UCLouvain

École polytechnique de Louvain

Louvain-la-Neuve, a Smart City: a LoRa based and LTE assisted bike tracker

Authors: **Olivier Latteur, Hadrien Libioulle**
Supervisors: **Yves Deville, Ramin Sadre**
Reader: **François De Keersmaeker**
Academic year 2021–2022
Master [120] in Computer Science and Engineering

Acknowledgements

Before diving into this master thesis, we would like to thank our supervisors, Prof. Yves Deville and Prof. Ramin Sadre, for their support, advices and reviews throughout the year without which this work would have been much more difficult.

We also would like to thank our external reader, François De Keersmaeker, for his time and involvement in this master thesis as well as the members of the WELCOME platform for their trust in the use of the equipment necessary to carry out the measurements

Finally, we would like to thank our families for having provided us great technical, psychological and logistical support.

Contents

Chapter 1

Introduction

The initial objective of this master thesis was to find and implement an application of the Internet of Things (IoT) in the context of smart cities, which is the concept of future connected, optimised and sustainable city thanks to digitisation. This application should be suitable and useful in the city of Louvain-la-Neuve. While some previous master thesis [64] [44] [53] were focused on the study of LoRaWAN, this one has the objective to realise a finished solution with a first proof-of-concept prototype. In this context, we have chosen to explore the domain of connected bicycles for theft prevention.

Indeed, 100.000 is the estimated number of stolen bikes in Belgium every year [34]. Even if this number is decreasing on a national scale, it remains important and is still growing in big cities. Moreover, the value of the bikes tends to be much higher as the number of electric bikes continues to grow. This problem is unfortunately not specific to Belgium and takes an important place in many European countries [62]. These thefts are a real problem in an environmental context where cars should be replaced as much as possible by soft mobility, that is public transport , low-emission means of transport and bikes.This problem has the effect of discouraging the use of bikes as means of transportation for daily travel and increases the feeling of insecurity.

In order to make a contribution and to fight against this worldwide problem, this master thesis aims to design and make a real life prototype of a personal device to detect bicycle theft and being able to track the bike to retrieve it. This device is intended to be used

as a standalone solution by using existing networks and will be battery operated to be as autonomous as possible.

Moreover, the usage of IoT technologies is very suitable in this case as it provides a wide coverage, cheap costs of operation, long battery life and a small spatial footprint, as IoT devices are often small.

Objectives

The system proposed in this master thesis should at least meet the following requirements in order to address effectively the problems of bicycle thefts:

- The device should be placed on the bike itself in a discreet manner so the thief will not be aware of the device;
- It should track the bike position depending on some parameters defined by the user. These parameters allow to personalise the device behaviour in order to play on the battery life of the device according to the user needs and circumstances ;
- It must be powered by a battery and last at least 1 week without recharge, few weeks or months would be obviously better;
- The user should be able to interact with the device through a user interface (a mobile application) to retrieve information and update/modify the device parameters.
- The device should have an adequate network reception to provide connectivity most of the time in order to send data and receive configuration updates.

.

Structure

Chapter 2: Technological context This chapter will explain the different used technologies and the important concepts, costs, benefits and opportunities related to them. This will serve as the baseline technology knowledge for this master thesis. This chapter will also present similar solutions that already exist as well as previous related master thesis.

Chapter 3: Design of the bike tracker This chapter will explain the whole design and design choices of the proposed solution. It will cover hardware and software aspects.

Chapter 4: Experiments The experiments done with the device to evaluate the solution and technologies will be explained in this chapter. It will be limited to explain the experiments and give their results.

Chapter 5: Analysis and discussion This chapter will discuss the experiment results and what we can conclude from them compared to the requirements.

Chapter 6: Conclusion This chapter will conclude this master thesis and explain the possible improvements that can be done to the proposed solution.

Chapter 2

Technological context

This chapter will introduce the different technologies used for this master thesis that have been explored and that have been implemented or not on the first prototype but which have been used for the design phase of the device.

The main technologies discussed hereunder are LoRa, LoRaWAN, LTE-M and NB-IoT as communication systems, GNSS and WiPS as positioning systems and MicroPython as embedded programming language.

2.1 Hardware

2.1.1 Embedded and IoT devices

An IoT (**I**nternet **O**f **T**hings) device [48] is a piece of hardware such as sensor(s), actuator(s), appliance(s), ... that is:

- Programmed for a certain application;
- Able to exchange data over the internet or other networks (thanks to Wi-Fi, Bluetooth, cellular networks, Low Power Wide Area Networks, ...)

However, even though the IoT term could implies that the device is using Internet, it does not mean that the device must use the public Internet. It could very well use any public or private network.

The choice of such a device highly depends on the application/context needs. What processing power is required? What communicating system is used? What are the power expectations and requirements? What is the cost/benefit of the device?

This master thesis will mainly focus on small, low power consumption constrained micro-controller based systems/devices. Most of these devices have a minimal OS (which is closer to a library than a fully-fledged OS) and run an embedded program (written with a programming language like C, C++, MicroPython, ...) that allows to access hardware functionalities. They are often composed of a MCU (micro-controller unit) which is a small computer that contains at least one CPU, memories (RAM, Flash, ROM) and programmable inputs/outputs peripherals [76].

2.1.2 MicroPython

As previously said, most IoT are using micro-controllers to operate. In the past, these micro-controllers were firstly only programmed in assembly language and then later on via low-level programming language such as C. But nowadays, they can also be programmed with higher-level programming language like C++, python/MicroPython, javascript, etc [76].

MicroPython is one of the possible languages that can run on micro-controllers today. It is an open-source python interpreter optimised to run on micro-controllers written in C [76]. MicroPython offers a python3 compiler to bytecode with access to low-level hardware features and some libraries from python3.

The pros of MicroPython are its open-source license, ease of use and learning. However, its main disadvantage is that it is not as fast and memory optimised as lower level languages like C.

2.2 Communication technologies

This section will present two different technologies of communication, the first one is LoRaWAN that uses LoRa and the second one is LTE-M/NB-IoT.

2.2.1 LoRa and LoRaWAN

2.2.1.1 LoRa

LoRa (for **Lo**ng **Ra**nge) is the physical proprietary radio modulation technique based on chirp spread spectrum (`CSS`) developed by Cycleo and later acquired by Semtech [80] [82].

It uses license-free sub-gigahertz radio frequency bands EU868 (863–870/873 MHz) in Europe, AU915/AS923-1 (915–928 MHz) in South America, US915 (902–928 MHz) in North America, IN865 (865–867 MHz) in India, AS923 (915–928 MHz) in Asia and 2.4GHz worldwide [45] [80] [82].

2.2.1.2 LoRaWAN

LoRaWAN is a **M**edium **A**ccess **C**ontrol (MAC) protocol layer built on top of LoRa. It is a software layer that defines how end devices should use the LoRa modulation in terms of bands, timings, message structure, identification etc. It allows multiple devices to use the LoRa modulation to communicate with a server or an application [16] [80] [82].

The LoRaWAN specifications, maintained by the LoRa Alliance, were defined with multiple elements in mind, the most important of which are [16]:

- Low power to provide great battery life for battery operated devices;
- Long range (it can reach 40 `km` to 60 `km` on flat terrain without obstacles);
- End to end security via `AES-128` encryption;
- Low cost infrastructures for the network and low cost end devices.

For even more characteristics, please see [16].

LoRaWAN architecture

Figure 2.1 gives a quick overview of the network architecture used to communicate between the end-device and the application server.

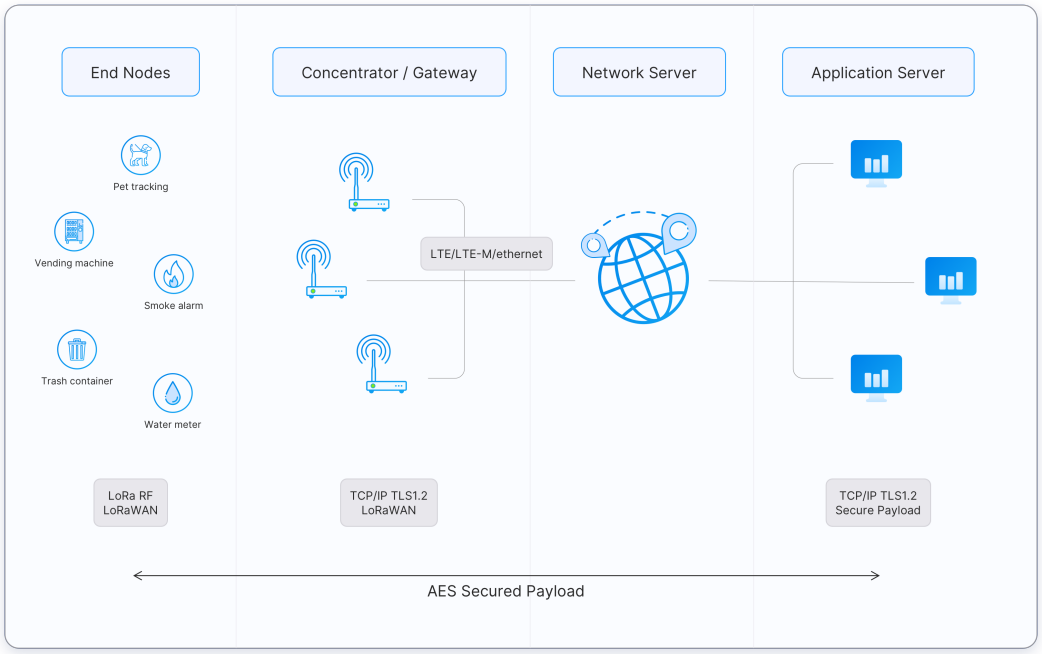


Figure 2.1: LoRaWAN architecture [18]

The gateways (located where the antennas are installed) operate as a middleman between the provider server (network server) and the end-device.

When the end-device wants to send data to the application server (often the backend of the application), the end-device sends packets to the gateway using LoRa/LoRaWAN, then the gateway sends the packets to the network server (using traditional communication technology like HTTPS) which finally transfers (via webhooks, MQTT or other IoT services depending on the LoRaWAN network provider) the packets to the application server.

For packets sent from the application server to the end-device, the application server sends packets to the network server with the end-device identifier which are then forwarded to the corresponding gateway. Finally, the gateway transfers the packets to the device using LoRa/LoRaWAN.

This architecture means that even if the end-device and the application server are geographically close, the traffic will maybe travel thousands of kilometres to the selected network server and only then travel to the application server (it depends on the provider and its configuration). The network server acts as a single entry and exit point for LoRaWAN com-

munications.

For this master thesis, the network server used is the one from The Thing Network. The Thing Network is a global collaborative Internet of Things ecosystem that follows the LoRaWAN specifications. It runs The Things Stack which is a crowdsourced, open and decentralised LoRaWAN network [28]. This means that both the gateways and the application server are configured to communicate with The Thing Network servers. The Thing Network offers three possible network servers around the world to work with (Sydney, Ireland and California). The one used in this master thesis is the Irish one because it is the closest to Belgium.

Duty cycle and fair use policy

The duty cycle is the proportion of time a system is active, e.g. if a device transmits 10% of the time, the duty cycle is 10%.

As the European LoRa bands (863 `MHz` - 870 `MHz`) are unlicensed bands, the duty cycles (transmission time proportion) are regulated by section 7.2.3 of the ETSI EN300.220 standard [39]. The maximum duty cycles in these bands range from 0.1% to 10% [70]. However, for public community networks, more strict policies, called *Fair Use Policy*, are often applied like a maximum of 30 seconds of uplink airtime per day per node and a maximum of 10 downlink messages per day per node for The Thing Network [70].

Device classes

The LoRaWAN specifications [47] [69] [81] define 3 end device types or classes: class A, class B and class C. Theses classes define the characteristics of the bi-directional communication over LoRa allowing to balance power consumption and latency.

The class A device is the most basic one, the class B device implements everything a class A device implements with more receiving slots and the class C device implements a class B device but the device keeps its receiving slot open unless it sends something.

Let's make a more in depth comparison of these 3 classes.

Firstly, the class A consists of 2 short receiving windows (aka. *Rx* windows) after an uplink transmission. The gateway can send packets during one of the two *Rx* windows or not

send anything as you can see on figures 2.2 and 2.3.

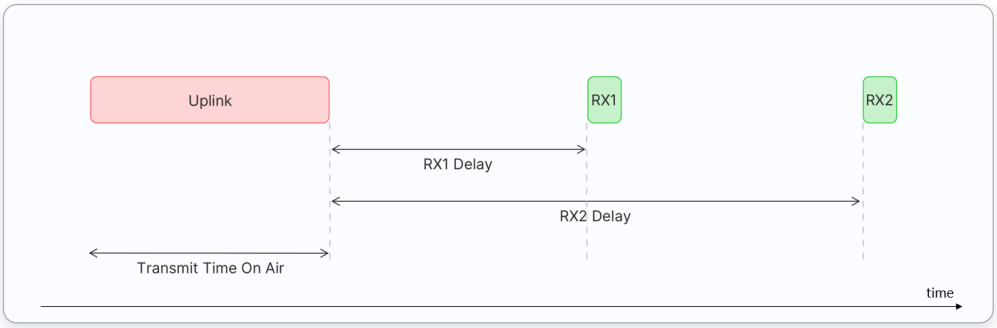


Figure 2.2: LoRaWAN class A device [9]

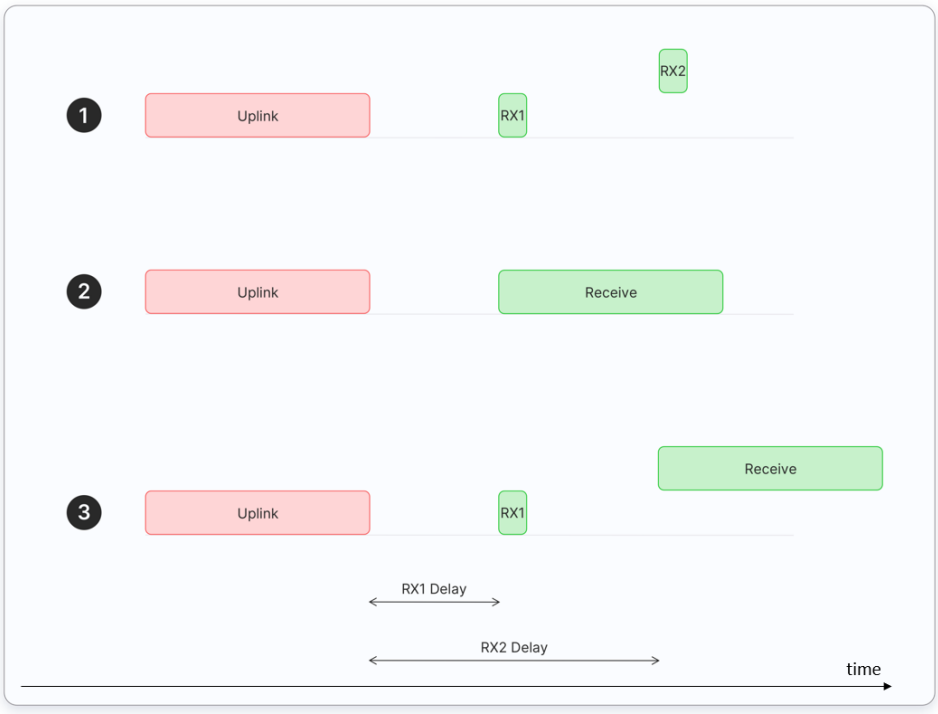


Figure 2.3: LoRaWAN class A device - 3 different cases [9]

Secondly, the class B (in addition to class A features) sends time-synchronised beacon packets (aka. *pings*) to the gateway to synchronise and setup periodic *Rx* windows independent from the transmission windows (aka. *Tx* windows). This helps reducing network

latency while not consuming too much power at the end device. See figure 2.4 for a visual representation

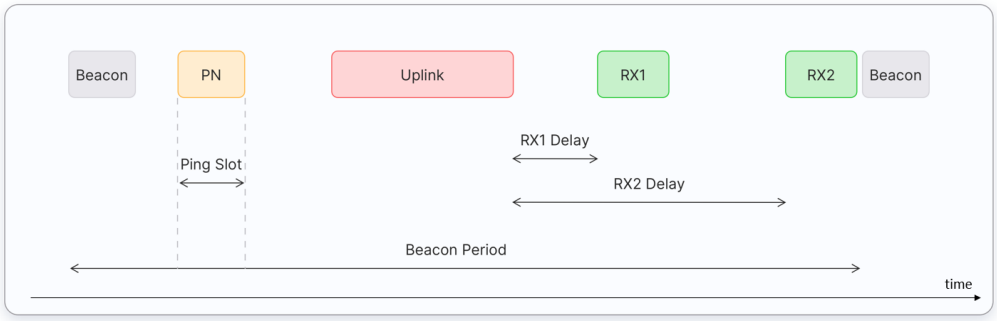


Figure 2.4: LoRaWAN class B device [9]

Finally, the class C device (in addition to class B features) keeps the *Rx* window open unless it is transmitting. This mode consumes much more power but reduces network latency. It is more advisable when the device is not on battery. See figure 2.5 for a graphical representation of a class C device.

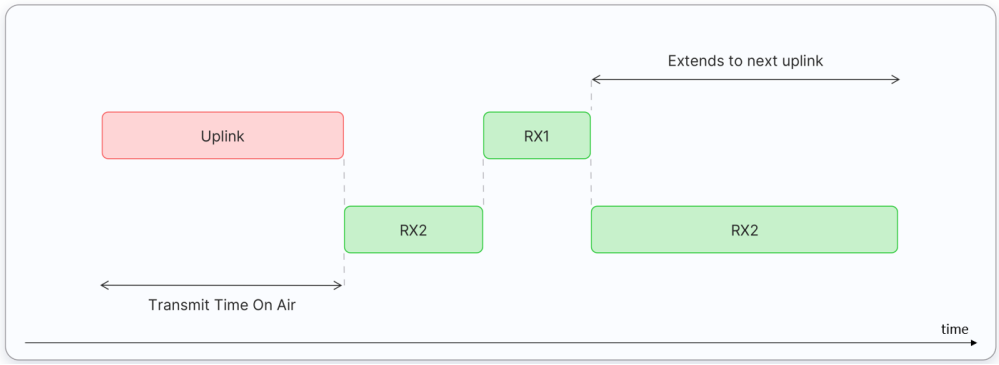


Figure 2.5: LoRaWAN class C device [9]

To sum up, a class A device has a higher network latency than a class B device which itself has a higher network latency than a class C device. However, a class A device consumes much less power than a class B device that itself consumes less power than a class C device.

End device activation

Before using a LoRaWAN network, each end device has to be registered following a procedure called "activation" [52] [83] [63] [82]. Two different procedures exist:

- **Over-The-Air-Activation (`OTAA`)** – This is the most secured technique. It consists of exchanging keys and assigning a dynamic address to the device;
- **Activation By Personalization (`ABP`)** – This is the less secured technique. It consists of hardcoding the security keys and address on the device which could be vulnerable to cipher attacks. Moreover, this technique does not allow the device to roam between networks as the keys and address have to be changed.

Over-The-Air-Activation (`OTAA`) In LoRaWAN specification 1.0.x, the procedure requires 2 different messages [52] [82]:

- `Join-request` – from the end device to the network;
- `Join-accept` – from the network to the end device.

Before starting the activation process, 3 different elements should be stored on the end device and configured on the network [52] [83] [63] [82]:

- `AppKey` – a `AES-128` bit secret, specific to each end device and used by it and the network to generate the different keys used to secure the connection;
- `AppEUI` – an `ID` representing the application used by the end device to identify the destination;
- `DevEUI` – an `ID` representing the device used by the network to identify the end device.

Figure 2.6 shows the `OTAA` procedure which occurs in 5 steps [52] [83] [63]:

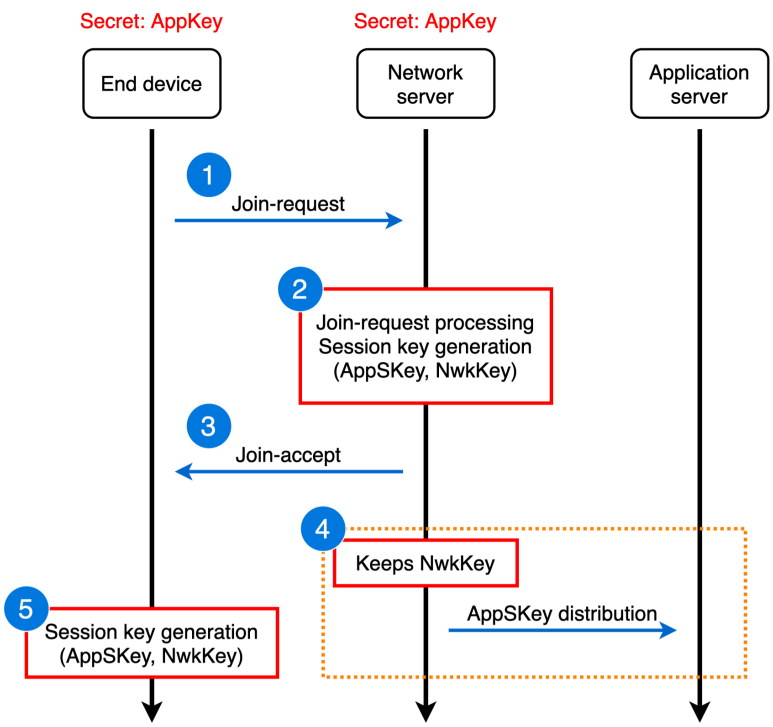


Figure 2.6: OTAA procedure as defined by LoRaWAN specification 1.0.x [52]

1. The end device sends the `join-request` (containing the `AppEUI`, the `DevEUI` and a nonce, `DevNonce`, used by the network to keep track of the connections) signed using the `AppKey` (signature known as Message Integrity Code (MIC));

2. The network server processes the `join-request` by verifying the signature and the information contained in it. If the device is allowed to join, 2 sessions keys are generated (`AppSKey` and `NwkKey`) and the server generates the `join-accept` response which contains useful information such as the `Tx` and `Rx` delays, a nonce (`AppNonce`) used to generate the corresponding keys on the device, the device address (`DevAddr`) and other information which are described in [52], [83] and [63]. Then, the packet is also signed using the `AppKey` (aka. Message Integrity Code (MIC)) but, unlike the `join-request`, the packet is encrypted using the same `AppKey` (using an AES decryption operation in ECB mode);

3. The network server sends the encrypted `join-accept`. However, if the device fails to activate, the request is quietly discarded and no response is sent;

4. The network server safely stores the `AppSKey` and `NwkKey`. Moreover, the network server sends the `AppSKey` to the application server (thus, it is now shared between the end device and the application server).

5. Finally, the end device processes the `join-accept` by deciphering it using the `AppKey`. Then, the same `AppSKey` and `NwkKey` are computed using the provided nonce and the `AppKey`.

Now, as the device is authenticated and the device, the network server and the application server are all sharing the keys needed to securely encrypt the communication.

NB: the procedure is a little different for LoRaWAN 1.1 but it is not described here as it is less common and not supported by the chosen hardware.

Activation By Personalization (ABP) It is a lot less secure than `OTAA` and the device is not able to switch network on-the-fly as the keys have to be manually changed on the device.

Actually, the procedure is straightforward: the user must precompute and preshare all the keys (`AppSKey` and `NwkKey` for LoRaWAN 1.0.x and `FNwkSIntKey`, `SNwkSIntKey`, `NwkSEncKey`, and `AppSKey` for LoRaWAN 1.1) and the end device address (`DevAddr`) on the end device, the network server and the application server as shown on figures 2.7 and 2.8 [52] [83] [63].

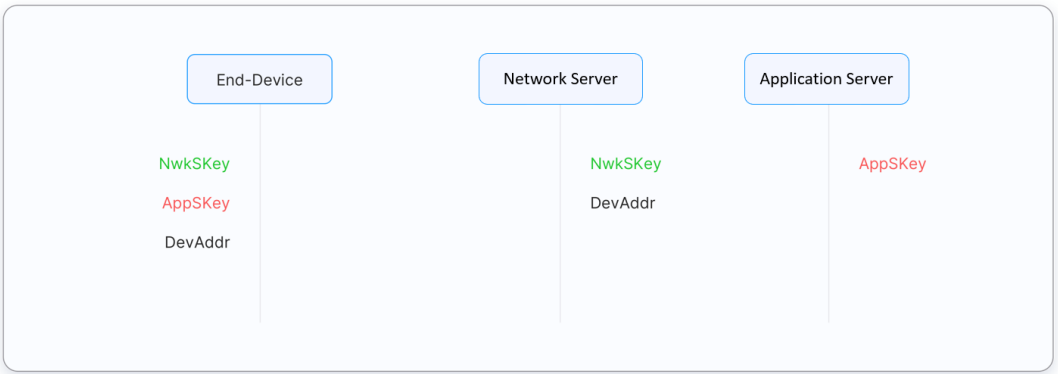


Figure 2.7: `ABP` procedure in LoRaWAN 1.0.x [52]

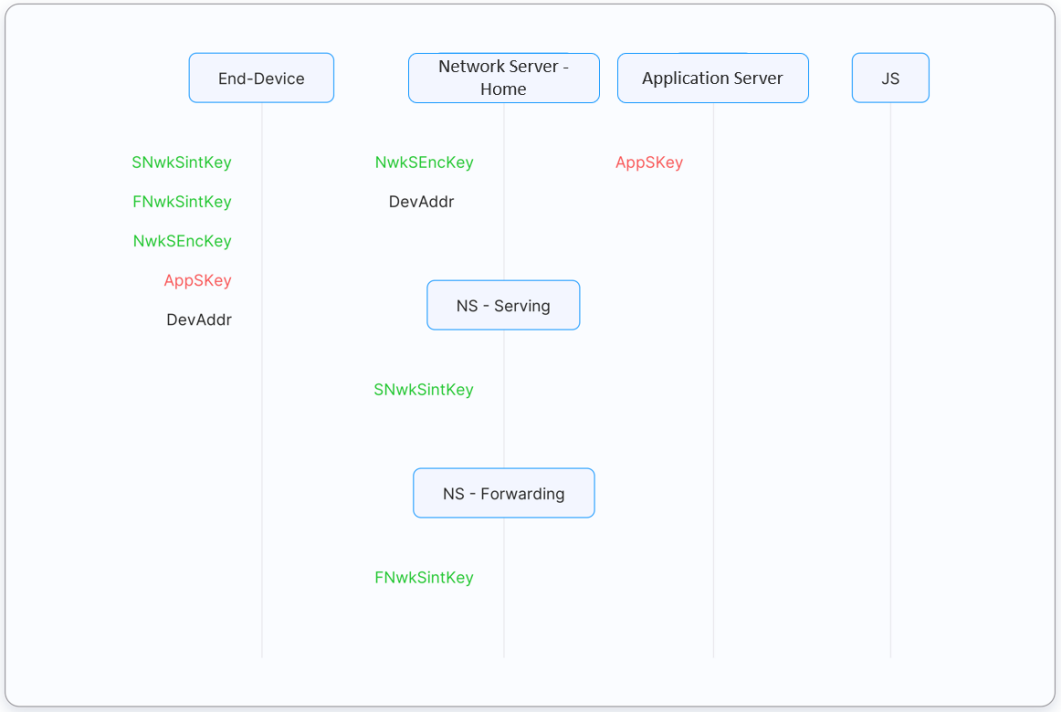


Figure 2.8: ABP procedure in LoRaWAN 1.1 [52]

Regional parameters: data rate and spreading factor

While using LoRaWAN, some parameters depend on the region the device is located. For the EU, these parameters associated with the European frequency bands (EU863-870) define data rates (DR) ranging from DR0 to DR15. However, DR8 to DR15 are reserved for future usage [82] [71].

These data rates are the effective number of bits per seconds available depending on the channel frequency (in the range 863 MHz to 870 MHz for the EU), channel bandwidth (BW and must be equal to 125 KHz, 250 KHz or 500 KHz) and the spreading factor (7 to 12 for LoRaWAN).

The spreading factor (SF) is the number of symbols (or chirps) sent when a bit of data is sent. As LoRa uses CSS modulation, the spreading factor corresponds to the duration of a sweep of frequency. See figure 2.9 for a visual representation of the spreading factor when CSS (modulation technique where the symbols or chirps are frequency sweeps) is used.

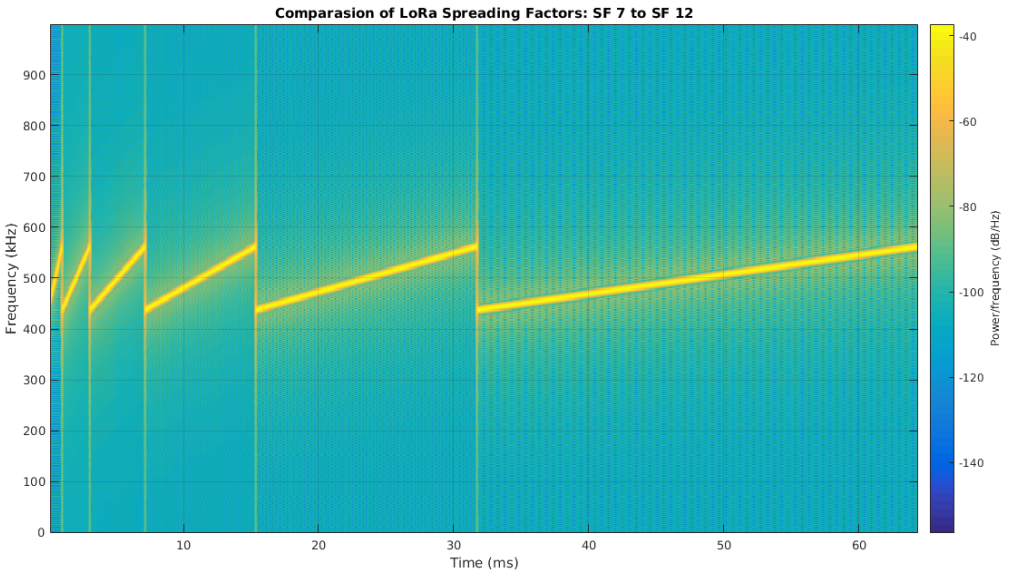


Figure 2.9: Spreading factor representation [66]

The parameter combinations are shown in table 2.1 and must be followed if you want to use LoRaWAN in the EU.

DR	Modulation	SF	BW	Bit rate	Max. payload size
0	LoRa	12	125	250 bps	51 bytes
1	LoRa	11	125	440 bps	51 bytes
2	LoRa	10	125	980 bps	51 bytes
3	LoRa	9	125	1760 bps	115 bytes
4	LoRa	8	125	3125 bps	242 bytes
5	LoRa	7	125	5470 bps	242 bytes
6	LoRa	7	250	11000 bps	242 bytes
7	FSK	N/A	N/A	50000 bps	N/A

Table 2.1: Data rates for EU863-870 as defined by LoRaWAN specification [46]

2.2.1.3 Sigfox and why not to use it?

Sigfox is a global network operator using Low Power Wide-area network (LPWAN) ultra-narrowband radio technology which the Belgium provider is Citymesh. It has been designed

for low power, long range and small data [24].

The main reasons why the use of Sigfox has not been considered in this master thesis is because it is a close, proprietary and paid network. In addition, Sigfox imposes a downlink limit of 4 messages/day of maximum 8 bytes and an uplink limit of 140 messages/day of 12 bytes [22] whereas LoRa limitation are less restrictive.

2.2.2 LTE-M and NB-IoT

LTE-M and NB-IoT are both cellular communication technologies developed by 3GPP (3rd Generation Partnership Project). They are both Low Power Wide Area Networks (LPWAN) network designed to reduce the power consumption, increase coverage and enhance the battery life. They were specifically designed for the IoT world that requires a low data rate, a low cost price and a great battery life.

They both use Orthogonal Frequency-Division Multiple Access (OFDMA) as main downlink modulation technique and Single-Carrier Frequency-Division Multiple Access (SC-FDMA) as main uplink modulation technique.

They are also both deployed in frequency bands between $\approx$ 700 MHz and $\approx$ 2 GHz.

2.2.2.1 LTE-M

LTE-M, which stands for LTE-Machine type communication, is also called LTE Cat M1 (LTE category M1) because its specifications have been frozen (in 2016) under this name. LTE-M allows to reuse the normal LTE base stations as the LTE frequency bands include the one of LTE-M. It uses a bandwidth of 1.4 MHz and offer a better coverage than LTE (improved penetration that improves indoor coverage and overall higher range). Moreover, LTE-M can have a downlink/uplink data rate up to 1 Mb/s and has a low latency (10 – 15 ms) [75].

2.2.2.2 NB-IoT

NB-IoT, which stands for Narrowband Internet of Things, is a wireless communication technology that focuses mainly on indoor coverage. This master thesis will be focused on NB-IoT release 13 (LTE Cat NB1).

NB-IoT uses a bandwidth of 180 KHz and offers a downlink and uplink data rate up to respectively 26 Kbit/s and 66 Kbit/s. Finally, it has higher latency than LTE-M (1.6–10s) [74].

2.2.2.3 Pros and cons of LTE-M and NB-IoT

Even if LTE-M and NB-IoT share a lot of features, characteristics and objectives, they differ in the following points:

- LTE-M has a higher data rate than NB-IoT;
- LTE-M has a lower latency than NB-IoT;
- NB-IoT has been designed for indoor environments;
- LTE-M supports better cells mobility/handover than NB-IoT. NB-IoT has been designed mainly for fixed devices;
- NB-IoT is designed to consume less energy than LTE-M;
- NB-IoT uses a smaller bandwidth.

2.2.2.4 PSM and eDRX mode

For LTE networks, as soon as the device (or UE for User Equipment as defined by 3GPP) is connected (or attached) to the network, different kind of messages are exchanged with the network during different phases of the session. One of these phases is the paging phase where the UE is idling and listens for paging messages that contains relevant information about the events in the network (incoming data, change in configuration, ...). 3GPP release 13 [31] defines LTE-M and NB-IoT as well as PSM and eDRX, two new technologies developed to reduce power consumption by tweaking the paging phases and their period.

PSM

`PSM` stands for Power Saving Mode. This feature is enabled and configure on the end device where the user must specify the active and sleep periods [55]. Then, instead of listening all the time when idling, the device can go to sleep during the specified amount of time and wake up without reattaching to the network.

By doing this, it helps reducing the network overhead produced by massive abrupt shutdowns without proper disconnection which decrease a lot the service on cellular networks [55].

PSM also shortens the attach time to the network as the antennas will remember that the device is there even though it is sleeping and not able to receive messages [55].

Figure 2.10 shows the PSM timings and cycles.

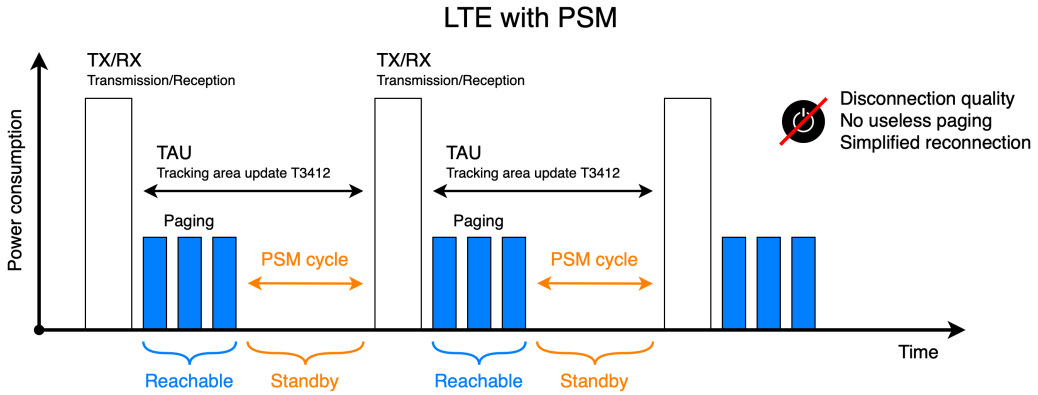


Figure 2.10: PSM [55]

eDRX

eDRX stands for extended Discontinuous Reception and can be used in conjunction with PSM. It has been designed to increase even more battery life by extending the period between paging message thus increasing the standby period length [54]. Paging messages are messages sent to UEs or antennas to advertise events.

Figure 2.11 shows eDRX timings and cycles.

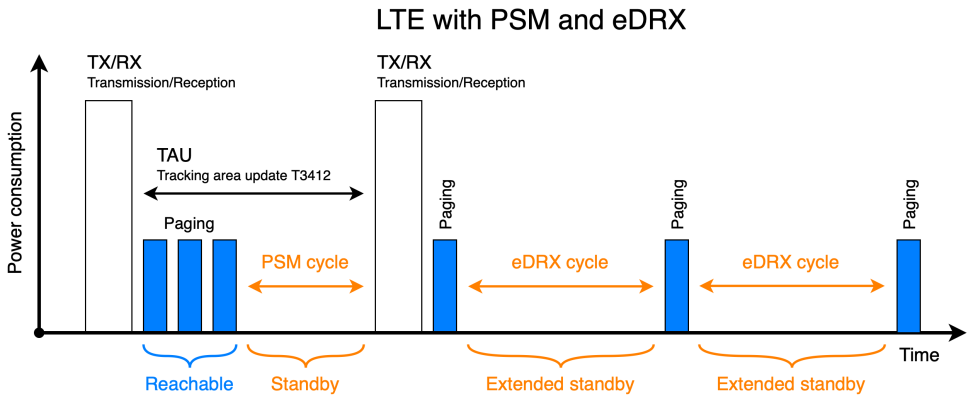


Figure 2.11: eDRX [54]

Comparison between PSM and eDRX

PSM (see figure 2.10) allows to specify a period without paging slots where the device can sleep while allowing the device to reattach faster as the antennas will remember the presence of the device for the specified duration.

eDRX (see figure 2.11) allows to specify the duration between the paging slots allowing the device to sleep or reduce its power consumption between two slots.

In conclusion, these 2 technologies are not exactly the same but they are very similar and they help to reduce the overall power consumption by reducing the impact of paging phases.

2.3 Positioning technologies

This section will give an overview on how do Global Navigation Satellite System (GNSS) positioning system and the Wi-Fi positioning system (WiPS) work.

2.3.1 GNSS

The abbreviation GNSS stands for **G**lobal **N**avigation **S**atellite **S**ystem [78] and include multiple positioning systems (GPS, GLONASS, Galileo, QZSS, ...). It is a system that provide a location service (latitude, longitude end elevation) for receiver and that can also be used to

acquire the current local time precisely.

This technology was first developed for military purposes in 1960s and later on for civilians.

The receiver receives multiple broadcast signals coming from satellites on a specific frequency. These signals contain the time it has been sent and some additional information about the satellite that sent it. By knowing the orbits of the satellites, the frequency offset between the different broadcast signals and the time each signal was sent (all satellites timing are synchronised), the receiver is able to calculate its geographic coordinates thanks to trilateration.

Multiple protocols exist to read the GNSS position from the GNSS chip. The one used for this master's thesis is the NMEA_0183 (controlled by the National Marine Electronics Association) [77] [20]. This means that NMEA messages (GPRMC, GPGGA and GPGLL messages) coming from the GNSS chip need to be processed to extract the GNSS position.

The GNSS module uses an antenna to receive signals from satellites. The antenna can be either active or passive. A passive antenna is a receiver that does not draw any power in contrast to an active antenna which includes an amplifier, using additional power from the chip to increase the range/signal reception.

Moreover, the receiver needs at least 4 different GNSS signal coming in straight line from different satellites [43] (signal multipath is possible). This requirement implies that if the receiver is in a bad environment (e.g. indoor), the GNSS module might not receive enough signals and therefore will not be able to compute its position. In addition, position accuracy depends of the capture time, as the amount of satellites signals in sight.

2.3.2 Wi-Fi positioning system

The abbreviation WiPS stands for **Wi**-Fi **P**ositioning **S**ystem [6]. It is a geolocalisation system mainly developed for indoor locations that uses the nearby Wi-Fi access points to localise itself. This technique requires to have a large database containing the real position of most Wi-Fi access points that exist and to know the signal strength of multiple Wi-Fi access points. The information used to identify the Wi-Fi access point is the BSSID (Basic Service Set Identifier) and not the SSID (service Set ID) as multiple access points can have the same SSID.

WiPS can use multiple techniques with different accuracy results. The most used technique and the simplest one is to use the RSSI (Received Signal Strength Indication) to measure the distance between the receiver and the transmitter (the position of the transmitter is known) and use a trilateration algorithm to find the receiver position.

However, this solution has some downsides:

- The user must often pay to have access to a database large enough to be able to implement a viable WiPS solution.
- The user cannot know in advance (before processing the collected Wi-Fi access points) the accuracy of the position obtained. Or it should process the Wi-Fi access points locally and therefore store the entire Wi-Fi database locally which is not a possible solution for an IoT device.
- No Wi-Fi access point means no possibility to know the position.
- Not all Wi-Fi access points are referenced in a database which means that even if there are Wi-Fi access points around the device, it does not mean that it is possible to extract a position of it.

Different providers like Google[21] and Mozilla [19] have mapped the location of Wi-Fi access points and are providing a WiPS service that takes care of the computation. These kind of services only require the input data coming from the device that wants to locate itself. The data the provider needs to localise the device are at least the BSSID and the RSSI of several Wi-Fi access points. For Google [21] and Mozilla [19], their services are available through a REST API.

These big providers play a very important role as acquiring a large database of Wi-Fi access point locations is expensive and this is therefore cheaper and easier to just use existing services.

2.4 Existing solutions

Similar solutions to the bike tracker proposed by this master thesis exist and try to solve the bike theft problem. Here are the most similar and interesting solutions:

- The most similar product is the biketrack by Invoxia (figure 2.12) [14]. This device is a tracker hidden inside a bicycle rear light which only uses LoRa as communication

technology. It also uses the GPS for general location and the Bluetooth for localising the device precisely when close to it.

However, as this solution only uses LoRa as communication system, the solution cannot work in an environment without LoRa coverage.

- Biketrax by PowUnity (figure 2.13) [5]. This device is specially designed for electric bikes but restricted to Brose, Shimano, Yamaha and Bosch e-bikes. The biggest advantage of this device is that it is placed inside the e-bike engine compartment and is therefore invisible to thieves and hard to remove. It also has a back-up battery in addition to the battery of the e-bike, allowing the device to have a huge battery life. The tracker uses 2G connectivity to communicate.

  This solution is great for the proposed bikes but is unfortunately not applicable to all electric bike and not to non electric bike. Moreover this solution is expensive as it costs around 200€ plus shipping and an annual subscription of around 40€.

- Yabby Edge by Digital Matter (figure 2.14) [7]. This device is an all purposes IP67 tracking device using both GNSS and Wi-Fi scanning to localise itself. It offers a web platform, periodic or movement wake up and an excellent battery life. This device is available in two models, one using LoRaWAN as communication technology and the other using LTE-M/NB-IoT.

  This device is the closest to the solution proposed by this master thesis, however we can observe some significant differences:

  - Yabby Edge does not offer to combine both LoRaWAN and LTE-M/NB-IoT. It can either use LoRaWAN (Yabby Edge LoRaWAN) or use LTE-M/NB-IoT (Yabby Edge Cellular) and pay a fee to use the network for each communication;
  - It has not be designed and advertised for everybody. It mainly targets the industrial world and does not provide an out-of-the-box solution for for the customer market;
  - It has not be designed for bicycles as it is a simple box designed for assets tracking.

- PyGo2 by Pycom (see figure 2.15) [11]: This device was launched on kickstarter in 2018 and, at the time of writing this master thesis, it enters its mass production phase. This device uses LTE-M/NB-IoT, LoRa, Sigfox and Bluetooth as communication systems as well as GPS and accelerometer for positioning systems. Although this device has not been designed for bikes, it is possible to design a case to attach it to a bike and use it with its mobile app (Pylife).

As additional information, a second device named PyGo1 [12] is also available for pre-order and offers the same features as the PyGo2 but without LTE-M/NB-IoT connectivity.

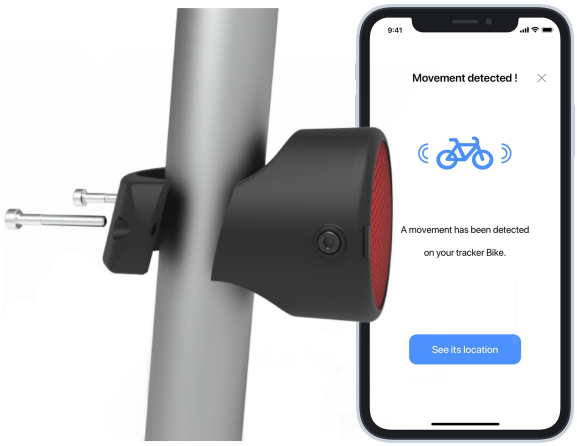


Figure 2.12: Invoxia bike tracker [14]

Figure 2.13: Biketrax by PowUnity [5]

Figure 2.14: Yabby Edge by Digital Matter[7]

Figure 2.15: PyGo by Pycom [11]

2.5 Related master theses on LoRa and LoRaWAN

This section will guide you through previous master theses that are highly related to this master thesis that would be interesting to read in addition to this master thesis.

- **Evaluating LoRa and LoRaWAN performance in Louvain-la-Neuve** [64]: This master thesis is about explaining how LoRa works, deploying LoRa and evaluate LoRa in Louvain-la-Neuve.

  This master thesis allows to understand the LoRa and LoRaWAN technologies from the low level of radio communication to the deployment in a real environment. It offers a good start to completely understand how LoRa and LoRaWAN operates. Moreover, it also evaluates the LoRaWAN reception in Louvain-la-Neuve.

- **Using LoRaWAN and Wi-Fi for smart city monitoring in Louvain-la-Neuve** [44]: This master thesis studied the combination of LoRaWAN and Wi-Fi as communication systems for IoT devices by deploying a end-to-end system allowing the monitoring of environmental data such as the temperature, humidity, air quality or noise level. This master thesis offers a good understanding of the design and architecture of an end-to-end solution using the LoRaWAN network as well as the traditional Wi-Fi connection.

- **Versatile and scalable IoT platform for efficient asset management** [53]: This master thesis goal was to have a user-friendly platform to visualise data coming from different sources (existing systems, LoRaWAN, Wi-Fi, ...) for the GTPL (Service de gestion technique du patrimoine de Louvain-la-Neuve) service at UCLouvain. This master thesis is a good way to understand how does an entire application with data acquisition run, operate and what are the important choices to make.

Chapter 3

Design of the bike tracker

This chapter will describe the design of the final solution that is an embedded device used to track the position of a bike (aka. a bike tracker). The proposed solution has the following general architecture (see figure 3.1 for graphical representation):

- A tracker (with an embedded software) located on the bike to locate it;
- Two underlying networks, LoRaWAN and LTE, used by the device to communicate with the backend;
- A backend server handling the data sent by the tracker and sending requests to the tracker;
- A mobile application to provide an interface to the user in order to show the location of the tracker, its battery percentage, etc. and to allow the user to remotely configure the tracker.

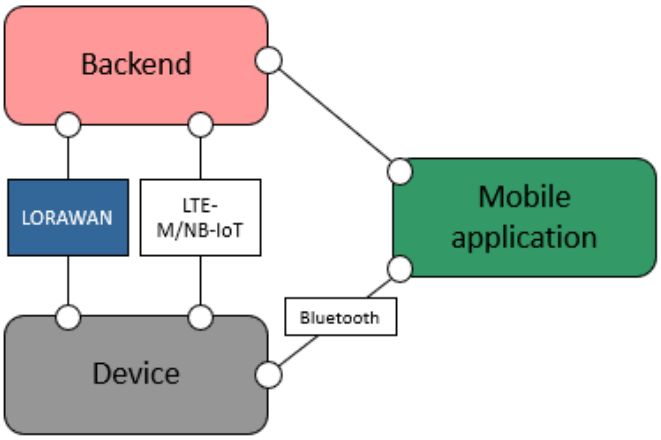


Figure 3.1: General architecture

3.1 Device design constraints

As explained in chapter 1, the device is intended to be used as a bike tracker to reduce bike theft. Therefore, the following requirements are expected for such a device:

- Small/compact design compatible with bikes;
- Discreet design to avoid being too visible and easy to remove;
- Battery operated with an expected battery life ranging from few days to few weeks/-months;
- Simple design to ease the recharge process;
- Reliable long-range communication to provide tracker data and configurability;
- Backend server to process tracker data and configure this tracker;
- User-friendly interface to show the device position/battery as well as to configure the tracker.

Moreover, at least for the first prototype, the technologies for the device, the backend server and the user interface should be wisely chosen by selecting familiar frameworks, technologies and languages. Indeed, it will ease and speed up the development, allowing us to take more time on the most important parts:

- Communicate using multiple networks (LoRa and LTE);
- Develop a device software logic allowing it to correctly behave against theft;

- Estimate and provide enough battery life (by using a sleep/active cycle);
- Provide device-related data through a user-friendly interface;
- Allow the user to configure the device using this same interface;
- Evaluate technologies coverage around Louvain-la-Neuve.

3.2 Hardware

Initially, for the embedded device, we tried to use the boards used in the previous master thesis on the same subject [36], that is the WiFi LoRa 32 (V2) by Heltec [41]. However, these boards do not include a GNSS module, an accelerometer nor a LTE modem.

The GNSS (Global Navigation Satellite Systems) module had been added to the Heltec board by adequately connecting a GT-U7 module [40]. However, this board was still lacking an accelerometer and a LTE modem. Adding an accelerometer would not have been a huge problem but adding a LTE module is quite more challenging. It would have required to find a compatible arduino shield that are often really big, expensive and are much more complex to use, as they are not designed to specifically work with Heltec's board.

All this convinced us to look for another development board and we came across a dutch hardware manufacturer called Pycom. They manufacture multiple development boards (LoRa/Sigfox, Wi-Fi/Bluetooth, LTE or all combined) based on MicroPython instead of arduino. Using MicroPython (see Section 2.1.2) makes the prototyping easier but reduces power efficiency. However, power efficiency is not critical for a first prototype.

3.2.1 FiPy development board

For the main development board, we decided to use the FiPy board [58] developed and manufactured by Pycom. This board has been created in 2016 and was the first development board to embark five networks (LTE-M/NB-IoT, LoRa, Sigfox, Wi-Fi 2.4 GHz and Bluetooth). Since then, it seems to still be the only one. A system block diagram of this board is shown on figure 3.2.

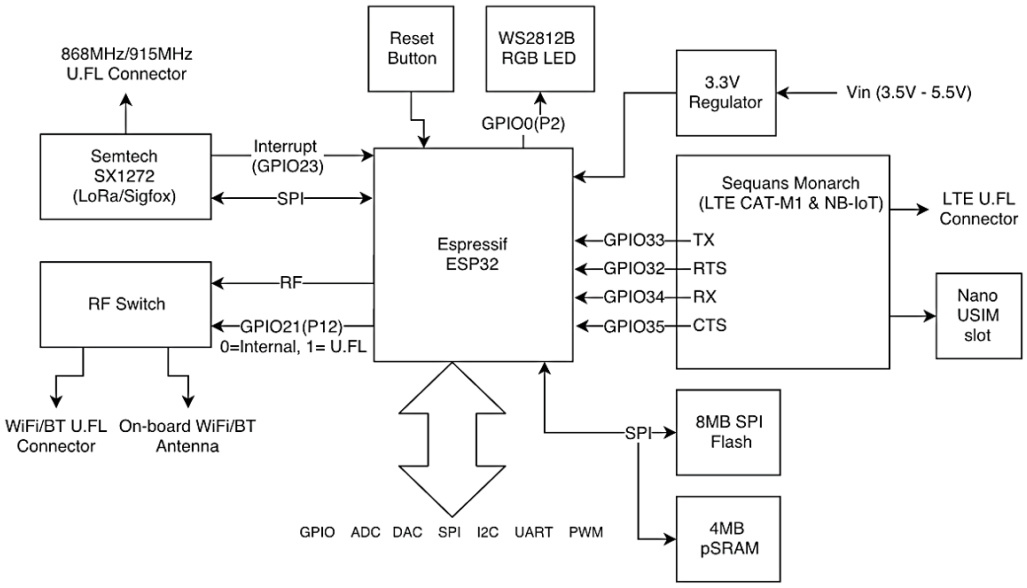


Figure 3.2: The FiPy development board by Pycom – Block diagram [58]

The FiPy uses an ESP32 as MCU, specifically an ESP32-PICO-D4 with a 32-bits dual core Xtensa LX6 CPU (up to 600 DMIPS) [73]. It has 520KB + 4MB of memory and 8MB of external flash memory (to store files and programs). Alongside the main CPU, this MCU has an ultra low power (ULP) coprocessor capable of waking up the main CPU in case of events when the MCU has been put into deep sleep mode. This coprocessor consumes up to 25μA [58] [73].

Figures 3.3 and 3.4 respectively show the top and the bottom of the FiPy board. This board has a RGB LED, an embedded LTE/LoRa/Sigfox transceivers and multiple connectors for external antennas. Speaking of which, external Wi-Fi and Bluetooth antennas are optional thanks to the on-board Wi-Fi and Bluetooth antennas. However, for the LTE/LoRa/Sigfox connectivity, the board requires external antennas.

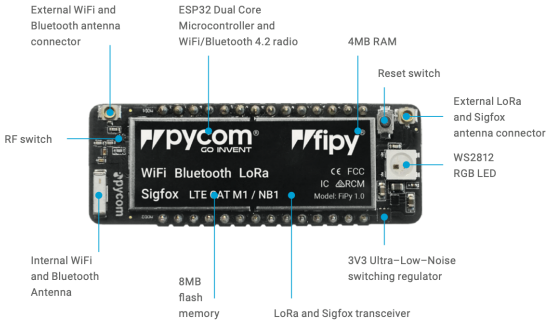


Figure 3.3: Top view of the FiPy

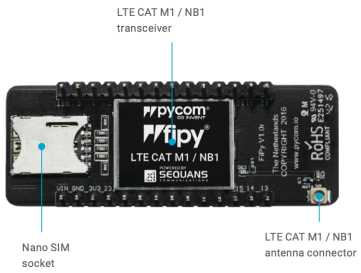


Figure 3.4: Bottom view of the FiPy

In terms of LoRa connectivity, the device supports frequencies from 860 MHz to 1020 MHz (that is, including the European frequency of 868 MHz). The device also supports and can be configured as a class A and a class C device. For our application, the device will be configured as a class A device because the network latency is not critical while the battery life is much more important.

The LTE modem is embedded in the FiPy. It supports both LTE-M and NB-IoT standards however to switch between the two, the modem must be flashed (one cannot switch *on-the-fly*). For lower consumption purposes, it supports *eDRX* and *PSM*.

The GNSS module and the accelerometer are embedded on the Pytrack 2.0X (see Section 3.2.2).

To conclude, the choice of this board instead of the Heltec's board with some expansion shields has been motivated by the following elements:

- Compact design since everything is embedded in one development board (except for the GNSS and accelerometer modules). No need to weld or connect external modules to provide connectivity.
- Faster and easier development because everything has been designed to work together (the manufacturer provides an API for each network). So we do not have to develop APIs for the board to interface with external connectivity module provided by a different manufacturer.

- Cheaper overall.
- MicroPython for faster prototyping.
- Highly used hardware with a lot of support and forum discussions.

3.2.2 Pytrack 2.0X expansion board

The Pytrack 2.0X [59] [60] is an expansion board (or shield) to the FiPy that provides:

- an SD card slot;
- a GNSS module with an internal antenna and an external active antenna connector;
- an accelerometer;
- a JST PHR-2 male connector to connect any Li-Ion/Li-Po battery (must provide between 3.5V and 5.5V for the development board to work);
- a USB interface to program/flash any Pycom's development board and to recharge a connected battery.

A picture of the top of the shield is presented in figure 3.5 and a technical view on figure 3.6.

The GNSS module is a L76-L from Quectel [61]. This module gives access to 4 different GNSS: GPS (USA), GLONASS/BeiDou (Russia/China), Galileo (Europe) and QZSS (Japan). Thanks to the on-board antenna (on the Pytrack, not in the module itself), this GNSS module can be used without an external active antenna. However, without it, localisation inside buildings can be tedious or even impossible because the shield's on-board antenna is too weak to work indoors. The use of an active antenna would solve the problem but it would make the device much more visible by potential thieves, as the antenna is quite large. This problem has been solved by using the WiPS (Wi-Fi Positioning System).

The accelerometer is a 3-axis linear LIS2HH12 [67] capable of measuring accelerations of up to $\pm 2g/\pm 4g/\pm 8g$ in all directions depending on the configuration. Unfortunately, the accuracy of the accelerometer is unknown. It also provides the relative orientation of the board. This accelerometer can be set up to wake up the device (that would be in deep sleep

mode) if an acceleration greater than a specified threshold (specified by the developer) is detected.

The USB interface is useful for debugging (send commands and act as *stdout* for the program running on the development board). The Pytrack (via the USB) is capable of flashing the development board connected to it, which is necessary if we want to develop and store any code on the development board. Finally, when a battery pack is connected to the board, the Pytrack is capable of recharging it if the USB connection can provide enough power.

The SD card slot is used to store data on a SD card when debugging or measuring network coverage (see Section 4). This feature is only useful during developmentand would not be present on the final prototype.

Figure 3.5: The Pytrack 2.0X expansion board

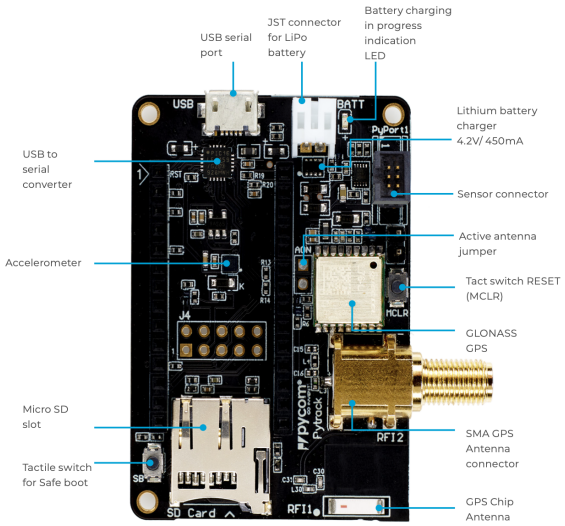


Figure 3.6: Top view of the Pytrack 2.0X

3.2.3 Batteries

For the battery pack, we have decided to use the same batteries as the ones used by F. De Keersmaeker and N. Van De Walle in their master thesis [36] that is 3.7V Li-Po batteries of

1100 mAh [50].

For the prototype, we have chosen to use only one battery to facilitate the design. In fact, we could imagine putting 2 or 3 batteries in parallel to increase the battery life of the final product. However, this has to be done with great precaution because the cells must have the same voltage. If not, it will cause continuous discharge of some batteries to compensate the lack of voltage of the others, leading to decreased performance, unnecessary heat and maybe, potential fire hazards. Putting batteries in series is not possible as the total tension would exceed the maximal tension allowed by the board and the shield.

This battery pack is connected to the tracker via the JST PHR-2 male connector on the Pytrack. As said previously, when the batteries are connected to the Pytrack and when the Pytrack is powered via USB, the batteries will be recharged.

3.2.4 Waterproof case

To attach the device to a bicycle without being too visible, we propose two solutions: one design for production with the assumption that we have the ability to reduce the size and shape of the hardware and a second design suitable with the current size and shape of the hardware of the prototype.

- Production solution:

  If we assume to be able to design the hardware shape with a reasonable small size, the production solution will consist of putting the device inside the bike itself to prevent anyone from removing the device. However, if we put the device inside the bike, we would have to study the impact of the bike material on the signal reception/emission and therefore choose the adequate position for the device (e.g. in the handlebar, frame, seat, etc).

  Figure 3.7 shows some existing solutions to put hardware into a bike frame or handlebar.

- Solution with the current hardware:

  This solution consists of hiding the device in a bicycle lamp at the back of the bike, below the seat. However, depending on lamp power consumption we could imagine just not turning-on the lamp and just having an external look that looks like a regular bicycle lamp. The case would use special/non-regular screws in order to make it harder to remove the case. The design of this solution is shown on figure 3.8 and a similar

(a) *Bike Hawk* tracker [8]

(b) *BikeFinder* [35]

Figure 3.7: Examples of actual production designs

design proposed and commercialised by *Invoxia* is presented in figure 3.9 (bike tracker designed as a bike rear light).

Figure 3.8: Design for a case with the actual hardware available

Figure 3.9: Similar design produced by *Invoxia* [14].

3.3 Embedded software

This section explains the logic running on the hardware presented in Section 3.2 and the communication technologies and protocols used to communicate with the user (see Section 3.4 for the backend and the user interface).

3.3.1 Functionalities

The embedded software is running on the FiPy & Pytrack 2.0X using MicroPython and must meet the following requirements:

- receive/send requests from/to LoRaWAN network.
- receive/send requests from/to LTE network.
- send the position when the accelerometer detects a movement.
- send the position regularly depending on the frequency if the device is *monitored*
- send the battery level regularly depending on the frequency
- receive request from the Bluetooth interface to change change and get the device state.
- change state (frequency and/or *monitored*) when receives appropriate request.

3.3.2 Software logic

In order to meet all requirements, the software running on the FiPy operates as follows when it wakes up:

1. The device wakes up according to the device frequency or because the accelerometer has triggered the device to wake up;
2. The device takes measurements about the network current reception quality:
    - If the LoRa network reception is considered sufficient, the device will switch to LoRa;

- If the LoRa network reception is considered **not** sufficient, the device will measure the LTE reception and switch to it if it is considered sufficient.
- If no reception of the two networks, the device will go to deep sleep and try again next time it wakes up .

3. The device will send a packet to the backend (via LoRa or LTE) containing:
   - Only the battery level if the device is **not** in *monitored* mode;
   - The battery level and the current position if the device is in *monitored* mode. The current position can be GNSS coordinates or a list of Wi-Fi SSIDs when WiPS is used.
4. The device waits for a potential reply of the backend (meaning the current device configuration has been changed via the mobile application);
5. The device goes to deep sleep mode for a duration based on the wake up frequency set by the owner of the tracker via the mobile application.

When the device is in deep sleep mode, in case of movements (detected via the accelerometer that can wake up the device at **any** time), even if the device is **not** in *monitored* mode, the device will wake up earlier than expected and send its position.

3.3.3 Communication

The following sections explain how the device and the backend communicate using LoRaWAN and LTE network.

3.3.3.1 Communication using LoRaWAN

For costs and availability reasons, we are using the open community LoRaWAN network of The Thing Network. When using this LoRaWAN network to send uplink packets to the backend, the device sends packets to The Thing Network which then forwards them to the backend using a Webhooks integration [27].

The device sends and receives LoRaWAN packet according to class A which is the class the most suitable for our application due to its low power consumption and that we do not need feature of class B or C.

When sending downlink packets from the backend to the device, the backend is using a downlink queue maintained by The Thing Network for each device [26]. When the device is sending an uplink packet to The Thing Network, it consumes one element in the queue and sends it to the device (see figure 3.10).

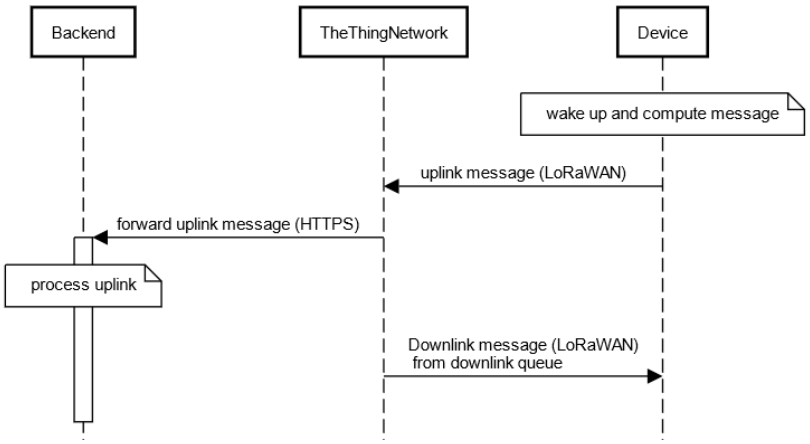


Figure 3.10: LoRaWAN - Device sending and pulling messages

3.3.3.2 Communication using LTE

To use LTE, we are using a sim card from Orange Belgium that provides an access to their network. For testing purposes, we subscribed to the *intense* subscription that gives us 30MB per month of data for 21€ per year. This subscription is suitable for ”starting a project” and not for industrial use. For industrial use, the subscriptions will cost at least between 6€ and 10€ per year with a real cost depending on the usage [56].

For the first prototype, the communication protocol over LTE was HTTP but it was not very suitable because of:

- Its big packet size;
- Its need to maintain a connection;
- The need to reduce power consumption;
- The limited data plan from Orange.

To solve this issue, we considered the use of UDP, DTLS, TCP and TLS. The solution that consumes less data is UDP in first position and TCP in second [57]. However, these two

solutions are not suitable for security reasons as the communication is not encrypted. In order to secure the connection between the device and the backend, we should therefore use DTLS or TLS. DTLS or TLS could be used with a pre-shared secret (PSK) instead of certificates to minimise the number of backend exchanged between the device and the Internet. Given all of these information, DTLS with PSK has been chosen for its ability to minimise the size and the number of packets exchanged when communicating with the backend.

Because the device is only online using LTE for a short period of time, downlink messages (from the backend to the device) are transmitted to the device through the response of the uplink message (see figure 3.11).

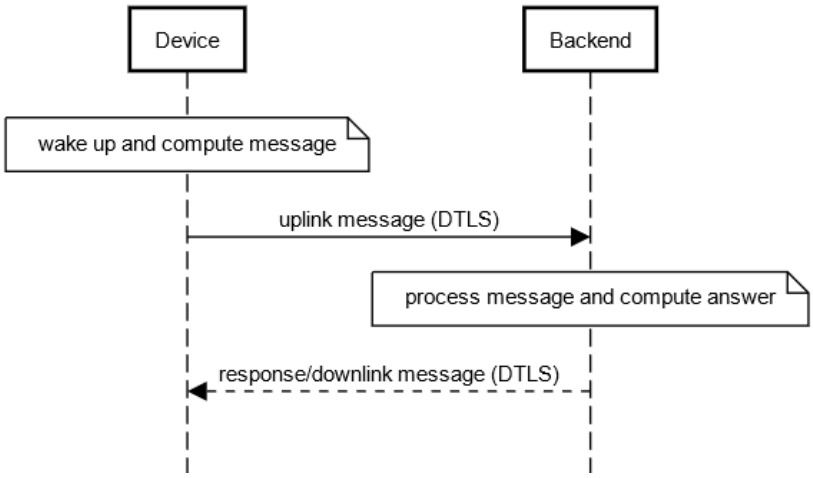


Figure 3.11: LTE - Device sending and pullin messages

3.3.3.3 Packet specification

To transport information/requests over LoRaWAN and LTE, we have designed a packet format which allows to transport all the information needed in a minimum number of bytes.

The main packet format is a little bit different if we use LoRaWAN or LTE.

- For LoRaWAN, the first byte is the *flag* and is used to define the packet type. The rest of the packet is the related payload associated to the packet type (see figure 3.12). Several packets can be sent at the same time.

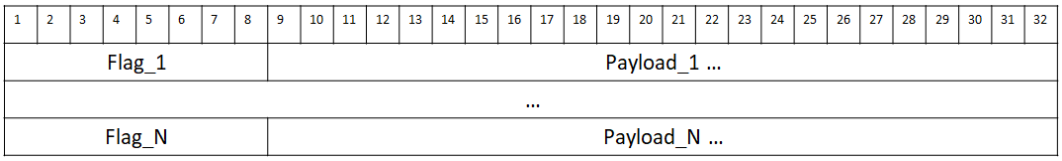


Figure 3.12: Packet format

- For LTE, because the source is not yet authenticated (which is automatically the case when using LoRaWAN), we need a way to identify and authenticate the device. Therefore, the first 256 bytes sent are used to sign the rest of the bytes with the device private key. Then, the next 6 bytes contain the device *DevEUI*. Finally, the rest of the bytes has the same format as the one used with LoRaWAN and can contain multiple packets.

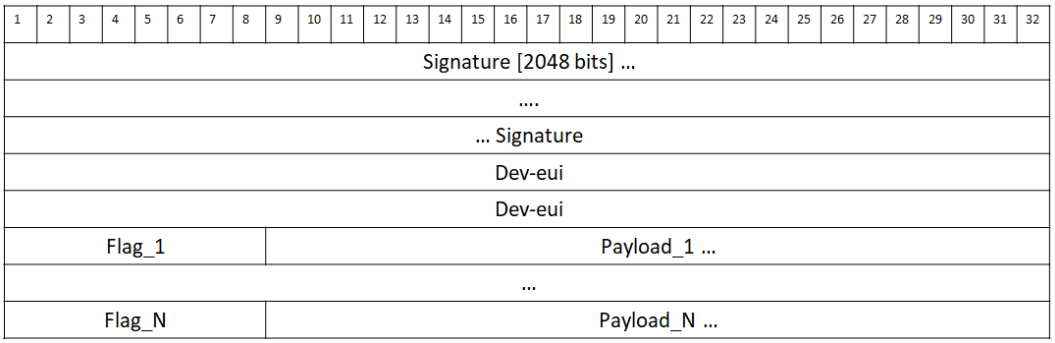
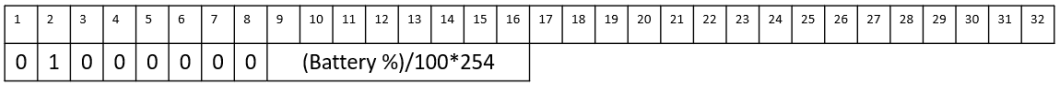


Figure 3.13: Packet format - LTE

Given the main packet format, we have designed message payloads to contain the necessary information. Because one of the device features is to use the Bluetooth to modify the device state, a 4 bytes timestamp has been added to all downlink message to solve device state conflicts due to Bluetooth interactions.

- A payload that contains the battery level: The flag used is 2 and the payload is a 1 byte integer that contains the battery level given that 100% is equals to 254 (255 means unknown value).



- A payload that contains the frequency at which the device sends its position. The flag is equal to 3 or 101 depending on whether we get or set the frequency and the payload is a 2 bytes integer that contains the frequency per day.

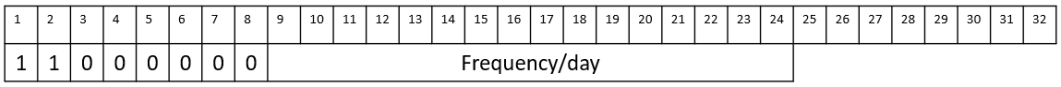


Figure 3.14: Frequency update - uplink message

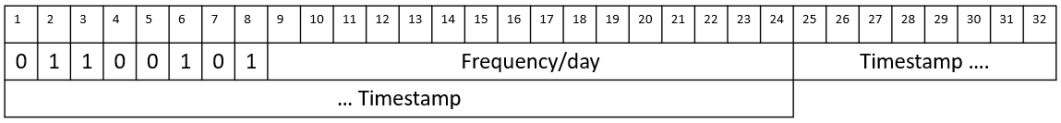
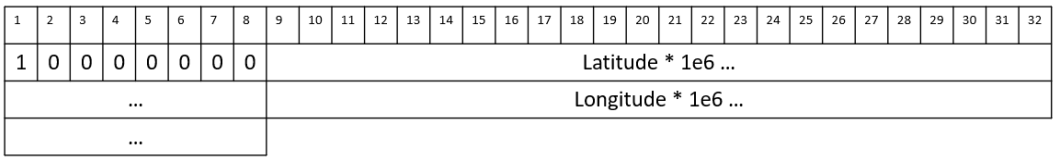


Figure 3.15: Frequency update - downlink message

- A payload that contains the GNSS position: The flag is equal to 1 and the payload contains respectively the latitude and longitude, each encoded on a 4 bytes integer in order to transmit the position with a high accuracy.



- A payload that contains the device *monitored* status: The flag is equal to 4 or 102 depending on whether we get or set the *monitored* status and the payload is a byte that is whether true (1) or false (0)

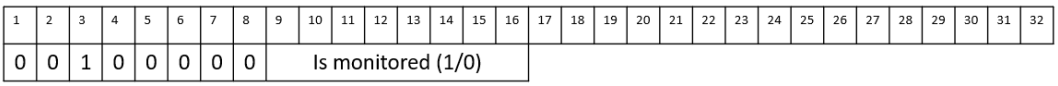


Figure 3.16: Update *monitored* status - uplink message

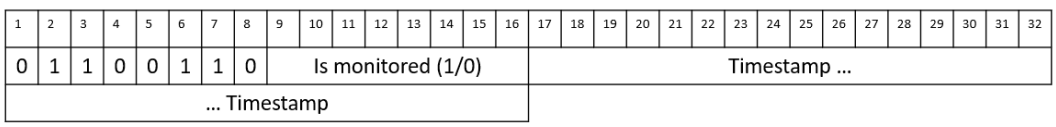
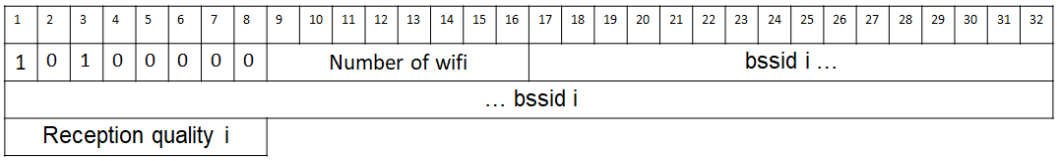


Figure 3.17: Update *monitored* status - downlink message

- A payload that contains BSSID and Wi-Fi reception quality in dB for each Wi-Fi access point that the device can found around him (used for WiPS, see section 2.3.2): The flag is equal to 5, the payload contains a byte to indicate the number of Wi-Fi contained in the payload, the BSSID (6 bytes) [1] and reception quality (1 byte) for each Wi-Fi.



- Others packets that do not need a payload can just be encoded in the flag with an optional 4 bytes timestamp. e.g. Request the device position, battery level or *monitored* status.

3.4 Backend and frontend softwares

Now that we have defined how the embedded software works and how it communicates with the backend, let's define how the backend and the mobile application (frontend) work and handle the communications with the device.

To give a quick overview of how the backend and frontend are composed and interact with each other and the device, see figure 3.18.

[1] The Basic Service Set Identifier (BSSID) is a 6 bytes MAC address that identifies the Wi-Fi. [23]

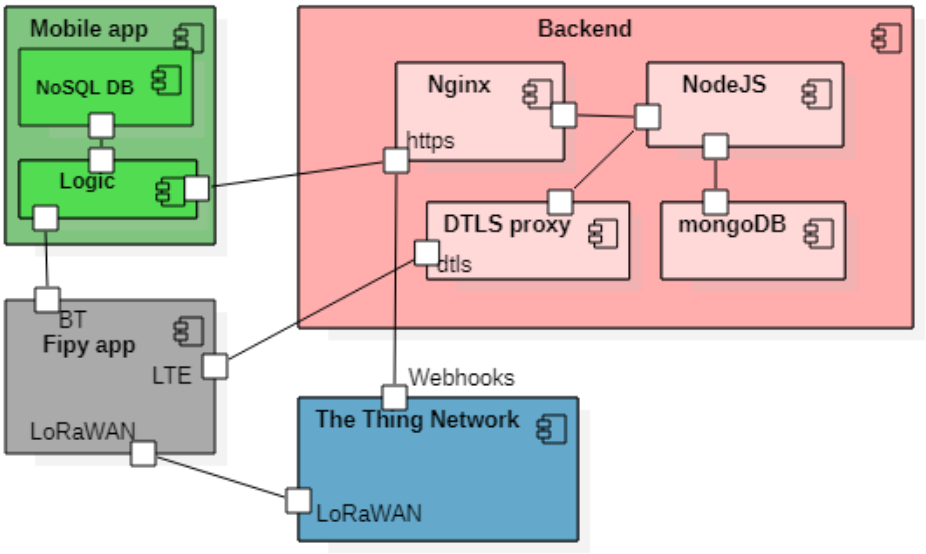


Figure 3.18: Overview of the software architecture

3.4.1 Backend

The backend has two main purposes: the first one is to store all data sent by the device and respond to the device. The second is to be an interface between the user (mobile application) and the device by allowing to send requests to the device via the appropriate network.

Let's now define the precise requirements that the backend must meet.
Interactions with the device:

- receive LoRaWAN (via The Thing Network) packets and store received data (location, battery level, frequency, monitored, etc).
- send requests to a specific device via the LoRaWAN network (The Thing Network) in order to inform the device to update its state or to request a specific information.
- respond to a device request via LoRaWAN network (The Thing Network)
- receive LTE requests from the internet and store received data
- respond to a device request coming from the LTE network.

- send requests to the device via LTE network in order to update the device state or to request a specific information.

Interface with the user: as almost all the requirements of the backend linked to the user interactions come from the mobile application features, see Section 3.4.2.2 for more information about these features.

3.4.1.1 General architecture

To make the backend suitable to all the requirements, we have divided the backend in different blocks. Each block is running docker containers in order to facilitate the development process among different environments and the deployment process. The use of docker also increases the security thanks to isolation (see more about security choices in Section 3.5).

The backend is composed of five main components (see figure 3.19 for the backend architecture and figure 3.18 for an architecture overview):

1. a NodeJS server: it contains all the logic of the backend. NodeJS has been chosen because of its capability of handle a lot of request simultaneously, huge packages library and our familiarity with the framework itself to optimise time in the prototype development.
2. a MongoDB storage: a NoSQL document database that contains devices information, state and data of each user. See Section 3.4.1.2 for more information.
3. a Nginx reverse proxy: the reverse proxy allows to secure the communication between the NodeJS server (and the swagger service) and the Internet.
4. a DTLS reverse proxy: it acts as a middleware between DTLS requests and the NodeJS server to serve requests done using LTE.
5. a Swagger openAPI: Swagger is an API documentation tool [25]. It is used to format and display the REST API shown by the NodeJS server. We will not cover this component further as it is mostly for development purposes.

The backend is hosted on the `Microsoft Azure` platform using virtual machines (during the development phase, the backend was self-hosted for easier and quicker debugging).

However, as all the services are running inside containers, it would not be difficult to use the `Azure Container Instance (ACI)` service. This allows to avoid maintaining or paying for a whole virtual machine. We have chosen Azure because we were already familiar with the service thanks to the course LINFO2145: Cloud Computing given by Prof. Étienne Rivière at UCLouvain. Nevertheless, we could very well have used `Amazon Elastic Container Service (ECS)` instead of ACI.

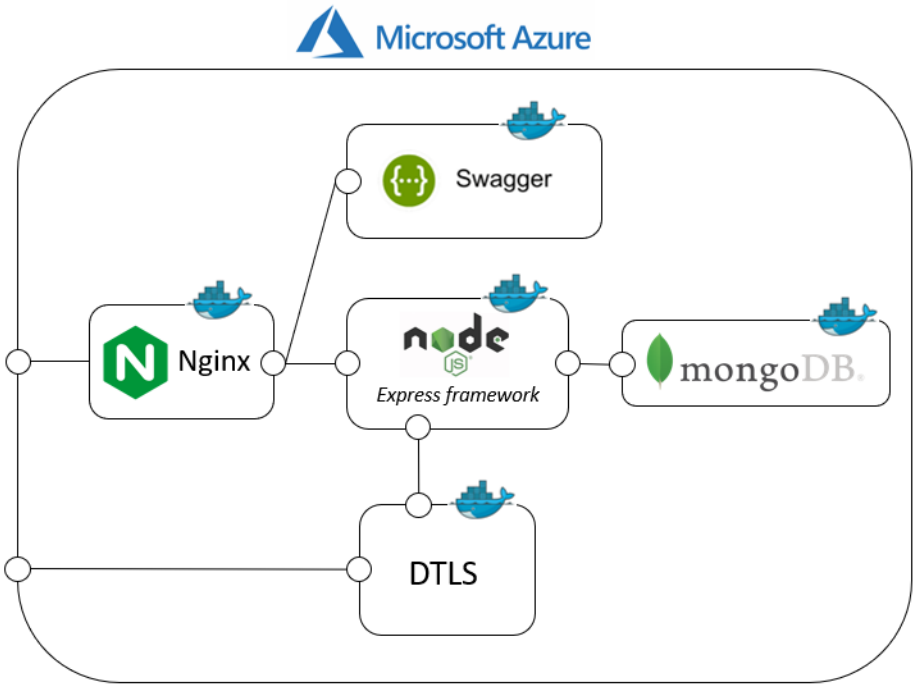


Figure 3.19: Backend architecture

Because the backend and the whole application have been built as a prototype, we assumed that the backend will not be deployed in a production environment. Therefore, we have not implemented scalability and deduplication in our design process. If the solution were to be adapted for production, we would have to rethink how the logic is running in the backend, maybe evaluate the possibility of introducing microservices to better serve scalability and reduce backend complexity. However, because the backend is already using a NoSQL database, the process of horizontal scaling should not be that hard to implement (see Section 3.4.1.2).

3.4.1.2 Storage

As already mentioned above, the chosen storage is MongoDB, a NoSQL document database.

The choice of using a NoSQL document database is mainly based on the possibility of constant change in the data schema as the master thesis and the prototype evolve. Using a relational database would cause to revisit the relational schema much more often as the prototype evolves. Moreover, the data are not strongly related to each other (these are only device states and configurations) so a relational database is not suitable.

MongoDB has been chosen because of the following advantages:

- Good scalability;
- Handles well a large number of requests, devices and users;
- Implements georeplication to avoid data loss [1]. This feature has not been implemented but might be relevant in a production environment.

Nevertheless, the use of another NoSQL database management system is possible as we do not have specific requirements regarding the NoSQL database.

In order to serve the device and the user interface, the following data are contained in the database:

- **user profile**: store all user information.
  - username
  - hashed password
  - devices identifier
- **device information**: store all device information.
  - device unique identifier
  - device name (can be changed by the user)
  - positions sent by the device - history
  - last battery level received by the device
  - last state received by the device, which contains the frequency at which the device sends position and if the device is monitored. When the device is monitored, it sends its position depending on the frequency.

- last frequency set by the user. It is a pending request waiting for the device to confirm the frequency change.
- last *monitored* status set by the user. It is a pending request waiting for the device to confirm the *monitored* status change.

We could imagine adding more fields related to the user to recover lost password, store user preferences, etc. But this is not the purpose of this thesis.

3.4.1.3 Communication with the device

As explained in Section 3.3.3.2, we are using DTLS as communication protocol to send information from the device to the backend and vice versa. To handle this protocol, we have implemented a DTLS service that acts as a reverse proxy between the DTLS requests and the NodeJS server.

The backend only sends data to the device via LTE when receiving requests (figure 3.22). Doing the same process over the LoRaWAN network is a little bit more complex. Indeed, when the backend wants to modify the on-device configuration, it must send the downlink request to the downlink queue of The Thing Network because it does not know when the device will pull the pending requests (figure 3.20 and 3.21).

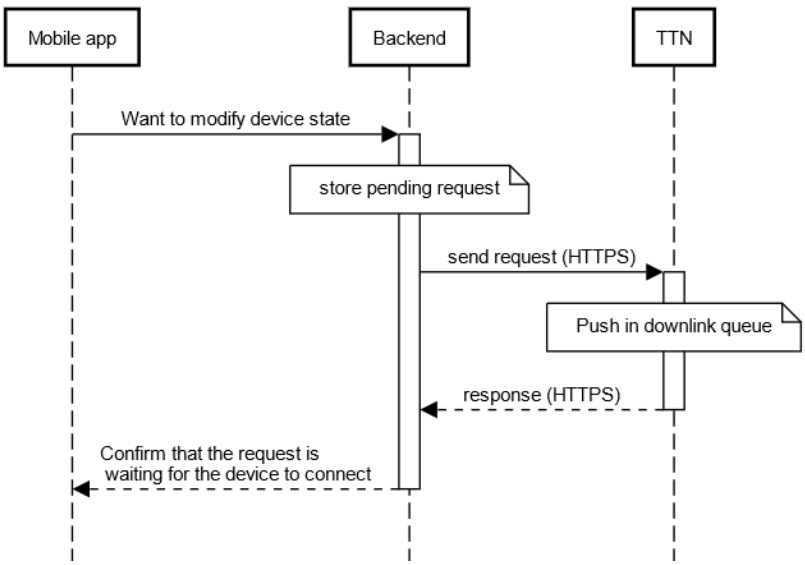


Figure 3.20: Push downlink packets from the backend

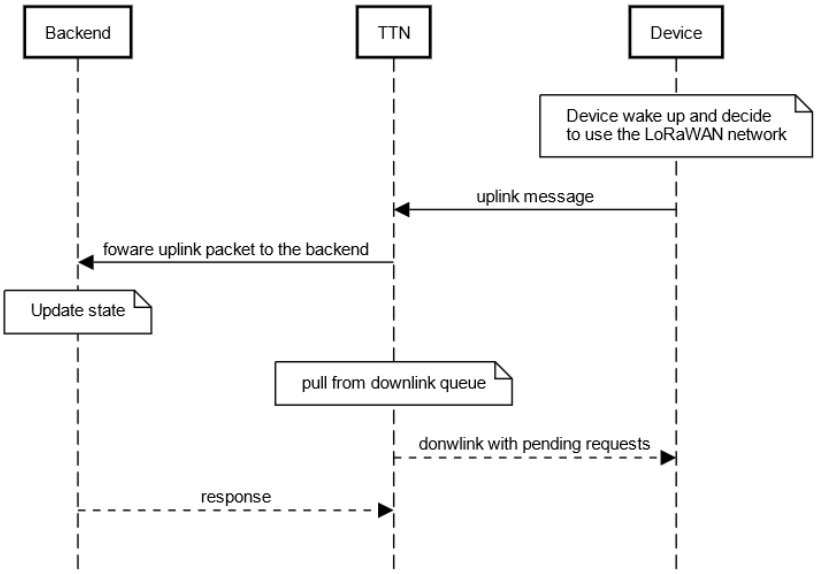


Figure 3.21: Pull downlink packets

If the backend keeps appending packets on the downlink queue, the queue may contain repetitive information and will cause the device to pull useless packets from the downlink queue, resulting in greater air time, higher bandwidth usage and increased power consumption of the device. That is why every time the backend wants to send a message to the device via LoRaWAN, it pulls the current downlink queue, extracts messages from packets, removes repetitive messages and replaces the downlink queue with a new packet containing the appropriate messages. This method ensures that there will have at most one packet (containing multiple messages) on the downlink queue and that the device will only have to pull one packet. In addition, also note that the backend clears the queue after sending all pending requests via LTE as the device does not need to pull this information from the LoRaWAN network anymore.

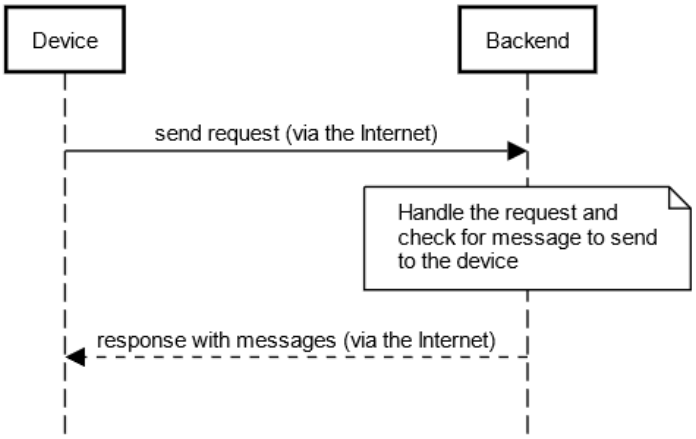


Figure 3.22: LTE message flow

3.4.2 Mobile application

The developed solution against thieves requires the user to be notified and to be able to quickly see the device location. As a result, the user interface has been implemented via a mobile application since we often have our phone at hand.

Therefore, developing the user interface on it makes perfect sense. Moreover, a smartphone allows to use Bluetooth to interact directly with the device.

3.4.2.1 Technology

Because the mobile application should run on all smartphones (both on iOS and Android), the use of a cross-platform framework seems more than appropriate to develop the mobile application.

The chosen cross-platform framework is Flutter. Flutter [13] in an open-source framework developed by Google using the Dart language (dart syntax is similar to java syntax). The framework has a great time-to-market, great performances, an easy way to build nice interfaces and universal look independent of the platform.

In addition, the database used by the mobile application to store data is a NoSQL document database.

3.4.2.2 Features

The mobile application includes the following features allowing the user to:

- Create an account in the application.
- Sign-in in the application.
- Sign-out of the application.
- Manage its devices by adding (only if the device is not already used by another user) or deleting one.
- Rename one device.
- See and change the frequency at which the device is sending its position, *monitored* status (if the device is sending its position or not) and the accelerometer feature (which enable the device to wake up when a movement occurs). When the user changes one of them, the modification is marked as pending and will be applied to the device as soon as the device pull the modification.
- See the last location of one device and its approximate accuracy.
- See and delete the history of location of one device.

Screenshots of these features implemented into the mobile application are available in appendix B.

3.5 Security and privacy

Firstly, the data collected by the application is limited to the device positions, username and password. No other user information is collected and stored by the application. The user can at any moment delete all these information by deleting the history position of its device and delete its account.

Secondly, when designing the solution with the device, backend and the communication between all components, security has been taken into account. Although we have not implemented all security features in the prototype, here are all safety-related design features:

1. Run the backend in a secured environment (azure, docker, nginx, ...): all the backend services are running in a secure environment, a nginx reverse proxy is being used to encrypt the communication between the user and the backend and a certificate to authenticate the backend.

2. Authenticate user: in order to prevent user to access unwanted data of other users, all requests specific to a user or a device must be done with an authentication token created with a secret that only the backend knows about. This allows to identify the user. The chosen token validity period is two weeks. However, we have not implemented a way to revoke token before the end of the validity period. If such a mechanism had been implemented, the backend should have stored the creation date in the token so that it could refuse tokens created before a certain date.

3. Secure sensible user data: as the position data are only shown and used by the user interface, we could imagine encrypting the position data with the user credential before sending the data on the network, therefore preventing anyone to see the device positions so that only the user could decrypt the location data of one of its devices thus providing end-to-end encryption between the device and the user. However, this would mean that the device should process encryption which could increase the payload length and the power consumption.

   The ”good enough” solution would be to encrypt the history of position only inside the database. We could go a little bit further by encrypting the data with a random encryption (e.g. AES), with an Order-preserving encryption to be able to detect if the device is moving or with both thanks to an onion layers encryption.

4. User data on the device: because the device sends its position directly to the server and does not store any previous movements or positions. The only data available to an attacker who would have physical access to the device would be the current device state and configuration (monitoring the position, wake up frequency and accelerometer enabled).

5. Secure communication between LoRaWAN and the backend: as the communication between the device and the The Thing Network through the LoRaWAN network is already secured thanks to the *AppKey* which is used to derive the session key used in the communication. And because the communication between The Thing Network and the backend is done using HTTPS and an API key to authenticate The Thing

Network, we can say that the communication between the device using LoRaWAN and our backend is secure.

6. Authenticate device from LoRaWAN network: As the device already uses a key to authenticate itself on the The Thing Network, the only way for an attacker to impersonate a device identity would be to know the pre-shared secret (AppKey) between the device and The Thing Network.

7. Secure communication between LTE network and the backend: Because we are using DTLS, we can say that as long as our pre-shared secret is not revealed, we can ensure that the communication between the device and the backend is secure.

8. Authenticate device from LTE network: As the device communicates directly with the backend via the Internet, we have decided that every uplink LTE message will be signed with the device private key. Therefore, the only way for an attacker to impersonate a device identity would be to know the device private key.

Finally, there are some security vulnerabilities that are difficult to solve due to the use of development boards.

- When communication using LoRaWAN or LTE-M/NB-IoT, encryption keys are stored in plain text on the device which would allow an attacker that has physical access to the device to extract these keys and use them to impersonate the device. This would allow the attacker to send fake data (device state and position) to the server/user. Unfortunately, even if encryption had been used to store these keys, the decryption key would always have been readable for the attacker as the USB port use for development purposes on the Pytrack 2.0X would also have been available for the attacker. Therefore, the attack would have been able to extract private keys and use them. A possible solution would be to manually remove the user interface (and recharge the battery other than with this port) so that the attack would not have directly access to the internal memory.
- An attacker that has physical access to the device could simulate GNSS messages on the board pins and therefore make the device believe that its position has not changed. This vulnerability is not fixable due to the use of development boards but could be solve or at least increase the difficulty of the hack by using a board with an integrated GNSS module.

3.6 Differences between the proposed solution and the first prototype

- Because of timing constraints, we decided to not implement the Bluetooth functionality in the prototype (backend, frontend and embedded software). Moreover, this feature is nice for the user in a final product (allows the user to interact with the device without using any network and it could be nice for initial configuration of the device) but does not bring much added value to the rest of the implementation.

  The timestamp in the downlink packets is also not implemented because it is only there to solve conflict between local configurations via Bluetooth and configurations via the mobile application.

- Because of embedded software limitation, implementing DTLS would have required to implement CoAP (Constrained Application Protocol). However, introducing CoAP into the embedded and backend design would have added unwanted complexity. Therefore, we have decided to simply use TLS to communicate between the device and the backend using LTE. However, due to the limitations of MicroPython, we could not implement TLS with PSK. Therefore, we used TLS with certificate.

- The security feature about protecting user sensible data (see Section 3.5, point 3) has not been implemented.

The implementation of the whole prototype can be found on its GitHub page (`https://github.com/hlibioulle/TFE-IoT`) and pictures of the prototype are shown in appendix A.

Chapter 4

Experiments

To determine which communication and positioning technology the device should use as well as to evaluate the provided service level of the first prototype (communication capabilities, battery life expectancy, movement detection capabilities, ...), some measurements had to be made. This chapter describes these experiments and their results. The two main experiments that have been made are:

- power consumption measurements to choose the technology that uses the least power, resulting in a better battery life;
- accuracy measurements to evaluate the feasibility of these technologies in Louvain-la-Neuve.

Before reading this chapter: this chapter only describes the experiments, how they have been made and their results but it does **not** analyse the results. The analysis and interpretations of the results and the comparisons between these technologies is covered in chapter 5.

4.1 Network services coverage

In term of networks, the device will use mainly 2 networks: LoRa or LTE-M/NB-IoT. Because of time constraints, NB-IoT has not been considered for network coverage as it requires to flash the LTE modem of the device in order to switch between LTE-M and NB-IoT. However, the use of the second device could have been imagined but it would have required a second

SIM card that we did not have. Moreover, the second device was use at the same time for the positioning services experiments (cfr. Section 4.2).

Nevertheless, some measurements of the NB-IoT network have been done to compare its coverage with the LTE-M coverage. See figure C.3 in appendix C.1 for the raw measurements.

4.1.1 Experiments

The experiments consisted of walking during a first part then cycling for a second part throughout the city with one of the two devices.

This device was configured to periodically check the LTE-M `RSSI` (Received Signal Strength Indication) and the LoRa `RSSI` then report both of them in a file on the SD card connected to the device.

To retrieve the LTE-M `RSSI` (Received Signal Strength Indicator), the following `AT` command (`AT` commands are the standard way to communicate with modems [79] [38] [32]) have been sent: `AT+CSQ` to get the signal strength. The answer has the following format: `+CSQ: <rssi (more=better)>, <ber (less=better)>`. The `rssi` value has to be converted to `dBm` using the table 4.1.

Original `RSSI` value	Converted `RSSI` [`dBm`]	Condition
1	< -111	Bad
$2 \rightarrow 9$	$-109 \rightarrow -95$	Marginal
$10 \rightarrow 14$	$-93 \rightarrow -85$	OK
$15 \rightarrow 19$	$-83 \rightarrow -75$	Good
$20 \rightarrow 30$	$-73 \rightarrow -53$	Excellent
31	> -51	Perfect
99	/	Unknown / Undetectable

Table 4.1: Conversion between value returned by `AT+CSQ` command and `dBm` [38] [32]

For LoRa, the default settings have been used for the measurements and for the prototype that is a spreading factor (`SF`) of 7, a coding rate (`CR`) of 1 and a bandwidth (`BW`) of 125 `kHz`.

To retrieve the LoRa `RSSI`, a packet has to be sent then a function from Pycom's API (`<LoRa object>.stats()`) is called which returns useful information (including an `RSSI`) about the **last** received LoRa/LoRaWAN packet, that is why the measurement requires a

packet to be sent over LoRa before checking the stats (to get the latest information).

In order to receive back a packet, the LoRa socket was set in `CONFIRMED` mode. The `CONFIRMED` mode is used to receive back a response from the network server (inside The Thing Network network) when it receives the packet.

As the device will go into deep sleep a lot then wake up to quickly send its state, a time limit to wait for the packet to be sent has been set to 10 seconds otherwise, the device would stay on for too long (instead of going into deep sleep) to have a good battery life. If the device does not receive a confirmation within 10s (and 2 retransmissions, which is the default number of retries for a `CONFIRMED` packet), the `send(...)` call will time out.

To have a more precise measurement of the LoRa coverage as seen by the device, the limit could have been increased to 30 seconds or more then plot the results depending on the time taken for the packet to be sent and received. However, this would have required a lot more time and the measurement would have been more tedious (because of changing weather condition and GNSS availability).

4.1.2 Results

Figures 4.1 and 4.2 show the coverage measurement result for LTE-M and LoRa networks respectively.

Cells are squares of 64m by 64m and an average of the data per cell has been computed from the raw data available at figure C.1 for the LoRa network and at figure C.2 for the LTE-M network in the appendices.

The legend (top right corner) is in `dBm` and a **RED** cell means no signal at all on the whole cell.

For the LoRa network, a `RSSI` of less than -120 `dBm` is often assumed to be bad while values between -120 `dBm` and -50 `dBm` are often assumed to be good [64] [65] [51]. For the LTE-M network, the `RSSI` scale is based on the scale returned by the `AT+CSQ` command [38] [32] [49] [42] [37].

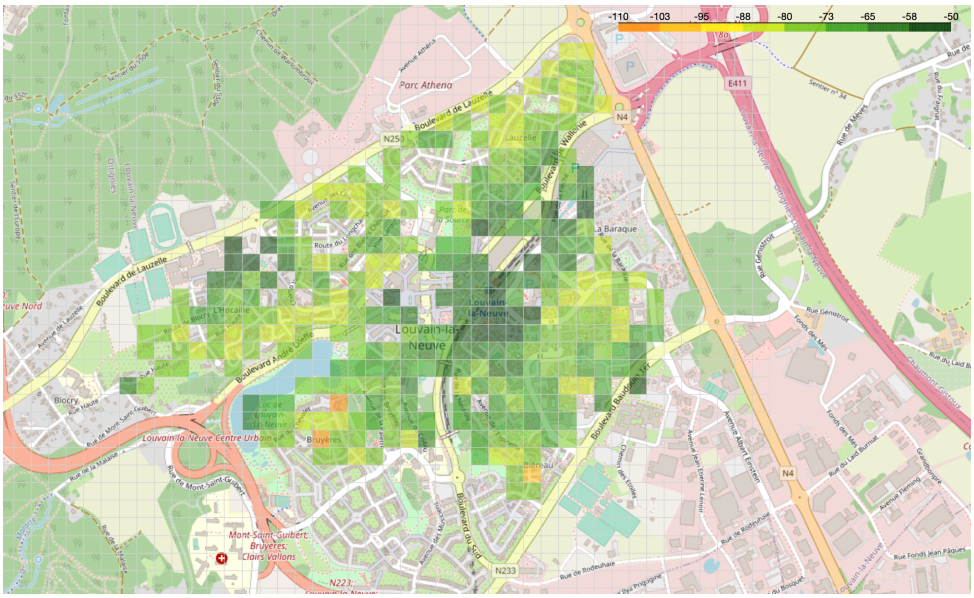


Figure 4.1: LTE-M coverage around Louvain-la-Neuve

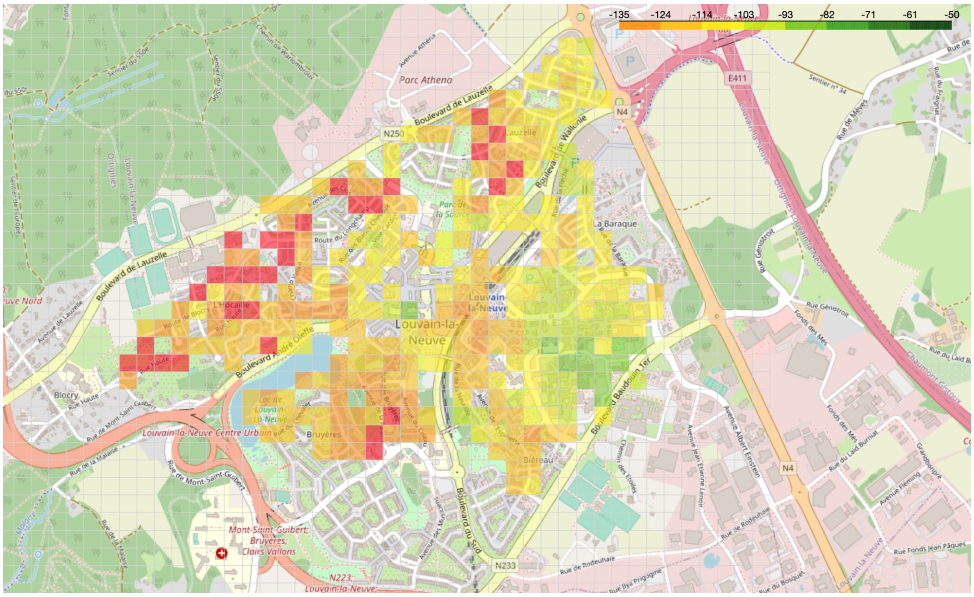


Figure 4.2: LoRa coverage around Louvain-la-Neuve

The analysis and interpretation of these results are done in Section 5.1.1.

4.2 Positioning services accuracy

As a reminder, the device is using two technologies to be localised, GNSS and WiPS. The goal of this experiment is to determine which one of these two technologies is the best in term of accuracy and reliability.

4.2.1 Experiments

The experiment to measure both GNSS and WiPS consisted of going all around Louvain-la-Neuve on a bike and making frequent stops to measure the GNSS position and scan Wi-Fi access points with the device itself in order to compare the position measurement with the actual position obtained with the phone.

The information collected during these measurements were:

- the position reported by the GNSS module
- the Wi-Fi access points that the device saw.
- the phone position which report our actual position (accuracy of $\approx$ 4 meters). This position will be used as the reference position to compare the accuracy of both GNSS and WiPS.

The positions obtained with the collected Wi-Fi access points were computed afterwards by requesting the Google geolocalisation API (see Section 2.3.2 for more information). For the GNSS measurements, a measure was marked as failed when the duration to acquire the GNSS position had exceeded 30 seconds.

4.2.2 Results

After capturing a lot of data all around LLN, here are the results obtained.

Figures 4.3 and 4.4 show the accuracy in meters of GNSS and WiPS depending on the location in Louvain-la-Neuve (no measurements have been made in squares **without** colour). Cells are squares of 64m by 64m and an average of the data per cell has been computed. The precise measurement points can be found on figure C.4 in appendix C.2.

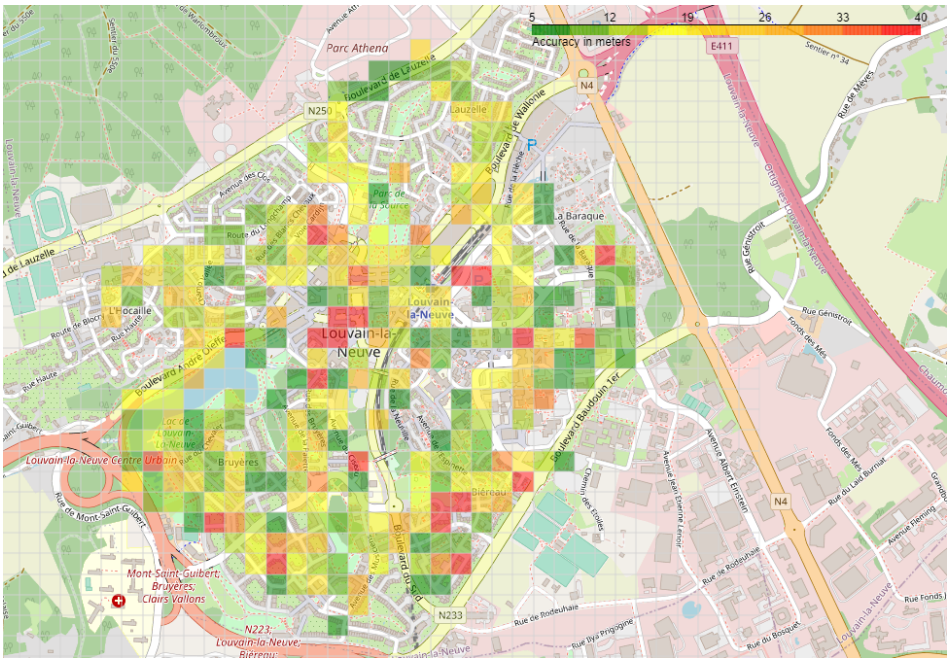


Figure 4.3: GNSS module accuracy (in meter) in LLN

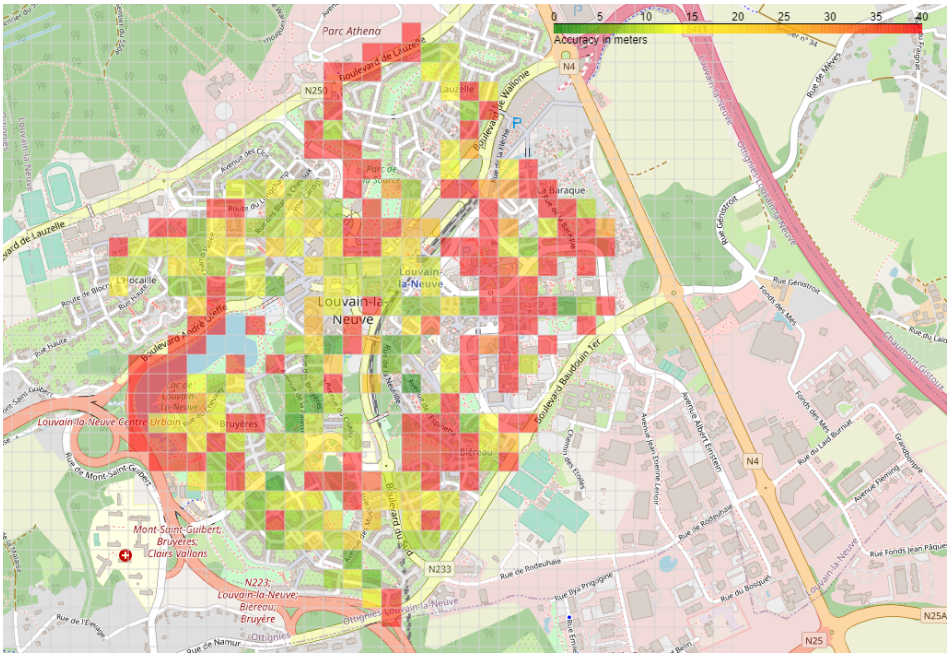


Figure 4.4: WiPS solution accuracy (in meter) in LLN

To give a general idea of the accuracy of both positioning systems, figure 4.5 shows the accuracy distribution for GNSS and WiPS based on measurements done on each cell of figures 4.3 and 4.4 (outliers are not visible).

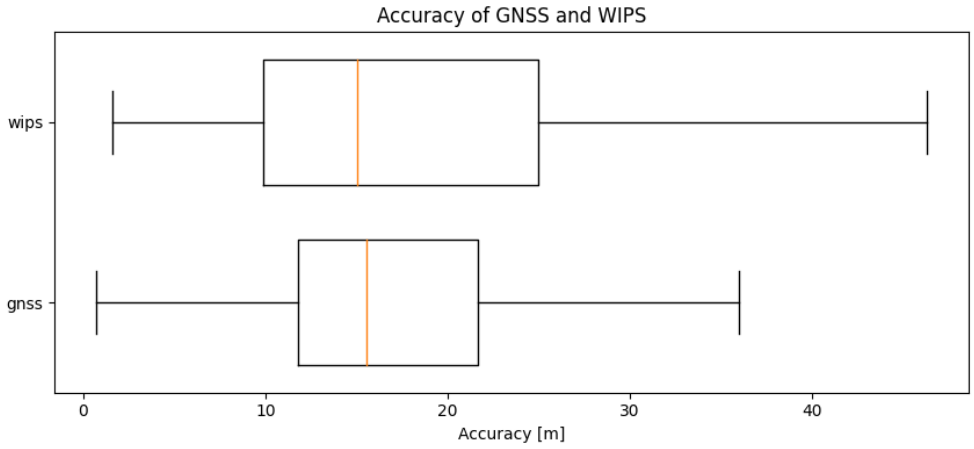


Figure 4.5: Accuracy of GNSS and WiPS

The analysis and interpretation of these results are done in Section 5.2.1.

4.3 Power consumption measurements

To evaluate the battery life of the device, multiple power consumption measurements have be done. This includes the following scenarios: sending/receiving/processing LoRaWAN or LTE packets, getting GNSS position, capturing close Wi-Fi, use accelerometer and measure the standard power consumption of the device.

All power consumption measurements were made with the Graphical Sampling Multimeter `Keithley DMM7510` [10] and the power supply `DC Power Supply PS3003` [15] via the UCLouvain WELCOME platform [30].

For all measurements done with the device (FiPy + Pytrack), measures have been done according to the schematic on figure 4.6 with a voltage of $3.8V$.

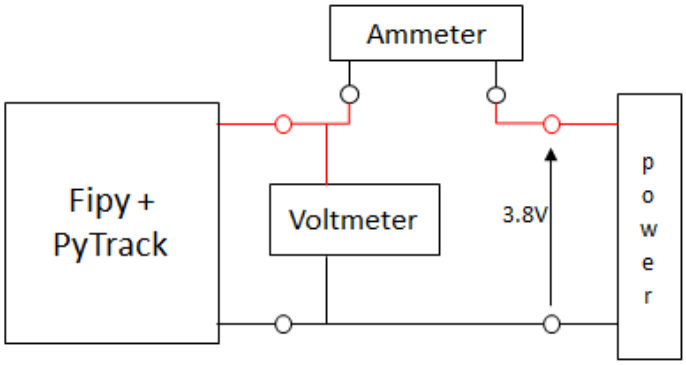


Figure 4.6: Schematic of power consumption measurement for the entire device

4.3.1 Development board and extension shield

4.3.1.1 Experiments

The following measurements have been made to know the different possible power consumption of the development board (FiPy) with the extension board (Pytrack):

- Idle consumption of Pytrack and FiPy together.
- Pytrack in deep sleep without maintaining power on the FiPy and with the GNSS module in standby mode.
- Pytrack in deep sleep without maintaining power on the FiPy and with the GNSS module in normal mode.
- Pytrack in deep sleep with the GNSS module in standby mode and the FiPy in sleep (light) mode.
- both the Pytrack and the FiPy (kept on power) are in deep sleep with the GNSS module in standby
- Pytrack in idle and the FiPy is put in deep sleep.
- Pytrack in idle and the FiPy is put in sleep (light) mode.

4.3.1.2 Results

Here are the results of the measurements regarding the consumption of the boards themselves without any communication service running.

As a first observation, measurements show that the idle power of both the Pytrack with the FiPy is 258 `mW` and that the power consumption goes a little bit higher when the devices boot. That is why the power consumption right before and after the deep sleep are different for a short time. Because this observation is recurrent, it will not be discussed in the following measurements.

Figures 4.7 and 4.8 show the power consumption of the device when the Pytrack is put on deep sleep without maintaining the FiPy on power and while putting the GNSS module in standby and normal mode. Although these two figures are very similar, the average power is quite different. The measurements show an average of 0.112 `mW` with the GNSS module in standby mode and 0.546 `mW` in normal mode.

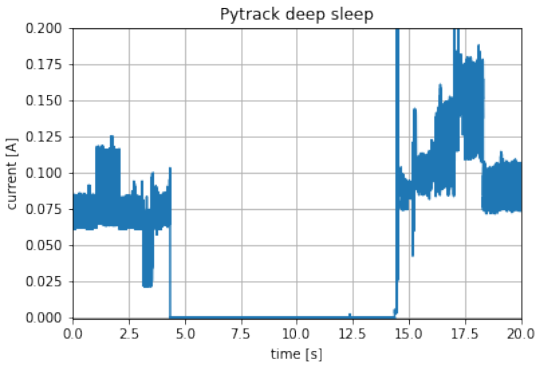


Figure 4.7: Pytrack in deep sleep, FiPy not powered and GNSS in standby mode

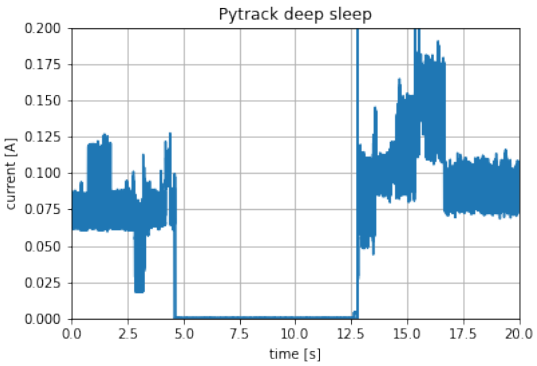


Figure 4.8: Pytrack in deep sleep, FiPy not powered and GNSS in normal mode

Figure 4.9 shows the power consumption of the device when the Pytrack is put in deep sleep and the FiPy in sleep (light sleep). The average power used during the sleep period is 16.806 `mW`.

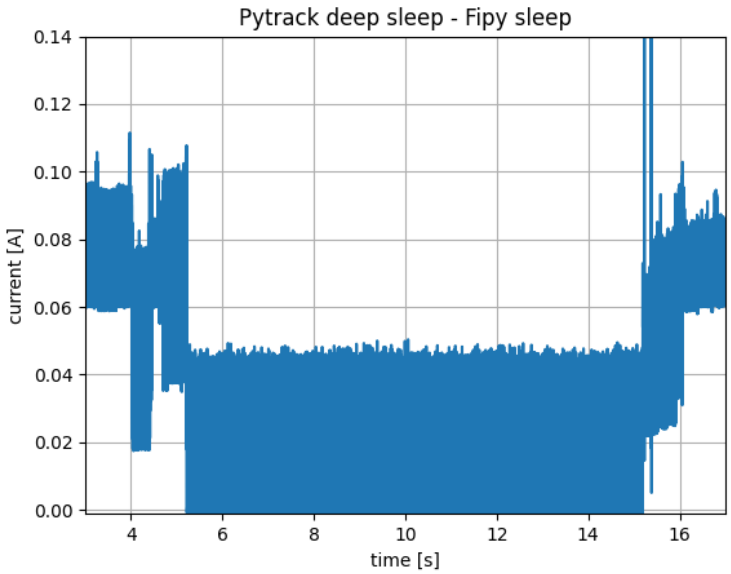


Figure 4.9: Pytrack in deep sleep and FiPy in (light) sleep

Figure 4.10 shows the power consumption of the device when the Pytrack and FiPy are put in deep sleep with the GNSS module in standby mode and the accelerometer ready to wake up the FiPy when a specific threshold of acceleration is exceeded.

Both the Pytrack and the FiPy are in deep sleep after the ≈5th second for ≈2.0 seconds. Then, the accelerometer detects an acceleration exceeding the threshold and sends a signal to the FiPy to wake it up. After the wake up, the power consumption did not come back to its initial value because only the FiPy woke up and the Pytrack is still in deepsleep. The Pytrack wakes up after the ≈21st second and the initial power consumption appears again after the ≈26th second.

The average power used by the device is 2.21 `mW` during its deep sleep period with the accelerometer powered.

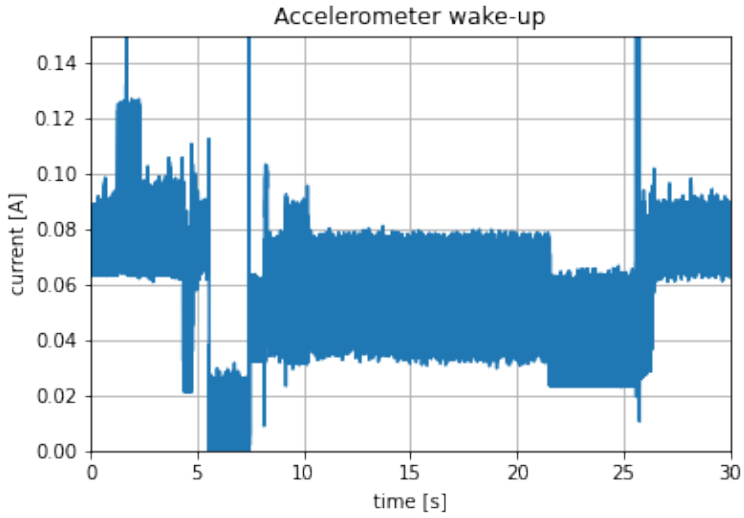


Figure 4.10: Accelerometer wakes up the FiPy and Pytrack sleeps for 20 seconds

Figure 4.11 shows the power consumption of the FiPy and the Pytrack when only the FiPy is put in sleep. The two high peaks have been added manually (led at highest intensity) in order to surround the sleep period. The average power used during the sleep period is 122.57 `mW`.

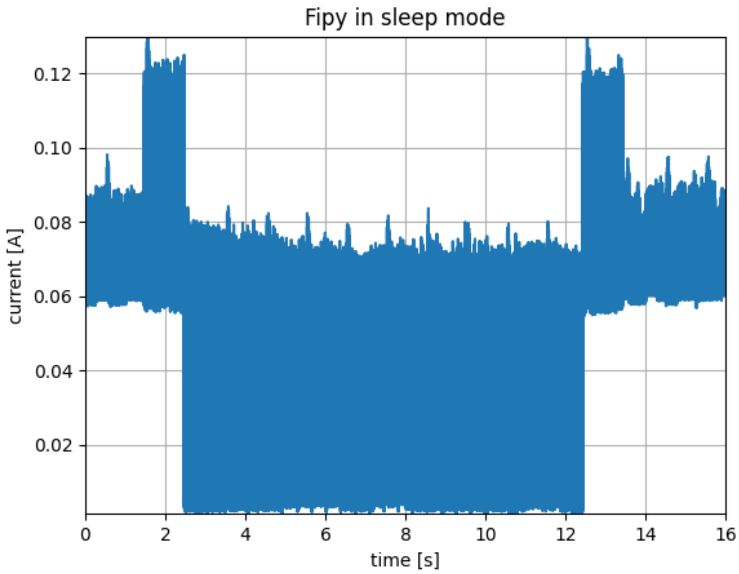


Figure 4.11: FiPy put in sleep

Figure 4.12 shows the power consumption of the FiPy and the Pytrack when only the FiPy is put in deep sleep. The average power used during the deep sleep period is 112.09 `mW`.

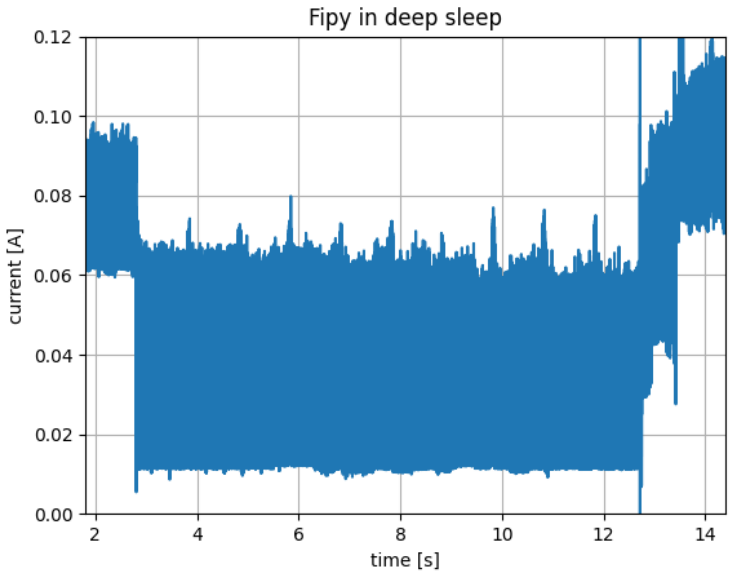


Figure 4.12: FiPy put in deep sleep

The analysis and interpretation of all these results are done in Section 5.3.

4.3.2 Communication services

The goal of these experiments is to determine the power efficiency of LoRa, LTE-M and NB-IoT.

4.3.2.1 Experiments

To test the power consumption of LoRa, LTE-M and NB-IoT, the following measurements have been made:

- The sending of a packet (push the battery level) through The Thing Network using LoRa (resulting in a payload size of 2 `bytes`) and directly to the backend using LTE-M and NB-IoT (resulting in a payload size of 266 `bytes`, including a 256 `bytes` signature).
- The reception of a packet (update the frequency $\Rightarrow$ 3 `bytes`) through The Thing Network (using LoRa) or directly from the backend (using LTE-M or NB-IoT).

4.3.2.2 Results

Figure 4.13 shows the sending and reception of a packet to and from TTN. The first highest pick is when the device joins the LoRaWAN network. The second highest pick is when the packet is sent. And the third one (≈ 14.5th second) is the reception.

The average power used when sending and receiving is 258.64 `mW` for a duration of 13.49 seconds, resulting in a consumption of 0.9871 `mWh`.

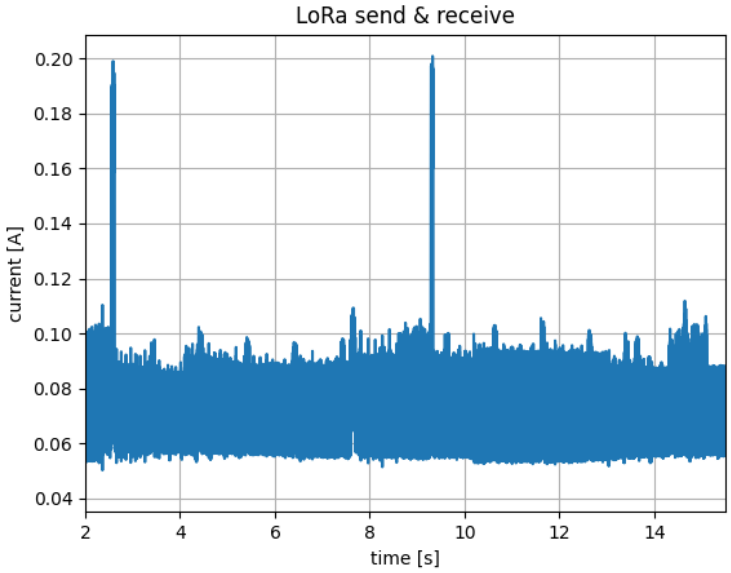


Figure 4.13: Send and receive packet using LoRaWAN

Figure 4.14 shows the sending and reception of a packet using LTE-M. The device starts to connect to the antenna during ≈5 seconds, then it resolves the server address, sends the packet and receives a packet. All picks with constant interval from the 5th second are TCP messages send to maintain the TCP/TLS connection. The pick after ≈11 seconds is the disconnection from the socket. Finally, the device detaches itself from the network and comes back to its initial state and consumption.

The average power used during these operation is 532.73 `mW` for a duration of 16.8 seconds which results in an energy of 2.486 `mWh`.

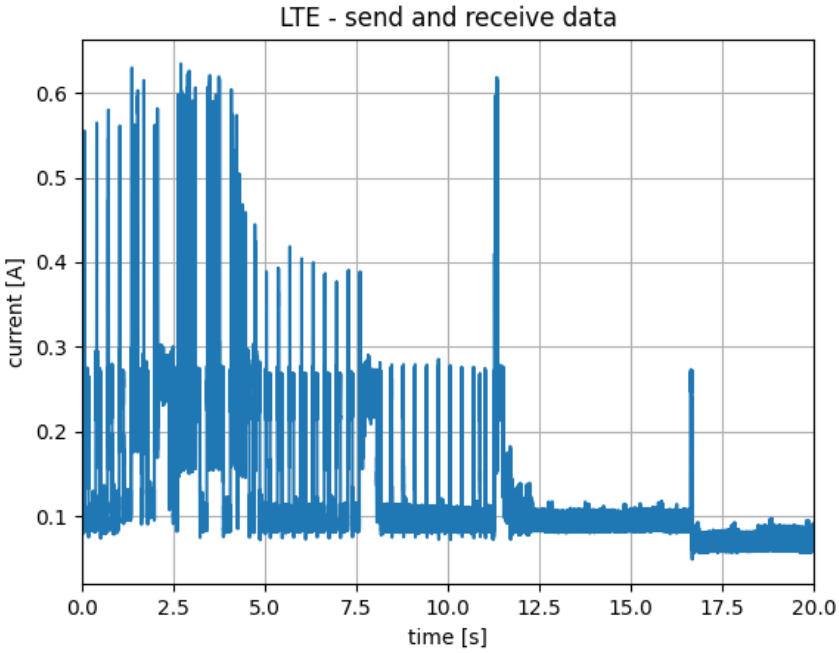


Figure 4.14: Send and receive packet using LTE-M

Figure 4.15 shows the sending and reception of a packet using NB-IoT. From the ≈8th second to the ≈22th second, the device attaches itself to the network, then the two dense picks at ≈27th second and ≈32nd second are the connection to the server and the sending of a packet. The two last picks at ≈38th second are the reception of the packet from the server. Finally, after the ≈45th second, the device detaches itself from the server and the network. The average power used during the exchange of packet in NB-IoT is 670.82 `mW` for a duration of 59.25 seconds, resulting in an energy of 11.0406 `mWh`.

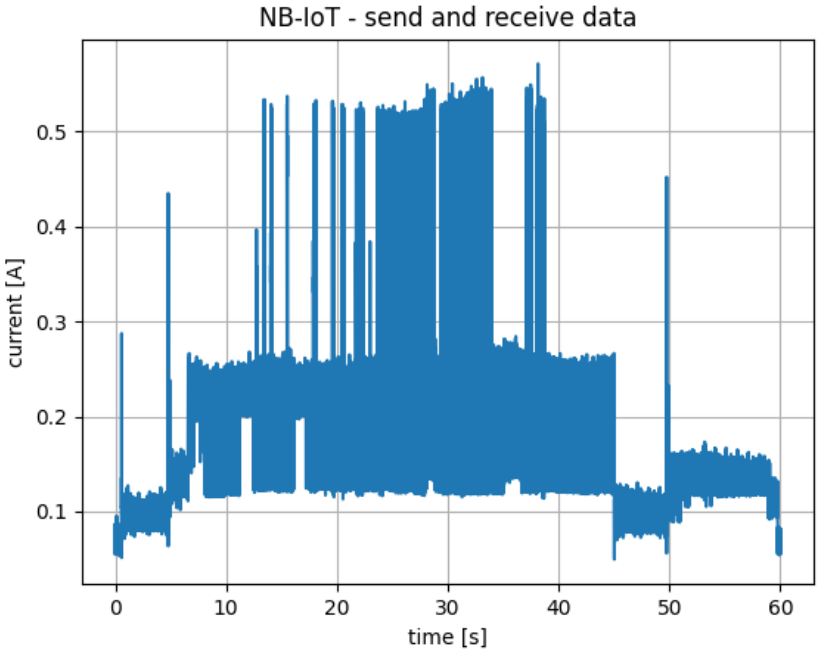


Figure 4.15: Send and receive packet using NB-IoT

The analysis and interpretation of all these results are done in Section 5.1.2.

4.3.3 Positioning services

4.3.3.1 Experiments

The following measurements have been made to measure the power consumption of both GNSS and WiPS:

- scan Wi-Fis around the device in active mode;
- scan Wi-Fis around the device in passive mode;
- get the GNSS position when the device has been kept on power (standby mode) since the last location;
- fail to get the GNSS position after the device has booted without being in deep sleep.

4.3.3.2 Results

Figure 4.16 shows the power when scanning Wi-Fis in active mode. Higher peaks as high as 1306 `mW` are visible when the device sends (broadcast) a packet on the network. The average power used by the scanning operation is 534 `mW` for a duration of 2.22 seconds, resulting in an energy of 0.329 `mWh`.

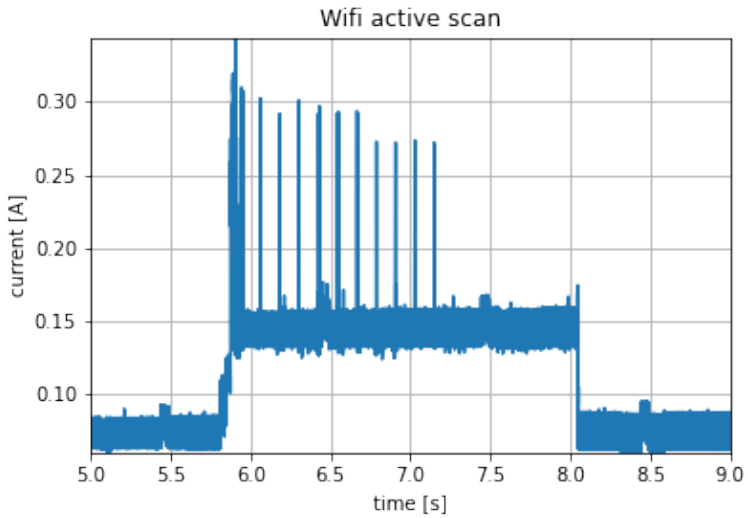


Figure 4.16: Active Wi-Fi scanning

Figure 4.17 shows the power when scanning Wi-Fis in passive mode. The average power used by the scanning is 527 `mW` for a duration 4.7 seconds, resulting in an energy of 0.6883 `mWh`.

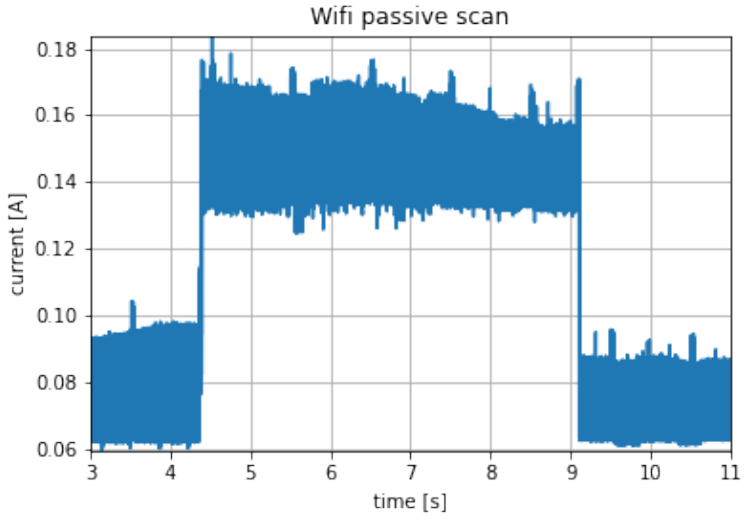


Figure 4.17: Passive Wi-Fi scanning

Figure 4.18 shows the power consumption when the device successfully obtains a GNSS position. The average power used is 298.285 `mW` for a duration of 2.2 seconds, resulting in an energy of 0.1822 `mWh`.

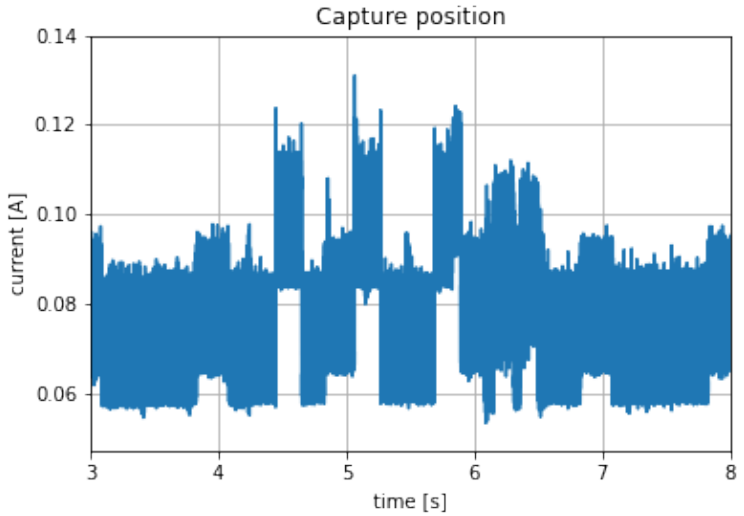


Figure 4.18: GNSS position obtained successfully

Figure 4.18 shows the power usage when the device fails to obtain a GNSS position. The average power used is 298 `mW` for a duration of 10.3 seconds, resulting in an energy of 0.8546 `mWh`.

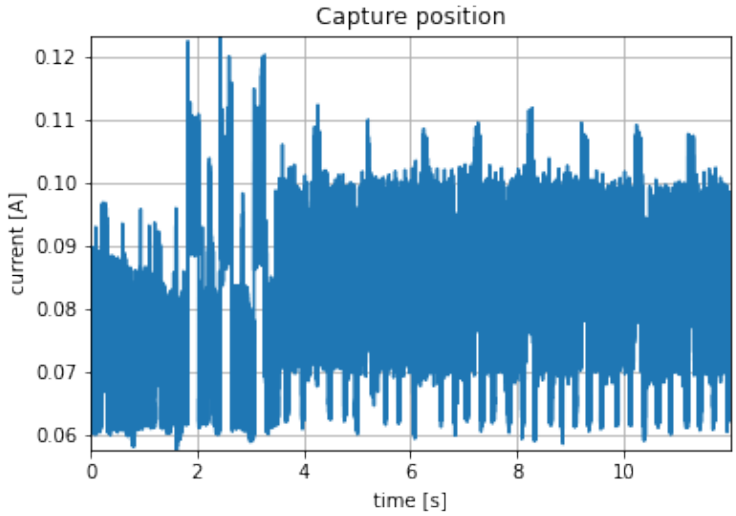


Figure 4.19: GNSS position failed

The analysis and interpretation of these results are done in Section 5.2.2.

Chapter 5

Analysis and discussion

The goal of this chapter is to compare multiple technologies with respect to the prototype objectives, evaluate which one is the most suitable for the device depending on the environment and conclude assumptions on the device final capabilities.

More precisely, this chapter will mostly discuss the coverage of communication technologies, the accuracy of positioning systems, power consumption of used technologies and the battery life of the prototype.

Most of the analysis done in the following chapter rely on the measurements done in chapter 4 and a lot of references to figures and observations of this chapter will be done.

5.1 Network services

5.1.1 Coverage

5.1.1.1 LoRa coverage

Unlike LTE-M, the LoRa network has a good coverage only in the south of the city, where the antenna is located. Throughout the rest of the city, the `RSSI` is not that good (-135 `dBm` to -105 `dBm`) but is enough for the application as the packets are really small (from 1 `bytes` to 222 `bytes` which is the maximum allowed for the used configuration [3]), thus requiring a low bandwidth for the packets to be sent in an acceptable amount of time.

For the LoRa measurements, as explained in Section 4.1.1, the socket was set to `confirmable` mode and a hard limit on the total sending time has been imposed (by using a timeout on the socket) to 10`s`. Let's see if this limit had an impact on the measurements accuracy/reliability.

Because we are using LoRa default settings, an uplink packet airtime is between ~41.2 `ms` for a payload of 1 `byte` and ~363.8 `ms` for a maximum payload of 222 `bytes` while a downlink packet airtime will be between ~41.2 `ms` and ~44.3 `ms` (the payload has a size between 1 `byte` and 3 `bytes`). A confirmation packet for an uplink packet (if not piggybacked $\iff$ no payload) would take ~39.2 `ms`.

These airtimes have been computed according to LoRaWAN standards using the following formulas [29] [4] [17]:

$$
\begin{aligned}
H &= 1 \Rightarrow \text{as the device uses LoRaWAN so the header is enabled} \\
DR_{opti} &= 0 \Rightarrow \text{as the LoRa low data rate optimisation is } \textbf{not} \text{ enabled because we are using } \texttt{DR5} \\
Time_{symbol} &= (2^{SF}/BW) * 1000 \\
Time_{preamble} &= (Length_{preamble} + 4.25) * Time_{symbol} \\
\#Symbol_{payload} &= 8 + max\left[ceil\left(\frac{8 * Size_{payload} - 4 * SF + 28 + 16 - 20 * H}{SF - 2 * DR_{opti}}\right) * \frac{CR + 4}{4}, 0\right] \\
Time_{payload} &= \#Symbol_{payload} * Time_{symbol} \\
Time_{packet} &= Time_{preamble} + Time_{payload}
\end{aligned}
$$

where $Size_{payload}$ is the full LoRaWAN payload size (including the LoRaWAN MAC header (about 9 `bytes` when no MAC commands are included), the application payload, and the MIC (4 `bytes`)), $Time_{symbol}$ is the time to send a symbol, $Time_{preamble}$ is the time to send the LoRa preamble (the 4.25 constant corresponds to synchronisation bytes[2] [68]), $\#Symbol_{payload}$ is the number of symbol in the payload, $Time_{payload}$ is the time to send the payload and $Time_{packet}$ is the time to send the whole packet.

For the measurements, the `confirmed` mode was enabled and the sent packet was a battery level packet (1 `byte`) so, with an error margin of 10% about the airtimes, the packet could have been sent ~220 times (during the period of 10`s`), which seems more than enough for it to arrive. The packet is not retransmitted if no confirmation is received because the measurements were automatic and the device was cycling between measurements of LoRa and measurements of LTE-M. Therefore, if the packet was lost because of an accidental reason,

the few previous and next measurements would correct this erroneous measurement as the results correspond to averages of all measures per cell.

In conclusion, this hard limit is not that constraining for the measurements.

For the prototype, the default LoRa settings seem adequate as they provide a maximum payload size of 222 `bytes` (required for a good accuracy using WiPS) and an airtime between ~41.2 `ms` and ~363.8 `ms` which is acceptable to achieve a good battery life (by avoiding a too long operation time).

5.1.1.2 LTE-M coverage

Unsurprisingly, figure 4.1 shows that there is a really good coverage of LTE-M throughout the city and even more in the center of Louvain-la-Neuve, as the antenna is situated on top of the main administration building of the university, Pl. de l'Université 1, 1348 Ottignies-Louvain-la-Neuve.

In some isolated places (south and west of the city), that you can more precisely see on the raw data at figure C.2 of the appendix, the signal can sometimes drop but only during a short amount of time. It is probably caused by inaccuracies of the device and some combination of external events (away from the antenna and surrounded by buildings).

The cellular connectivity (LTE-M and NB-IoT) is mostly configured by the provider/antenna and the modem in order to provide the best quality. This is why the device does not provide much configurability except for the PSM/eDRX features (which do not influence the signal quality). This explains why only the signal quality has been analysed and not different configurations as the developer does not have access to the modem configuration.

5.1.1.3 A word about NB-IoT coverage

The measurements (figure C.3 in appendix C.1) have been made to prove that its coverage is similar to the coverage of the LTE-M network. The results show that NB-IoT coverage in Louvain-la-Neuve is similar to LTE-M coverage which is not surprising as the technology and modulation behind NB-IoT and LTE-M are not completely different as explained in Section 2.2.2.

5.1.2 Power consumption comparison of LoRa, LTE-M and NB-IoT

After having discussed the LoRa and LTE-M coverage (assumption that NB-IoT coverage is similar to LTE-M coverage), let's talk about LoRa, LTE-M and NB-IoT power consumption.

The table 5.1 (based on experiments in chapter 4) clearly shows that sending and receiving a packet using LoRa is the most power efficient technology compared to LTE-M or NB-IoT. The second power efficient technology is LTE-M. Finally, NB-IoT is the one that uses the most energy. The very high power consumption of NB-IoT can be explain by the fact that it takes a lot of time to attach itself to the cellular antenna and more time to exchange TCP packets as the bandwidth is smaller than LTE-M.

With a more optimised version, that is without using TCP/TLS and using PSM/eDRX, NB-IoT should consumes less than LTE-M.

	Power [mW]	**Duration [s]**	**Energy [mWh]**
Send and receive packet with LoRa	258.64	13.49	0.9871
Send and receive packet with LTE-M	532.73	16.8	2.486
Send and receive packet with NB-IoT	670.82	59.25	11.0406

Table 5.1: Device power measurement - communication technologies

For the network technologies, it is important to compare energies and not just powers as the network technology will influence a lot the total duration of the communication for a same scenario (in this case, sending a battery level packet and receiving a frequency change request).

Given these observations, the device should first try to send packets using LoRa and then try to use LTE-M if the LoRa signal is too weak. This execution order prevents using a lot of energy (using LTE-M) for nothing if the device is capable of sending using LoRa.

5.2 Positioning services

5.2.1 Accuracy and reliability

Firstly, lets discuss the accuracy and the reliability of both GNSS and WiPS technologies. Figures 4.3 and 4.4 show that WiPS has a much weaker accuracy outside of the center of LLN

than GNSS. This can be easily explained by the fact that the number of Wi-Fi access points available is higher in the center.

Figure 4.5 also shows that the GNSS is more reliable and stable as its IQR (Interquartile Range) is smaller than the WiPS one and the maximal value (excluding outliers) is higher with WiPS than with GNSS.

Moreover, no experiments have been made but we can assume because of the number of Wi-Fi access points, that the use of WiPS is not suitable in rural environment.

5.2.2 Power consumption

Regarding the power usage of these two technologies, here is a small reminder of the measures in chapter 4:

	Duration [s]	**Power [mW]**	**Energy [mWh]**
Successfully getting GNSS position outside in ideal conditions	2.2	298.285	0.1823
Failed to get GNSS position	10.3	298	0.8547
Active scan of Wi-Fi access points	2.2	534	0.329
Passive scan of Wi-Fi access points	4.7	527	0.688

Table 5.2: Device power measurement - positioning system

These data show that in ideal conditions, when the GNSS module already got a fix position, requesting the GNSS position consumes less than WiPS. However, if the GNSS position acquisition takes too long then the power consumed will exceed the power consumed by the Wi-Fi scan.

The measurements also show that the active Wi-Fi scanning is more power efficient than the passive mode. Indeed, even if the active scan uses more power (not by a significant margin), the total energy used to retrieve Wi-Fi is smaller than using passive scanning because of the duration.

5.2.3 Which one to use ?

Given that the GNSS is more reliable than WiPS and that WiPS can be useful when we cannot have GNSS position, the solution for the device is to use WiPS as a backup positioning system when the GNSS position is not available or takes too long to be acquired.

This prevent the GNSS to consume to much energy for nothing by waiting for a GNSS fix for too long.

The type of Wi-Fi scan used by the device will be an active scanning as it uses less energy than passive scanning.

5.3 Battery life expectancy

This section is going to evaluate the power consumption of the device and therefore estimate its battery life in days depending on different factors related to the use of the device.

As a small reminder of the different power consumption of the possible operations that the device is doing, here is an additional table to tables 5.1 and 5.2 (data coming from experiments chapter):

	Power [mW]
Idle consumption	258
Pytrack in deepsleep (GNSS in standby) and FiPy OFF	0.112
Both Pytrack and FiPy in deepsleep	2.21

Table 5.3: Device power measurement - Board

The battery life of the device given that the battery has a voltage of $3.7V$ can be given by

$$\text{Expected battery life} = \frac{Capacity_{battery}\ [mAh]}{Energy_{cycle}\ [mWh]} * 3.7 * Duration_{cycle} = \frac{Energy_{battery}\ [mWh]}{Energy_{average}\ [mW]} \tag{5.1}$$

where a cycle is defined as the period from one wake-up of the device to another depending on the set frequency (frequency/day). The energy used during one cycle is given by:

$$Energy_{cycle} = Energy_{awake}\ [mWh] + \frac{Power_{sleep}\ [mW]}{3600} * \frac{24 * 3600}{frequency} \tag{5.2}$$

$$Power_{sleep} = \begin{cases} 0.545\ \texttt{mW} & \text{if the accelerometer feature is disabled} \\ 2.21\ \texttt{mW} & \text{if the accelerometer feature is enabled} \end{cases} \tag{5.3}$$

The power used during the sleep period (equation 5.3) are values with the GNSS in normal mode and **not** in standby mode. Even if the standby mode consumes less power than the normal mode, the use of the normal mode in deep sleep can be justified by the fact that acquiring a fix GNSS position can be costly and it is therefore worth it to try keeping the fix GNSS by using the normal mode. Moreover, the power consumption with the most impact on battery life is $Energy_{awake}$.

Equation 5.2 still depends on the energy used by the device to operate when it is awake (retrieving position, send received data, update its state, ...). This energy can be defined as:

$$Energy_{Awake} = isMonitored * Energy_{getPosition} + Energy_{sndRcvPacket} + Energy_{computation} \tag{5.4}$$

$$isMonitored = \begin{cases} 1 & \text{if the device monitors its position} \\ 0 & \text{if the device do not monitor its position} \end{cases} \tag{5.5}$$

$$Energy_{getPosition} = \begin{cases} \frac{298.285\ [mW]}{3600} * Time_{positioning} & \text{if succeed retrieving the gnss position} \\ \frac{298.285\ [mW]}{3600} * Timeout + 0.329\ \text{mWh} & \text{if failed retrieving the gnss position} \end{cases} \tag{5.6}$$

where $Time_{positioning} < Timeout$ and $Timeout = 5\ \texttt{s}$.

$$Energy_{sndRcvPacket} = \begin{cases} 0.9871\ \texttt{mWh} & \text{if the device uses LoRa} \\ 2.486\ \texttt{mWh} & \text{if the device uses LTE-M} \end{cases} \tag{5.7}$$

$$Energy_{computation} = \frac{258.647\ [mW]}{3600} * Time_{computation} \tag{5.8}$$

Where $Energy_{computation}$ is the energy used for $Time_{computation}$ seconds to process information in addition to the other device task.

$Duration_{cycle}$ in equation 5.1 is the sum of time used by each operation done by the device depending on the situation and can be defined as:

$$Duration_{cycle} = Time_{sleep} + isMonitored * Time_{positioning} + Time_{snd/rcv\ pkt} + Time_{computation} \tag{5.9}$$

$$Time_{positioning} = \begin{cases} < Timout & \text{if succeed retrieving the gnss position} \\ timeout + 3.33 & \text{if failed retrieving the gnss position} \Rightarrow WiPS \end{cases} \tag{5.10}$$

$$Time_{snd/rcv\ pkt} = \begin{cases} 13.5 & \text{if the device decides to use LoRa} \\ 16.8 & \text{if the device decides to use LTE-M} \end{cases} \tag{5.11}$$

Timing measurements on the device prototype show that the computation time ($Time_{computation}$) is around 27 seconds. This timing is highly dependent on the prototype implementation and could be improved in an optimised prototype (better code logic and a lower-level language, e.g. C/C++, instead of MicroPython).

With all these equations and values, the estimated battery life depending on the frequency, the monitored status and whether or not the accelerometer feature is enabled can be computed.

However, if the user often changes the state of the device, this will produce additional uplink and downlink connections and therefore decrease the expected battery life. This use case has not been taken into account into the model as it highly dependent on the user, therefore it is very difficult to predict.

The evaluation of LoRa/LTE-M signal quality has not been taken into account in the battery life estimation as this operation is not done at every wake-up (it can be configured in the code to do the measurement every wake-up, 2 wake-ups, ...).

Moreover, the expected battery life (with a margin of 5 % on both the cycle energy consumption and the cycle duration) with a lithium battery of 1100 `mAh` is given depending on the three main states in which the device can be in. The user is free to choose (via the mobile application) the one that best suits his needs. These options/behaviours/states are the following :

1. The device **is not** monitoring/sending its position **nor** monitoring its movements and just wake up according to the set frequency to send its estimated battery percentage and to check if its state must be update.

2. The device **is** monitoring/sending its position periodically depending to the set frequency but **not** its movements.

3. The device **is not** monitoring/sending its position periodically but **will wake up** and send its position if the device is moving (thanks to the accelerometer).

4. The device **is** monitoring/sending its position periodically and **will wake up**, send its position if the device is moving (thanks to the accelerometer).

However, in the tables 5.4 to 5.7, the `accelerometer` option assumes the `monitoring` is **disabled**. Indeed, both can be enabled at the same time but it does not make sense as the device will wake up and advertise its position if it moves, as configured in the tracker software (so advertising the position periodically is not useful).

The following battery life expectations are given in the best case (table 5.4 - using LoRa and GNSS), the worst case scenario (table 5.7 - using LTE-M and WiPS) and two additional intermediate cases (table 5.5 - using LoRa & WiPS and table 5.6 - using LTE-M & GNSS).

Wake-up frequency		1m	5m	15m	30m	1h	2h	5h	12h	1d
Monitoring	`ON` 1.6	5.2	14.0	26.2	47.9	82.6	147.4	212.6	252.6	
	`OFF`	1.7	5.5	14.8	27.6	50.3	86.3	152.1	216.6	255.4
Accelerometer	`ON`	1.6	5.2	12.6	20.7	31.1	41.8	52.7	58.8	61.3

Table 5.4: Expected battery life in days - best case (LoRa and GNSS)

Wake-up frequency		1m	5m	15m	30m	1h	2h	5h	12h	1d
Monitoring	`ON`	1.4	4.5	12.0	22.6	41.6	73.0	134.7	201.2	244.4
	`OFF`	1.7	5.5	14.8	27.6	50.3	86.3	152.1	216.6	255.4
Accelerometer	`ON`	1.6	5.2	12.6	20.7	31.1	41.8	52.7	58.8	61.3

Table 5.5: Expected battery life in days - intermediate case 1 (LoRa and WiPS)

Wake-up frequency		1m	5m	15m	30m	1h	2h	5h	12h	1d
Monitoring	`ON`	1.1	3.6	9.6	18.2	34.0	61.1	117.7	184.5	231.6
	`OFF`	1.1	3.7	10.0	18.9	35.3	63.0	120.6	187.5	234.0
Accelerometer	`ON`	1.1	3.6	8.9	15.4	24.6	35.5	48.4	56.4	59.9

Table 5.6: Expected battery life in days - intermediate case 2 (LTE-M and GNSS)

Wake-up frequency		1m	5m	15m	30m	1h	2h	5h	12h	1d
Monitoring	`ON`	1.0	3.3	8.7	16.4	30.8	55.7	109.4	175.9	224.7
	`OFF`	1.1	3.7	10.0	18.9	35.3	63.0	120.6	187.5	234.0
Accelerometer	`ON`	1.1	3.6	8.9	15.4	24.6	35.5	48.4	56.4	59.9

Table 5.7: Expected battery life in days - worst case (LTE-M and WiPS)

Appendix C.3 gives the battery life expectation results but with the GNSS module in standby mode during the deep sleep period.

5.4 Analysis conclusion

The battery life results are acceptable for a first prototype/proof of concept as it can achieve few weeks to few months of lifetime.

As an example, let's imagine that 40% of time the LoRa network is available and 70% of the time the GNSS is available, the combined expected battery life is:

$$0.4*0.7*Case_{best}+0.4*0.3*Case_{intermediate_1}+0.6*0.7*Case_{intermediate_2}+0.6*0.3*Case_{worst}$$

Moreover, the configuration using the accelerometer with a wake-up frequency of 1h seems reasonable and relevant for the problem we are trying to solve. Therefore, the expected battery life would be in the example:

$$0.4 * 31.5 + 0.6 * 24.9 \approx 27.5 \texttt{ days}$$

which is an acceptable battery life considering that the device is a prototype.

An improvement could be the use of the standby mode in deep sleep resulting in approximately twice the battery life when a low frequency is set (see appendix C.3). No significant improvement should be noticed with high frequency (every 1,5 or 10 minutes) as the sleep time is therefore smaller than lower frequency. However, the impact of the standby mode on the time and condition to acquire the GNSS position should be studied to prevent the inefficiency of the GNSS module when locating the device.

The results also clearly show that LTE-M and NB-IoT offer a real alternative to LoRa in terms of coverage but its usage will impact the battery life (depending of the frequency). Actually, the usage of LTE-M all the time instead of LoRa would reduce the expected battery life by ~2.3% to ~35.3%. However, LTE-M/NB-IoT in conjunction with LoRa is mandatory to have a reliable connection with the device as the current coverage of LoRa, shown in figure 4.2 of Section 4.1.2, is quite low (even more outside Louvain-la-Neuve or in more rural areas).

Chapter 6

Conclusion

This present master thesis aimed to design a new solution to detect and prevent bicycle theft in Louvain-la-Neuve using LoRa and cellular networks. Then, the challenge was to evaluate this solution by building a first prototype to estimate battery life, coverage and localisation accuracy.

Even though similar out of the box solutions, like the one developed by Invoxia [14], exist today on the market, they often cannot be highly reliable in terms of connectivity coverage as they often only use LoRa network. Because of the coverage limitation of LoRa, the usage of LTE-M/NB-IoT has been explored as an additional communication technology to increase the coverage and communication reliability.

What has been achieved?

To achieve the master thesis goals, we designed an embedded device supporting LoRa, LTE-M/NB-IoT and Bluetooth (but not used in the prototype) connectivity as well as an accelerometer and GNSS capabilities to detect movements and track positions.

The implemented device uses hardware development boards from Pycom (FiPy and Pytrack 2.0X) which already have all the required hardware features. Both LoRa and LTE-M/NB-IoT network are used thanks to respectively The Thing Network and Orange cellular network provider.

Moreover, we developed a MicroPython embedded software, an application server as well

as a mobile application. The device sends its position periodically or when the device detects a movement. The user can visualise the position and positions history on its mobile application as well as modifying the device configuration.

Through this master thesis, we successfully managed to realise a proof-of-concept prototype to detect bicycle theft and localise bikes. We also successfully evaluated the feasibility of both LoRa and LTE-M/NB-IoT as viable radio communication systems and concluded that the use of both technologies is a must to offer a suitable bike tracking solution. Indeed, using both LoRa and LTE-M/NB-IoT makes it possible to overcome the respective defects of each of these technologies in terms of coverage and power consumption. However, the usage of LoRa should be privileged as often as possible over LTE-M/NB-IoT as it consumes less energy.

Areas of improvements

Although the final solution is complete and answers to all the desired requirements, for the first prototype we had to make compromises in some areas. Therefore, here are the areas of the current implementation where improvements can be made and which we felt it was important to mention and which are important to hightight:

- Implement the Bluetooth to get rid of the daily limit of downlink message imposed by the LoRaWAN provider (named "The Thing Network") and therefore only use LTE and Bluetooth for most downlink messages;
- Switch from TLS to DTLS in the prototype to reduce the traffic and power consumption used by LTE communications;
    - A more optimal implementation would communicate via LTE using Non-IP Data Delivery (NIDD) instead of IP [72]. This will reduce packet size and transfer responsibility for authentication and encryption from the application server to the network provider. For this, the application server would have to use the Network Exposure functions of the LTE network provider (if they are available) in order to communicate with the end device [33].
- Design an external GNSS antenna integrated into the bike to improve the GNSS reception and accuracy.
- Make the device compatible with a bike dynamo to recharge the device and therefore highly increase the battery life.
- Improve privacy in the application server such as explained in chapter 3 (Section 3.5).

It would also be interesting in the future to design a case and the prototype around the PyGo2 (by Pycom) [11] when it is available as this device answers to all the hardware requirements, contains all the needed technologies and its form factor is very small and therefore interesting. However with this kind of small form factor, the battery life could be lower.

Closing thoughts

Even if this device has been designed for a bike and in the context of bike theft, one could very well imagine a use of this device with very small adjustments for something other than its original purpose. The device could be used as a simple tracker for high worth assets (e.g. construction or industrial assets).

Finally, with the quick rise of IoT, smart cities, green transportation and electric bikes, it is highly suitable that the future can only foresee an increase in similar solutions to the one realised in this master thesis to fight against theft and encourage commuting by bicycle. Hoping that this master thesis and proof of concept will encourage future initiatives to not only rely on closed proprietary networks and hardware but also use public accessible networks as well as open hardware and to invest in a community to increase the power (coverage) of such collaborative networks which LoRa is part of.

Bibliography

[1] How To Scale MongoDB. https://www.mongodb.com/basics/scaling. [Online; accessed 12. Mar. 2022].

[2] LoRa packet structure @ ResearchGate. https://www.researchgate.net/figure/LoRa-packet-structure_fig1_316999094, May 2017. [Online; accessed 30. Apr. 2022].

[3] LoRaWAN - Packet size considerations. https://lora-developers.semtech.com/documentation/tech-papers-and-guides/the-book/packet-size-considerations, Oct. 2017. [Online; accessed 30. Apr. 2022].

[4] LoRaWAN airtime calculator. https://github.com/avbentem/airtime-calculator/blob/master/src/lora/Airtime.ts, July 2020. [Online; accessed 30. Apr. 2022].

[5] BikeTrax GPS tracker e-bike - GPS tracking with PowUnity. https://powunity.com/en/product/e-bike-gps-tracker, Dec. 2021. [Online; accessed 29. Apr. 2022].

[6] Wi-Fi positioning system - Wikipedia. https://en.wikipedia.org/w/index.php?title=Wi-Fi_positioning_system&oldid=1062972571, Dec. 2021. [Online; accessed 17. Apr. 2022].

[7] Yabby Edge - Indoor/Outdoor Battery-Powered GPS Tracker. https://www.digitalmatter.com/our-devices/yabby-edge, Aug. 2021. [Online; accessed 4. May 2022].

[8] Bike Hawk GPS Tracker and Computer. https://www.coolshityoucanbuy.com/2015/03/bike-hawk-gps-tracker-and-computer.html, Feb 2022. [Online; accessed 12. Mar. 2022].

[9] Device Classes. https://www.thethingsnetwork.org/docs/lorawan/classes, Apr. 2022. [Online; accessed 13. May 2022].

[10] DMM7510 7.5 Digit Graphical Sampling Multimeter. https://www.tek.com/en/products/keithley/digital-multimeter/dmm7510, Apr 2022. [Online; accessed 2. Apr. 2022].

[11] Fipy - Pycom. https://pycom.io/product/pygo2, Mar. 2022. [Online; accessed 7. May 2022].

[12] Fipy - Pycom. https://pycom.io/product/pygo1, Mar. 2022. [Online; accessed 7. May 2022].

[13] Flutter. https://flutter.dev, Feb 2022. [Online; accessed 8. Mar. 2022].

[14] Invoxia Bike Tracker. https://www.invoxia.com/uk/bike-tracker, Mar 2022. [Online; accessed 12. Mar. 2022].

[15] Lab power supply 0-30v / 0-3a dual lcd display. https://www.velleman.eu/products/view/?country=fr%E2%9F%A8=en&id=340425, Apr. 2022. [Online; accessed 16. Apr. 2022].

[16] LoRa vs LoRaWAN. https://www.thethingsnetwork.org/docs/lorawan/what-is-lorawan/, 2022. [Online; accessed 6. May 2022].

[17] LoRaWAN airtime formula and calculator. https://www.rfwireless-world.com/calculators/LoRaWAN-Airtime-calculator.html, 2022. [Online; accessed 30. Apr. 2022].

[18] LoRaWAN Architecture. https://www.thethingsnetwork.org/docs/lorawan/architecture/, Apr. 2022. [Online; accessed 6. May 2022].

[19] MLS - Overview. https://location.services.mozilla.com, Apr. 2022. [Online; accessed 27. Apr. 2022].

[20] National Marine Electronics Association - NMEA. https://www.nmea.org/content/STANDARDS/NMEA_0183_Standard, May 2022. [Online; accessed 1. May 2022].

[21] Overview | Geolocation API | Google Developers. https://developers.google.com/maps/documentation/geolocation/overview, Apr. 2022. [Online; accessed 27. Apr. 2022].

[22] Qualification | Sigfox build. https://build.sigfox.com/study, May 2022. [Online; accessed 13. May 2022].

[23] Service set (802.11 network) - Wikipedia. https://en.wikipedia.org/w/index.php?title=Service_set_(802.11_network), Feb 2022. [Online; accessed 11. Mar. 2022].

[24] SIGFOX.COM. https://www.sigfox.com/en, Apr. 2022. [Online; accessed 9. May 2022].

[25] Swagger UI. https://swagger.io/tools/swagger-ui, Mar 2022. [Online; accessed 8. Mar. 2022].

[26] The Thing industries Downlink Queue. https://www.thethingsindustries.com/docs/devices/downlink-queue-ops, Mar 2022. [Online; accessed 8. Mar. 2022].

[27] The Thing industries WebHooks. https://www.thethingsindustries.com/docs/integrations/webhooks, Mar 2022. [Online; accessed 8. Mar. 2022].

[28] The Things Network. https://www.thethingsindustries.com/docs/getting-started/ttn, May 2022. [Online; accessed 27. May 2022].

[29] The Things Network - LoRaWAN airtime calculator. https://www.thethingsnetwork.org/airtime-calculator, 2022. [Online; accessed 30. Apr. 2022].

[30] Welcome - Wallonia Electronics & Communications Measurements. https://sites.uclouvain.be/welcome/index.php?page=qui-sommes-nous&l=fr, Apr 2022. [Online; accessed 2. Apr. 2022].

[31] 3GPP. 3GPP - Release 13. https://www.3gpp.org/release-13, 2015. [Online; accessed 27 May 2022].

[32] 3GPP. AT command set for User Equipment (UE). https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=1515, 2022. [Online; accessed 27 May 2022].

[33] 3GPP. Network Exposure function and NIDD. https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3642, 2022. [Online; accessed 31 May 2022].

[34] Allaer, Emmanuel and Sadutto, Maurizio. #Investigation : vol de vélos, un fléau hors de contrôle. https://www.rtbf.be/article/investigation-vol-de-velos-un-fleau-hors-de-controle-10998532, June 2022.

[35] BikeFinder. BikeFinder — GPS-sporing & forsikring. https://www.youtube.com/watch?v=f6B0EoIfFHg, Jul 2019. [Online; accessed 12. Mar. 2022].

[36] De Keersmaeker, François & van de Walle, N. Using LoRaWAN and Wi-Fi for smart city monitoring in Louvain-la-Neuve. http://hdl.handle.net.proxy.bib.ucl.ac.be/2078.1/thesis:30517, 2021. Promotors: Deville, Yves; Sadre, Ramin. [online, accessed 12th March 2021].

[37] EMnify. At commands for cellular modules. https://www.emnify.com/developer-blog/at-commands-for-cellular-modules. [Online; accessed 23-April-2022].

[38] ETSI. AT command set for User Equipment (UE). https://www.etsi.org/deliver/etsi_ts/127000_127099/127007/08.11.00_60/ts_127007v081100p.pdf, 2022. [Online; accessed 27 May 2022].

[39] European Telecommunications Standards Institute (ETSI). ETSI EN300.220. https://www.etsi.org/deliver/etsi_en/300200_300299/30022002/03.02.01_60/en_30022002v030201p.pdf, June 2018. [Online; accessed 4. May 2022].

[40] Goouuu Tech. GT-U7 GPS Module. https://images-na.ssl-images-amazon.com/images/I/91tuvtr02jL.pdf. [online, accessed 12th March 2022].

[41] Heltec Automation. Wi-Fi LoRa 32 (V2). https://heltec.org/project/wifi-lora-32/. [online, accessed 8th March 2022].

[42] Induo. At commands. https://www.induo.com/wp-content/uploads/2014/03/at-kommandon-conel-router.pdf. [Online; accessed 23-April-2022].

[43] Jeffrey, C. *An Introduction to GNSS: GPS, GLONASS, Galileo and Other Global Navigation Satellite Systems*. NovAtel, 2010.

[44] Keersmaeker, F. D., and van de Walle., N. Using LoRaWAN and Wi-Fi for smart city monitoring in Louvain-la-Neuve. http://hdl.handle.net/2078.1/thesis:30517, 2021. Promotors: Deville, Yves; Sadre, Ramin. [Online; accessed 23-April-2022].

[45] LoRa alliance. Lorawan regional parameters. https://lora-alliance.org/resource_hub/rp2-1-0-3-lorawan-regional-parameters/, 2021. [Online; accessed 4-May-2022].

[46] LoRa alliance. LoRaWAN specification v1.0.3. https://lora-alliance.org/wp-content/uploads/2021/05/RP-2-1.0.3.pdf, 2022. [Online; accessed 13. May 2022].

[47] LoRa alliance. LoRaWAN specification v1.1. https://lora-alliance.org/resource_hub/lorawan-specification-v1-1/, 2022. [Online; accessed 4. May 2022].

[48] Ltd., A. What are IoT Devices. https://www.arm.com/glossary/iot-devices, May 2022. [Online; accessed 6. May 2022].

[49] M2M Support. At+csq command. https://m2msupport.net/m2msupport/atcsq-signal-quality/, 2022. [Online; accessed 23-April-2022].

[50] MakerFocus. 3.7V 1100mAh Li-Po Batteries. https://tinyurl.com/makerfocus-1100-mah-battery. [online; accessed 8. Mar. 2022].

[51] Mobilefish. Lora network. https://lora.readthedocs.io/en/latest/. [Online; accessed 23-April-2022].

[52] Network, T. T. End device activation. https://www.thethingsnetwork.org/docs/lorawan/end-device-activation/, 2022. [Online; accessed 7. May 2022].

[53] Olivier Perdaens, M. D., and Borges, M. A. D. C. Versatile and scalable IoT platform for efficient asset management. http://hdl.handle.net/2078.1/thesis:30610, 2021. Promotors: Sadre, Ramin. [Online; accessed 23-April-2022].

[54] Orange. eDRX. https://iotjourney.orange.com/fr-FR/support/faq/qu'est-ce-que-l'edrx, 2022. [Online; accessed 13. May 2022].

[55] Orange. PSM. https://iotjourney.orange.com/fr-FR/support/faq/qu%27est-ce-que-le-psm-ou-mode-psm-(power-saving-mode), 2022. [Online; accessed 13. May 2022].

[56] Orange Belgium. Orange IT solutions - IoT. https://business.orange.be/fr/it-solutions/internet-things. [online, accessed 12th March 2022].

[57] Ortholand, J.-M. Connectivity, protocols, security and IoT needs: how to find the right path. https://www.orange-business.com/en/blogs/connectivity-protocols-security-and-iot-needs-how-find-right-path, Dec 2020. [Online; accessed 19. Mar. 2022].

[58] Pycom. FiPy specsheet. https://development.pycom.io/gitbook/assets/specsheets/Pycom_002_Specsheets_FiPy_v2.pdf, Feb 2022. [online, accessed 8th March 2022].

[59] Pycom. Pytrack 2.0X product page. https://pycom.io/product/pytrack-2-0-x/, Feb 2022. [online, accessed 8th March 2022].

[60] Pycom. Pytrack 2.0X specsheet. https://docs.pycom.io/gitbook/assets/PyTrack2X_specsheet.pdf, Feb 2022. [online; accessed 8. Mar. 2022].

[61] Quectel. GNSS L76-L. https://www.quectel.com/product/gnss-l76-l, Feb 2022. [online; accessed 8. Mar. 2022].

[62] RTL Info. 85.000 vélos sont volés chaque année en Belgique: voici la somme astronomique que cela ... https://www.rtl.be/info/belgique/societe/85-000-velos-sont-voles-chaque-annee-en-belgique-voici-la-somme-astronomique-que-cela-c aspx, Dec. 2021.

[63] Sanchez-Iborra, R., Sanchez-Gomez, J., Pérez, S., Fernandez, P., Santa, J., Hernández-Ramos, J., and Skarmeta, A. Enhancing lorawan security through a lightweight and authenticated key management approach. *Sensors 18* (06 2018).

[64] Schoenmaeckers, R. Evaluating LoRa and LoRaWAN performance in Louvain-la-Neuve. http://hdl.handle.net/2078.1/thesis:19386, 2019. Promotors: Deville, Yves; Sadre, Ramin. [Online; accessed 23-April-2022].

[65] Sensing Labs. Understand how rssi and snr are considered as good radio level. https://sensing-labs.com/f-a-q/a-good-radio-level/. [Online; accessed 23-April-2022].

[66] sghoslya.com. LoRa and CSS. http://www.sghoslya.com/p/lora-is-chirp-spread-spectrum.html, 2022. [Online; accessed 13. May 2022].

[67] STMicroelectronics. LIS2HH12 accelerometer. https://www.st.com/en/mems-and-sensors/lis2hh12.html. [online; accessed 8. Mar. 2022].

[68] Talla, V., Hessar, M., Kellogg, B., Najafi, A., Smith, J., and Gollakota, S. Lora backscatter: Enabling the vision of ubiquitous connectivity. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies 1* (05 2017).

[69] The Things Network. Device classes. https://www.thethingsnetwork.org/docs/lorawan/classes/, 2022. [Online; accessed 4. May 2022].

[70] The Things Network. Duty Cycle. https://www.thethingsnetwork.org/docs/lorawan/duty-cycle/, 2022. [Online; accessed 4. May 2022].

[71] The Things Network. Regional parameters. https://www.thethingsnetwork.org/docs/lorawan/regional-parameters/, 2022. [Online; accessed 13. May 2022].

[72] Verizon. Non-IP Data Delivery - NIDD. https://thingspace.verizon.com/documentation/apis/connectivity-management/working-with-verizon/about-non-ip-data-delivery.html, 2022. [Online; accessed 31 May 2022].

[73] Wikipedia. ESP32. https://fr.wikipedia.org/wiki/ESP32#Base_Xtensa. [online, accessed 8th March 2022].

[74] Wikipedia. Narrowband iot — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Narrowband_IoT&oldid=1049307567, 2021. [Online; accessed 7-May-2022].

[75] Wikipedia. Lte-m — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=LTE-M&oldid=1070091683, 2022. [Online; accessed 7-May-2022].

[76] Wikipedia. Microcontroller — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Microcontroller&oldid=1084544679, 2022. [Online; accessed 3-May-2022].

[77] Wikipedia. Nmea 0183 — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=NMEA_0183&oldid=1084781746, 2022. [Online; accessed 1-May-2022].

[78] Wikipedia. Satellite navigation. https://en.wikipedia.org/w/index.php?title=Satellite_navigation&oldid=1084022869, 2022. [Online; accessed 30-April-2022].

[79] Wikipedia contributors. Hayes command set — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Hayes_command_set&oldid=1052901861, 2021. [Online; accessed 23-April-2022].

[80] Wikipedia contributors. Lora — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=LoRa&oldid=1082958203, 2022. [Online; accessed 4-May-2022].

[81] Wikipedia contributors. LoRaWAN. https://fr.wikipedia.org/wiki/LoRaWAN, 2022. [Online; accessed 4. May 2022].

[82] Wikipédia. Lorawan — wikipédia, l'encyclopédie libre. http://fr.wikipedia.org/wiki/LoRaWAN, 2022. [Online; accessed 4-May-2022].

[83] You, I., Kwon, S., Choudhary, G., Sharma, V., and Seo, J. T. An enhanced lorawan security protocol for privacy preservation in iot with a case study on a smart factory-enabled parking system. *Sensors 18* (06 2018).

Appendix A

Device prototype

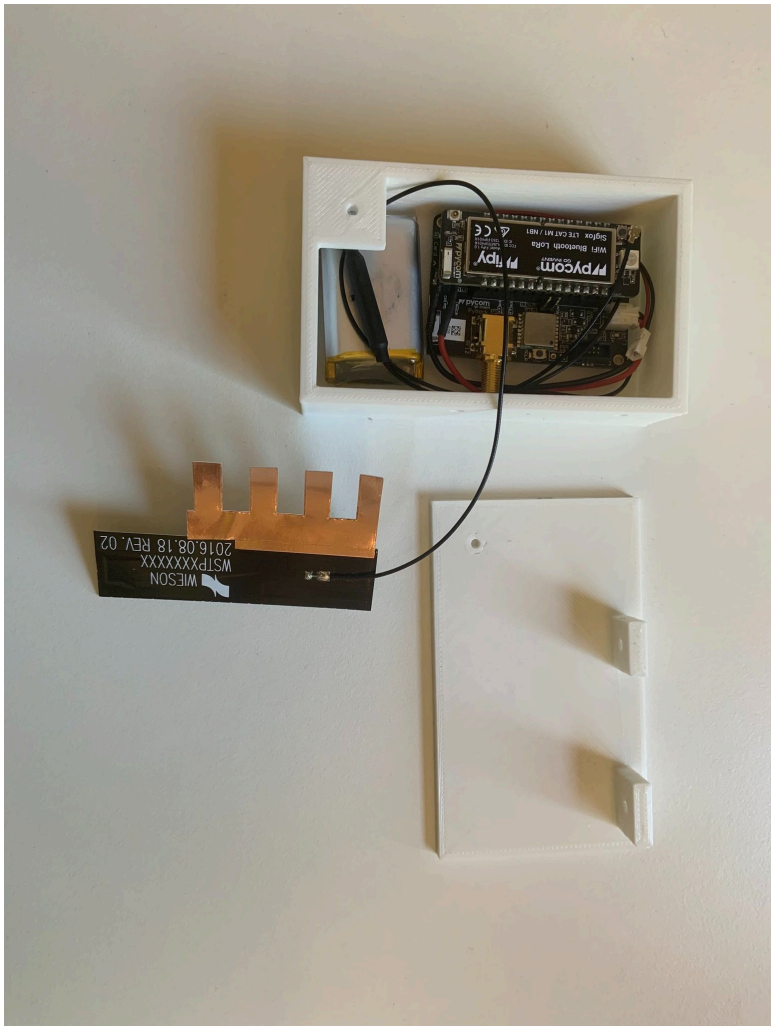


Figure A.1: Prototype

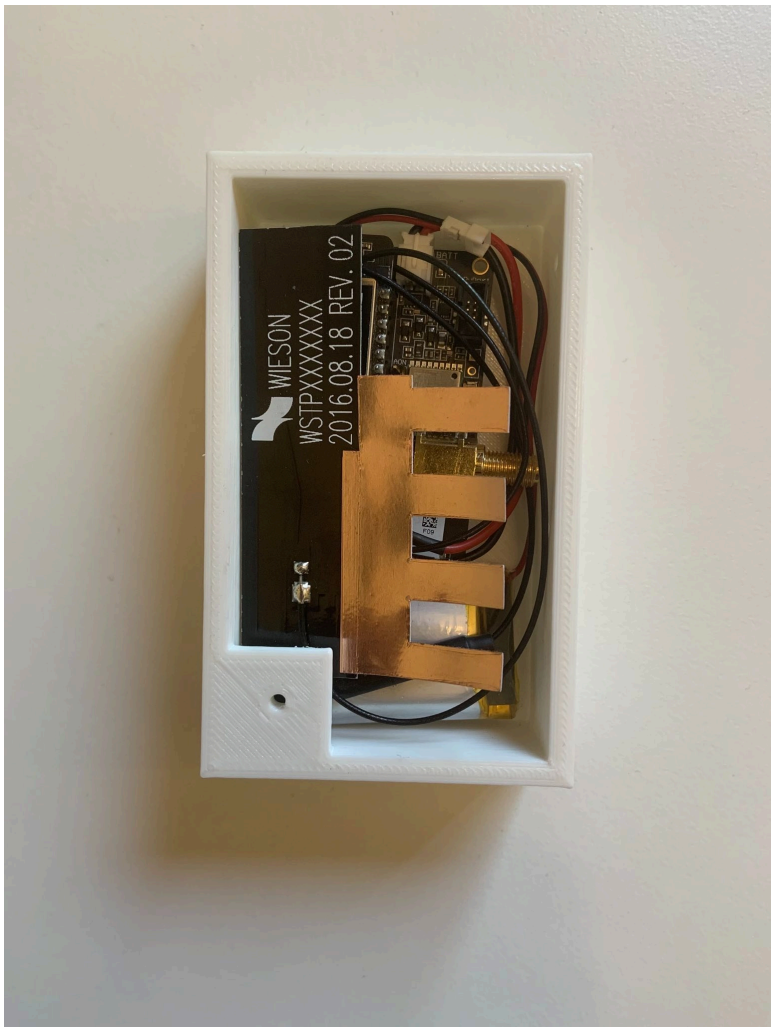


Figure A.2: Prototype

Appendix B

Mobile application

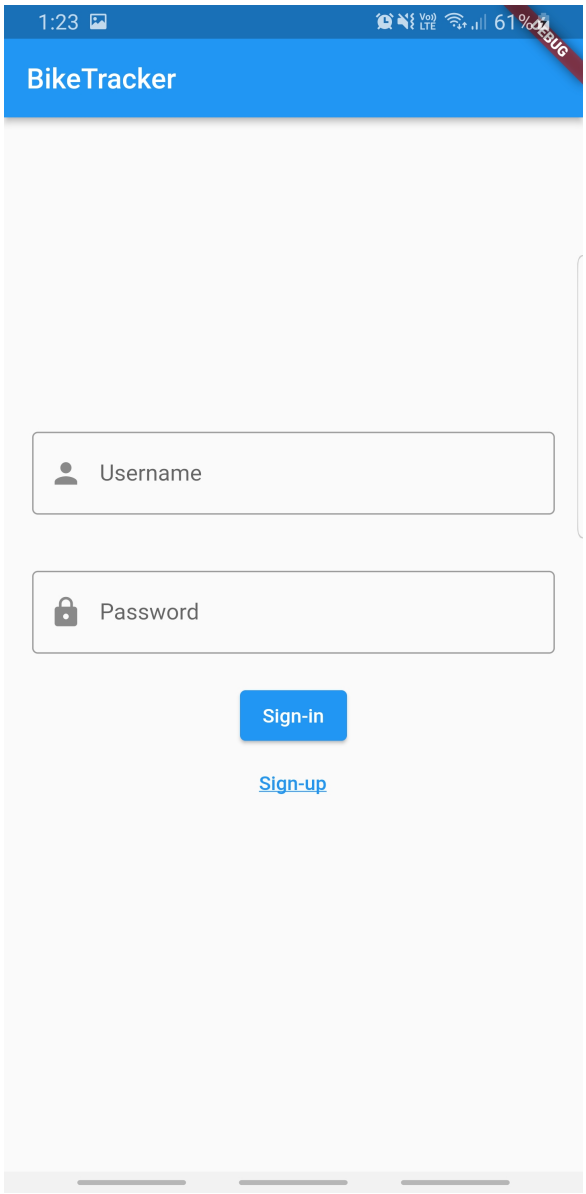


Figure B.1: Sign-in screen

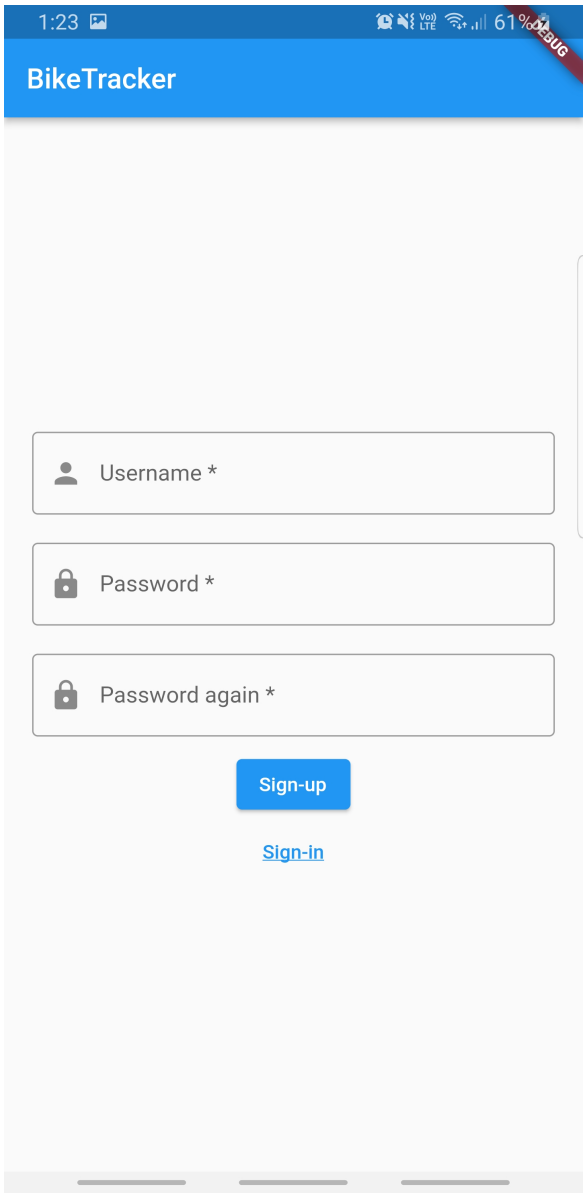


Figure B.2: Sign-up screen

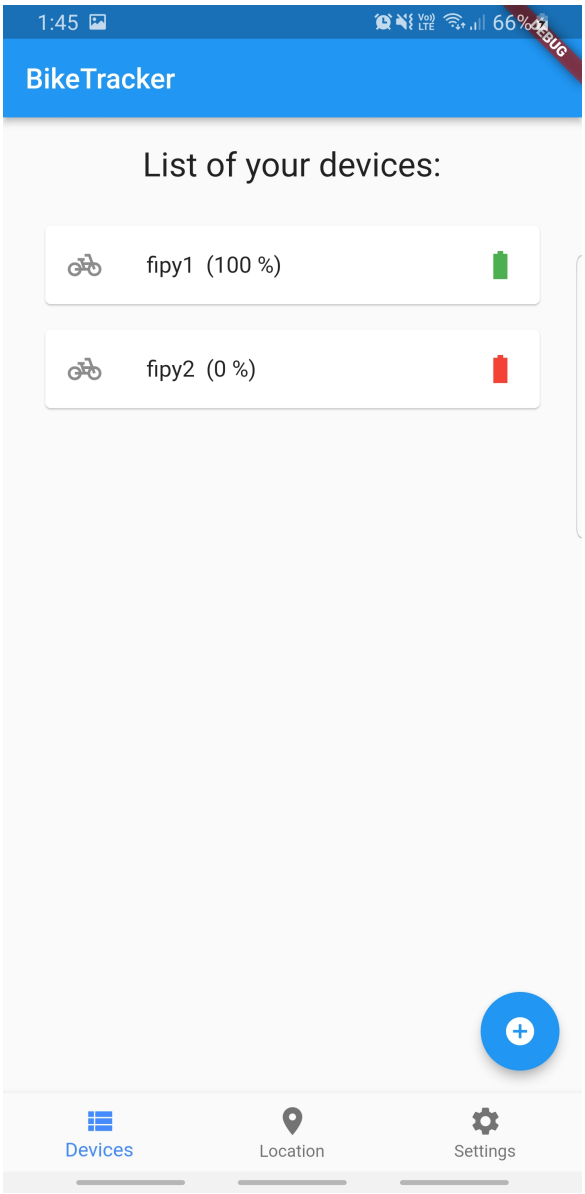


Figure B.3: Device list screen

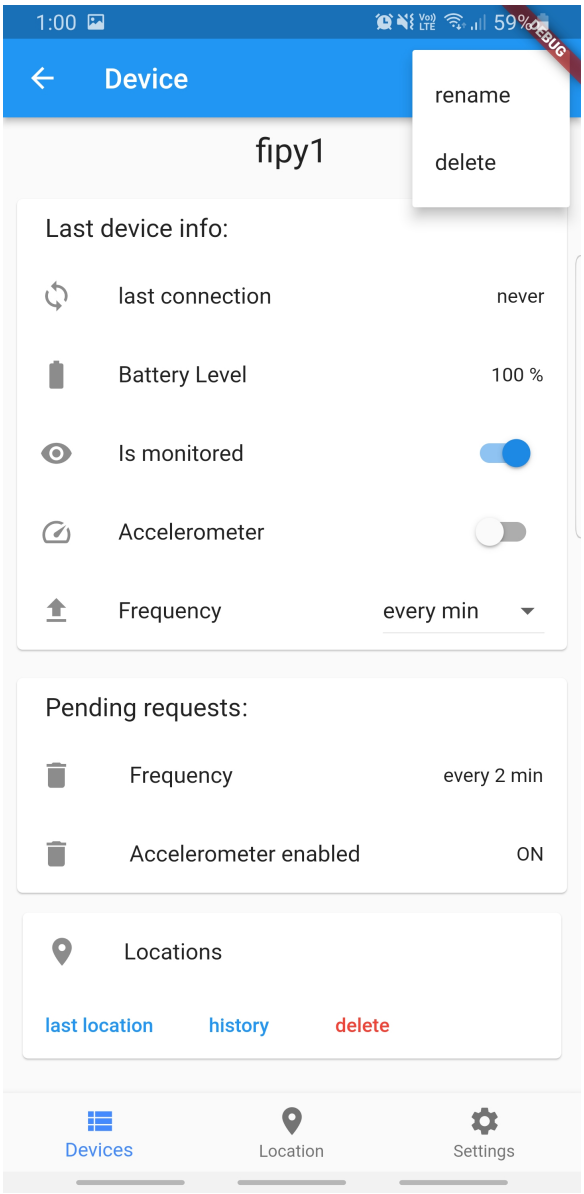


Figure B.4: Show device information screen

Figure B.5: Location of devices

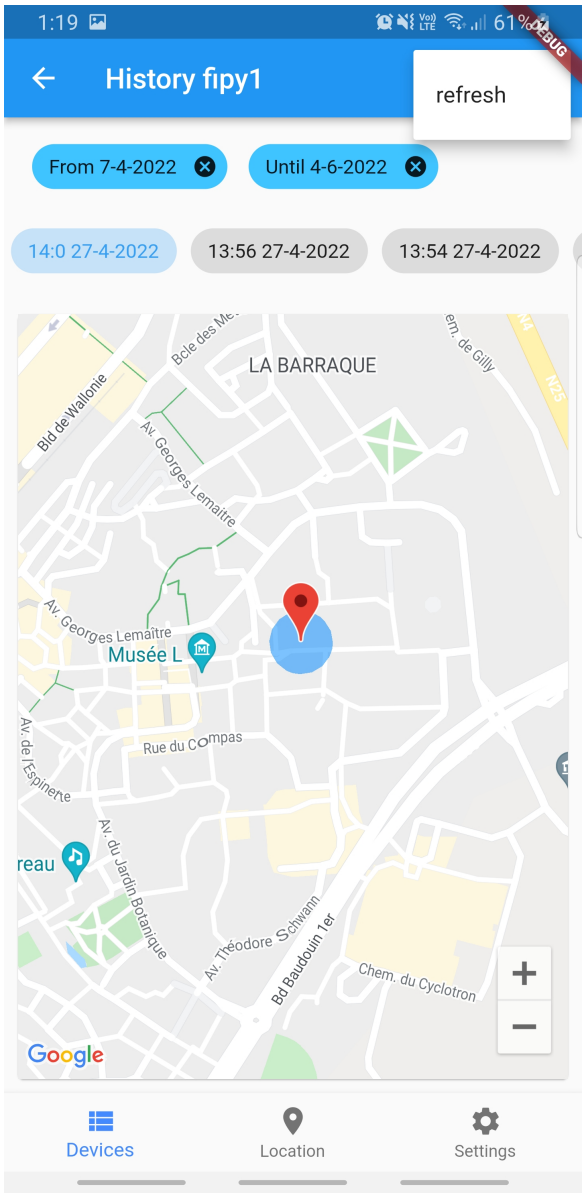


Figure B.6: History of positions for a device

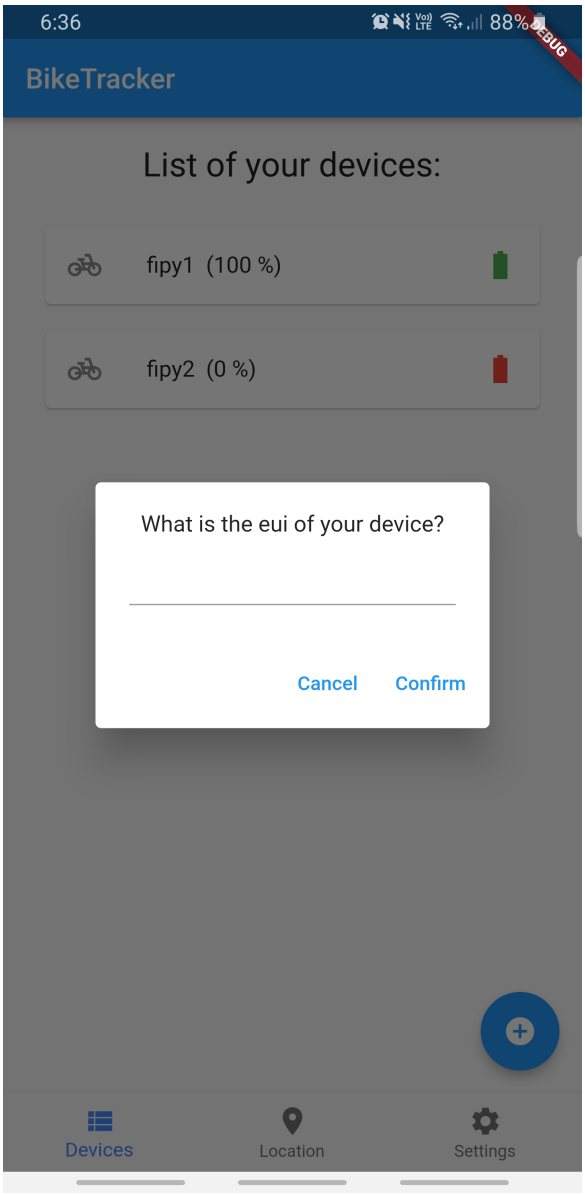


Figure B.7: Add a device

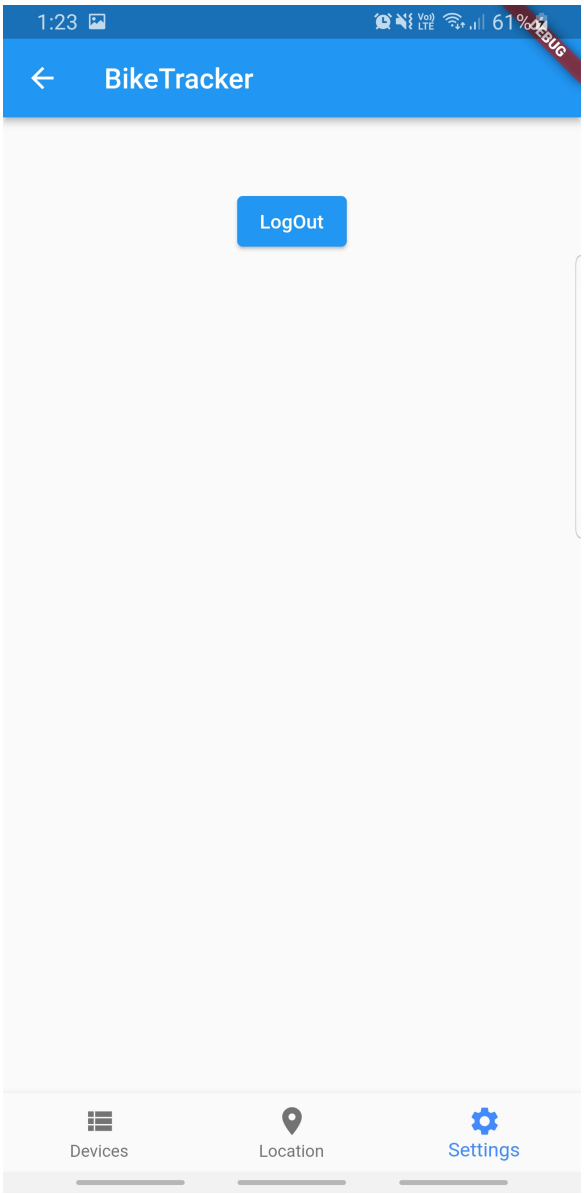


Figure B.8: Settings screen

Appendix C

Experiments

C.1 Network services

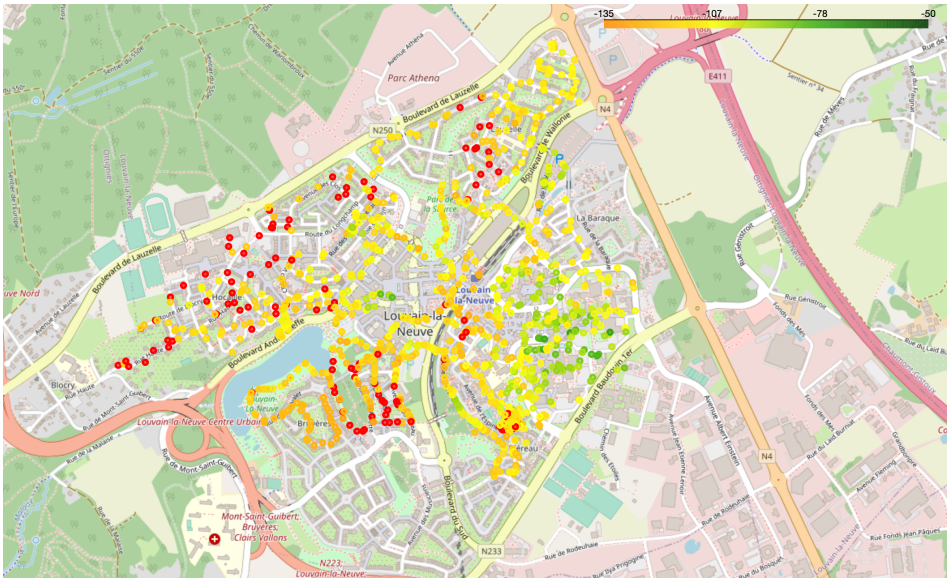


Figure C.1: LoRa coverage - raw data

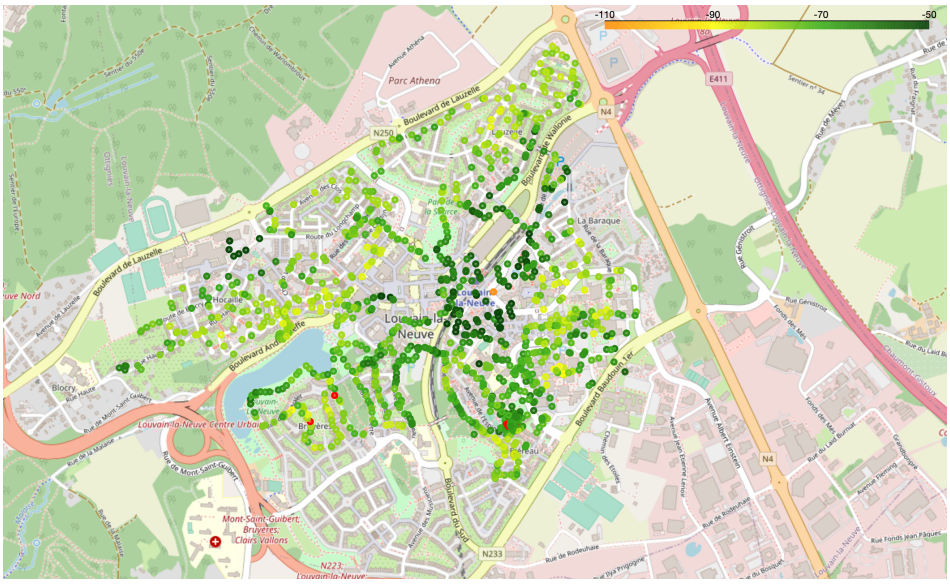


Figure C.2: LTE-M coverage - raw data

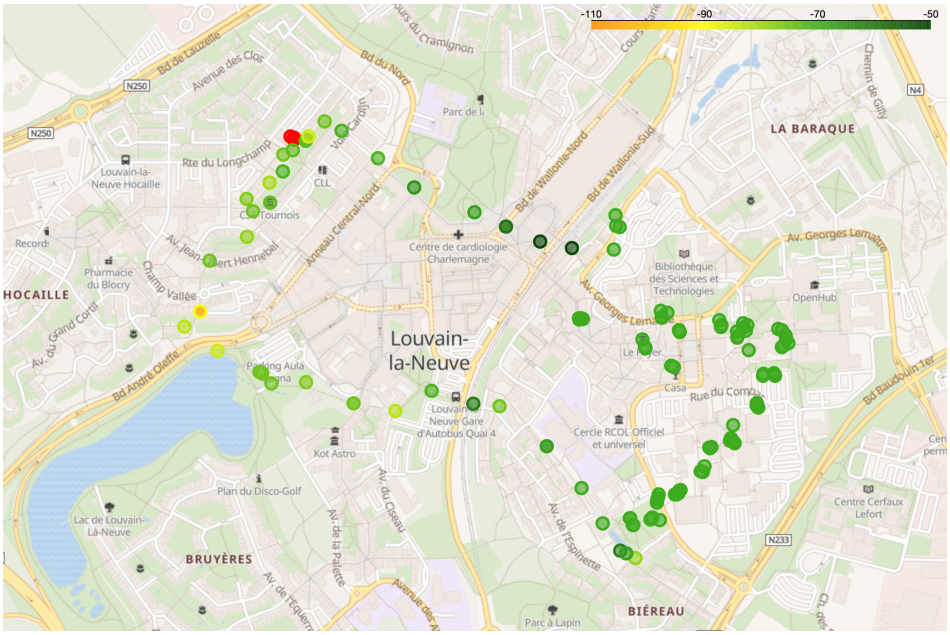


Figure C.3: NB-IoT coverage - raw data

C.2 Positioning services

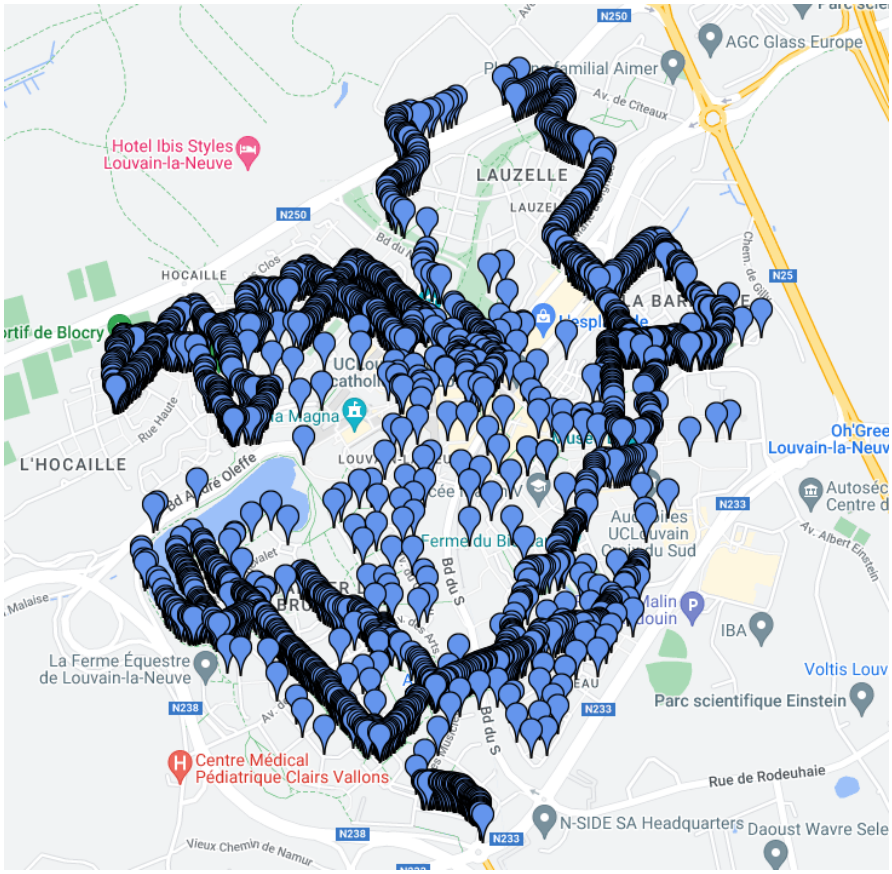


Figure C.4: Measurement points for WiPS and GNSS

C.3 Battery life expectation

Wake-up frequency		1m	5m	15m	30m	1h	2h	5h	12h	1d
Monitoring	`ON`	1.6	5.3	14.5	28.0	54.4	104.5	236.1	464.6	710.9
	`OFF`	1.7	5.6	15.3	29.7	57.6	110.5	248.2	484.1	733.4
Accelerometer	`ON`	1.6	5.3	13.0	21.8	33.8	46.7	60.9	69.1	72.6

Table C.1: Expected battery life in days - best case (LoRa and GNSS) with GNSS in standby mode

Wake-up frequency		1m	5m	15m	30m	1h	2h	5h	12h	1d
Monitoring	`ON`	1.0	3.3	8.8	17.1	33.4	64.8	151.7	318.9	526.6
	`OFF`	1.1	3.7	10.2	19.8	38.7	75.0	174.1	359.4	580.8
Accelerometer	`ON`	1.1	3.6	9.1	16.0	26.2	39.0	55.2	65.9	70.8

Table C.2: Expected battery life in days - worst case (LTE-M and WiPS) with GNSS in standby mode

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
École polytechnique de Louvain
Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | **www.uclouvain.be/epl**