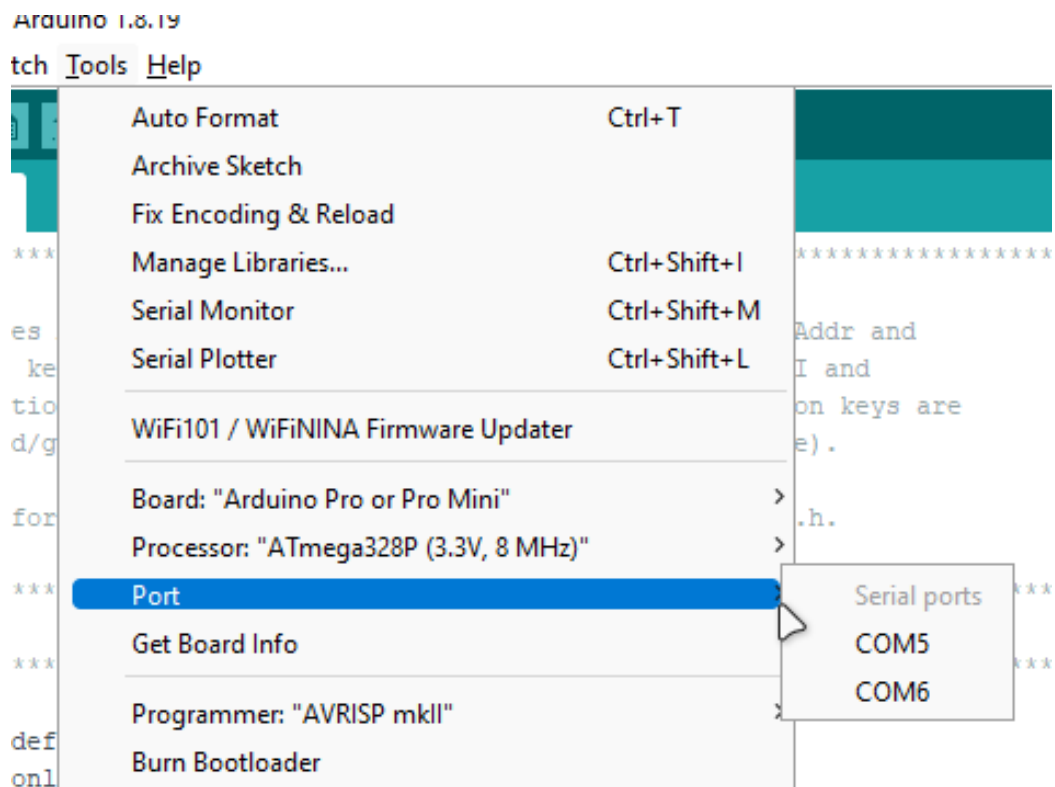


Figure 14 - Card specific parameters

Now that we had set up our work environment for programming embedded systems, we now had to set up the programming environment for our scripts that would run on the Raspberry Pi server.

2.3. Setting up the Raspberry Pi server

2.3.1. Installation of a database

Since our Raspberry Pi runs on a Debian-based operating system, we could easily install packages and software.



Figure 15 – Debian logo

To access it remotely, our supervisor Mr. Tran Van Lic set up an SSH connection via the public IP of the device. All that was left to do was to install a database and a script running continuously that would listen to the data sent and received via an API, that of TTN.

We therefore opted for a MariaDB database (a community branch of MySQL) which we had to configure in such a way that it was accessible outside the local network.



Figure 16 – MySQL logo



Figure 17 – MariaDB logo

For our tests, we made a rather simplistic database that allows us to store the messages and the main information.

```

MariaDB [(none)]> use ttn;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
MariaDB [ttn]> SHOW TABLES;
+-----+
| Tables_in_ttn |
+-----+
| Message       |
| Payload       |
+-----+
2 rows in set (0.001 sec)

MariaDB [ttn]> describe Message;
+-----+-----+-----+-----+-----+-----+
| Field      | Type      | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| message_id | int(11)   | NO   | PRI | NULL    | auto_increment |
| message    | varchar(100) | YES  |     | NULL    |               |
+-----+-----+-----+-----+-----+-----+
2 rows in set (0.003 sec)

MariaDB [ttn]> describe Payload
-> ;
+-----+-----+-----+-----+-----+-----+
| Field      | Type      | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| uplink_token | varchar(100) | NO   | PRI | NULL    |               |
| time         | datetime  | YES  |     | NULL    |               |
| message      | varchar(100) | YES  |     | NULL    |               |
| device_id    | varchar(100) | YES  |     | NULL    |               |
| gateway_id   | varchar(100) | YES  |     | NULL    |               |
| device_eui   | varchar(100) | YES  |     | NULL    |               |
| device_address | varchar(100) | YES  |     | NULL    |               |
| application_id | varchar(100) | YES  |     | NULL    |               |
+-----+-----+-----+-----+-----+-----+
8 rows in set (0.003 sec)

MariaDB [ttn]>

```

Figure 18 – Description of our database

2.3.3 Listening to uplinks via The Things Network API

Once that we knew where we could retrieve the received messages from, we had to think of a way to transfer the data from TTN to the database. For this, we decided to use the Python programming language with the MQTT library, a messaging protocol called « publish-subscribe » based on the TCP/IP protocol which is natively integrated in the TTN application, and which has a partial but usable documentation.

For programming in Python, we chose to use the Visual Studio Code editor which is a simple and efficient choice, and which is very popular nowadays especially within the community of this language.



Figure 19 – MQTT logo

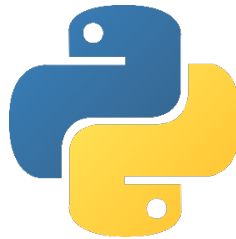


Figure 21 –
Python logo

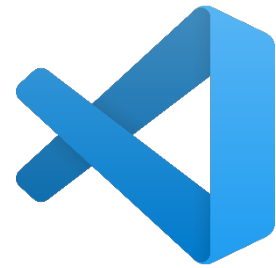


Figure 20 - Visual
Studio Code logo

We also wanted to make the Python script to start automatically at the server startup, which I achieved it by using a service and the systemctl package usually integrated in Linux distributions such as Debian.

Now that all the necessary elements for the communication have been decided and gathered, we'll now see what I did for my part in detail, what I had to think to find solutions and what I configured.

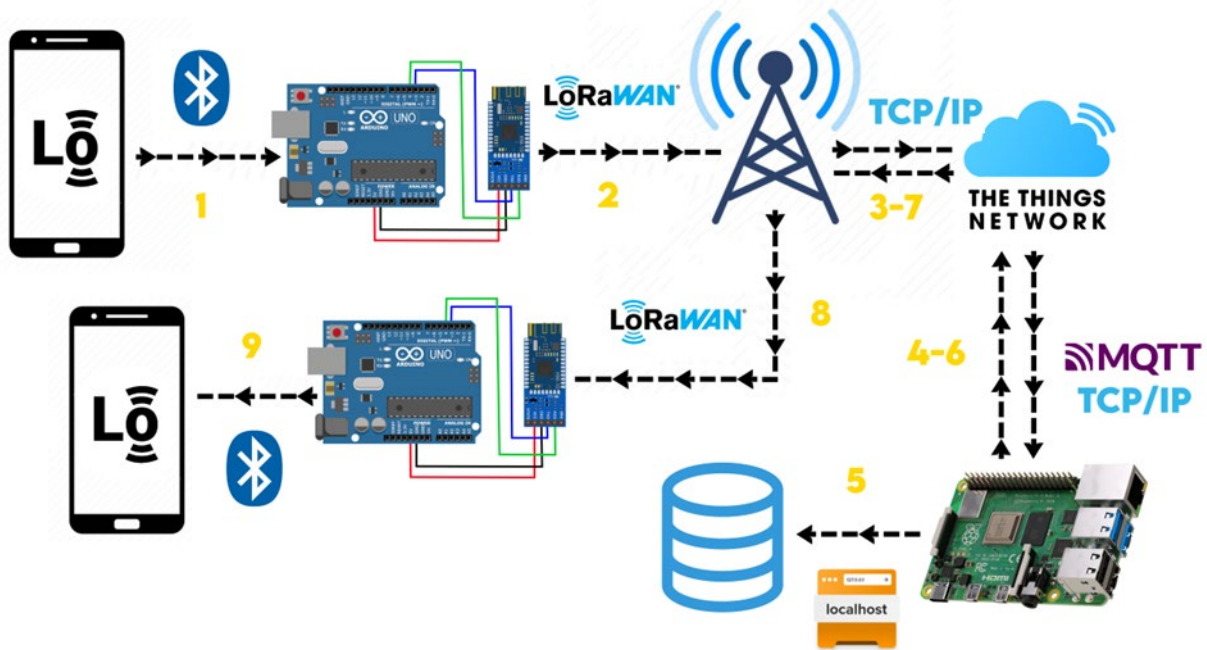


Figure 22 - Diagram of the complete communication between all devices

Note that in the past, communication between a gateway and TTN was done using UDP. Fortunately, the TTN technical team has implemented a TCP-based protocol to avoid data loss (which would be problematic in the case of emergency messaging).

III. Project implementation and in-depth analysis

1. Bluetooth Low Energy 4.0 (BLE)

1.1. Module HM-10

The module I chose to use for our boards is based on Bluetooth Low Energy technology (BLE) 4.0.

Used in various fields such as medicine, security, connected objects or recently for Covid-19 case finding (in the TousAntiCovid application), this technology aims to be a low power version of simple Bluetooth, despite the amount of data it can transmit.

Specifications	Classic Bluetooth	Bluetooth Low Energy (BLE)
Range	100 m	Greater than 100 m
Data Rate	1-3 Mbps	1 Mbps
Application Throughput	0.7 -2.1 Mbps	0.27 Mbps
Frequency	2.4 GHz	2.4 GHz
Security	56/128-bit	128-bit AES with Counter Mode CBC-MAC
Robustness	Adaptive fast frequency hopping, FEC, fast ASK	24-bit CRC, 32-bit Message Integrity Check
Latency	100 ms	6 ms
Time Lag	100 ms	3 ms
Voice Capable	Yes	No
Network Topology	Star	Star
Power Consumption	1 W	0.01 to 0.5 W
Peak Current Consumption	less than 30 mA	less than 15 mA

Figure 23 - Comparative sheet between BLE and Bluetooth

Indeed, without MTU* negotiation, the transfer and reception limit of this protocol does not exceed 20 bytes of payload per packet. So, I had to adapt to the technical problems of the application team which failed in implementing this functionality to bypass this limit.

In order not to slow down the progress of the project, we chose to continue with our tasks without taking this detail into account.

1.2. Bluetooth communication

1.2.1. Connections and configuration at the code level

When we want to develop code for Arduino hardware, or even in programming embedded systems in general, we must assign to the different pins* the features we want to implement.

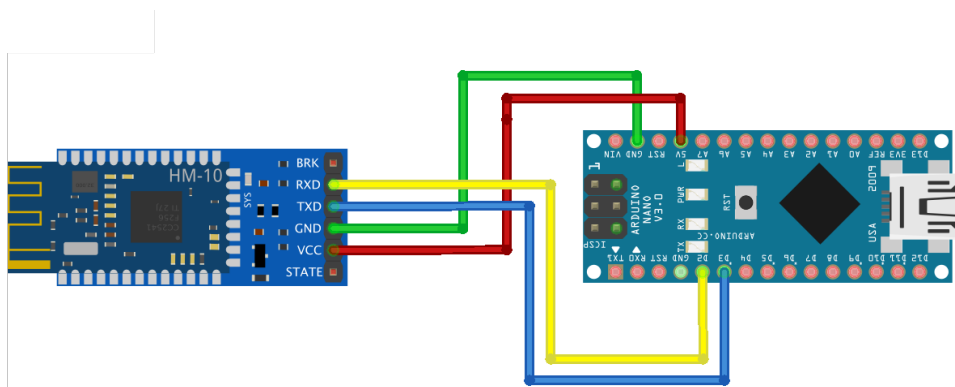
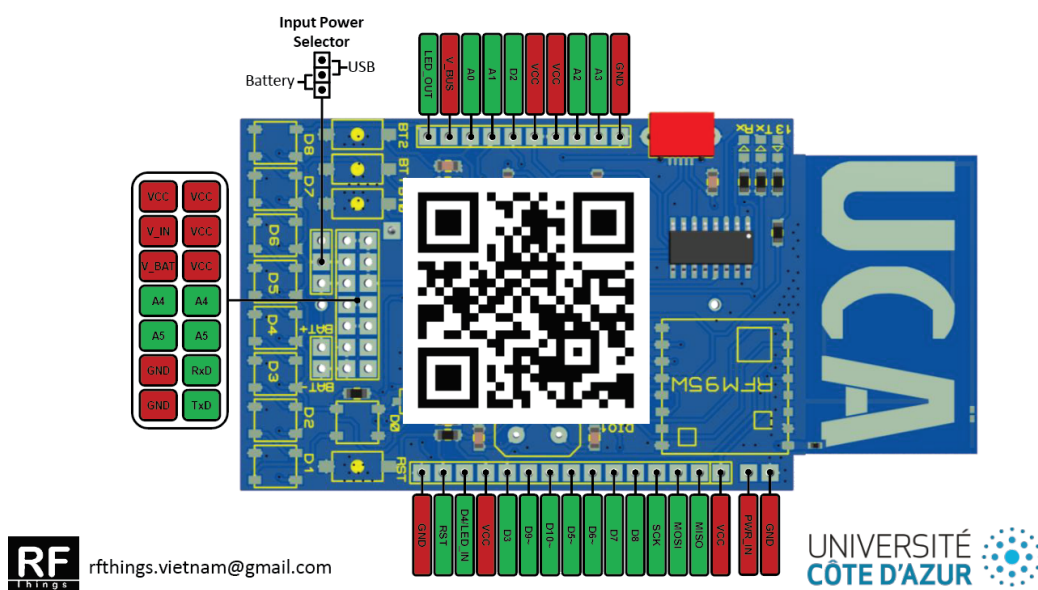


Figure 24 - Connecting the Bluetooth module to the UCA Education Board



rfrthings.vietnam@gmail.com

Figure 25 - UCA Education Board Datasheet

For example, the HM-10 module has 4 separate pins:

- TX: this is the transmission port, the one that will send messages from the board.
- RX: the receiving port, the one that will receive and send messages to the card.
- VCC: Voltage Common Collector is the positive supply voltage of an integrated circuit containing bipolar junction transistors.
- GND: abbreviation for Ground is the reference point for all signals where all voltages can be measured from it.

The pins I'm mainly interested in are the TX and RX pins.

Once the wiring was made, and the activation LED lit up (visual signal validating my wiring), I was able to start, at first, the Bluetooth communication.

```
1  #include <SoftwareSerial.h>
2
3  // Setting up the bluetooth serial
4  SoftwareSerial HM10(2, 3); // RX = 2, TX = 3
5
6  char appData;
7  String inData = "";
8
9  void setup() {
10     Serial.begin(9600);
11     HM10.begin(9600);
12     Serial.println("Started successfully.");
13 }
14
15 void loop() {
16     HM10.listen(); // listen the HM10 port
17     while (HM10.available() > 0) { // if HM10 sends something then read
18         appData = HM10.read();
19         inData += String(appData); // save the data in string format
20     }
21
22     if (inData.indexOf("|") != -1) {
23         Serial.println("Message envoyé : " + inData); // Will be replaced by send function in the future
24         inData = "";
25     }
26 }
27
```

Figure 26 - Minimalist code for Bluetooth data reception

For you to understand how an Arduino program works, I will explain in detail how this code works and how to use the IDE.

To begin with, its syntax is identical to that of C and C++. We can also distinguish two essential functions for the code compilation: the `setup()` and `loop()` functions. The first function will be executed when the board is started, while the second will be executed after the first and will run continuously until the board is shut down.

The Arduino IDE allows listening to the serial ports (USB port where the board is connected) to retrieve and send data. The interface allowing this is called "Serial Monitor" and works in much the same way as the standard input/output consoles we are used to.

To be able to display text with a correct encoding, it is necessary to define the transmission speed on the port (also called "Baud Rate" of unit "baud" or "bps" for byte-per-second) to avoid displaying bad encoded characters like this:

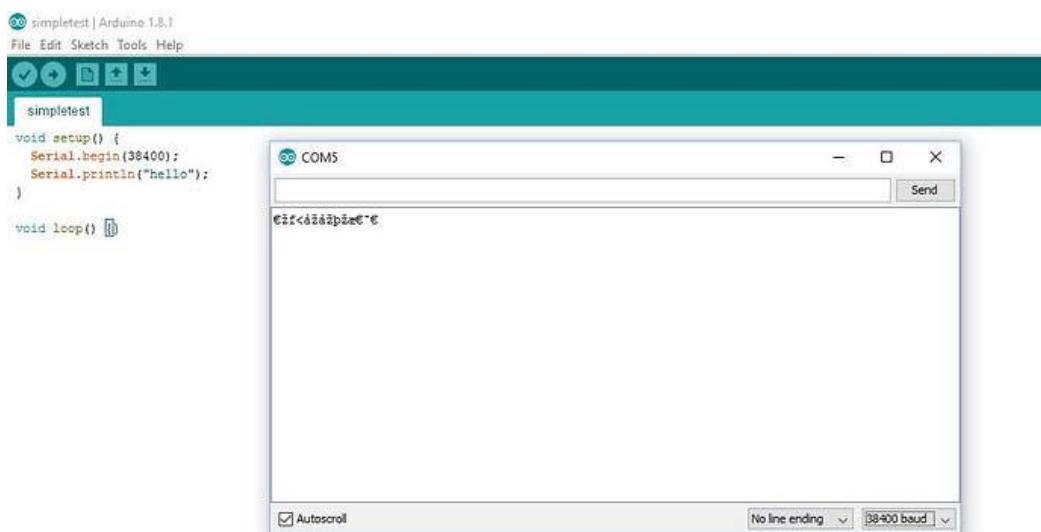


Figure 27 - Serial Monitor and Arduino IDE with a bad Baud Rate

It sometimes happens that, even if you have selected a baud rate identical to the one written in the code, encoding problems may occur. In this case, it is because the port or cable is not adapted to such a speed. Indeed, the higher the baud rate, the higher the transmission and reception speed. If you go too high, you start to see errors on reception, because the clocks and sampling periods* cannot keep up. The safest and most common value for microcontrollers today is 9600 bps.

Once initialized using the `begin()` method, the Serial will be ready to be used. We do the same so that the HM-10 module can be synchronized at the same speed as the board using the `SoftwareSerial.h` library, an Arduino native library that allows serial communication on other pins of an Arduino board, using software to reproduce the functionality (hence the name « SoftwareSerial »).

1.2.2. How the receiving works

To test the data reception of our program, we used the mobile application "BLE Scanner (Connect & Notify)" which allowed us to connect very quickly to the board and to send messages in the form of character strings or bytes to the board. It is also thanks to this application that we noticed the failure of sending data from the board to the phone.

Receiving in BLE using the HM-10 module works in the same way as a stream that is read character by character. Therefore, I had to adapt my code to be able to distinguish two messages between each other using a specific separator. I chose the ' | ' character (vertical bar).

1.2.3. Sending issue

Despite all our efforts and multiple tests on various applications, we were unable to send messages via Bluetooth (only receive them). On my side, I couldn't understand what the problem could be. The only method proposed by the library to send via Bluetooth was not functional.

In order not to fall behind, we decided to temporarily stop working on sending messages via Bluetooth and to focus on sending messages from the board to TTN, for the moment. We could afford it since this feature will only be needed for receiving from a board to the phone (downlink part).

2. Sending to The Things Network from a board

2.1. First adaptation of the code

2.1.1. Adaptation methodology

While Lyne was configuring our TTN application, I took care of the program that allows us to send our messages to it from a board. I started my program from an already existing file that I could find in a GitHub repository of Mr. Fabien Ferrero.

Originally written by Thomas Telkamp and Matthijs Kooijman, two former employees of The Things Network, this script allows the creation of uplinks at regular intervals, as well as the reception of downlinks in parallel. Among other things, this program allows communication between the board and TTN.

This program requires the installation of a particular library: "lmic.h". Be careful not to install the obsolete version of IBM but rather to use the one with the full name: "MCCI Catena Arduino LoRaWAN LMIC Library". This library allows the

use of the basic Arduino LMIC library, which allows LoRaWAN communications between devices, to communicate with TTN. It is also a more elaborate, complete and more maintained version of its predecessor by IBM.

The first step in modifying this program was to change the values of the European frequency bands to those used in Vietnam. The radio bands to use were given to me by Mr. Souébou Bouro.

```
LMIC_setupChannel(0, 921400000, DR_RANGE_MAP(DR_SF12, DR_SF7), BAND_CENTI); // g-band
LMIC_setupChannel(1, 921600000, DR_RANGE_MAP(DR_SF12, DR_SF7B), BAND_CENTI); // g-band
LMIC_setupChannel(2, 921800000, DR_RANGE_MAP(DR_SF12, DR_SF7), BAND_CENTI); // g-band
LMIC_setupChannel(3, 922000000, DR_RANGE_MAP(DR_SF12, DR_SF7), BAND_CENTI); // g-band
LMIC_setupChannel(4, 922200000, DR_RANGE_MAP(DR_SF12, DR_SF7), BAND_CENTI); // g-band
LMIC_setupChannel(5, 922400000, DR_RANGE_MAP(DR_SF12, DR_SF7), BAND_CENTI); // g-band
LMIC_setupChannel(6, 922600000, DR_RANGE_MAP(DR_SF12, DR_SF7), BAND_CENTI); // g-band
LMIC_setupChannel(7, 922800000, DR_RANGE_MAP(DR_SF12, DR_SF7), BAND_CENTI); // g-band
LMIC_setupChannel(8, 922700000, DR_RANGE_MAP(DR_FSK, DR_FSK), BAND_MILLI); // g2-band
```

Figure 28 - Lines of code associated with Vietnamese radio frequencies

After that, I simply put the right identifiers and keys in the right places, which were provided to me by Lyne, to establish the communication between the board and our TTN application.

```
// LoRaWAN end-device address (DevAddr)
static const u4_t DEVADDR = 0x260B9BCB;

// LoRaWAN NwksKey, network session key
// This is the default Semtech key, which is used by the early prototype TTN
// network.
static const PROGMEM u1_t NWKSKEY[16] = {0x6F, 0x7A, 0xAC, 0x3C, 0x34, 0x9B, 0x2C, 0x5F, 0x25, 0xDB, 0x6C, 0x7C, 0x2B, 0xFD, 0x85, 0x53};

// LoRaWAN AppSKey, application session key
// This is the default Semtech key, which is used by the early prototype TTN
// network.
static const PROGMEM u1_t APPSKEY[16] = {0x41, 0x79, 0xB0, 0x21, 0x14, 0x29, 0x9D, 0xAD, 0x10, 0x31, 0x71, 0x7A, 0xA8, 0xDC, 0xFF, 0x0C};
```

Figure 29 - Hexadecimal values to fill in the code

The meanings of all these global variables are:

- DEVADDR: This is the hexadecimal address of the device, randomly generated in the TTN application and then associated in the code with the device.
- NWKSKYKEY: The network session address is used to validate the integrity of messages.
- APPSKYKEY: The session application address that allows encryption and decryption of the sent and received packet.

Now that everything was set up, I could start sending test messages to the TTN application that Lyne had finished configuring.

2.1.2. Results obtained

The first tests showed that the communication was working well. The code sent « Hello, World! » every 30 seconds

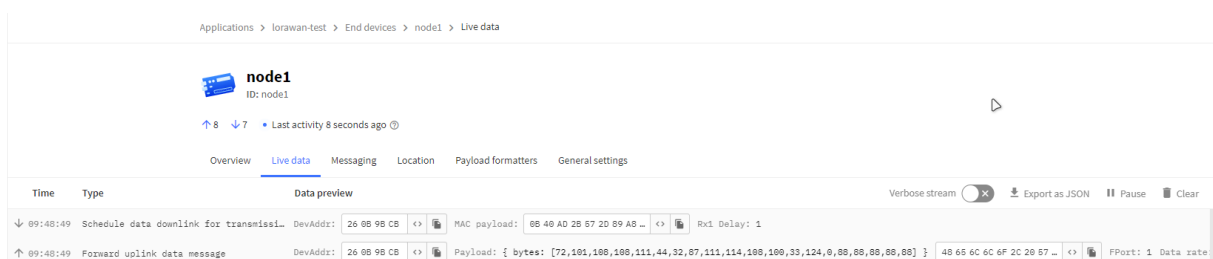


Figure 30 - First results obtained in the console of the TTN application by node1

We can see that the bytes correspond well to the message sent if we trust the ASCII table: 72=H, 101=e, 108=l, 111=o, etc. The 88 represent the character 'X', the end character of the character table which is used as a buffer in the code.

2.2. Second adaptation of the code

2.2.1. Merge the program with the Bluetooth program

The tests now conclusive, I had to modify the sending function of the program which functioned on a time interval with queue so that this one only sends at once.

Once this was done, all I had to do was add my Bluetooth reception program so that, as soon as the board receives a message in Bluetooth, it transmits it to TTN.

```
void loop() {
  os_runloop_once();

  HM10.listen(); // listen the HM10 port
  while (HM10.available() > 0) { // if HM10 sends something then read
    appData = HM10.read();
    inData += String(appData); // save the data in string format
  }

  if (inData.indexOf("|") != -1) {
    Serial.println("Message envoyé : " + inData);
    inData.toCharArray(mydata, inData.length() + 1);
    do_send(&sendjob);
    inData = "";
    print_data();
  }
}
```

Figure 31 - Final code running continuously on a board

The way the code works is as follows:

1. The board waits for the HM-10 module to send a message that it has received from a device connected to it,

2. When a message is received, it is recovered character by character to form a string,
3. This string, once it contains the separator character, is written in a global variable of type character array (like a buffer) of fixed size. The characters not filled by the string in the array are 'X',
4. The message is sent in the form of an array of integers representing the ASCII code of the characters (format required by the method that allows sending).

You can of course find the complete commented code in the appendix.

2.2.2. Results obtained

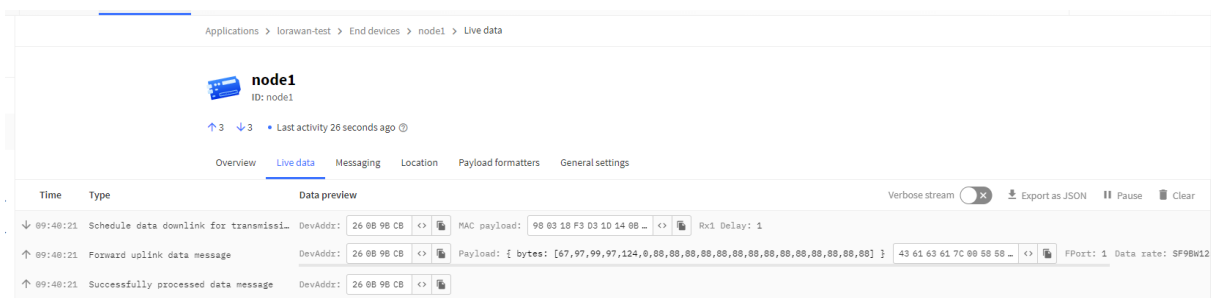


Figure 32 - Payload received on the TTN application console by node1

Not surprisingly, the two codes, separately functional, are also functional when put together. We can see that the message has been received and that the bytes correspond to the array of characters in the Arduino code (message + character '|' + sequence of 'X' that completes the empty indices of the array).

3. Python script and Raspberry Pi server

3.2. Python Script

Before writing the code, I first created an environment with the command `python -m venv .env` to avoid any conflict between the libraries I will install and those I already had installed on my computer.

Python's `virtualenv` is a tool used to create an isolated workspace for a Python application. It has various advantages such as the ability to install modules locally, export a working environment, and execute a Python program in that environment.

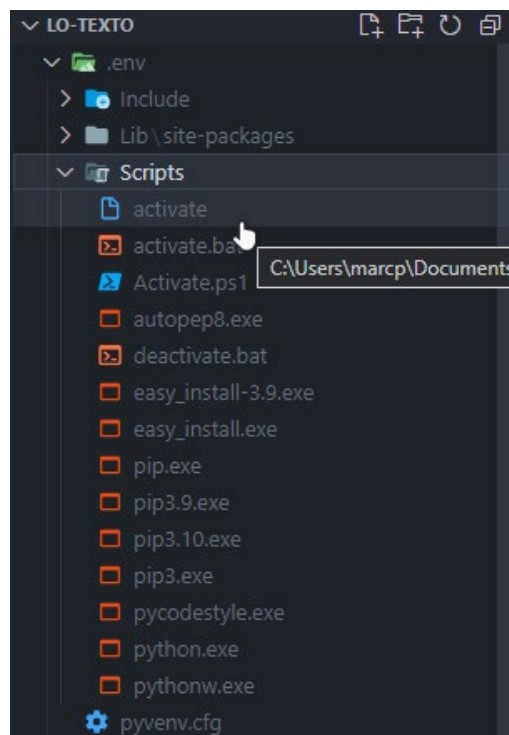


Figure 33 - .env folder created with the activation file of the latter

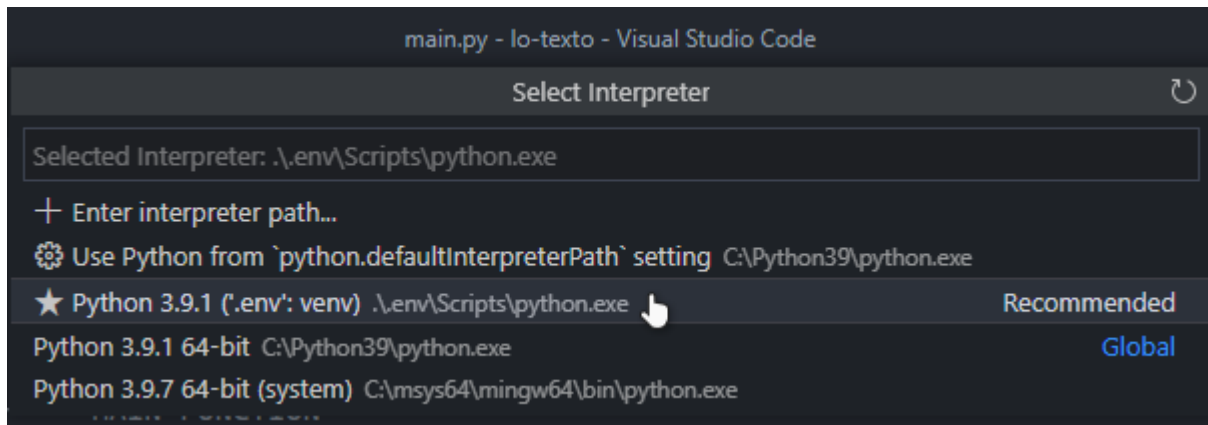


Figure 34 - Selection of the environment interpreter

Once this was done, I could launch the installation of the MQTT library with the command `pip install paho-mqtt` (Eclipse Paho being an implementation of MQTT).

3.1.1. Uplink listening

To access the TTN application from the Python script, MQTT requires an authorization key (or API key). So, I asked Lyne to generate one for me so that I could perform my tests.

Once the key was generated, I followed the documentation of Eclipse Paho MQTT for Python to do, at first, the listening on the uplinks.

```
1 import paho.mqtt.client as mqtt
2
3
4 QOS = 0 # Quality of Service (0 = sent at most 1 time, 1 = sent at least, 2 = sending until response of receiver)
5 TTN_APP_ID = "lorawan-test-stage@ttn"
6 TTN_API_KEY = "NNSXS.CJSXORVTBM2DMPGDLWSWASMAGKA7QJPIC7EHOY.D24W07IV72YZTBPJGPF720XBEM7NKRXS363RBWPUCG6KYVGH30TQ"
7 TTN_HOSTNAME = "eu1.cloud.thethings.network"
8 TTN_PORT = 8883
9
10 def on_message(client, userdata, msg):
11     print(msg.topic, msg.payload)
12
13 def main():
14
15     # Creating client
16     client = mqtt.Client()
17     client.on_message = on_message
18     client.username_pw_set(TTN_APP_ID, TTN_API_KEY)
19
20     # IMPORTANT - this enables the encryption of messages
21     client.tls_set()
22
23     # Connecting to the broker
24     client.connect(TTN_HOSTNAME, TTN_PORT)
25
26     # Subscribing to ALL topics
27     client.subscribe("#", QOS)
28
29     # Looping for ever
30     client.loop_forever()
31
32
33 if __name__ == "__main__":
34     main()
```

Figure 35 - Minimalist code for listening to uplinks in Python

This code allows me to display in the Visual Studio Code console all the messages that the TTN application receives in the form of a JSON. The '#' represents all the « topics » (or broadcast strings) on which the client will listen.

To get the important information in this message, I used the `eval()` function of Python which allows to evaluate a string in order to eventually get an object or to execute it if it is valid.

Indeed, as dictionaries in Python are very similar to JSON objects, I was able to do the conversion by putting the payload of type `str` and JSON-formatted into the function so it can return me a value of type `dict`.

3.1.2. Database

To insert into the database, I did not use any particular library to perform these actions. As the script would be directly hosted on the same server that hosts the database, I didn't find this necessary and opted for a command-line approach.

It is from Bash commands that I perform all the actions I want to do, thanks to the native library called « `os` » which allows the use of commands of the operating system on which Python is located.

The procedure for adding to the database from my script is simple:

1. Retrieving interesting data from the dictionary created from the JSON object,
2. Open MySQL, select the database to use,
3. SQL query of type `INSERT INTO`.

You can find in appendix the whole Python code to see the different commands and redirections to keep log files.

3.1.3. Sending downlinks

To send downlinks, I also followed the documentation and came up with the following piece of code:

```

120 def send_downlink(client: mqtt.Client, payload: dict):
121     """Send a downlink message to the TTN API"""
122
123     fport = 3
124
125     # Converting payload content (array of int) to string (ascii) and then to hexadecimal
126     hexadecimal_payload = binascii.hexlify(
127         bytes(bytes_to_ascii(payload["uplink_message"]["decoded_payload"]["bytes"]), encoding='utf-8')).decode()
128
129     # Target end_device_id // TODO : CURRENTLY DEVICE_ID IS THE SENDER AND WE NEED TO FIND A WAY TO PUT THE RECEIVER INSTEAD
130     end_device_id = "node2"
131
132     # Preparing the downlink message's topic (channel where it will be sent)
133     topic = "v3/" + TTN_APP_ID + "/devices/" + \
134         end_device_id + "/down/push"
135
136     # Convert hexadecimal payload to base64
137     base64_payload = base64.b64encode(
138         bytes.fromhex(hexadecimal_payload)).decode()
139
140     # Publishing downlink
141     msg = '{"downlinks":[{"f_port": ' + \
142         str(fport) + ', "firm_payload": "' + \
143         base64_payload + ', "priority": "NORMAL"}]}'
144     client.publish(topic, msg, QoS)

```

Figure 36 - Minimum code to perform a downlink

First, we convert the packet in hexadecimal bytes, then we convert it in base64, and we send the message on the downlink of the application.

Mr. Nhan Le Thanh told me that I only had to send the downlink to one end device (node2), for the moment.

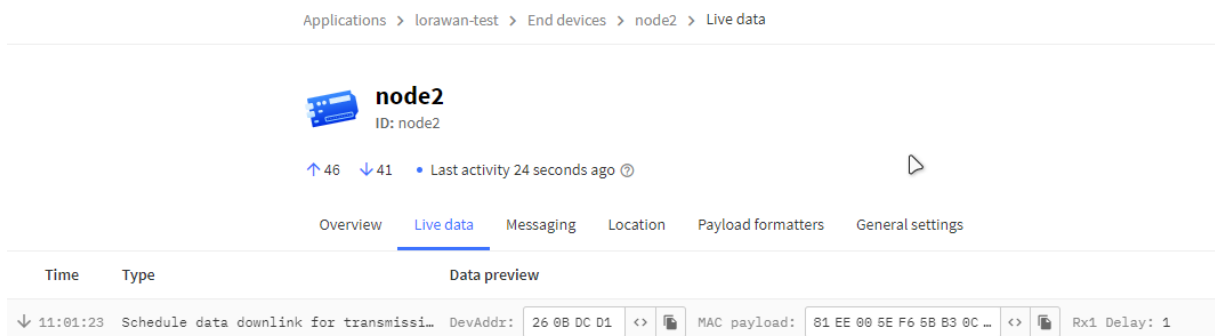


Figure 37 - Downlink display in the application console for node2

Unfortunately, the message was not received by the board, we are currently waiting for an answer from Mr. Fabien Ferrero who had assured us that his code allowed boards to receive downlinks.

3.2. Setting up the Raspberry Pi server

While Lyne was configuring the database on this server, I oversaw installing the necessary tools to make it work, including the use of Python and Git. We use the SSH protocol to connect and manipulate this server remotely with an administrator account.

3.2.1. Installation

To install tools and software on the server, you must use the command « apt-get » specific to Linux operating systems.

I first created a folder in which I store our GitHub repository. For that, I used the Git tool that I installed on the server beforehand. I then installed Vim to be able to modify files on the server if needed.

Finally, I installed Python in its 3.9 version which is the one I use on my computer, and which is considered as the most stable at the moment.



Figure 39 - Vim logo, text editor on console



Figure 38 - Logo of Git, decentralized version management tool

I also installed the Python modules I use in my script with pip commands. Pip is the Python package manager tool.

Once these tools were installed, all I had to do was to make the Python script launch automatically at each server startup.

4. Adaptation with the mobile application

The mobile application is now operational, so we were able to test the sending of messages very quickly since it was able to connect to our boards automatically.

4.1. Limited messages and firmware

Because of the Bluetooth Low Energy limitation, we had to find a way around this restriction. I proposed the idea of assigning hexadecimal identifiers to the predefined messages (of which there are 13 now). The messages sent would then be like something like « 1579ACE| ». I then had to add the corresponding messages to the database so that I could retrieve these identifiers and store the real messages in the database.

4.2. Adaptation of the Python script

I modified my Python script and added a SQL query that converts from hexadecimal to decimal and then performs a query for each identifier to retrieve the corresponding messages before inserting the newly created message.

5. Return on the remaining features to be implemented

At this point, there is not much left to do. The two remaining features to implement are the retrieval of data from TTN to a board, and the sending of the board via Bluetooth to the mobile application.

IV. Obtained results

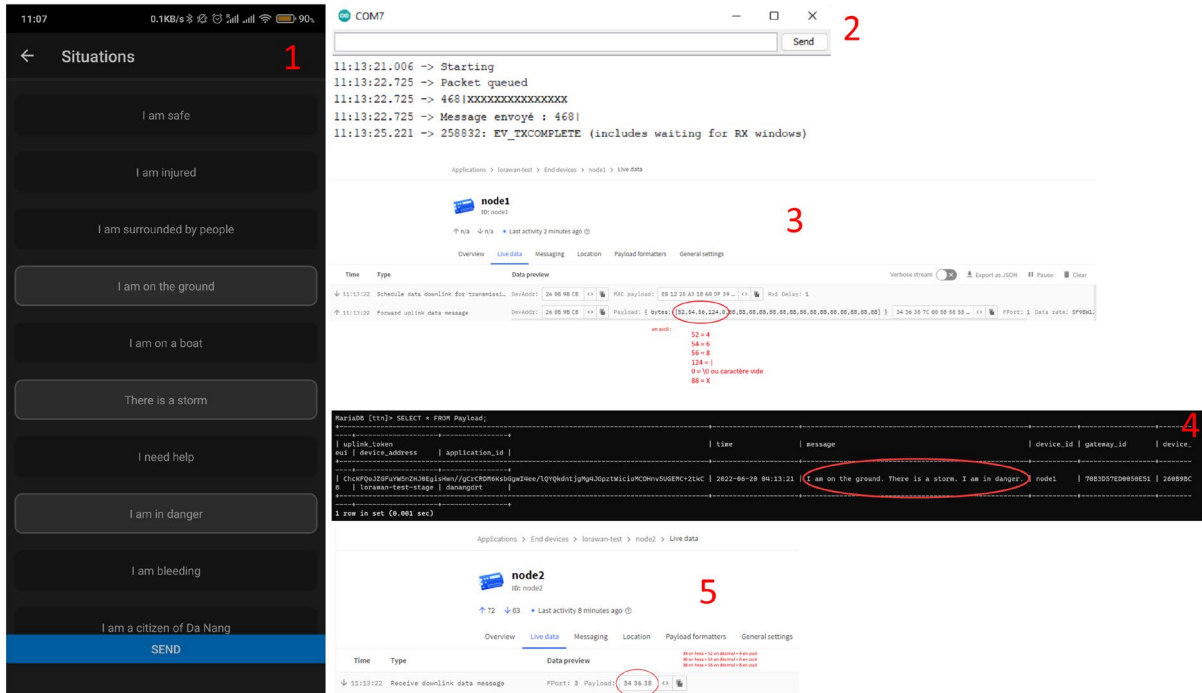


Figure 40 - Complete picture of the communications made so far

As shown in this picture, we managed to achieve the communication from the application to the server which sends back a downlink. We did not manage to go further in the project.

V. Conclusion

This internship was very enriching for me, both in terms of technical skills and the discovery of another culture.

Despite the impediment we encountered, we are still trying to figure out a solution to fix the downlink and Bluetooth issues until the end of our internship on July 17th. Furthermore, I was allowed to learn a lot about LoRa and LoRaWAN communications, embedded system programming, and the use of TTN's API with the MQTT protocol during this internship. I'm truly thankful for this opportunity.

References

LMIC for Arduino Documentation:

<https://github.com/mcci-catena/arduino-lmic>

LoRa and LoRaWAN code examples for UCA Education Board:

https://github.com/FabienFerrero/UCA_Education_Board

TheThingsNetwork v3 Documentation:

<https://www.thethingsindustries.com/docs/getting-started/>

Eclipse Paho (MQTT Integration) documentation for Python:

<https://www.eclipse.org/paho/index.php?page=clients/python/docs/index.php>

Using HM-10 Module with an Arduino board:

<https://how2electronics.com/bluetooth-low-energy-tutorial-with-hm-10-ble-4-0-arduino/>

DNIIT website

<https://dniit-i3s.cnrs.fr/fr/node/118>

Python documentation

<https://docs.python.org/fr/3.9>

Arduino documentation

<https://www.arduino.cc/reference/en/>

Technical appendices

```

.....
This uses ABP (Activation-by-personalisation), where a DevAddr and
Session keys are preconfigured (unlike OTAA, where a DevEUI and
Application key is configured, while the DevAddr and session keys are
assigned/generated in the over-the-air-activation procedure).

Do not forget to define the radio type correctly in config.h.
.....
//.....
// Region definition (will change de frequency bands)
// Define only 1 country
//
//#define CFG_EU 1
#define CFG_VN 1
//.....
#include <lmic.h>
#include <hal.h>
#include <SPI.h>
#include <SoftwareSerial.h>

// LoRaWAN end-device address (DevAddr)
static const u1_t DEVADDR = 0x260B98CB;

// LoRaWAN NwkSKey, network session key
// This is the default Semtech key, which is used by the early prototype TTN
// network.
static const PROGMEM u1_t NWSKEY[16] = {0x6F, 0x7A, 0x9C, 0x3C, 0x34, 0x9B, 0x2C, 0x5F, 0x25, 0x0B, 0x6C, 0x7C, 0x2B, 0x7D, 0x85, 0x53};

// LoRaWAN AppSKey, application session key
// This is the default Semtech key, which is used by the early prototype TTN
// network.
static const PROGMEM u1_t APPSKEY[16] = {0x41, 0x79, 0xB0, 0x21, 0x14, 0x29, 0x9D, 0xAD, 0x10, 0x31, 0x71, 0x7A, 0xA8, 0xDC, 0xFF, 0x0C};

// Setting up the bluetooth serial
SoftwareSerial HM10(2, 3); // RK = 2, TX = 3

char appData;
String lnData = "";

// These callbacks are only used in over-the-air activation, so they are
// left empty here (we cannot leave them out completely unless
// DISABLE_JOIN is set in config.h, otherwise the linker will complain).
void os_getEui48(u1_t* buf) {}
void os_getDevEui(u1_t* buf) {}
void os_getDevKey(u1_t* buf) {}

static uint8_t mydata[] = "XXXXXXXXXXXXXXXXXXXX"; // BLK limits to 20 max length for payload
static osjob_t sendjob;

// Schedule TX every this many seconds (might become longer due to duty
// cycle limitations).
const unsigned TX_INTERVAL = 1;

// Pin mapping
const lmic_pinmap lmic_pins = {
    .nss = 10,
    .miso = LMIC_UNUSED_PIN,
    .mosi = 8,
    .dio = {3, 7, 6},
};

void onEvent(ev_t ev) {
    Serial.print(os_getTime());
    Serial.print(" ");
    switch (ev) {
        case EV_SCAN_TIMEOUT:
            Serial.println(F("EV_SCAN_TIMEOUT"));
            break;
        case EV_BEACON_FOUND:
            Serial.println(F("EV_BEACON_FOUND"));
            break;
        case EV_BEACON_MISSED:
            Serial.println(F("EV_BEACON_MISSED"));
            break;
        case EV_BEACON_TRACKED:
            Serial.println(F("EV_BEACON_TRACKED"));
            break;
        case EV_JOINING:
            Serial.println(F("EV_JOINING"));
            break;
        case EV_JOINED:
            Serial.println(F("EV_JOINED"));
            break;
        case EV_RFU1:
            Serial.println(F("EV_RFU1"));
            break;
        case EV_JOIN_FAILED:
            Serial.println(F("EV_JOIN_FAILED"));
            break;
        case EV_REJOIN_FAILED:
            Serial.println(F("EV_REJOIN_FAILED"));
            break;
        case EV_TXCOMPLETE:
            Serial.println(F("EV_TXCOMPLETE (includes waiting for RX windows)"));
            if (LMIC.txrxFlags & TXRX_ACK)
                Serial.println(F("Received ack"));
            if (LMIC.dataLen) {
                Serial.print(F("Received "));
                Serial.print(LMIC.dataLen);
                Serial.println(F(" bytes of payload"));
                for (int i = 0; i < LMIC.dataLen; i++)
                    if (LMIC.frame[LMIC.dataBeg + i] < 0x10) {
                        Serial.print(F("0"));
                    }
                Serial.print(LMIC.frame[LMIC.dataBeg + i], HEX);
            }
            Serial.println("");
        }
        break;
        case EV_LOST_TSYNC:
            Serial.println(F("EV_LOST_TSYNC"));
            break;
        case EV_RESET:
            Serial.println(F("EV_RESET"));
            break;
        case EV_RXCOMPLETE:
            // data received in ping slot
            Serial.println(F("EV_RXCOMPLETE"));
            break;
        case EV_LINK_DEAD:
            Serial.println(F("EV_LINK_DEAD"));
            break;
        case EV_LINK_ALIVE:
            Serial.println(F("EV_LINK_ALIVE"));
            break;
        default:
            Serial.println(F("Unknown event"));
            break;
    }
}

void do_send(osjob_t s) {
    // Check if there is not a current TX/RX job running
    if (LMIC.opmode & OP_TXRXPEND) {
        Serial.println(F("OP_TXRXPEND, not sending"));
    } else {
        // Prepare upstream data transmission at the next possible time.
        LMIC_setTxData2(1, mydata, sizeof(mydata) - 1, 0);
        Serial.println(F("Packet queued"));
    }
    // Next TX is scheduled after TX_COMPLETE event.
}

```

Arduino code (part 1)

```

void setup() {
  Serial.begin(9600);
  HM10.begin(9600);
  Serial.println(F("Starting"));

  // LMIC init
  os_init();
  // Reset the MAC state. Session and pending data transfers will be discarded.
  LMIC_reset();

  /* This function is intended to compensate for clock inaccuracy (up to ±10% in this example),
   but that also works to compensate for inaccuracies due to software delays.
   The downside of this compensation is a longer receive window, which means a higher battery drain.
   So if this helps, you might want to try to lower the percentage (i.e. lower the 10 in the above call),
   often 1% works well already. */

  LMIC_setClockError(MAX_CLOCK_ERROR * 10 / 100);

  // Set static session parameters. Instead of dynamically establishing a session
  // by joining the network, precomputed session parameters are provided.
#ifdef PROGMEM
  // On AVR, these values are stored in flash and only copied to RAM
  // once. Copy them to a temporary buffer here, LMIC_setSession will
  uint8_t appskey[sizeof(APPSKEY)];
  uint8_t nwkskey[sizeof(NWKSKEY)];
  memcpy_P(appskey, APPSKEY, sizeof(APPSKEY));
  memcpy_P(nwkskey, NWKSKEY, sizeof(NWKSKEY));
  LMIC_setSession(0x1, DEVADDR, nwkskey, appskey);
#else
  // If not running an AVR with PROGMEM, just use the arrays directly
  LMIC_setSession(0x1, DEVADDR, NWKSKEY, APPSKEY);
#endif

#ifdef CFG_EU
  // Set up the 8 channels used
  LMIC_setupChannel(0, 868100000, DR_RANGE_MAP(DR_SF12, DR_SF7), BAND_CENTI); // g-band
  LMIC_setupChannel(1, 868300000, DR_RANGE_MAP(DR_SF12, DR_SF7B), BAND_CENTI); // g-band
  LMIC_setupChannel(2, 868500000, DR_RANGE_MAP(DR_SF12, DR_SF7), BAND_CENTI); // g-band
  LMIC_setupChannel(3, 867100000, DR_RANGE_MAP(DR_SF12, DR_SF7), BAND_CENTI); // g-band
  LMIC_setupChannel(4, 867300000, DR_RANGE_MAP(DR_SF12, DR_SF7), BAND_CENTI); // g-band
  LMIC_setupChannel(5, 867500000, DR_RANGE_MAP(DR_SF12, DR_SF7), BAND_CENTI); // g-band
  LMIC_setupChannel(6, 867700000, DR_RANGE_MAP(DR_SF12, DR_SF7), BAND_CENTI); // g-band
  LMIC_setupChannel(7, 867900000, DR_RANGE_MAP(DR_SF12, DR_SF7), BAND_CENTI); // g-band
  LMIC_setupChannel(8, 868800000, DR_RANGE_MAP(DR_FSK, DR_FSK), BAND_MILLI); // g2-band

#elif defined(CFG_VN)
  // Set up the 8 channels used
  LMIC_setupChannel(0, 921400000, DR_RANGE_MAP(DR_SF12, DR_SF7), BAND_CENTI); // g-band
  LMIC_setupChannel(1, 921600000, DR_RANGE_MAP(DR_SF12, DR_SF7B), BAND_CENTI); // g-band
  LMIC_setupChannel(2, 921800000, DR_RANGE_MAP(DR_SF12, DR_SF7), BAND_CENTI); // g-band
  LMIC_setupChannel(3, 922000000, DR_RANGE_MAP(DR_SF12, DR_SF7), BAND_CENTI); // g-band
  LMIC_setupChannel(4, 922200000, DR_RANGE_MAP(DR_SF12, DR_SF7), BAND_CENTI); // g-band
  LMIC_setupChannel(5, 922400000, DR_RANGE_MAP(DR_SF12, DR_SF7), BAND_CENTI); // g-band
  LMIC_setupChannel(6, 922600000, DR_RANGE_MAP(DR_SF12, DR_SF7), BAND_CENTI); // g-band
  LMIC_setupChannel(7, 922800000, DR_RANGE_MAP(DR_SF12, DR_SF7), BAND_CENTI); // g-band
  LMIC_setupChannel(8, 922700000, DR_RANGE_MAP(DR_FSK, DR_FSK), BAND_MILLI); // g2-band

#endif

  // Disable link check validation
  LMIC_setLinkCheckMode(0);

  // TTN uses SF9 for its RX2 window.
  LMIC.dn2Dr = DR_SF9;

  // Set data rate and transmit power for uplink (note: txpow seems to be ignored by the library)
  LMIC_setDrTxpow(DR_SF9, 20);
}

void print_data() {
  for (int i = 0; i < sizeof(mydata) / sizeof(mydata[0]); i++)
    Serial.print(char(mydata[i]));
  Serial.println("");
}

void loop() {
  os_runloop_once();

  HM10.listen(); // listen the HM10 port
  while (HM10.available() > 0) { // if HM10 sends something then read
    appData = HM10.read();
    inData += String(appData); // save the data in string format
  }

  if (inData.indexOf("|") != -1) {
    Serial.println("Message envoyé : " + inData);
    inData.toCharArray(mydata, inData.length() + 1);
    do_send(&sendjob);
    inData = "";
    print_data();
  }
}

```

Arduino code (part 2)

```

1 | # IMPORTS
2 |
3 |
4 | import paho.mqtt.client as mqtt
5 | import base64
6 | import binascii
7 | import os
8 |
9 |
10 | # ----- GLOBAL VARIABLES -----
11 |
12 | # Quality of Service :
13 | # 0 = sent at most once,
14 | # 1 = sent at least once,
15 | # 2 = sent until acknowledged (kind of handshake)
16 | QOS = 0
17 |
18 | # Findable in the TTN Application Overview tab and around (V3)
19 | TTN_APP_ID = "lorawan-test-stage@ttn"
20 | TTN_API_KEY = "NNSXS_C3JSORV7B2DHPGDLMSWASMGKA7Q3PIC7EH0V.D24M071V72Y2TBPJGPF720XBEM7NKR5X363RBPWUCG6KVVGH30TQ"
21 | TTN_HOSTNAME = "eu1.cloud.thethings.network"
22 | TTN_PORT = 8883
23 |
24 | # Name of the database where received uplinks infos will be stored
25 | DB_NAME = "ttn"
26 |
27 | # Debugs enabled
28 | DEBUG = True
29 |
30 | # ----- GLOBAL FUNCTIONS -----
31 |
32 |
33 |
34 | def bytes_to_ascii(bytes: list, separator: str = '|') -> str:
35 |     """Converts a list of bytes into a string, removes the first separator and take the left part
36 |
37 |     Args:
38 |         bytes (list): A list of bytes
39 |
40 |     Returns:
41 |         str: A string representing the bytes
42 |     """
43 |     a = ''.join([chr(c) for c in bytes]).split(separator)[0]
44 |     if DEBUG:
45 |         print(f"bytes={bytes}, ascii={a}")
46 |     return a
47 |
48 | def add_to_database(payload: dict) -> bool:
49 |     """Parse the payload into a sql query and send it to the database
50 |
51 |     Args:
52 |         payload (dict): The payload received from the TTN API
53 |
54 |     Returns:
55 |         bool: Whether the operation succeed or not
56 |     """
57 |     uplink_token = payload["uplink_message"]["rx_metadata"][0]["uplink_token"]
58 |     time = payload["uplink_message"]["rx_metadata"][0]["time"].split(
59 |         "+" )[0] + " " + payload["uplink_message"]["rx_metadata"][0]["time"].split("+")[1].split(".")[0]
60 |     decoded_message = bytes_to_ascii(
61 |         payload["uplink_message"]["decoded_payload"]["bytes"])
62 |     device_id = payload["end_device_ids"]["device_id"]
63 |     device_eui = payload["end_device_ids"]["dev_eui"]
64 |     device_address = payload["end_device_ids"]["dev_addr"]
65 |     application_id = payload["end_device_ids"]["application_ids"]["application_id"]
66 |     gateway_id = payload["uplink_message"]["rx_metadata"][0]["gateway_ids"]["gateway_id"]
67 |
68 |     msg_ids = tuple([int(x, 16) for x in decoded_message])
69 |     if len(msg_ids) <= 1:
70 |         msg_ids = f"({msg_ids[0]})"
71 |
72 |     # Preparing the select statement
73 |     select_statement = f"""
74 |     SELECT message FROM Message where message_id IN {msg_ids};
75 |     """
76 |
77 |     # Storing the query output to the result.txt file
78 |     os.system(
79 |         f"echo \"${select_statement}\" | sudo mysql ttn > /home/tvlic/query/result.txt")
80 |
81 |     # Retrieving the result from the file
82 |     full_message = None
83 |     with open("/home/tvlic/query/result.txt", "r") as f:
84 |         lines = f.readlines()
85 |         full_message = " ".join([line.replace("\n", '') for line in lines])
86 |
87 |     # Preparing the insert statement
88 |     insert_statement = f"""
89 |     INSERT INTO Payload VALUES ('{uplink_token}', '{time}', '{full_message}', '{device_id}', '{device_eui}', '{device_address}', '{application_id}', '{gateway_id}');
90 |     """
91 |
92 |     if DEBUG:
93 |         print(select_statement, insert_statement, sep='\n')
94 |
95 |     os.system(f"""
96 |     echo "${insert_statement}" | sudo mysql ttn > /home/tvlic/query/logs.txt;
97 |     """)
98 |
99 |     return True
100 |
101 |
102 |
103 | def on_message(client, userdata, msg):
104 |     """Callback function when a message is received from TTN"""
105 |
106 |     # Getting the payload, parsing it and sending it to the database
107 |     payload_as_dict = eval(msg.payload.decode("utf-8").replace("true", "True"))
108 |
109 |     # Adding to the database the payload and its content
110 |     try:
111 |         add_to_database(payload_as_dict)
112 |
113 |         send_downlink(client, payload_as_dict)
114 |
115 |     except Exception as e:
116 |         if DEBUG:
117 |             print("Error at:", e)
118 |
119 |
120 | def send_downlink(client: mqtt.Client, payload: dict):
121 |     """Send a downlink message to the TTN API"""
122 |
123 |     fport = 3
124 |
125 |     # Converting payload content (array of int) to string (ascii) and then to hexadecimal
126 |     hexadecimal_payload = binascii.hexlify(
127 |         bytes(bytes_to_ascii(payload["uplink_message"]["decoded_payload"]["bytes"]), encoding='utf-8')).decode()
128 |
129 |     # Transmit and device id /// TODO : CURRENTLY DEVICE_ID IS STATIC, WE NEED TO FIND A WAY TO MAKE IT CHANGEABLE
130 |     end_device_id = "node2"
131 |
132 |     # Preparing the downlink message's topic (channel where it will be sent)
133 |     topic = "v3/" + TTN_APP_ID + "/devices/" + \
134 |         end_device_id + "/down/push"
135 |
136 |     # Convert hexadecimal payload to base64
137 |     base64_payload = base64.b64encode(
138 |         bytes.fromhex(hexadecimal_payload)).decode()
139 |
140 |     # Publishing downlink
141 |     msg = {"downlinks": [{"f_port": fport, \
142 |         str(fport) + "f_payload": base64_payload, \
143 |         base64_payload + "priority": "NORMAL"}]}
144 |     client.publish(topic, msg, QOS)
145 |

```

```
146
147 # -----MAIN FUNCTION-----
148
149
150 def main():
151     # Creating client
152     client = mqtt.Client()
153     client.on_message = on_message
154     client.username_pw_set(TTN_APP_ID, TTN_API_KEY)
155
156     # IMPORTANT - this enables the encryption of messages
157     client.tls_set()
158
159     # Connecting to the broker
160     client.connect(TTN_HOSTNAME, TTN_PORT)
161
162     # Subscribing to ALL topics
163     client.subscribe("#", QOS)
164
165     # Looping for ever
166     client.loop_forever()
167
168
169 # -----MAIN CALL-----
170
171
172
173 if __name__ == "__main__":
174     main()
175
```

Python code (part 2)

Abstract

It is with a team of 3 other students that we develop an application allowing the transmission and reception of emergency messages using a specific communication protocol: the LoRaWAN.

Based on radio communication and belonging to the LoRa layer, LoRaWAN is a protocol that uses gateways to cross very large distances at a minimal cost.

The limits of LoRa communications are felt as soon as a few kilometers separate the two devices. Therefore, as this is a project aimed at people who may be far from the city center, it would be beneficial to them to have a more extensive communication tool.

Our project is therefore based on several gateways set up upstream in the city so that messages can be sent and received over greater distances. The idea is to connect the devices in LoRa to create a LoRaWAN network.