WAZIUP: Co-funded by the Horizon 2020 Framework Programme of the European Union

Open Innovation Platform for IoT-Big data in Sub-Sahara Africa

Grant Agreement Nº 687607

Report D2.1

Deliverable Title: Hardware selection and integration of long-range connectivity for IoT and heterogeneous networking building blocks

Responsible Editor:	UPPA
Contributors:	UPPA, UI
Document Reference:	WAZIUP D2.1 – Hardware selection and integration of long-range connectivity for IoT and heterogeneous networking building blocks
Distribution:	Public
Version:	1.1
Date:	07/02/2017

CONTRIBUTORS TABLE

DOCUMENT SECTION	AUTHOR(S)	REVIEWER(S)
Section 1	C. PHAM, UPPA	M. SHEIKHALISHAHI
Section 2	C. PHAM, UPPA	S. FATNASSI
Section 3	T. TEIXEIRA, UI	C. PHAM
Section 4	C. PHAM, UPPA	O. THIARÉ
Section 5	C. PHAM, UPPA	C. DUPONT
Section 6	C. PHAM, UPPA	T. TEIXEIRA
Section 7	M. DIOP AND C. PHAM, UPPA	F. BAIDOO
Section 8	C. PHAM, UPPA	P. COUSIN

DOCUMENT REVISION HISTORY

Version	Date	Changes
v1.1	FEB 6TH, 2017	PUBLIC RELEASE
v1.0	JAN 31ST , 2017	FIRST DRAFT VERSION FOR INTERNAL APPROVAL
v0.5	JAN 30TH , 2017	INTEGRATION OF REVIEWS FOR SECTION 7
v0.4	JAN 20TH , 2017	INTEGRATION OF REVIEWS FOR SECTION 1 AND 5
v0.3	JAN 19TH , 2017	INTEGRATION OF REVIEWS FOR SECTION 3, 6 AND 8
v0.2	JAN 15TH , 2017	INTEGRATION OF REVIEWS FOR SECTION 2, 4, AND 6
v0.1	JAN 5TH , 2017	FIRST RELEASE FOR REVIEW

EXECUTIVE SUMMARY

This deliverable 2.1 entitled « Hardware selection and integration of long-range connectivity for IoT and heterogeneous networking building blocks » covers the achievements of the first 12 months of T2.1 and T2.2. It will be structured as follows:

- 2. SENSING SYSTEMS AND IOT DEVICES: HARDWARE PLATFORM : reviews the concept of WAZIUP IoT devices and describes the hardware platforms that will be targeted by WAZIUP.
- 3. IOT CATALOGUE FOR HARDWARE SELECTION : describes the IoT catalogue tools for hardware selection and simple code generation.
- 4. IOT CONNECTIVITY : reviews IoT connectivity technologies and challenges. Will focus on the long-range LoRa technology that will be used in WAZIUP.
- 5. WAZIUP low-cost IoT platform : describes the software libraries for long-range communications and sensor management.
- 6. HETEROGENEOUS NETWORKING : describes how heterogenous networking will be managed, especially at the gateway level.
- 7. TESTS WITH LORAWAN AND MINIMUM INTEROPERABILITY : describes our LoRaWAN tests and how miminum interoperability is provided.

TABLE OF CONTENTS

1. Brief review of WAZIUP and WP2 11

1.1. WAZIUP 11

1.2. WP2 11

1.3. Deliverable D2.1 12

2. Sensing systems and IoT devices: hardware platform 13

2.1. Sensor platform 13

2.1.1. Sensing systems and IoT devices 13

2.2. Low-cost hardware platforms for DIY IoT 15

2.3. Hardware selection process 17

3. IoT catalogue for hardware selection 18

3.1. Concepts 18

3.2. Back-end 19

3.2.1. MongoDB 19

3.2.2. MongoDB Connector 20

3.2.3. Authentication 21

3.2.4. API 21

3.2.5. Price crawler 21

3.3. Front-end 22

3.3.1. Data Input/Output 23

3.3.2. Data processing 24

3.3.3. User interface 24

3.4. Manual 27

3.4.1. MVP’s menu 28

3.4.2. Hardware menu 31

3.4.3. Other menu 32

3.4.4. New element 34

4. IoT connectivity 36

4.1. Introduction 36

4.2. Brief review of long-range technologies 38

4.3. LoRa long-range technology 39

4.3.1. Physical layer 39

4.3.2. LoRaWAN 40

4.3.3. LoRa radio modules 41

4.4. Transmission in the unlicensed frequency band 42

5. WAZIUP low-cost IoT platform 45
5.1. IoT platform's objectives 45
5.1.1. Deployment scenarios 45
5.1.2. Why are we not LoRaWAN compliant? 46
5.2. Low-cost gateways 46
5.3. Long-range communication library 47
5.3.1. Improvements to the Libelium SX1272 library 47
5.3.2. LoRa modules that have been tested 48
5.3.3. Radio regulations such as frequency bands and duty-cycle 48
5.3.4. LoRa modes 48
5.3.5. Minimum function set to build a long-range end-device 49
5.3.6. Packet format 49
5.4. Building a low-cost IoT device 50
5.4.1. Software integration for long-range IoT device 51
5.4.2. Software templates for quick and easy appropriation 52
5.4.3. Programming the board 52
5.4.4. Customization of templates 55
5.4.5. IoT device power consumption tests 56
5.4.6. IoT device integration for outdoor usage 57
5.5. Building a low-cost gateway 58
5.5.1. Gateway hardware and architecture 58
5.5.2. Lower level radio bridge and post-processing block interaction 60
5.5.3. What clouds for low-cost IoT? 61
5.5.4. Uploading to clouds 62
5.6. Energy requirements of the whole system 65
5.6.1. Measured energy consumption 65
5.6.2. Investigating usage of solar panels 66
5.6.3. Internet connectivity with 2G/3G shield/dongle 66
5.7. Running without Internet connectivity 67
5.8. Software and tutorial materials available on the github 68
5.9. WAZIUP demo kit 69
5.10. Minimum Viable Products based on the IoT platform 72
5.11. Overview of the WAZIUP platform architecture 74
6. Heterogeneous networking 75
6.1. Heterogeneity with legacy short-range radio technologies 75
6.2. Taking into account data from other radio interfaces 77

6.2.1. Use case with IEEE 802.15.4 ... 78
6.2.2. Generalization to other type of interfaces ... 79
7. Tests with LoRaWAN and minimum interoperability ... 80
7.1. LoRaWAN packet format and requirements ... 80
7.1.1. LoRaWAN packet format ... 80
7.1.2. LoRaWAN encryption ... 80
7.2. Setting up a LoRaWAN gateway : the Multitech mConduit ... 81
7.2.1. The Multitech mConduit ... 81
7.2.2. The mDot ... 87
7.3. Providing a simple LoRaWAN interoperability level ... 91
7.3.1. Low-cost IoT device ... 91
7.3.2. Low-cost gateway ... 92
7.3.3. From LoRaWAN mDot to low-cost gateway ... 95
8. Annexes ... 97
8.1. Our LoRa FAQs ... 97
8.2. Low-cost LoRa IoT platform part list ... 98
8.3. Tutorials materials ... 99
8.4. Keynote talks ... 100
8.5. Scientific publications ... 101

LIST OF FIGURES

Figure 1 – a variety of physical sensors 13

Figure 2 – Arduino UNO microcontroller board with DHT22 physical sensor 13

Figure 3 – IoT Device Integration 14

Figure 4 – Fish Farming 15

Figure 5 – Arduino ecosystem 16

Figure 6 – Original Sparkfun Arduino Pro Mini (left) and a Chinese clone (right) 16

Figure 7 – Teensy boards : LC and 3.1 17

Figure 8 – Back-end overview 19

Figure 9 – Database representation 20

Figure 10 – Price crawler 22

Figure 11 – Front-end overview 23

Figure 12 – Web interface layout 25

Figure 13 – Listing of all the sensors 26

Figure 14 – Parameter information 26

Figure 15 – Help mode 27

Figure 16 – Consulting an use case 27

Figure 17 – Agriculture MVP's 28

Figure 18 – Agriculture use case 28

Figure 19 – List of all actors 29

Figure 20 – Cattle Rustling report 30

Figure 21 – Hardware solutions for Cattle rustling 30

Figure 22 – List of all boards 31

Figure 23 – Arduino Nano 32

Figure 24 – Radio communication protocols 33

Figure 25 – Bluetooth v4.2 33

Figure 26 – New unit 34

Figure 27 – Navigation overview 35
Figure 28 – Remote sensing scenario with 1-hop connectivity to the gateway 36
Figure 29 – Energy consumption for traditional cellular technologies 37
Figure 30 – Simplified view of the link budget of a wireless transmission 37
Figure 31 – IEEE 802 wireless technologies and LPWAN 38
Figure 32 – LoRa throughput and sensitivity for various BW and SF values 39
Figure 33 – spectrogram example of a LoRa transmission 40
Figure 34 – LoraWAN typical topology 41
Figure 35 – some radio module built from SX127x Semtech chips 42
Figure 36 – ISM band per region 42
Figure 37 – 868MHz unlicensed band restrictions 43
Figure 38 – additional notes 43
Figure 39 – Traditional Internet access scenario (top), without Internet access (bottom) 45
Figure 40 – LoRa mode as combination of BW and SF 49
Figure 41 – The long-range library for Arduino-compatible boards and LoRa radio modules 50
Figure 42 – Dragino LoRa product line based on HopeRF RFM95W 51
Figure 43 – Software building blocks for easy integration of long-range IoT devices 51
Figure 44 – Simple temperature sensor with periodic sensing and transmission 55
Figure 45 – power consumption and low-power tests 56
Figure 46 – Low-cost hardware integration 58
Figure 47 – Low-cost single channel LoRa gateway 59
Figure 48 – Gateway architecture 59
Figure 49 – post-processing block template 60
Figure 50 – Post-processing stage with Internet connectivity. 61
Figure 51 – From gateway to IoT clouds 62
Figure 52 – Raspberry power consumption (left), 10000mAh USB charging pack (right) 65
Figure 53 – Mass-market integrated USB solar charging pack 66
Figure 54 – 3G shield (left) and 3G dongle (right) 66
Figure 55 – Gateway without Internet access 67

Figure 56 – Autonomous gateway for no-Internet access deployment scenarios 67
Figure 57 – Fully autonomous cattle rustling application 68
Figure 58 – github repository for IoT devices and gateway 69
Figure 59 – WAZIUP LoRa demo kit 69
Figure 60 – demo gateway (left) and demo IoT device (right) 70
Figure 61 – Using the demo kit 71
Figure 62 – Visualizing real-time data with the embedded web server 71
Figure 63 – distribution of the WAZIUP demo kit 72
Figure 64 – Generic sensor IoT device platform 72
Figure 65 – Minimum Viable Products defined in WAZIUP 73
Figure 66 – Exemple of an active beacon collar system for cattle rustling applications 73
Figure 67 – IoT platform general architecture 74
Figure 68 – IEEE 802.15.4 as PHY/MAC layer in various short-range technologies 75
Figure 69 – Various integrated sensors with embedded IEEE 802.15.5 radios 76
Figure 70 – Short-range (IEEE 802.15.4) for legacy sensors mixed with long-range LoRa 76
Figure 71 – General approach for heterogeneous networking 77
Figure 72 – XBee serial gateway 79
Figure 73 – LoRaWAN packet format 80
Figure 74 – LoRaWAN frame assemby process 81
Figure 75 – The Multitech Conduit and some available mCards 82
Figure 76 – Gateway settings 87
Figure 77 – MultiConnect mDot 87
Figure 78 – Device settings on TTN 88
Figure 79 – Receiving data on TTN 90

LIST OF TABLES

Table 1 – Summary of energy consumption for Pro Mini and Teensy boards 57

1. Brief review of WAZIUP and WP2

1.1. WAZIUP

WAZIUP is a collaborative research project using innovative technological research applications on IoT and related big data management and advanced data analytic techniques. It has the support of multiple African stakeholders and public bodies with the aim of defining a new innovation space to advance the African Rural Economy. The potential of IoT and Big Data, in Sub-Saharan Africa, can be realized only if the cost is reasonable as most of the rural population in the Africa is at the poverty level. This is the main challenge that WAZIUP will address. In addition, WAZIUP is creating developer communities and innovation hubs to train, adapt, validate and disseminate results. WAZIUP's technical partners will develop methodologies, tools, software libraries and "recipes" for building low-cost IoT and data analysis platforms. Tightly involving end-users communities in the loop, namely rural African communities of selected pilots, will ensure quick appropriation and easy customization by third parties. Furthermore, WAZIUP organizes frequent training and Hackathon sessions in the sub-Saharan African region. WAZIUP will tackle the challenges enlisted below:

- Challenge 1: Innovative design of the IoT platform for the Rural Ecosystem. Low-cost, generic building blocks for maximum adaptation to end-applications in the context of the rural economy in developing countries.
- Challenge 2: Network Management. Facilitate IoT communication and network deployment. Lower cost solutions compared to state of the art technology: privilege price and single hop dedicated communication networks, energy autonomous, with low maintenance costs and long lasting operations.
- Challenge 3: Long distance. Dynamic management of long range connectivity (e.g., cope with network & service fluctuations), provide devices identification, abstraction/virtualization of devices, communication and network resources optimization.
- Challenge 4: Big-data. Exploit the potential of big-data applications in the specific rural context.

1.2. WP2

The objectives of WP2 are to co-design, adapt and deploy sensing systems with low-power and low-cost long-range (LR) communication and networking infrastructure. WP2 will set a test-bed in UGB for validation and performance evaluation of use cases in various rural areas. The outcomes from WP2 with corresponding tasks are listed below:

- T2.1 Design and adaptation of sensing systems considering societal and environmental threat - design, adapt and develop sensing systems for IoT nodes that will be deployed and tested in the use cases. Produce open-source plug-&-sense platforms for fast, easy deployment & customization to use cases. Methodology and tools for low-cost design.

- T2.2 Design and integration of heterogeneous IoT networking - integrate the hardware components and develop the software building blocks for generic usage of LR radio technologies. Develop and evaluate IoT activity scheduling algorithms that ensure fairness and efficiency in heterogeneous radio technologies communications. Data management for seamlessly supporting intermittent networking.

- T2.3 Low-latency and low-energy MAC protocols - address the issue of achieving low-latency and low-energy communications with the combination of LR and short/medium range radios. Propose and evaluate IoT node activity scheduling based on the timing requirement of the end application.

- T2.4 Open IoT test-bed setup and benchmark - deploy the IoT test-bed featuring LR IoT device. Provide the validation infrastructure for internal sub-tasks (T2.1, T2.2 and T2.3) and the demonstrator infrastructure for the use cases.

- T2.5 Training materials and tools for developer community - address the training and dissemination part of WP2. Developed sensing systems and software will be presented and explained to end-developers and end-users. Organize dedicated Living Labs, Tech Events and Hackaton to boost innovations around the IoT technologies and the WAZIUP developed solutions.

1.3. Deliverable D2.1

This deliverable 2.1 entitled « Hardware selection and integration of long-range connectivity for IoT and heterogeneous networking building blocks » covers the achievements of the first 12 months of T2.1 and T2.2. It will be structured as follows :

- 2. SENSING SYSTEMS AND IOT DEVICES: HARDWARE PLATFORM : reviews the concept of WAZIUP IoT devices and describes the hardware platforms that will be targeted by WAZIUP.
- 3. IOT CATALOGUE FOR HARDWARE SELECTION : describes the IoT catalogue tools for hardware selection and simple code generation.
- 4. IOT CONNECTIVITY : reviews IoT connectivity technologies and challenges. Will focus on the long-range LoRa technology that will be used in WAZIUP.
- 5. WAZIUP low-cost IoT platform : describes the software libraries for long-range communications and sensor management.
- 6. HETEROGENEOUS NETWORKING : describes how heterogenous networking will be managed, especially at the gateway level.
- 7. TESTS WITH LORAWAN AND MINIMUM INTEROPERABILITY : describes our LoRaWAN tests and how miminum interoperability is provided.

2. Sensing systems and IoT devices: hardware platform

2.1. Sensor platform

WAZIUP mainly focuses on sensing systems for so-called telemetry applications where physical parameters of the environment is measured. In this section we will briefly introduce such sensing systems that will be at the core of WAZIUP Use Cases.

WAZIUP has several Use Cases that focus on completely different issues, therefore imposing the need for WAZIUP to be very flexible in terms of the Hardware required to tackle each Use Case. So, taking this into account, WAZIUP needs to set up a way so that each piece of hardware deployed can be very specific to the particular Use Case, so that WAZIUP can reduce the costs to a minimum. [D1.1, section 6.2.1]

2.1.1. Sensing systems and IoT devices

Sensing the environment is performed by specialized physical sensors. Figure 1 shows a variety of physical sensors, each one designed to measure a particular physical parameter: sound level, temperature, air humidity, soil conductivity/humidity, dissolved oxygen level,...

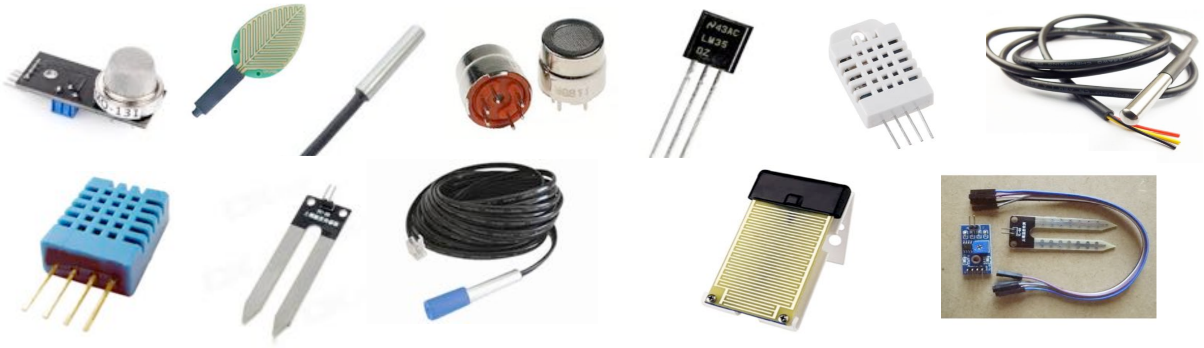


Figure 1 – a variety of physical sensors

Sensing systems are then generally built with a microcontroller board where physical sensors will be connected and driven by an appropriate program running on the board. Figure 2 shows an Arduino UNO microcontroller board with a DHT22 digital physical sensor.

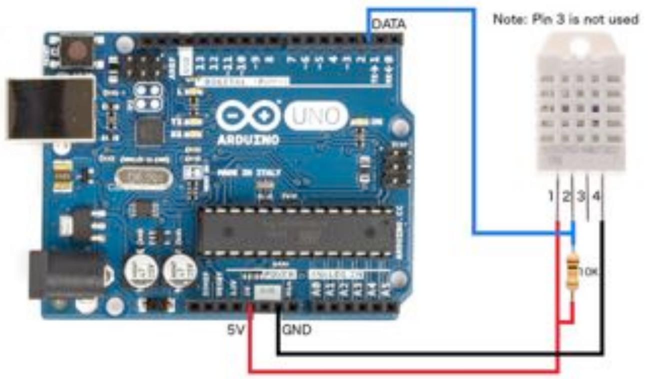


Image from http://cactus.io/hookups/sensors/temperature-humidity/dht22/hookup-arduino-to-dht22-temp-humidity-sensor

Figure 2 – Arduino UNO microcontroller board with DHT22 physical sensor

Physical sensors are either analog or digital. The program running on the microcontroller board will « read » the analog or digital value from the I/O pins and will generally convert this value into meaningful data according to the physical sensor reference datasheet.

The IoT Device in WAZIUP is the device that will be monitoring the environment, collecting and processing the data necessary for the Use Cases and then communicate them to the WAZIUP Platform. So, it will have communication, processing and sensing capabilities, that means an IoT Device can measure one or more parameter from the environment, processing those parameters by generating meaningful data and then communicates the data by (possible) different means (radio, cable link, local storage, etc). [D1.1, section 6.2.1]

These IoT Devices are composed by a group of hardware components, such as a board, sensor, radio and interfaces (when it's needed). Each IoT Device can have multiple sensors (and sensor interfaces) and also multiple radios (and radio interfaces), as far as the board supports it. The domains in WAZIUP rely on data gathered from the environment by the IoT Devices, and to be able to do so, the IoT Devices need to be composed by specific hardware:

- ***Sensor:** used to collect data of physical parameters from the environment (e.g. Temperature, Humidity, etc.).*
- ***Interface:** Required by some hardware to communicate (eg, converting sensor conductivity into an analog signal).*
- ***Board:** Main component of an IoT Device. It's used to receive the data gathered from the sensor(s), process it as defined and forward it using the radio(s) available.*
- ***Radio:** This component is used for communication purposes, so that the IoT Device can receive requests and send data from/to the WAZIUP Platform.*

In terms of readings from the IoT Device, it uses the sensor (or each one, in case of having more than 1) to gather data. This process is made through measurements, which are composed by the following elements:

- ***Parameter:** Physical input that is read from the environment (e.g., Temperature, Humidity, etc.).*
- ***Unit:** How the data obtained by the measurement is represented (e.g., ℉, ℃).*

Figure 3 shows the overall picture regarding the hardware components of an IoT Device and how it relates to the WAZIUP domains and to the measurements gathered form the environment.

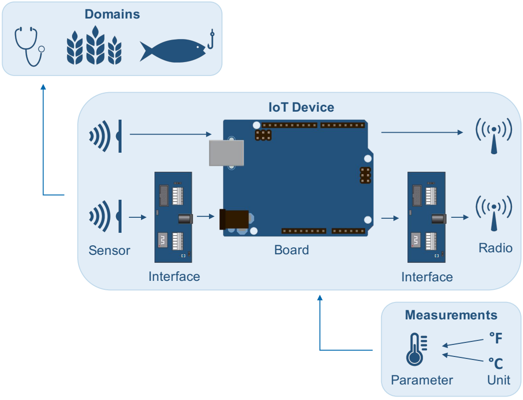


Figure 3 – IoT Device Integration

The following example describes a use case from the "Fish Farming" domain, presenting an IoT device that can be used on this situation. The main purpose of the IoT Device here specified is to collect data from the water. Among several parameters, the water temperature (℃) is measured by using an IoT Device composed by the following hardware:

- ***PT-1000 Temperature Sensor:*** *Its conductivity changes with the water temperature.*
- ***PT-1000 Temperature Module:*** *It translates the sensor conductivity into a temperature value*
- ***Arduino UNO Board:*** *Main board of the IoT Device*
- ***Lora RF Module:*** *Enables the communication from/to the IoT Device*

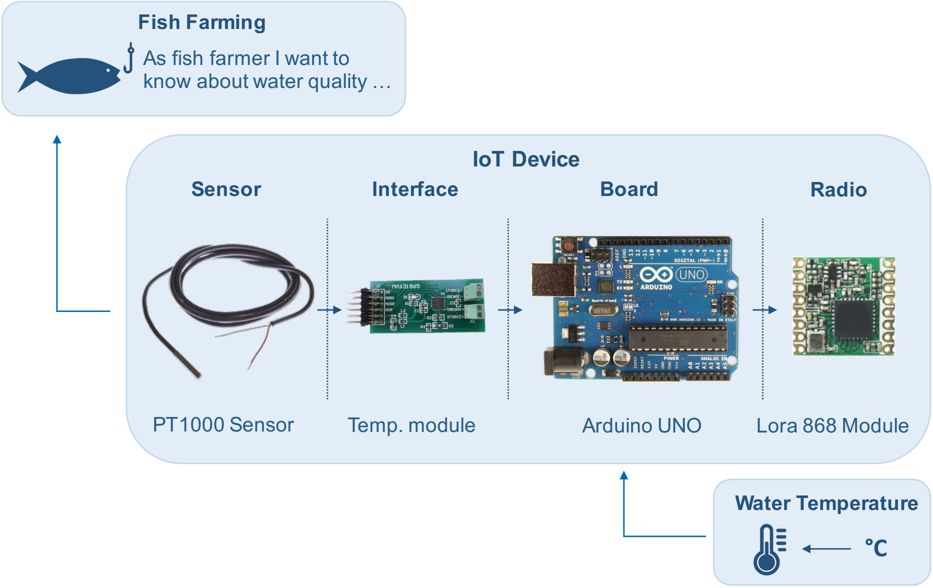


Figure 4 – Fish Farming

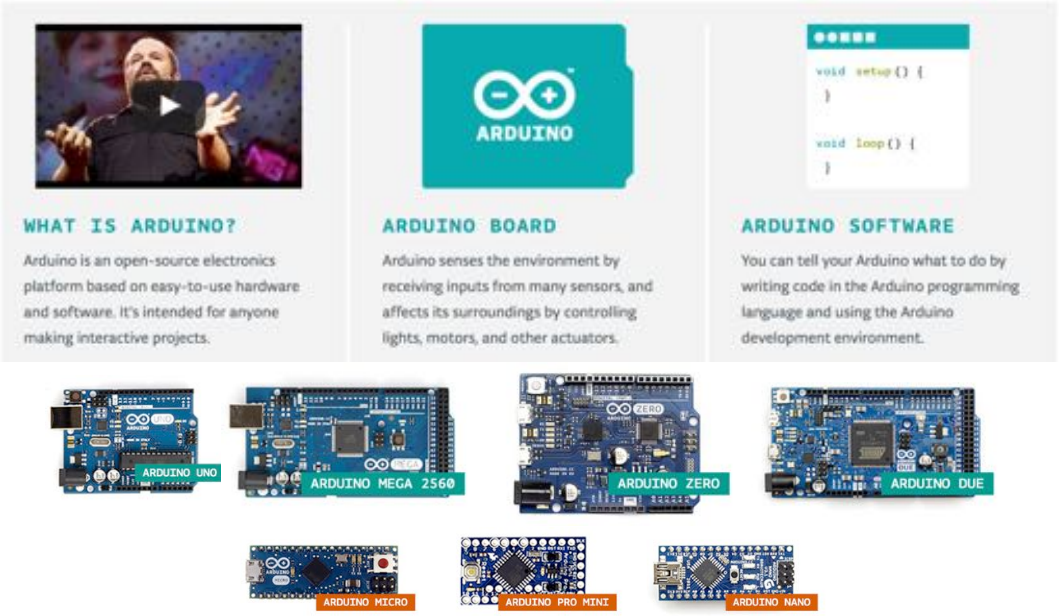
[D1.1, section 6.2.2]

2.2. Low-cost hardware platforms for DIY IoT

The maturation of the IoT market is happening in many developed countries. While the cost of IoT devices can appear reasonable within developed countries standards, they are definitely still too expensive for very low-income sub-saharan ones. The cost argument, along with the statement that too integrated components are difficult to repair and/or replace definitely push for a Do-It-Yourself (DIY) and "off-the-shelves" design orientation.

The world-wide availability of low-cost, open-source hardware platforms such as Arduino-like boards is clearly an opportunity for building low-cost IoT devices from consumer market components. In addition to the cost argument such mass-market board greatly benefits from the support of a world-wide and active community of developers. **This last argument is very important and it must be kept in mind that software can sometime be more important than hardware because developing drivers and specific libraries is a very hard and time-consuming task**.

Figure 5 shows a selected set of the genuine Arduino ecosystem with various Arduino board models and the programming IDE that accepts pluggins from a larger variety of Arduino-compatible boards : Teensy, Sparkfun, Seeeduino, SODAQ, Adafruit Feather, RFduino, Ideetron Nexus, Galileo,…



WHAT IS ARDUINO?

Arduino is an open-source electronics platform based on easy-to-use hardware and software. It's intended for anyone making interactive projects.

ARDUINO BOARD

Arduino senses the environment by receiving inputs from many sensors, and affects its surroundings by controlling lights, motors, and other actuators.

ARDUINO SOFTWARE

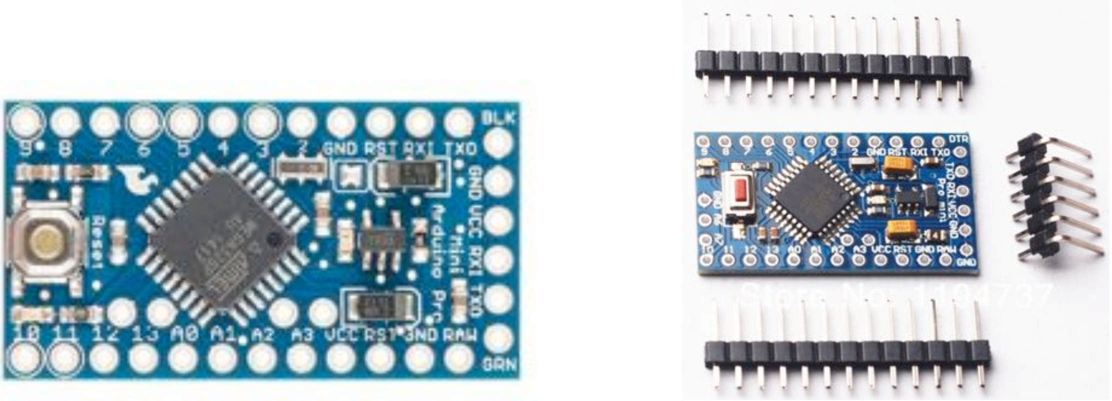
You can tell your Arduino what to do by writing code in the Arduino programming language and using the Arduino development environment.

```
void setup() {
}

void loop() {
}
```

Figure 5 – Arduino ecosystem

A non-exhaustive list of Arduino-compatible boards is available at https://en.wikipedia.org/wiki/List_of_Arduino_boards_and_compatible_systems. These boards can be compatible either at the hardware or at the software level.

The core Arduino board ecosystem consists in large form boards (UN0, MEGA, ZERO, DUE) and small form boards (MICRO/MINI, PRO MINI, NANO). Large boards are suitable for testing and first step prototyping while small boards can be used for higher integrated prototypes, close to final product. For instance, a small board (18mmx33mm) like the Arduino Pro Mini based on an ATmega328 microcontroller offers an excellent price/performance/energy tradeoff and can provide a low-cost platform for generic sensing IoT with LoRa long-range transmission capability for a total of less than 15 euro. Figure 6 shows the original Sparkfun Pro Mini board on the left. Thanks to its open-source design and high versatility, there are many Chinese clones available at a very low-cost (less than 1.5 euro), see Figure 6 (right).



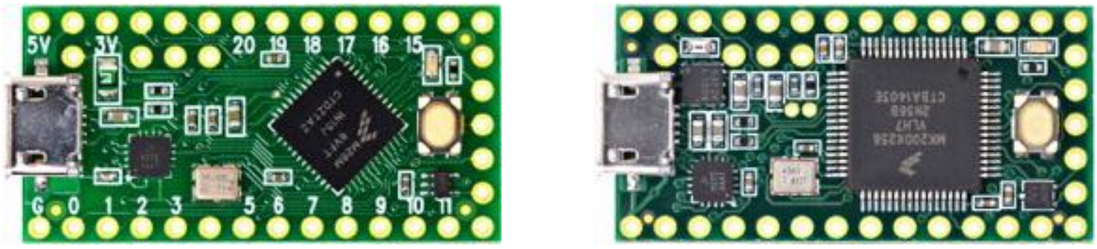
9 8 7 6 5 4 3 2 GND RST RXI TXO BLK
GND VCC RXI TXO
Reset
Arduino Pro Mini
10 11 12 13 A0 A1 A2 A3 VCC RST GND RAW GRN

Figure 6 – Original Sparkfun Arduino Pro Mini (left) and a Chinese clone (right)

The Arduino Pro Mini exists in 3.3v and 8MHz version to further reduce the energy consumption. **For all these reasons, we selected the Arduino Pro Mini as the generic platform for developing low-cost IoT device for WAZIUP Use Cases.**

However, the core ATmega328 microcontroller has only 2KB of memory which can become a limiting factor for some memory-demanding applications. In searching for a board with more memory, we also have to keep in mind the Arduino-compatible issue, the energy consumption issue and the availability of libraries. **We selected the Teensy boards that are compatible at the software level with most of the Arduino libraries and can be programmed within the Arduino IDE. They feature more recent microcontrollers with more memory, more power efficiency and more advanced power management possibilities.** Figure 7 shows the Teensy LC (left) and the Teensy 3.1 (right) boards that have the same size than the Pro Mini.

The Teensy LC costs about 10 euro while the Teensy 3.1 costs about 16 euro. Apart from the cost factor, the amount of memory required by the application will be the determining factor for choosing between the LC or the 3.1 board as both have processing capabilities that are well above what if usually needed for sensing applications. The Teensy LC has much lower power consumption than the 3.1 therefore if 8KB of memory is sufficient, the Teensy LC is the board of choice.



Teensy LC
48 MHz Cortex-M0+
8K of RAM memory

Teensy 3.1
72 MHz Cortex-M4
64KB of RAM memory

Figure 7 – Teensy boards : LC and 3.1

2.3. Hardware selection process

In WP2, the integration of sensing hardware for a specific application domain will be realized by an IoT Catalogue tool that will list a large variety of physical sensors with their approximate price and requirements to ease the decision-making and design process. The final choice will be left to the integrator person but once the choice has been realized, the IoT Catalogue can also help in generating code for the management the physical sensors. These generated code can then be integrated into the IoT generic templates.

The IoT Catalogue is described in the next Section.

3. IoT CATALOGUE FOR HARDWARE SELECTION

In the last years, we have witnessed an unprecedented growth in the number of IoT devices used among different domains including healthcare, agriculture, smart cities, etc. Each domain has its own set of applications where it is necessary to collect and process information about an environment with the purpose of producing an output which is used to improve the knowledge for that specific domain.

Finding a solution could be challenging, considering the complexity and variety of devices available on the market. The purpose of the IoT Catalogue is to help users on the process of choosing the most suitable devices. In this chapter, we explain which are the components represented in the tool, how they are related each other, how the tool stores information of each element and how stored data is processed and combined with the purpose of producing meaningful information. Instructions are also provided regarding the usage of the tool.

3.1. Concepts

For each case, when information is being collected from the environment, a specific solution must be assembled which consists in a group of IoT devices that are working as a single solution capable of sensing, processing, storing and transmitting data. The IoT Catalogue, based on a problem defined by the user, can present several solutions with different costs, levels of complexity, etc.

All the components used in a solution are represented by the IoT Catalogue with detailed information, including manufacturer, product page and its vendors allowing the user to choose where to buy the component based on the store location, price, etc. Currently the components are divided among the following types: sensor, radio, interface and board. Each type is also associated with specific information.

- **Sensor:** is used to perform measurements by collecting information from the environment. In the IoT Catalogue each sensor is represented by the parameters it measures and which units are used (eg. Atmospheric temperature using °C).
- **Interface:** is a component, usually a shield, required by some sensors to communicate with the board.
- **Board:** receives data coming from the sensors/interfaces, processes them. Data can be stored and/or transmitted.
- **Radio:** is used to transmit data processed by the board. This component could be part of a solution when considering the deployment in remote areas.

In the IoT Catalogue several applications are considered taking into account different environments, so all the physical inputs measured have to be taken into consideration and they are represented using parameters.

- **Parameter:** represents the physical input collected from the environment and which units are used.

- **Unit:** gives information on how to interpret a parameter and it could change taking into account the order of magnitude of the measured value, the type of measurement, etc. A unit could be related with another (eg: V <-> 1000mV).

One of the purposes of the IoT Catalogue is to propose solutions based on issues described by use cases and domains:

- **Use case:** gives information about a specific problem along with the application domain, the target and the parameters that are useful to measure, taking into account the context of the problem.
- **Domain:** the issues described by the various use cases of the IoT Catalogue are grouped into several applications domains: Agriculture, Environment, etc.

3.2. Back-end

The back-end is responsible for storing all the data used by the IoT Catalogue and also to provide an interface to the end user. This back-end is based on Node.js a JavaScript runtime environment, which provides all the required services such as storage, user authentication and rest API.

The Figure 8 gives an overview of all the components that are part of the back-end which are responsible for data storage, data access, user authentication and JSON rest API. All of these components are working together and are combined into a Node.JS application.

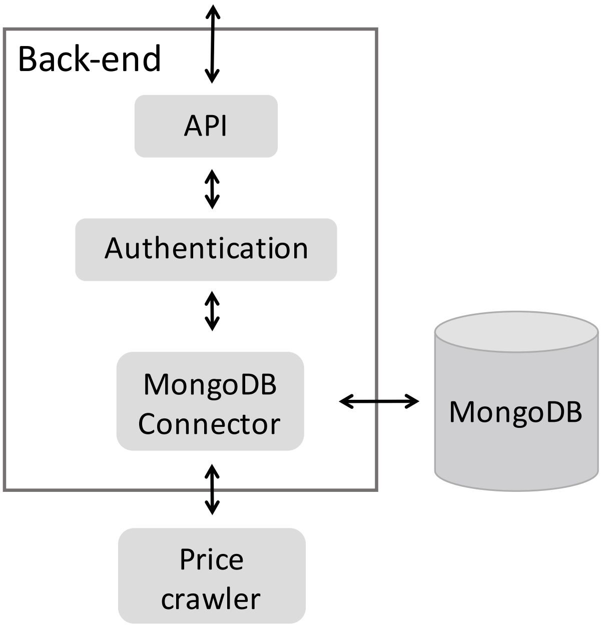


Figure 8 – Back-end overview

3.2.1. MongoDB

Data used in the IoT Catalogue are represented through JSON objects that are stored on a document-oriented database. Each document can contain one or more fields, including arrays, binary data and sub-documents. MongoDB database is used as it is a widely used open-source database. Figure 9 shows the data model used to represent and modulate the existing objects.

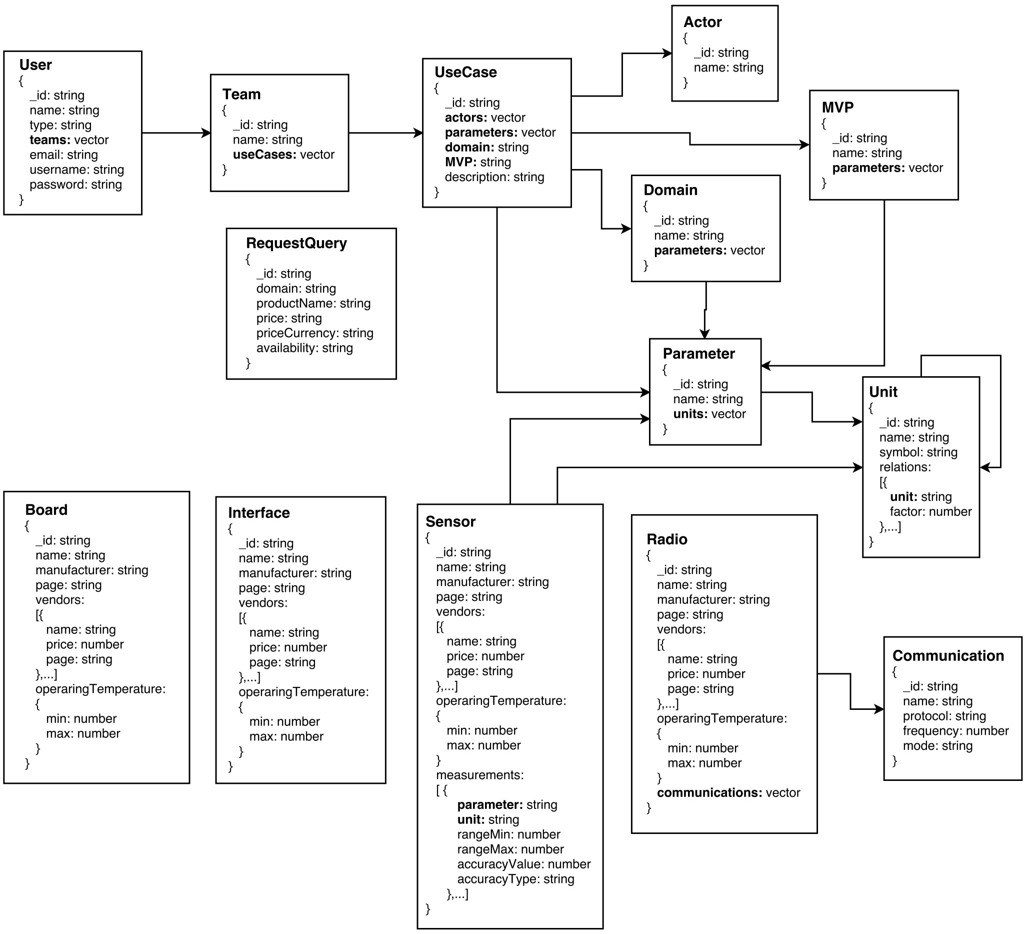


Figure 9 – Database representation

3.2.2. MongoDB Connector

This component establishes the communication between the mongoDB server and the node.js application, both running in the back-end. It contains the settings required to establish the connection with the mongoDB server and the methods used to insert, delete and update data from the database using the following methods.

- **mongoConnect:** establishes the connection with the mongoDB server. In case of failure the function is going to retry the connection every second.
- **insertMongoDoc:** adds new data to the database.
- **updateMongoDoc:** updates an existing document based on a given id.
- **deleteMongoDoc:** deletes a document from the database with the given id.

3.2.3. Authentication

A user can be authenticated on the IoT Catalogue. Without authentication, the tool is available in public mode where the user has only limited access to features. With authentication the user has access to further information and, depending on the privileges, the user will be able to edit and insert information on the IoT Catalogue. Actually the authentication is supported by username/password or using a google account and there are currently three types of users:

- **Read:** has read only access to all the components including the use cases.
- **Write:** has read and write access to all components including the use cases.
- **Admin:** besides the read and write access, has also access to a user administration page.

The authentication is based on Passport.JS which is an authentication middleware that uses the passport-local strategy for user/password login and the passport-google-oauth strategy for login with a google account.

3.2.4. API

It specifies the communication and interaction of the IoT Catalogue with the user and external applications. This is a REST API which works with JSON messages through the following services:

- **POST - /login:** to sign in using a username/password.
- **GET - /auth/google:** to sign in using a google account.
- **GET - /data:** to retrieve information stored in the database.
- **POST - /data:** to add, delete or modify information from the database. Only available for authenticated users.

The API is hosted using the library Express.JS which is a web application framework that provides a set of features for web applications.

3.2.5. Price crawler

The IoT Catalogue contains information about the vendors for each component (sensor, interface, radio and board) and stores data with its name, price and a link to the vendor's page. Taking into account that the information of a specific vendor could be outdated (new price, retired product, etc) a crawler is available to ensure the product contains the last updated price.

- The price crawler receives from the IoT Catalogue information about the commercial details of a product.
- The price crawler requests the information about the prices from the product pages for each vendor.
- The price crawler receives the new prices.

- If the new prices are different from the previous prices, the updated prices are sent back to the IoT Catalogue to update the product info.
- This process is then repeated for the next product.

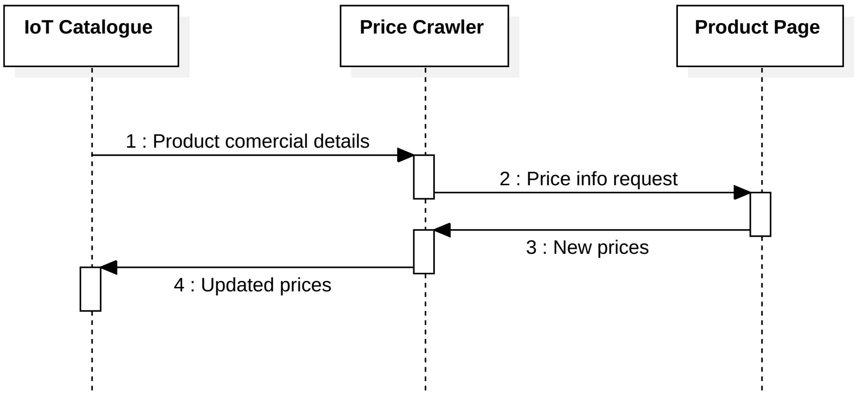


Figure 10 – Price crawler

The price crawler is scheduled to run every day and is supported by the information available in **RequestQuery** which tells to the price crawler how to retrieve data from each vendor site including the price value, currency and availability.

This crawler is based on the Cheerio library, a jQuery implementation on the server side allowing to extract information from HTML documents, and on the node-schedule library which is a job scheduler for Node.JS used to run the price crawler automatically every day at a specific time.

3.3. Front-end

The front-end allows the interaction between the IoT Catalogue and the end-user by providing a web interface. When access to the tool is available in guest mode it is possible to consult information about different elements. When logged in and authenticated, it is possible to add or edit data about the stored components. The use cases which are available under the MVP's are also available to edit and consult.

This front-end is divided into 3 components: data input/output which is used to receive and send data to the back-end; data-processing which is used to process and combine the data available and user interface which receives the inputs given by the user and displays the information requested by the user. An overview is given in the Figure 11.

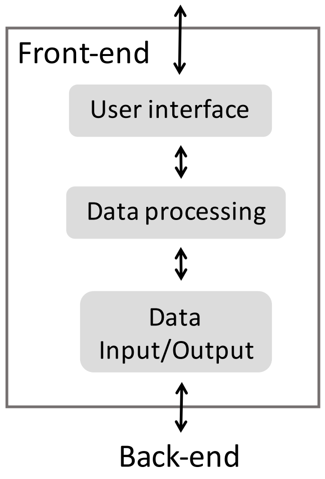


Figure 11 – Front-end overview

Each component is implementd with different libraries:

- **jQuery**: a JavaScript library used on the manipulation of HTML documents, event handling, AJAX and animation.
- **jQuery UI**: a set of user interface interactions, effects, widgets and themes built on the top of the jQuery library.
- **Bootstrap**: an HTML, CSS and JavaScript framework used on the design of responsive web interfaces.
- **DataTables**: a jQuery plugin used to provide advanced interaction controls and enhanced data visualization to the HTML tables of the catalogue.
- **Mustache.JS**: a JavaScript based template system which is used for form rendering.
- **Chosen**: a search engine for select boxes, making its usage simpler.
- **Showdown**: a JavaScript based markdown to HTML converter.
- **Moment.JS**: a parser to manipulate dates in JavaScript (eg: 12:00-14:00 -> "Two hours ago").

3.3.1. Data Input/Output

This component is used in the communication between the front-end and the back-end. It is used not only for the authentication process (send username/password) but also to receive and send data about new elements and use cases. The communication is done through the following functions which are calling the REST API available on the back-end:

- **getData:** Used to retrieve data about the elements.
  - **AJAX URL:** ./data/<ElementType>/<ElementId> (For a single element)
  - **AJAX Method:** GET
- **insertData:** Used to add or edit information about an element.
  - **AJAX URL:** ./data/<ElementType>/<ElementId> (When editing)
  - **AJAX Method:** POST
  - **Data:** {"action":"insert","data":<ElementData>}

- **deleteElement:** Used to delete an element.
    - **AJAX URL:** ./data/<ElementType>/<ElementId>
    - **AJAX Method:** POST
    - **Data:** {“action”:”delete”}

In some cases, it is necessary to process the data before sending it through the API or to process the data received through the API before showing it to the user. This is ensured by the following component.

3.3.2. Data processing

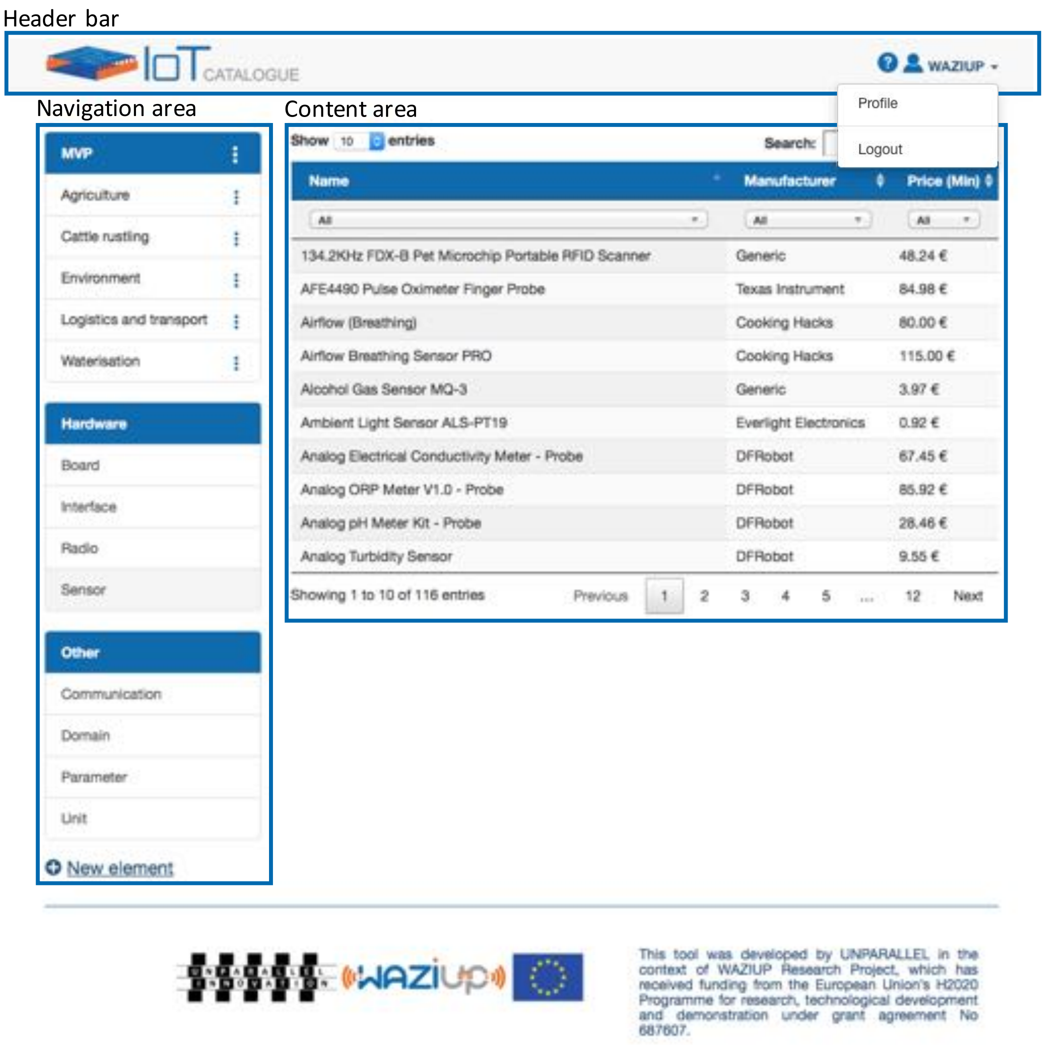
This component is responsible in processing the data available on the IoT Catalogue, in handling the data entered by the user, in preparing them to be sent and stored on the back-end and in arranging and combining the data coming from the back-end according to the request made by the user through the user interface. The following functions are used on this component.

- **newElement:** reads the data entered by the user, validate and combines them into an object, ready to be sent to the back-end.
- **getHardwareInfo:** gives information about the hardware that is able to measure from a list of given parameters.
- **sortObjects:** sorts objects from a vector taking into account a given attribute (eg: sort the vector vendors by the attribute ‘price’).

After processing the data, information will be displayed to the user and this is done through a user interface.

3.3.3. User interface

This is a web based interface which provides the mechanisms necessary to display to the user information about several components and also allows to enter new information. On IoT Catalogue the information is accessed by the user through tables and forms and this interface is provided in the following layout.



Name	Manufacturer	Price (Min)
All	All	All
134.2KHz FDX-B Pet Microchip Portable RFID Scanner	Generic	48.24 €
AFE4490 Pulse Oximeter Finger Probe	Texas Instrument	84.98 €
Airflow (Breathing)	Cooking Hacks	80.00 €
Airflow Breathing Sensor PRO	Cooking Hacks	115.00 €
Alcohol Gas Sensor MQ-3	Generic	3.97 €
Ambient Light Sensor ALS-PT19	Everlight Electronics	0.92 €
Analog Electrical Conductivity Meter - Probe	DFRobot	67.45 €
Analog ORP Meter V1.0 - Probe	DFRobot	85.92 €
Analog pH Meter Kit - Probe	DFRobot	28.46 €
Analog Turbidity Sensor	DFRobot	9.55 €

Showing 1 to 10 of 116 entries

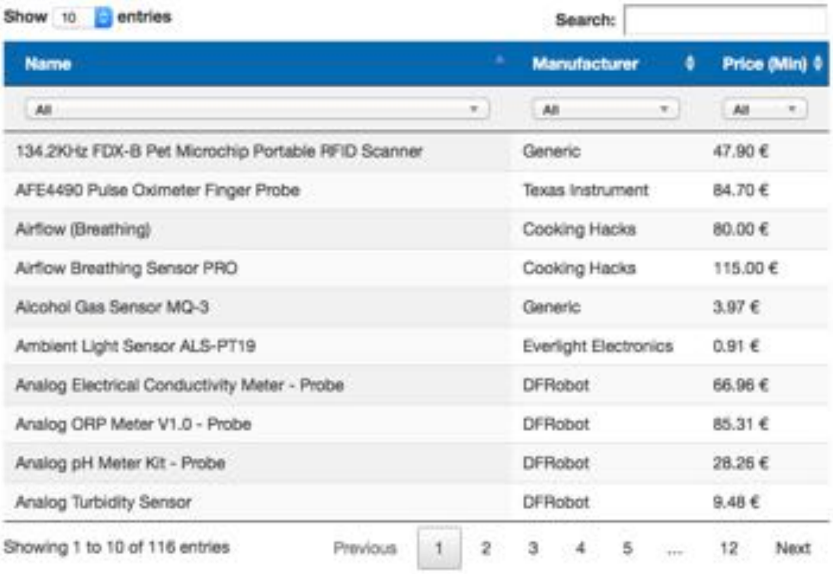
This tool was developed by UNPARALLEL in the context of WAZIUP Research Project, which has received funding from the European Union's H2020 Programme for research, technological development and demonstration under grant agreement No 687607.

Figure 12 – Web interface layout

The user interface is divided into 3 parts:

- A header bar located on the top of the page which contains a menu and information for the logged user;
- the navigation area which contains an MVP menu allowing to see the use cases defined in each one. A menu will contain the different categories for the hardware and an 'other' menu will contain information about miscellaneous components;
- a content area which is where the information is shown and entered by the user through tables and forms.

A table is used to display information about a group of objects and could be used for example to show all the components of specific type (list all the sensors). Clicking on a single line of the table opens a form containing extra information about the object and it is also possible to perform a search using the search text box filter list by attribute name using the dropdowns available. Figure 13 shows an example where information about several sensors is being displayed along with the name, the manufacturer and its price.



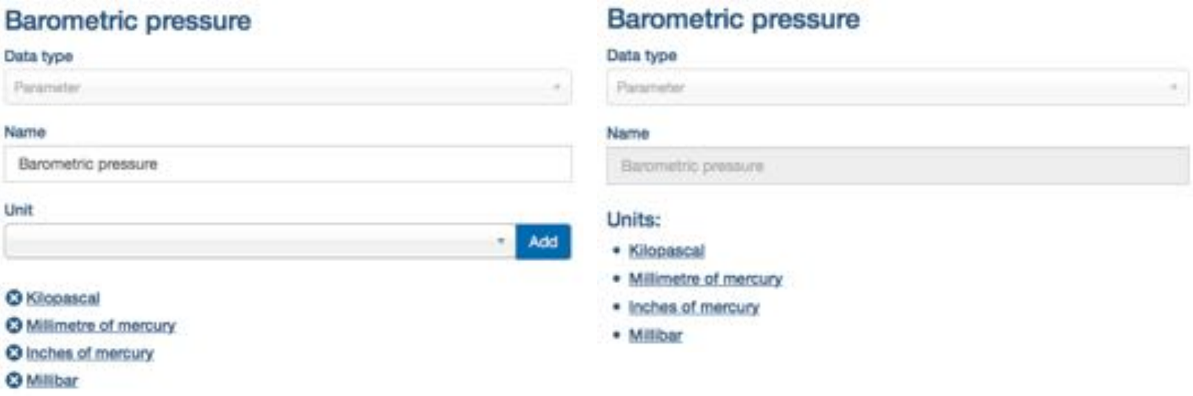
Show 10 entries Search:

Name	Manufacturer	Price (Min)
All	All	All
134.2KHz FDX-B Pet Microchip Portable RFID Scanner	Generic	47.90 €
AFE4490 Pulse Oximeter Finger Probe	Texas Instrument	84.70 €
Airflow (Breathing)	Cooking Hacks	80.00 €
Airflow Breathing Sensor PRO	Cooking Hacks	115.00 €
Alcohol Gas Sensor MQ-3	Generic	3.97 €
Ambient Light Sensor ALS-PT19	Everlight Electronics	0.91 €
Analog Electrical Conductivity Meter - Probe	DFRobot	66.96 €
Analog ORP Meter V1.0 - Probe	DFRobot	85.31 €
Analog pH Meter Kit - Probe	DFRobot	28.26 €
Analog Turbidity Sensor	DFRobot	9.48 €

Showing 1 to 10 of 116 entries

Figure 13 – Listing of all the sensors

A form can be used both in edit and view mode when information is being added/entered or when detailed information is being consulted from a component. The following figure shows an example where detailed information is shown about a parameter, both in edit and view mode.



Barometric pressure

Data type: Parameter

Name: Barometric pressure

Unit

- Kilopascal
- Millimetre of mercury
- Inches of mercury
- Millibar

Barometric pressure

Data type: Parameter

Name: Barometric pressure

Units:

- Kilopascal
- Millimetre of mercury
- Inches of mercury
- Millibar

Figure 14 – Parameter information

The following functions are used to handle the forms and the tables available in the IoT Catalogue.

- **newFormLoad:** loads in the content area the form layout taking into account the data that is being entered by the user.
- **fillElementForm:** fills all the fields of the loaded form taking into account the data available for a specific element.
- **fillElementsTable:** lists in a table all the components of a specific type, including information of specific of each type (eg: Name, Manufacturer and Price).
- **fillVendorsTable:** list in a table all the vendors for a specific component, including information about the vendor name, the price and the date of the last price update.

- **fillHardwareTable:** lists in a table all the hardware that can be used to measure a group of parameters, including information about the sensor name and about its price.
- **exportDoc:** displays on the content area all the use cases organized by MVP's listing all the actors, parameters and hardware for each one.

This front-end also provides an administration page for users with admin privileges. The next section provides instruction on the usage of the tool and the instructions are illustrated by several examples.

3.4. Manual

The IoT Catalogue provides a help mode which is intended to aid the user during its navigation. This mode can be activated in the top bar as shown in the Figure 15.

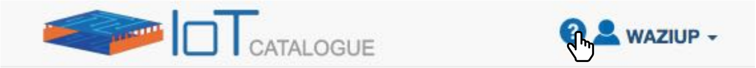
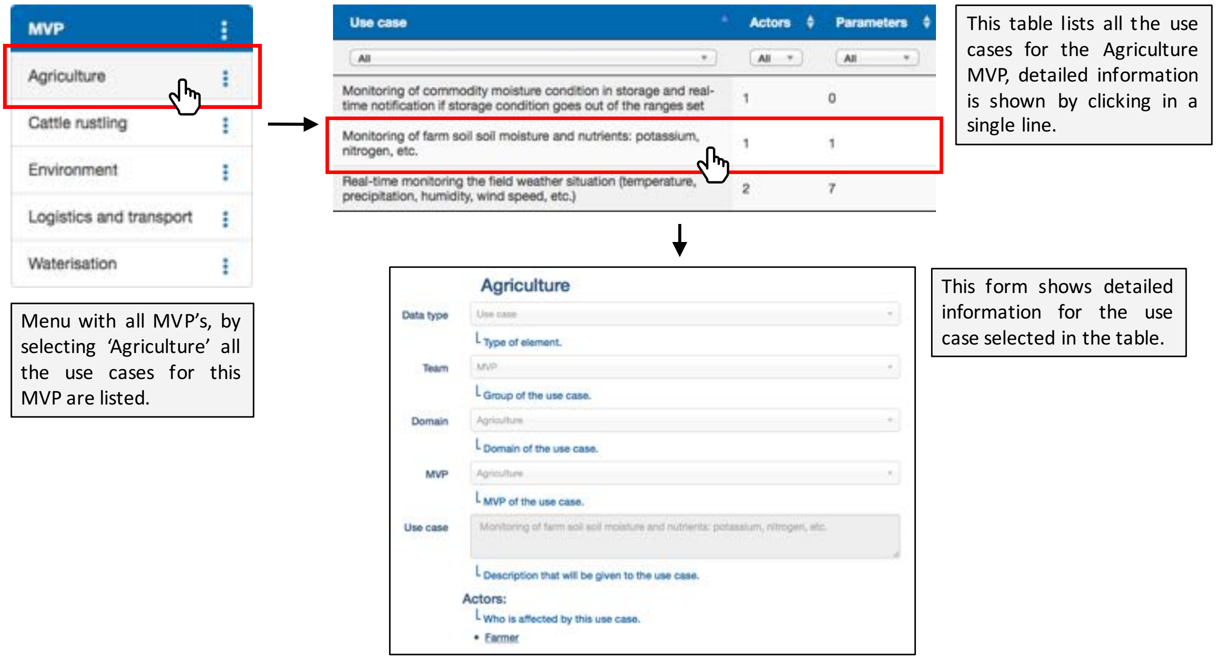


Figure 15 – Help mode

The web interface for each page includes extra instructions, making easier to consult and enter information. Navigation on the IoT catalogue is made through menus, tables and forms. Menus are used to select which type of information are listed in the table. It is then possible to select a line from the table to consult detailed information about a single element. Figure 16 shows how information about a single use case can be consulted.



Menu with all MVP's, by selecting 'Agriculture' all the use cases for this MVP are listed.

Use case	Actors	Parameters
All	All	All
Monitoring of commodity moisture condition in storage and real-time notification if storage condition goes out of the ranges set	1	0
Monitoring of farm soil soil moisture and nutrients: potassium, nitrogen, etc.	1	1
Real-time monitoring the field weather situation (temperature, precipitation, humidity, wind speed, etc.)	2	7

This table lists all the use cases for the Agriculture MVP, detailed information is shown by clicking in a single line.

Agriculture

Data type: Use case
└ Type of element.

Team: MVP
└ Group of the use case.

Domain: Agriculture
└ Domain of the use case.

MVP: Agriculture
└ MVP of the use case.

Use case: Monitoring of farm soil soil moisture and nutrients: potassium, nitrogen, etc.
└ Description that will be given to the use case.

Actors:
└ Who is affected by this use case.
- Farmer

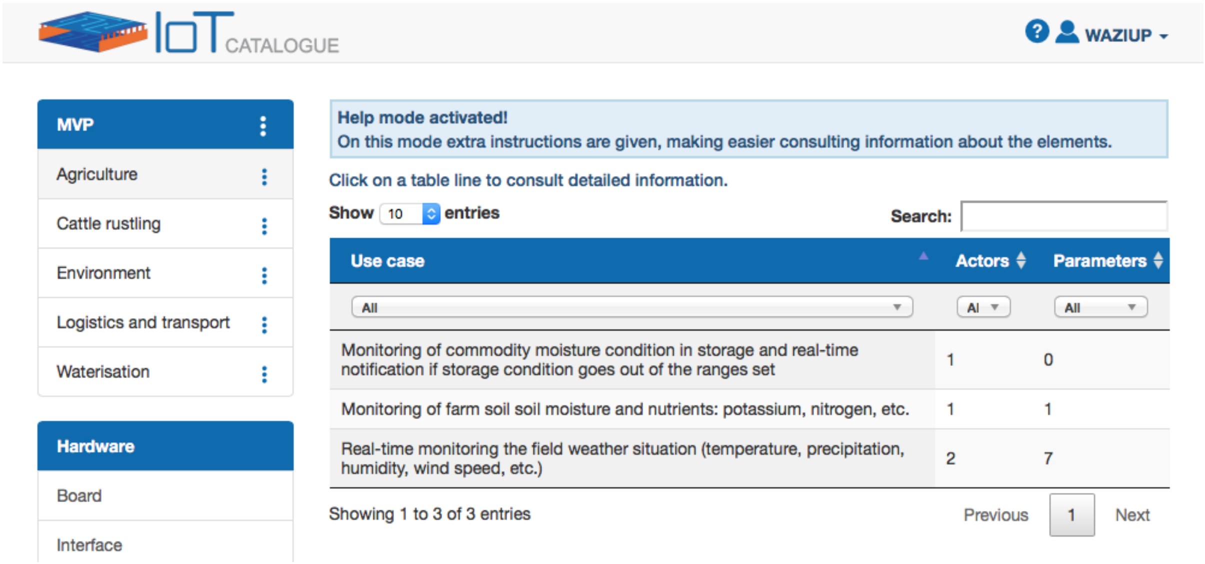
This form shows detailed information for the use case selected in the table.

Figure 16 – Consulting an use case

In the next paragraphs, examples are given for all the menus available under the navigation area, showing all the options and the types of information that are retrieved by using each menu.

3.4.1. MVP's menu

The IoT Catalogue in the context of WAZIUP organizes the information from the use cases among different MVP's. Under the first menu located in the navigation area it is possible to list all the use cases for a specific MVP. The following example shows how the agriculture related use cases are being listed.



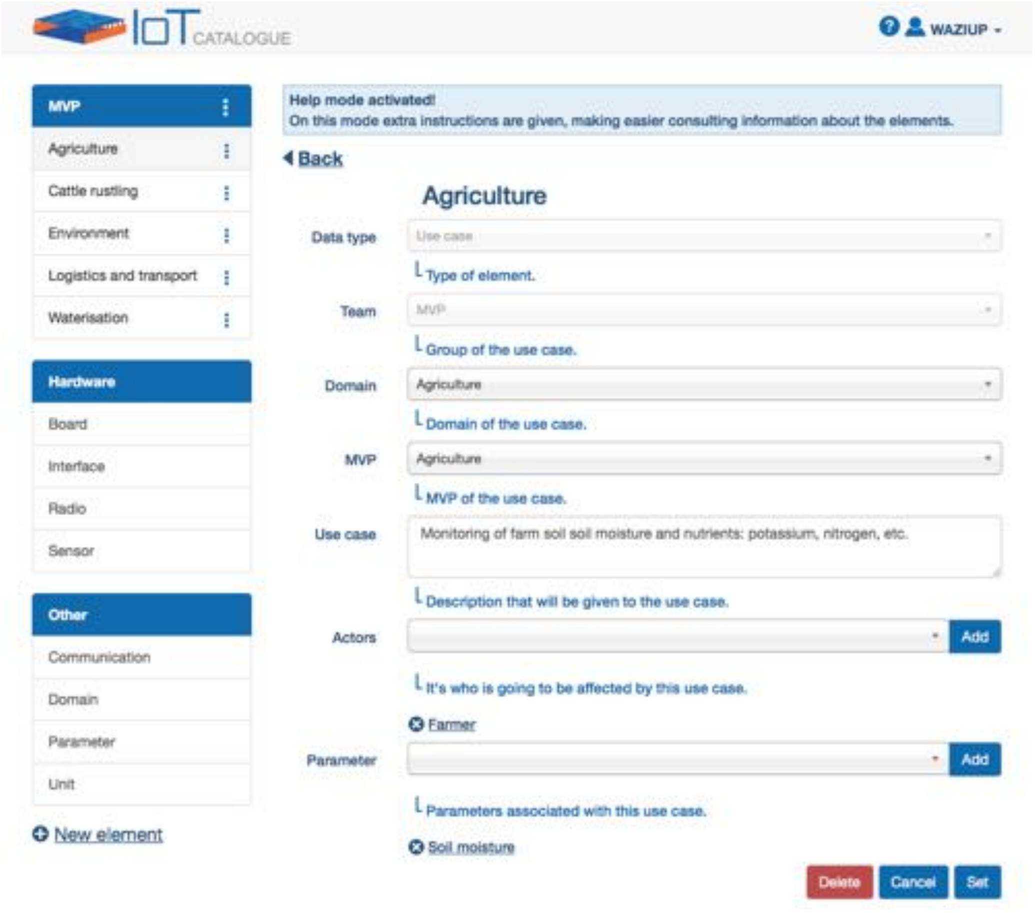
Help mode activated!
On this mode extra instructions are given, making easier consulting information about the elements.

Click on a table line to consult detailed information.

Use case	Actors	Parameters
All	Al	All
Monitoring of commodity moisture condition in storage and real-time notification if storage condition goes out of the ranges set	1	0
Monitoring of farm soil soil moisture and nutrients: potassium, nitrogen, etc.	1	1
Real-time monitoring the field weather situation (temperature, precipitation, humidity, wind speed, etc.)	2	7

Showing 1 to 3 of 3 entries

Figure 17 – Agriculture MVP's



Help mode activated!
On this mode extra instructions are given, making easier consulting information about the elements.

Agriculture

Data type: Use case
Type of element.

Team: MVP
Group of the use case.

Domain: Agriculture
Domain of the use case.

MVP: Agriculture
MVP of the use case.

Use case: Monitoring of farm soil soil moisture and nutrients: potassium, nitrogen, etc.
Description that will be given to the use case.

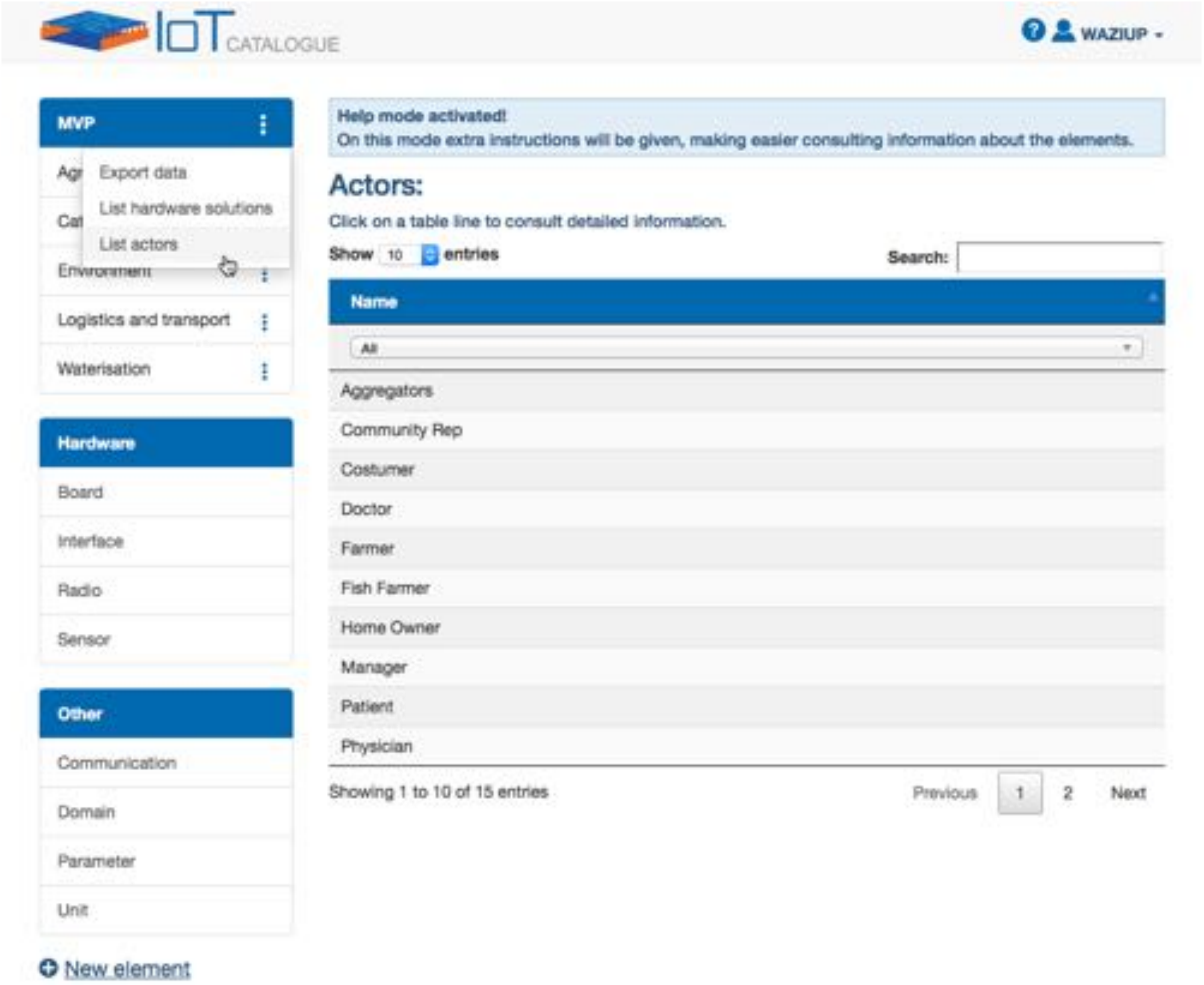
Actors:
It's who is going to be affected by this use case.
Farmer

Parameter:
Parameters associated with this use case.
Soil moisture

Figure 18 – Agriculture use case

The table in Figure 18 shows all the use cases related with the Agriculture MVP along with information about description of each one, the number of actors that are affected and the parameters that are measured under the context of the problem. By clicking in a line of the table it is possible to consult detailed information about the use case. An example is provided in Figure 18.

The form shown in Figure 19 details information about a use case of the Agriculture MVP. In the MVP menu it is also possible to list all the actors, the hardware used or to generate a report containing detailed information of each use case. Figure 19 shows a list of all the actors available on the tool.



Help mode activated!
On this mode extra instructions will be given, making easier consulting information about the elements.

Actors:

Click on a table line to consult detailed information.

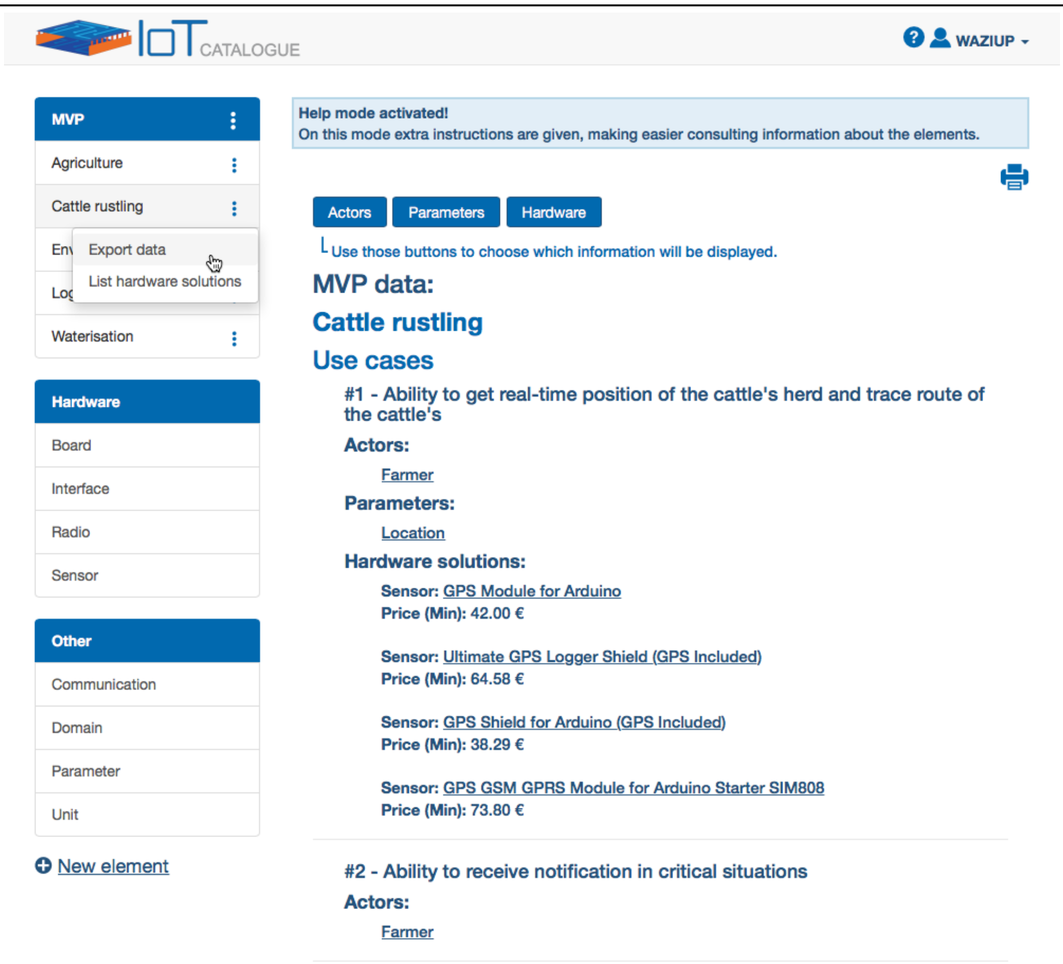
Show 10 entries Search:

Name
All
Aggregators
Community Rep
Costumer
Doctor
Farmer
Fish Farmer
Home Owner
Manager
Patient
Physician

Showing 1 to 10 of 15 entries Previous 1 2 Next

Figure 19 – List of all actors

All the actors used on the use cases are being listed along with its name. Clicking on a table line allows through a form to consult or edit its name. The information of the actors used is displayed when consulting information about a use case or a MVP. As shown in Figure 20 it is possible to see a report for Cattle Rustling MVP being generated including the information of its actors.



Help mode activated!
On this mode extra instructions are given, making easier consulting information about the elements.

Use those buttons to choose which information will be displayed.

MVP data:

Cattle rustling

Use cases

#1 - Ability to get real-time position of the cattle's herd and trace route of the cattle's

Actors:
Farmer

Parameters:
Location

Hardware solutions:

Sensor: GPS Module for Arduino
Price (Min): 42.00 €

Sensor: Ultimate GPS Logger Shield (GPS Included)
Price (Min): 64.58 €

Sensor: GPS Shield for Arduino (GPS Included)
Price (Min): 38.29 €

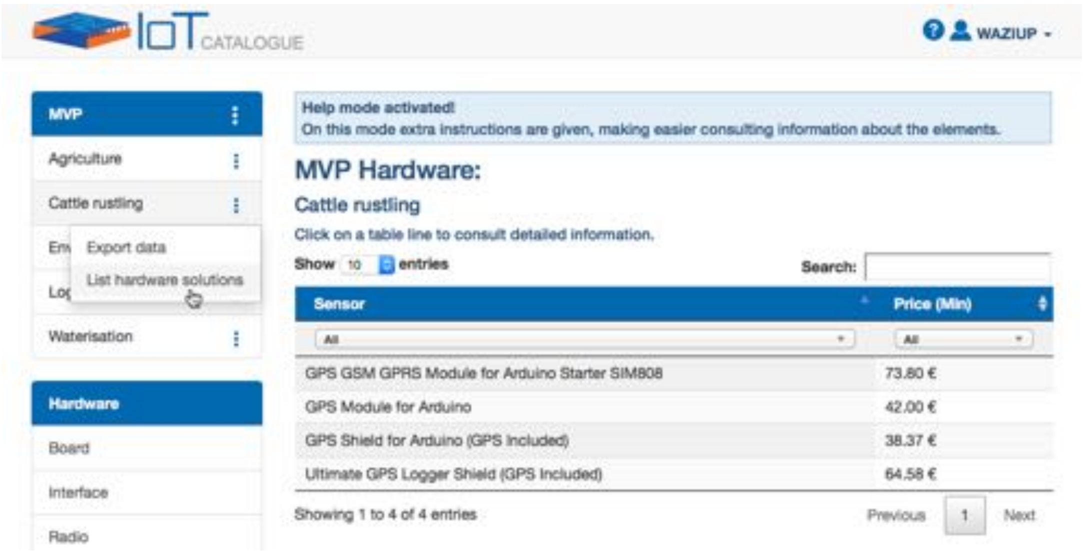
Sensor: GPS GSM GPRS Module for Arduino Starter SIM808
Price (Min): 73.80 €

#2 - Ability to receive notification in critical situations

Actors:
Farmer

Figure 20 – Cattle Rustling report

In this report it is possible to see for each use case detailed information along with its actors, the parameters and the hardware solutions. There are buttons (located on the top) used to hide/unhide information per category and a button (located on the top right) used to print the information displayed. It is also possible to list the hardware solutions per MVP.



Help mode activated!
On this mode extra instructions are given, making easier consulting information about the elements.

MVP Hardware:

Cattle rustling

Click on a table line to consult detailed information.

Show 10 entries Search:

Sensor	Price (Min)
All	All
GPS GSM GPRS Module for Arduino Starter SIM808	73.80 €
GPS Module for Arduino	42.00 €
GPS Shield for Arduino (GPS Included)	38.37 €
Ultimate GPS Logger Shield (GPS Included)	64.58 €

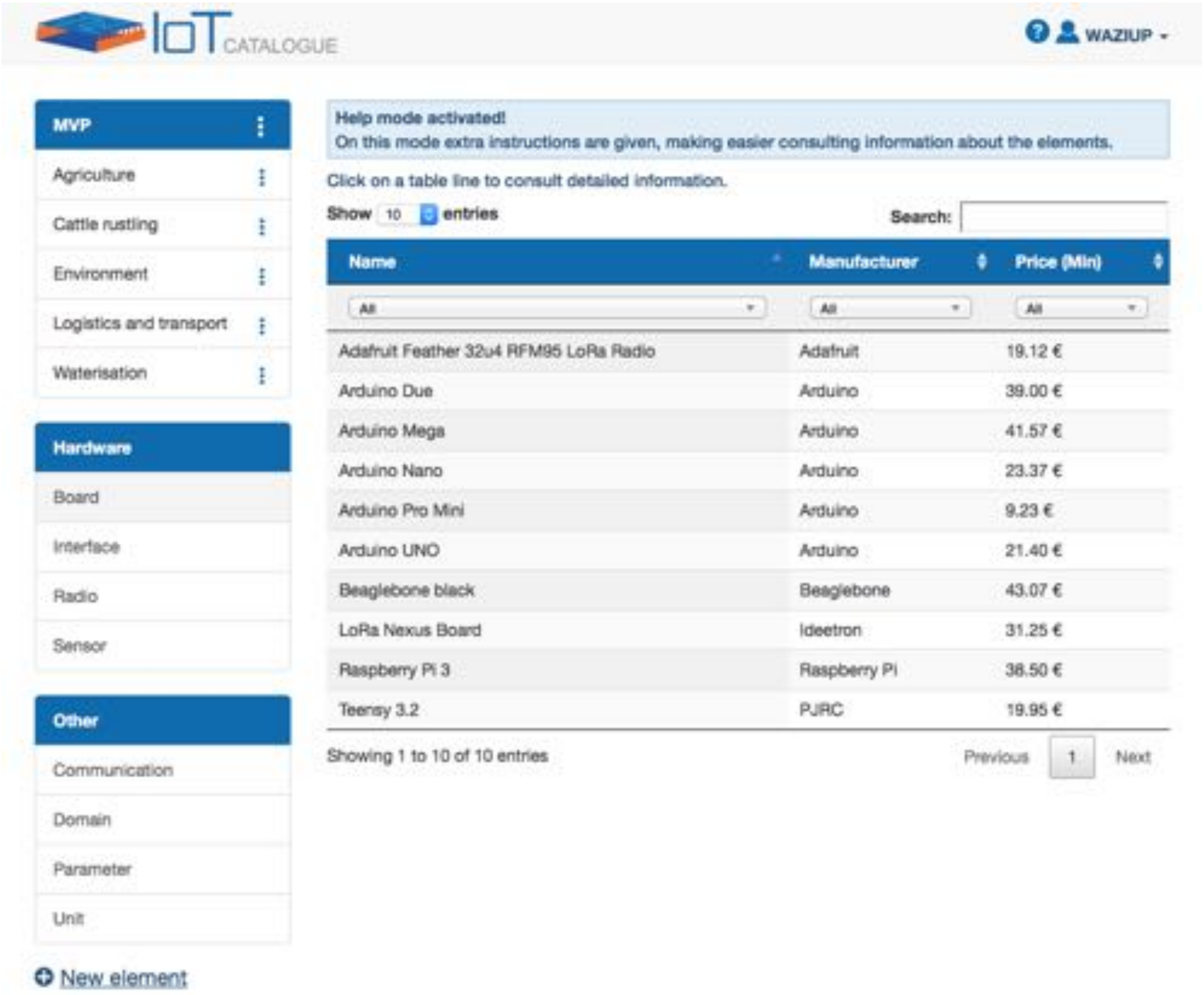
Showing 1 to 4 of 4 entries Previous 1 Next

Figure 21 – Hardware solutions for Cattle rustling

The table includes all the sensors used on Cattle Rustling MVP along with information about its name and price.

3.4.2. Hardware menu

The Hardware menu lists the hardware stored on the IoT Catalogue to choose among Board, Interface, Radio and Sensor to list the components of a specific type. The information is displayed in a table and it includes the name of the component, the manufacturer and the cost considering the vendor with the lowest price. In Figure 22 we can see a example where information about the all the boards is being listed.



Help mode activated!
On this mode extra instructions are given, making easier consulting information about the elements.

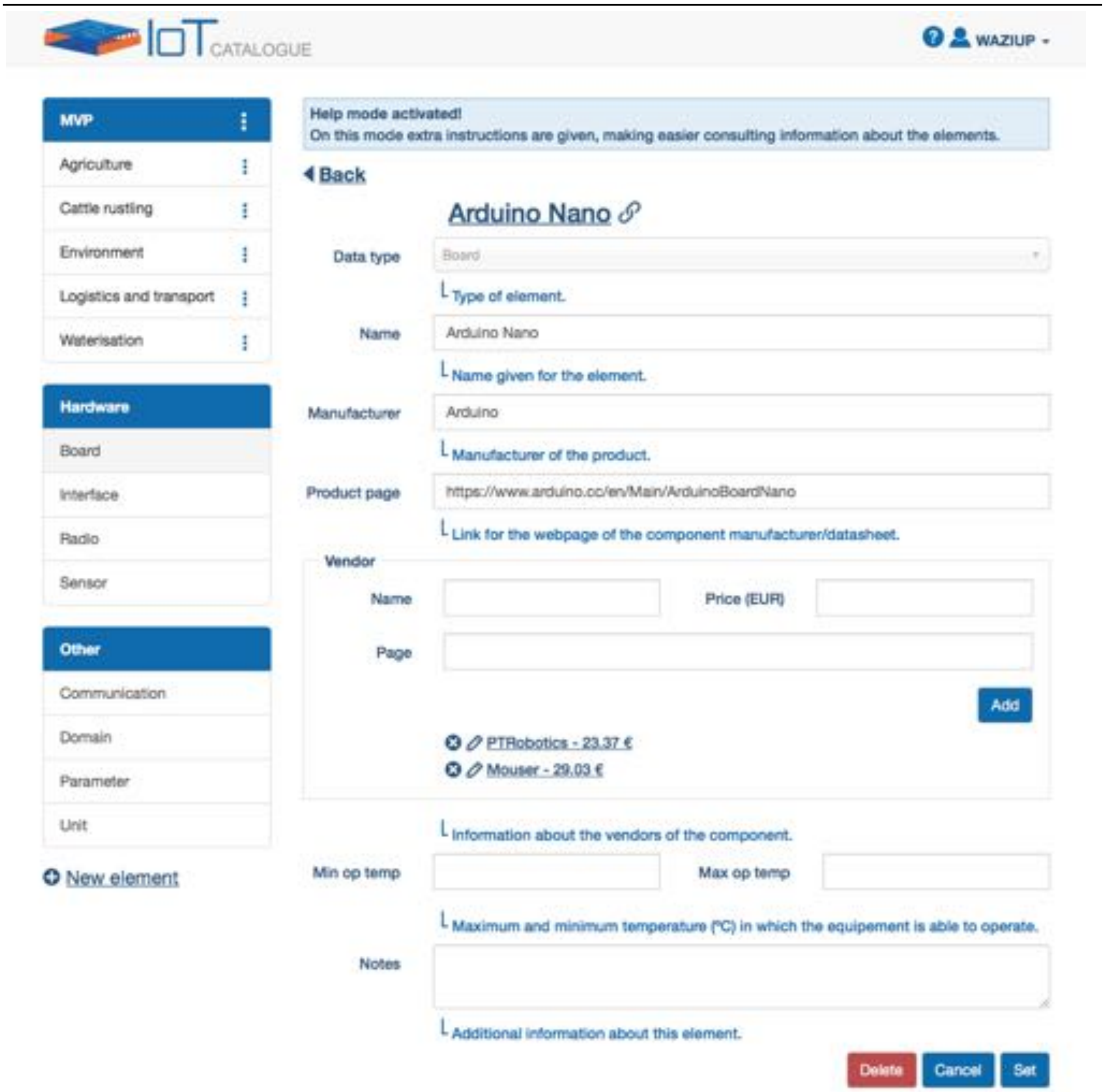
Click on a table line to consult detailed information.

Name	Manufacturer	Price (Min)
All	All	All
Adafruit Feather 32u4 RFM95 LoRa Radio	Adafruit	19.12 €
Arduino Due	Arduino	39.00 €
Arduino Mega	Arduino	41.57 €
Arduino Nano	Arduino	23.37 €
Arduino Pro Mini	Arduino	9.23 €
Arduino UNO	Arduino	21.40 €
Beaglebone black	Beaglebone	43.07 €
LoRa Nexus Board	Ideetron	31.25 €
Raspberry Pi 3	Raspberry Pi	38.50 €
Teensy 3.2	PJRC	19.95 €

Showing 1 to 10 of 10 entries

Figure 22 – List of all boards

All the boards are being listed along with basic information. By clicking in a table line it is possible to see/edit detailed information about the component. Figure 23 shows an example where detailed information about the “Arduino Nano” is being shown.



Help mode activated!
On this mode extra instructions are given, making easier consulting information about the elements.

Arduino Nano

Data type: Board
Type of element.

Name: Arduino Nano
Name given for the element.

Manufacturer: Arduino
Manufacturer of the product.

Product page: https://www.arduino.cc/en/Main/ArduinoBoardNano
Link for the webpage of the component manufacturer/datasheet.

Vendor
Name: Price (EUR):
Page:
PTRobotics - 23.37 €
Mouser - 29.03 €
Information about the vendors of the component.

Min op temp: Max op temp:
Maximum and minimum temperature (°C) in which the equipement is able to operate.

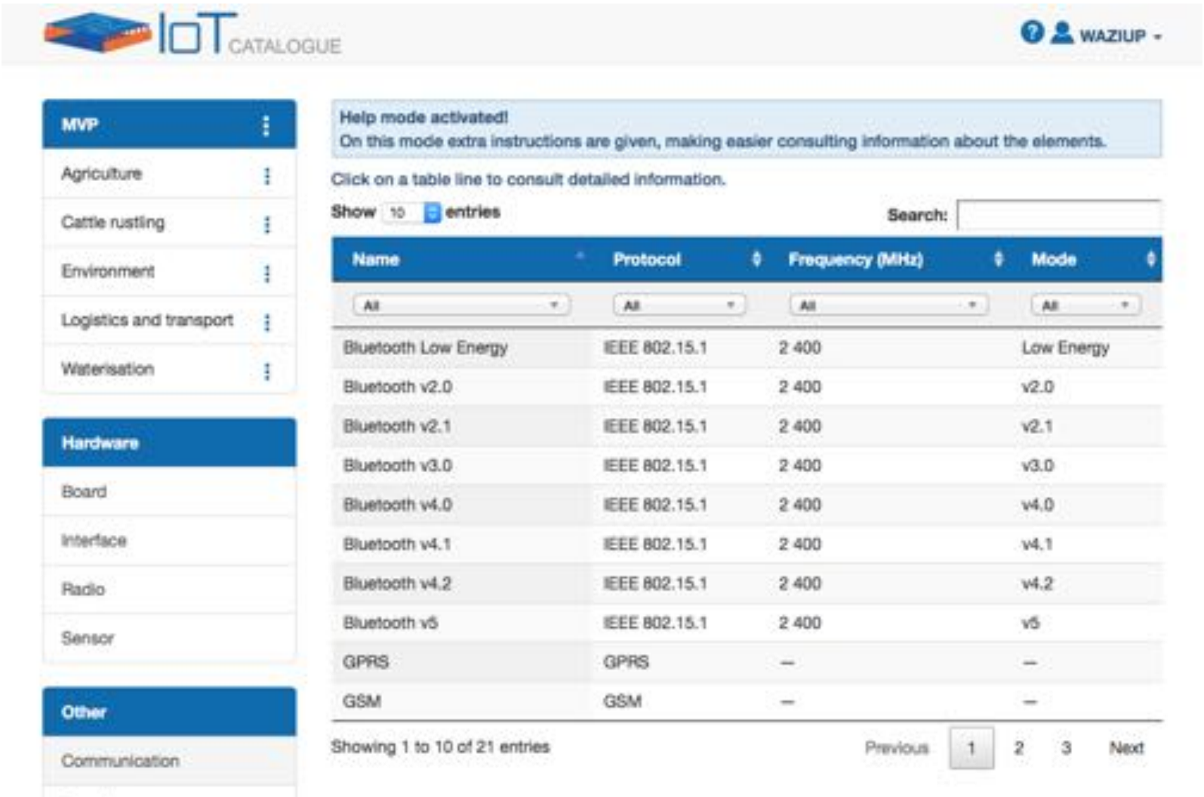
Notes:
Additional information about this element.

Figure 23 – Arduino Nano

On this screen it is possible to see and edit all the attributes for the sensor “Analog pH Meter Kit - Probe". It is also possible to remove the document from the database by using the “Delete” button located at the bottom of the form.

3.4.3. Other menu

The last menu of the navigation gives access to miscellaneous components such as radio communication protocols, domains, parameters and units. In the following example it is possible to see the listing of all radio communication protocols.



Help mode activated!
On this mode extra instructions are given, making easier consulting information about the elements.

Click on a table line to consult detailed information.

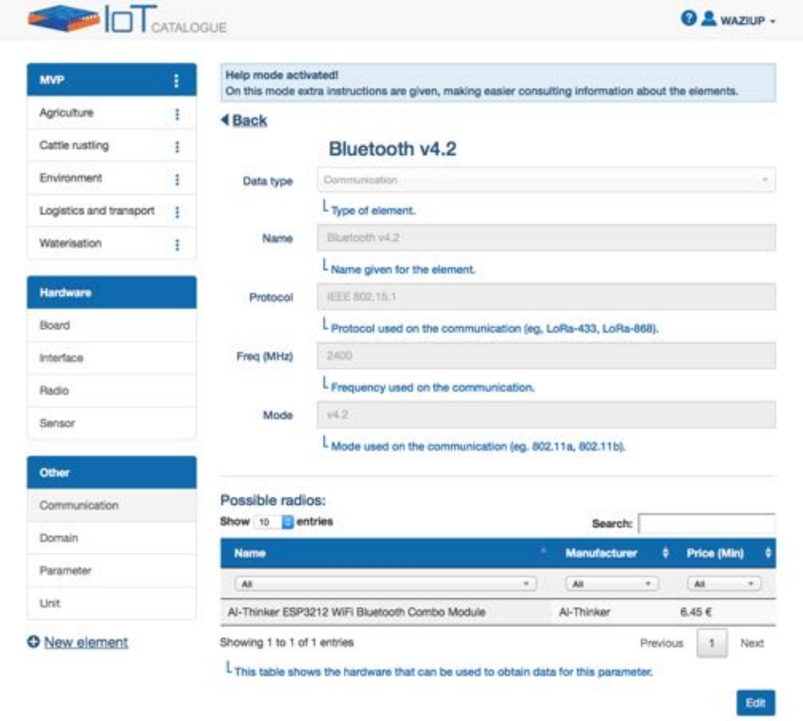
Show 10 entries Search:

Name	Protocol	Frequency (MHz)	Mode
All	All	All	All
Bluetooth Low Energy	IEEE 802.15.1	2 400	Low Energy
Bluetooth v2.0	IEEE 802.15.1	2 400	v2.0
Bluetooth v2.1	IEEE 802.15.1	2 400	v2.1
Bluetooth v3.0	IEEE 802.15.1	2 400	v3.0
Bluetooth v4.0	IEEE 802.15.1	2 400	v4.0
Bluetooth v4.1	IEEE 802.15.1	2 400	v4.1
Bluetooth v4.2	IEEE 802.15.1	2 400	v4.2
Bluetooth v5	IEEE 802.15.1	2 400	v5
GPRS	GPRS	—	—
GSM	GSM	—	—

Showing 1 to 10 of 21 entries Previous 1 2 3 Next

Figure 24 – Radio communication protocols

For each communication protocol it is possible to see information regarding its name, the protocol standard, the frequency and the mode of operation. By selecting a single element it is possible to see detailed information about the protocol and also the hardware that is associated with the protocol. Figure 25 shows an example for the "Bluetooth v4.2".



Help mode activated!
On this mode extra instructions are given, making easier consulting information about the elements.

Back

Bluetooth v4.2

Data type: Communication
└ Type of element.

Name: Bluetooth v4.2
└ Name given for the element.

Protocol: IEEE 802.15.1
└ Protocol used on the communication (eg. LoRa-433, LoRa-868).

Freq (MHz): 2400
└ Frequency used on the communication.

Mode: v4.2
└ Mode used on the communication (eg. 802.11a, 802.11b).

Possible radios:

Show 10 entries Search:

Name	Manufacturer	Price (Min)
All	All	All
AI-Thinker ESP3212 WiFi Bluetooth Combo Module	AI-Thinker	6.45 €

Showing 1 to 1 of 1 entries Previous 1 Next

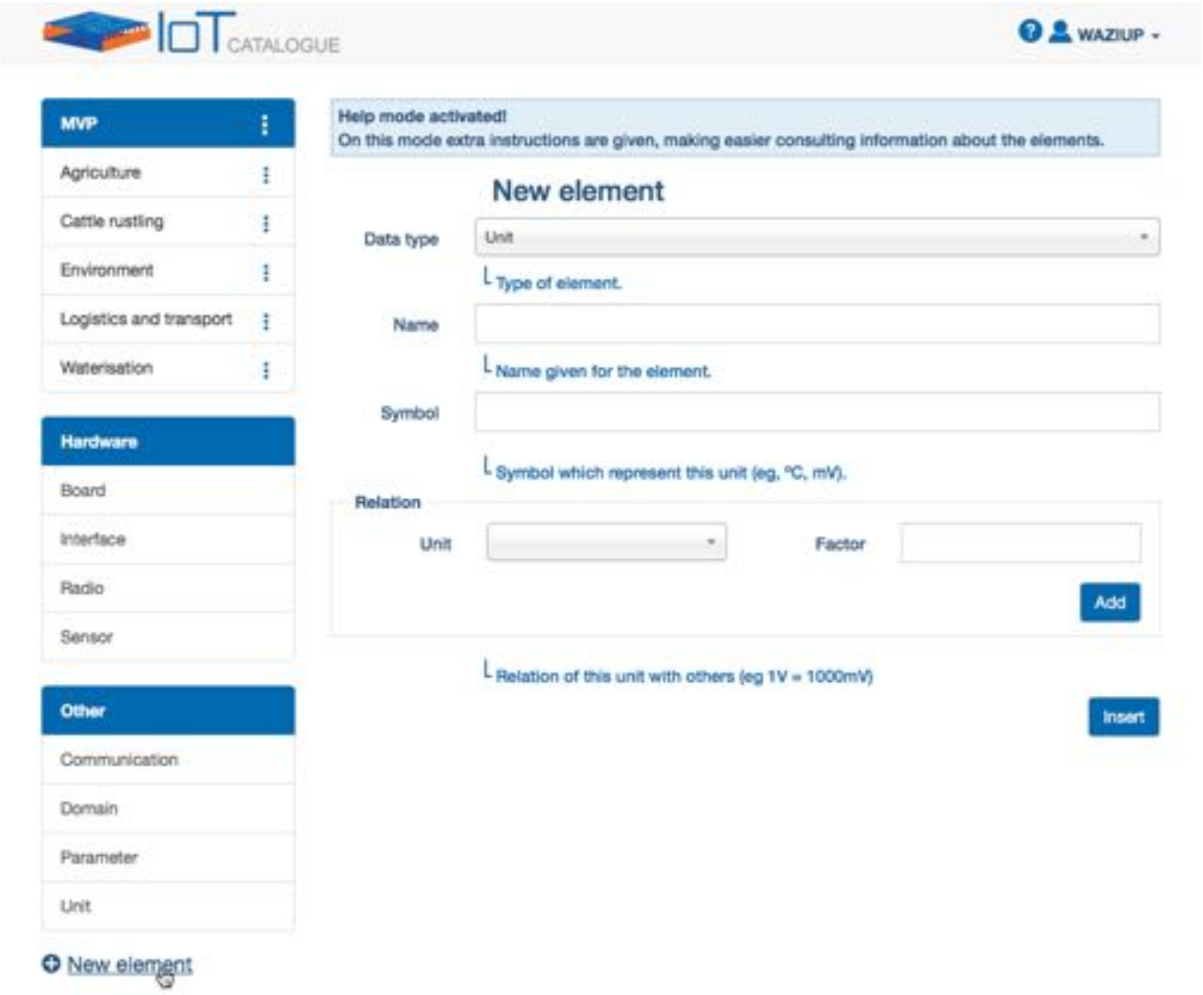
└ This table shows the hardware that can be used to obtain data for this parameter.

Figure 25 – Bluetooth v4.2

3.4.4. New element

In the IoT Catalogue, adding new information about a component is done through a form. For each type of component taking into account its attributes a different template is used to generate the form that will allow the user to input information about the component. Figure 26 shows an example for a case where a new unit is being added to the IoT Catalogue. There are templates for the following component types:

- Actor
- Board
- Communication
- Domain
- Interface
- MVP
- Parameter
- Radio
- Sensor
- Unit
- Use case



Help mode activated!
On this mode extra instructions are given, making easier consulting information about the elements.

New element

Data type: Unit
Type of element.

Name:
Name given for the element.

Symbol:
Symbol which represent this unit (eg. ºC, mV).

Relation
Unit: Factor:

Relation of this unit with others (eg 1V = 1000mV)

Figure 26 – New unit

On this example the loaded form was based on the “Unit” template where each field of the form is related to an attribute of the produced JSON object for this component and where some of the fields are mandatory. The following page shows a summary of all option that are available through the navigation area.

The navigation area in the IoT Catalogue contains several options and each one has its own type of interaction with the user to consult or edit information about the elements stored on the tool. The following figure gives an overview of the navigation area where it is possible to see all the options available, how they are related each other and how they interact with the user (by forms, table, menus, etc.).

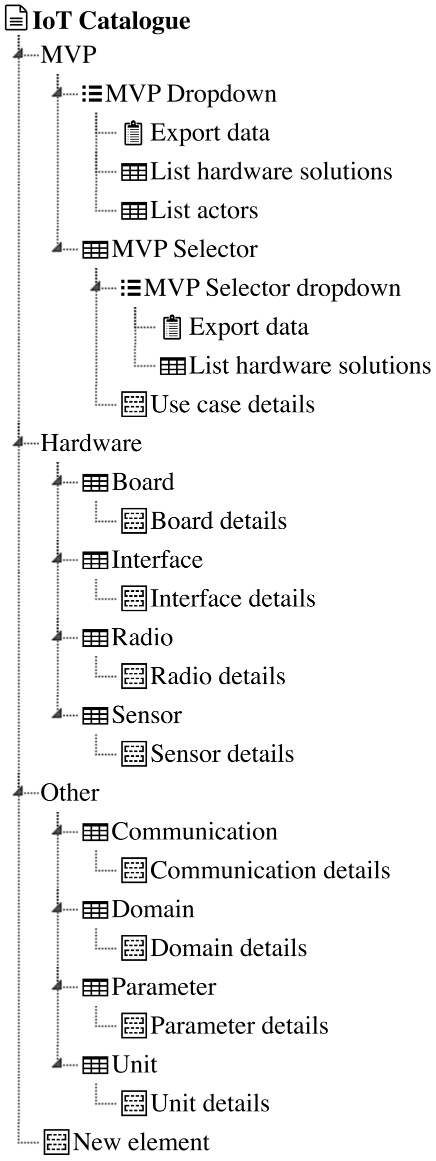


Figure 27 – Navigation overview

4. IoT CONNECTIVITY

4.1. Introduction

IoT devices naturally use wireless technologies to communicate. IoT communications are categorized as Machine-to-Machine communications to differentiate them from traditional human-oriented communications.

Many IoT applications are generally remote sensing systems where measures (e.g. temperature, acceleration, contact time,...) from the physical world, or from entities in the physical world, are collected for monitoring and possibly analysis & instrumentation tasks. The application model of IoT is very large and WAZIUP will not address all of them. **In the WAZIUP project, the remote sensing scenarios are will be the main scenarios for deployed applications** and Figure 28 shows a typical IoT remote sensing using either simple 1-hop connectivity (red links) or multi-hop connectivity (black dotted links) from end-device to a gateway.

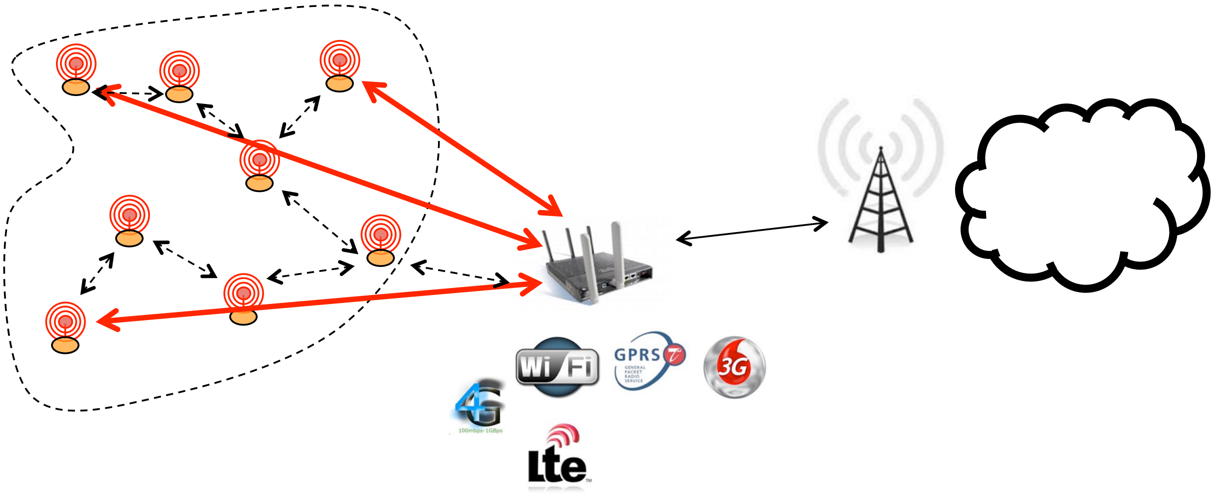
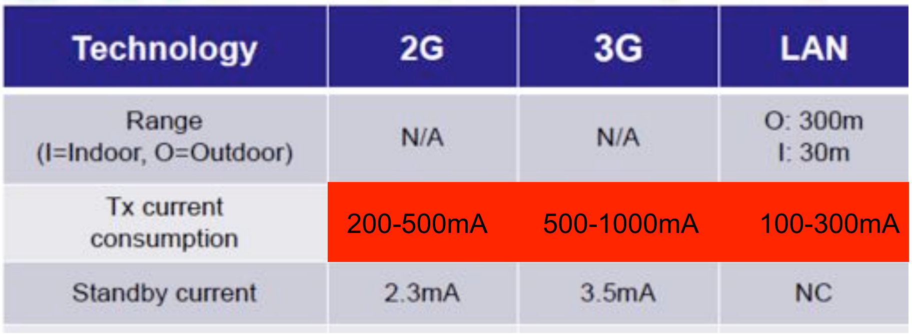


Figure 28 – Remote sensing scenario with 1-hop connectivity to the gateway

1-hop connectivity has clear advantages over the multi-hop case because of its simplicity when deploying the devices on a large-scale. Multi-hop communications typically need more complex mechanisms to be implemented such as routing protocols, synchronization issues when duty-cycling is applied, funneling effects for devices close to the gateway,... Over the last decade, the main radio technology for multi-hop IoT is IEEE 802.15.4 which serves for instance as the physical layer of ZigBee, WirelessHart & ISA100 networks and products to name a few.

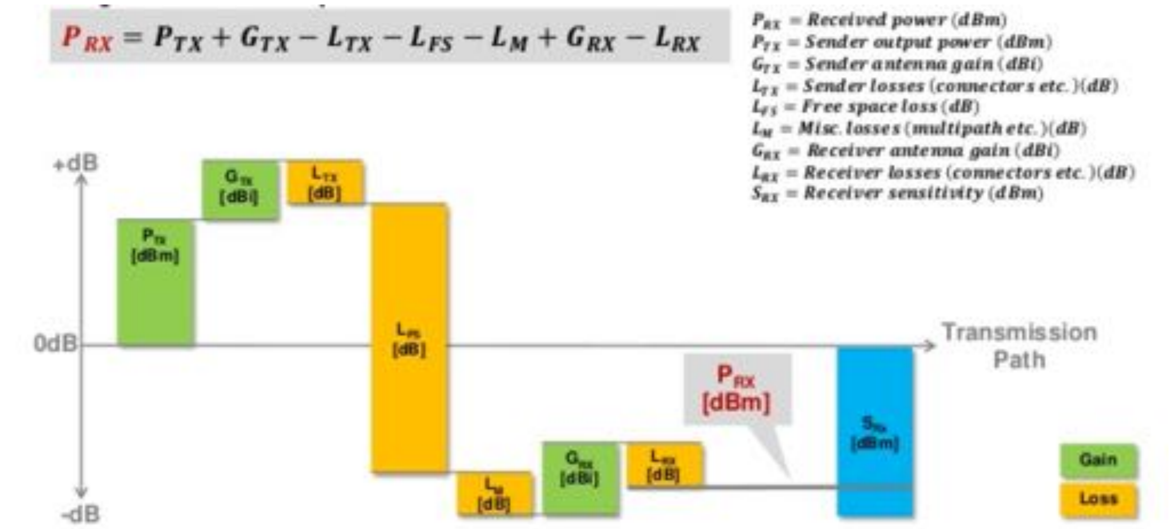
However, 1-hop connectivity in long-range scenarios is very energy consuming with traditional cellular radio technologies such as GSM or GPRS as shown in Figure 29. Therefore it is difficult to deploy battery-operated devices under these technologies that have a long lifetime. In addition to the energy cost, cellular technologies rely on mobile cellular operators where the cost of the network/service subscription fees is far from being negligible.



Technology	2G	3G	LAN
Range (I=Indoor, O=Outdoor)	N/A	N/A	O: 300m I: 30m
Tx current consumption	200-500mA	500-1000mA	100-300mA
Standby current	2.3mA	3.5mA	NC

Figure 29 – Energy consumption for traditional cellular technologies

An important feature for an IoT 1-hop radio technology candidate is therefore low power consumption. The other feature is of course the long-range transmission capability. Regarding transmission (and reception) range, Figure 30 shows in a very simplified view the transmission/reception chain of a wireless transmission where the notion of "receiver's sensitivity" for correct packet reception is illustrated.



$$P_{RX} = P_{TX} + G_{TX} - L_{TX} - L_{FS} - L_M + G_{RX} - L_{RX}$$

P_{RX} = Received power (dBm)
P_{TX} = Sender output power (dBm)
G_{TX} = Sender antenna gain (dBi)
L_{TX} = Sender losses (connectors etc.)(dB)
L_{FS} = Free space loss (dB)
L_M = Misc. losses (multipath etc.)(dB)
G_{RX} = Receiver antenna gain (dBi)
L_{RX} = Receiver losses (connectors etc.)(dB)
S_{RX} = Receiver sensitivity (dBm)

From Peter R. Egli, INDIGOO.COM

Figure 30 – Simplified view of the link budget of a wireless transmission

The receiver's sensitivity is usually defined as the lowest input power with acceptable link quality, typically 1% PER (Packet Error Rate). Generally, the receiver's sensitivity, at a given transmission power, can be greatly improved by reducing the throughput.

Recently, 2 radio technologies have initiated an incredible movement towards low-power long-range wide area networks (LPWAN): SigFox^{TM} and LoRa^{TM} where the range in 1-hop can be more than 15kms in Line-of-Sight (LOS) conditions. These LPWAN technologies are definitely more suitable for battery-operated IoT devices than traditional cellular technologies as their power consumption can be compared to previous short-range radio technologies (such as IEEE 802.15.4) but, with the increase in reception range, can avoid the complexity of multi-hop networks. They both follow the "talk slower": throughput is in the order of several kbps which is by far much smaller than traditional wireless technologies. Following these precursor technologies, many other technologies or adaptation of existing technologies are also trying to address the low-power, long-range and lower throughput IoT market. The next section will provide a brief review of these LPWAN technologies.

4.2. Brief review of long-range technologies

Figure 31 shows the main IEEE 802 wireless technologies and their corresponding target data rate and range. The region addressed by the recent LPWAN technologies is typically the one providing very low data rate but longer range. This region has not been significantly addressed in the past. SigFox and LoRa are the first true long-range technologies with a high performance/price tradeoff, making them very promising for large-scale deployment of IoT.

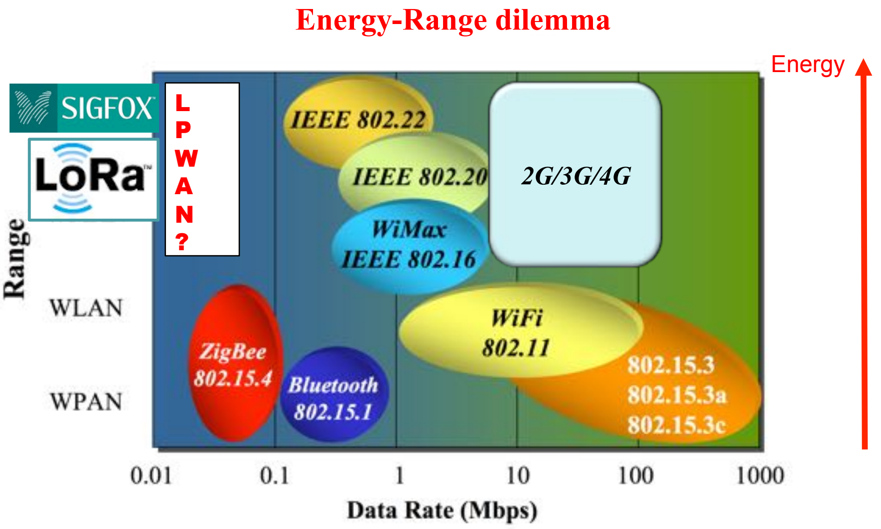


Figure 31 – IEEE 802 wireless technologies and LPWAN

There are other technologies that claim to address the LPWAN segment. It is beyond the purpose of this document to describe them is detail but we can mention some of them:

- Weightless-N, Weightless-P
- LTE-CatM1
- RPMA
- IEEE 802.11ah
- NWave
- Telensa
- NB-IoT
- Amber-Wireless
- WaveIoT

Many of them are not available yet on a mass-market perspective. SigFox and LoRa are still the 2 major LPWAN technologies that received important support from a large number of industrials and operators. Readers willing to know more about these technologies can first refer to the LinkLab white paper [1] and then to the EDN.com article [2].

SigFox is very different from LoRa in the sense that SigFox position itself mainly as an operator wheras LoRa is only the physical modulation techniques that is patented by Semtech. Third parties can use (buy) Semtech LoRa chips to deploy their own LPWAN networks. Some third parties may be themselves operators with a traditional business model but some others can be simply end-users deploying their own ad-hoc LPWAN networks. This is why **WAZIUP project uses only LoRa technology to build low-cost IoT infrastructures and platforms.**

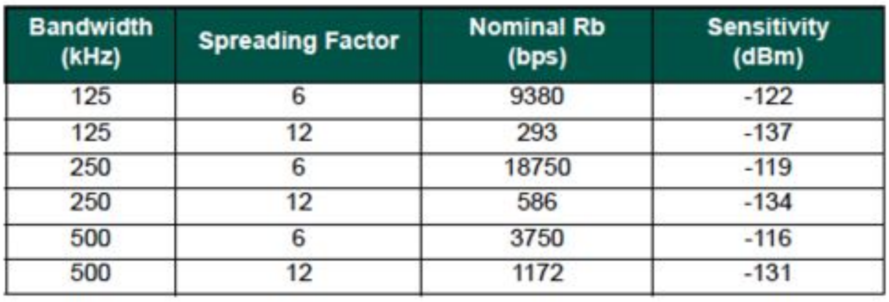
4.3. LoRa long-range technology

LoRa is a long-range radio technology developed by Semtech. Here is a definition from Semtech's LoRa FAQ:

"*LoRa™ (Long Range) is a modulation technique that provides significantly longer range than competing technologies. The modulation is based on spread-spectrum techniques and a variation of chirp spread spectrum (CSS) with integrated forward error correction (FEC). LoRa significantly improves the receiver sensitivity and as with other spread - spectrum modulation techniques, uses the entire channel bandwidth to broadcast a signal, making it robust to channel noise and insensitive to frequency offsets caused from the use of low cost crystals. LoRa can demodulate signals 19.5dB below the noise floor while most frequency shift keying systems (FSK) need a signal power of 8-10dB above the noise floor to demodulate properly. The LoRa modulation is the physical layer (PHY), which can be utilized with different protocols and in different network architecture – Mesh, Star, point to point, etcetera.*" [http://www.semtech.com/wireless-rf/lora/LoRa-FAQs.pdf]

In LoRa, throughput and range depend on 3 main LoRa parameters: BW, CR and SF. BW is the physical bandwidth for RF modulation and can take values of 500kHz, 250 kHz, 125kHz and 62.5kHz. Larger signal bandwidth allows for higher effective data rate, thus reducing transmission time at the expense of reduced sensitivity. CR is the coding rate for forward error detection and correction. Such coding incurs a transmission overhead and the lower the coding rate, the higher the coding rate overhead ratio, e.g. with coding_rate=4/(4+CR) the overhead ratio is 1.25 for CR=1 which is the minimum value. Finally, SF is the spreading factor that can be set from 6 to 12. The lower the SF, the higher the data rate transmission but the lower the immunity to interference thus the smaller is the range.

Figure 32 from Semtech shows for various bandwidth and spreading factor values the expected throughput and receiver sensitivity.



Bandwidth (kHz)	Spreading Factor	Nominal Rb (bps)	Sensitivity (dBm)
125	6	9380	-122
125	12	293	-137
250	6	18750	-119
250	12	586	-134
500	6	3750	-116
500	12	1172	-131

Figure 32 – LoRa throughput and sensitivity for various BW and SF values

4.3.1. Physical layer

At the physical layer, LoRa modulation is a variation of Chirp Spread Spectrum (CSS) that was long before been developed in radar technologies. We will not describe in detail the LoRa modulation as it is also beyond the scope of this document but the readers can refer to the Semtech document [3], the RevSpace article [4] and the Reversing LoRa article [5] that explains in a comprehensive manner the LoRa underlying modulation approach. However, for a fast glance on LoRa modulation, we include below a text taken from [3].

From Semtech's AN1200.22

"LoRa is a proprietary spread spectrum modulation scheme that is derivative of Chirp Spread Spectrum modulation (CSS) and which trades data rate for sensitivity within a fixed channel bandwidth. It implements a variable data rate, utilizing orthogonal spreading factors, which allows the system designer to trade data rate for range or power, so as to optimize network performance in a constant bandwidth."

"In LoRa modulation the spreading of the spectrum is achieved by generating a chirp signal that continuously varies in frequency. An advantage of this method is that timing and frequency offsets between transmitter and receiver are equivalent, greatly reducing the complexity of the receiver design. The frequency bandwidth of this chirp is equivalent to the spectral bandwidth of the signal."

We also include here an illustration of a LoRa transmission taken from [4].

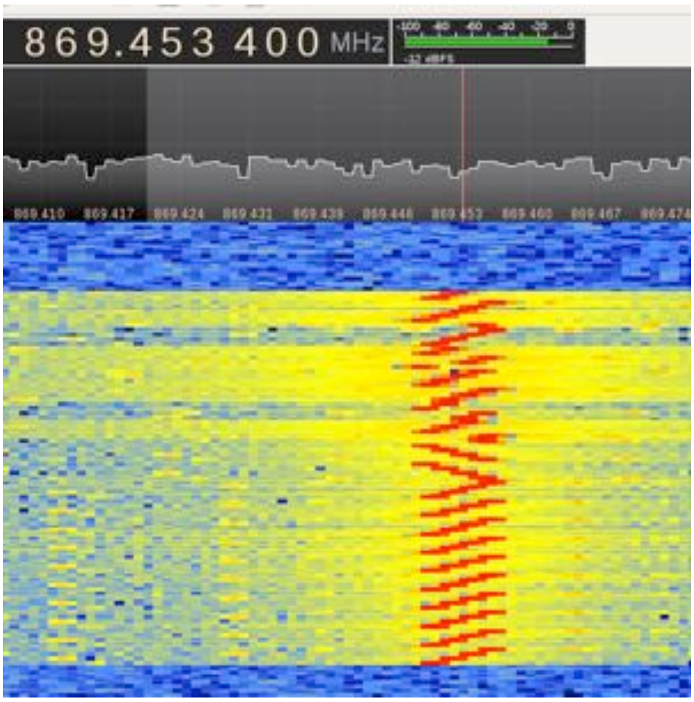


Figure 33 – spectrogram example of a LoRa transmission

"The image [...] shows the LoRa spectrogram for a short message as recorded by gqrx, when sending a 1-byte payload (with settings SF=12,BW=8,CR=4/8, implicit header). At the bottom of the spectrogram you can see the preamble consisting of 10 up-chirps and 2 down-chirps. At the top of the spectrogram you see the data portion of the signal, consisting solely of up-chirps."

4.3.2. LoRaWAN

As LoRa defines only the physical layer, the modulation, the LoRaWAN specification [6] [7] additionally defines common data and control channels (frequency and spreading factors), packet format, MAC commands,... for large-scale deployment with network servers and application servers. LoRaWAN also defines several classes for the end-device depending on the communication needs. Each class has its own requirements also defined by LoRaWAN specifications.

From Semtech's LoRa FAQ:

"The LoRa modulation is the PHY, and LoRaWAN is a MAC protocol for a high capacity long range and low power star network that the LoRa Alliance is standardizing for Low Power Wide Area Networks (LPWAN). The LoRaWAN protocol is optimized for low cost, battery operated sensors and includes different classes of nodes to optimize the tradeoff between network latency and battery lifetime. It is fully bi-directional and was architected by security experts to ensure reliability and safety. The architecture of LoRaWAN was also designed to easily locate mobile objects for asset tracking, which is one of the fastest growing volume applications for Internet of Things (IoT). LoRaWAN is being deployed for nationwide networks by major telecom operators, and the LoRa Alliance is standardizing LoRaWAN to make sure the different nationwide networks are interoperable."

A typical LoRaWAN network is depicted in Figure 34. Network servers will handle all communications between end-devices and gateways and therefore implement most of the LoRaWAN functionalities by interacting at a lower level with the gateways. Application servers can serve to store the end-user application data.

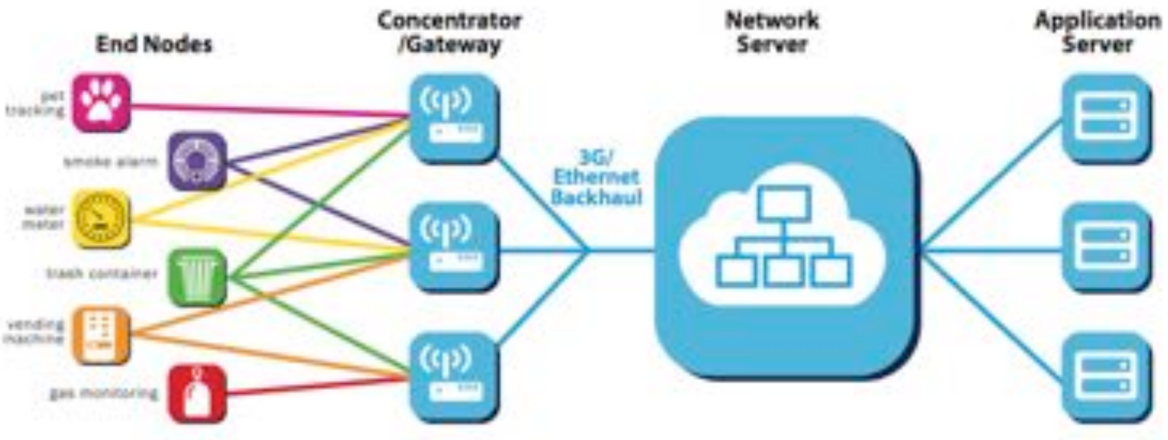


Figure 34 – LoraWAN typical topology

4.3.3. LoRa radio modules

The LoRa modulation is implemented in so-called LoRa transceiver chips provided by Semtech. They belong to the SX127x family and LoRa modulation is implemented in SX1272/73/76/77/78 chips as documented in [8].

Various manufacturers then use the Semtech chips to propose LoRa radio modules that can further be integrated/connected into microcontroller boards. Note that although an SX1276 chip can for example support several frequency bands, i.e. 433MHz, 868MHz and 915MHz, the design of a radio module to suit a given frequency band can be different as documented in the Semtech Reference Design Overview document [9]. This is mainly due to how the different amplifier components/lines of the LoRa chips are used/wired to power the radio transmission.

Figure 35 shows some radio modules that we fully tested in the WAZIUP project, as it will be explained later on in section 5.3.2.

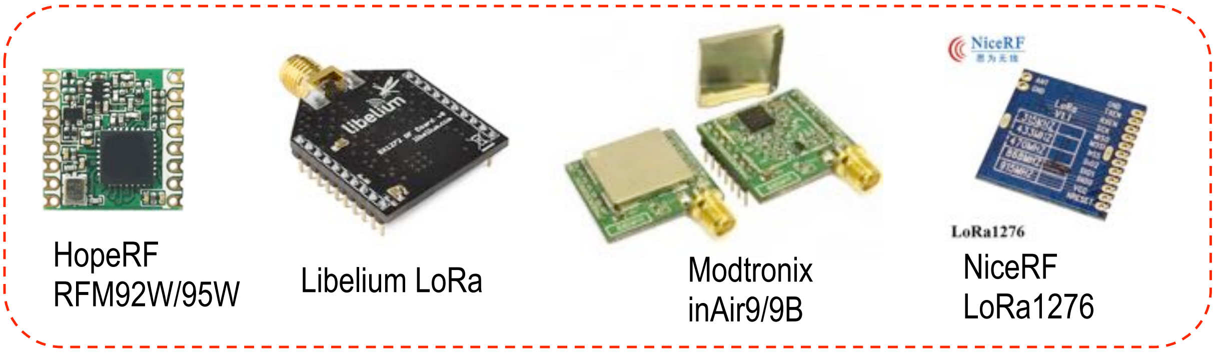


Figure 35 – some radio module built from SX127x Semtech chips

LoRa radio modules can either expose the raw LoRa transceiver features or add a LoRaWAN protocol stack on top of the Semtech chip. The 6 modules shown above only expose the raw LoRa transceiver functionalities.

4.4. Transmission in the unlicensed frequency band

Wireless communications are usually highly regulated in most countries and it is not allowed to transmit freely in a frequency band except or some so-called unlicensed bands. The ISM (Industry, Scientific and Medical) band is one of the most well-known unlicensed band. For instance, the 2.4GHz ISM band is an internationally agreed unlicensed band so that WiFi operating in this band can be deployed free of license authorization and charges. It is the same reason that allows most people on Earth to be able to have microwave ovens as they use 2.4GHz microwaves. Figure 36 shows a brief overview of the unlicensed frequency bands per region.

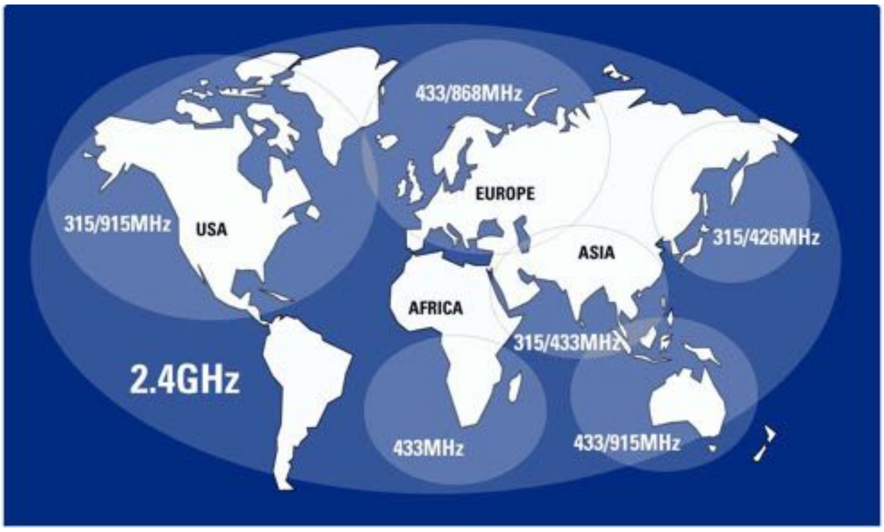
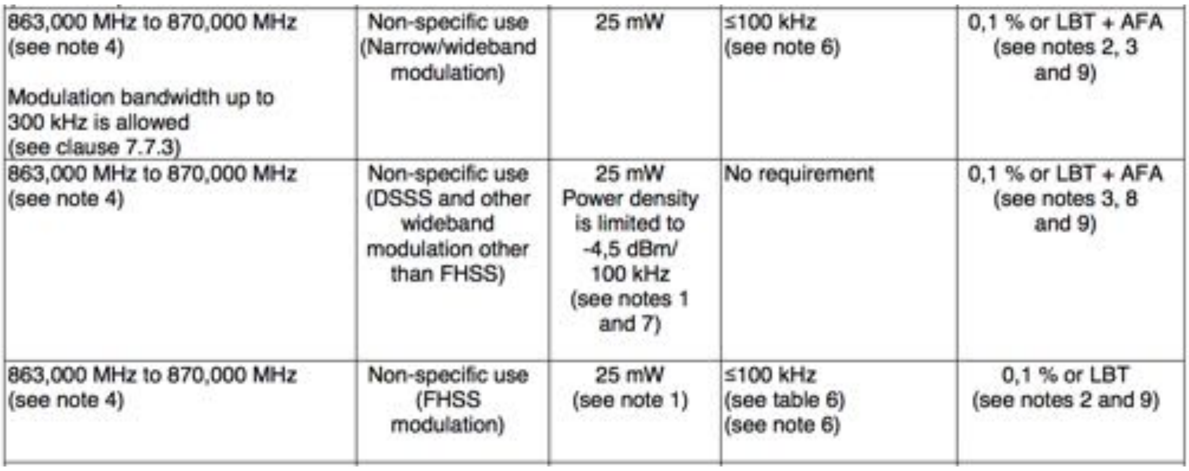


Figure 36 – ISM band per region

Many recent long-range radio technologies (thus including LoRa) do operate in the unlicensed frequency bands that usually change from one country to another. The choice of unlicensed band is not really a technical constraint because these long-range technologies can also operate in other frequency bands that are licensed (for instance those uses by mobile telephony operators). The choice of unlicensed band is rather motivated by the ease of deployment of these initial long-range devices in order to boost the market. It is foreseen that many operators will finally reuse GSM/2G frequency bands for deploying LPWAN technologies in licensed bands.

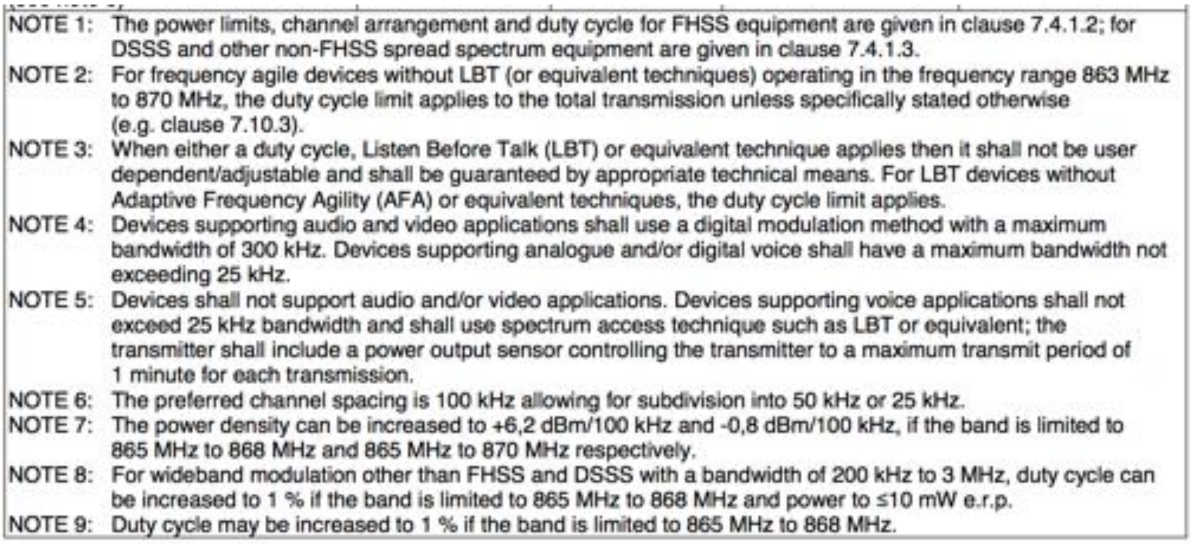
In most of European countries, the unlicensed bands for LoRa usage is 433MHz and 868MHz. These countries usually adhere to the ETSI regulations for Short Range Device (SRD) in sub-GHz bands defined in the ETSI EN 300-220-1 document [10]. The 868MHz band is usually preferred for IoT as the 433MHz band is quite saturated by existing radio devices such as audio, alarms and other remote control devices. Additionally, as these bands are “free”, some constraints are to be put to avoid excessive usage and saturation of the frequency bands. The main regulation mechanisms are transmission power limitation and transmission time limitation. The latter is usually refer to as “duty-cycle” limitation expressed for instance as 1% per hour. In this case, the total transmission time of an SRD device under 1% duty-cycle is 36s per hour period. More details on SRD regulations can be read in [10]. For transmission power limitation, typical limitation is 14dBm (25mW) in Europe in the 868MHz band as illustrated in Figure 37 below taken from ETSI EN 300-220-1 document [10].



863,000 MHz to 870,000 MHz (see note 4) Modulation bandwidth up to 300 kHz is allowed (see clause 7.7.3)	Non-specific use (Narrow/wideband modulation)	25 mW	≤100 kHz (see note 6)	0,1 % or LBT + AFA (see notes 2, 3 and 9)
863,000 MHz to 870,000 MHz (see note 4)	Non-specific use (DSSS and other wideband modulation other than FHSS)	25 mW Power density is limited to -4,5 dBm/ 100 kHz (see notes 1 and 7)	No requirement	0,1 % or LBT + AFA (see notes 3, 8 and 9)
863,000 MHz to 870,000 MHz (see note 4)	Non-specific use (FHSS modulation)	25 mW (see note 1)	≤100 kHz (see table 6) (see note 6)	0,1 % or LBT (see notes 2 and 9)

Figure 37 – 868MHz unlicensed band restrictions

Note that 1% duty cycle can be used provided that the band is restricted to 865-868MHz as indicated mainly by note 9 below, also taken from ETSI EN 300-220-1 document [10].



NOTE 1: The power limits, channel arrangement and duty cycle for FHSS equipment are given in clause 7.4.1.2; for DSSS and other non-FHSS spread spectrum equipment are given in clause 7.4.1.3.

NOTE 2: For frequency agile devices without LBT (or equivalent techniques) operating in the frequency range 863 MHz to 870 MHz, the duty cycle limit applies to the total transmission unless specifically stated otherwise (e.g. clause 7.10.3).

NOTE 3: When either a duty cycle, Listen Before Talk (LBT) or equivalent technique applies then it shall not be user dependent/adjustable and shall be guaranteed by appropriate technical means. For LBT devices without Adaptive Frequency Agility (AFA) or equivalent techniques, the duty cycle limit applies.

NOTE 4: Devices supporting audio and video applications shall use a digital modulation method with a maximum bandwidth of 300 kHz. Devices supporting analogue and/or digital voice shall have a maximum bandwidth not exceeding 25 kHz.

NOTE 5: Devices shall not support audio and/or video applications. Devices supporting voice applications shall not exceed 25 kHz bandwidth and shall use spectrum access technique such as LBT or equivalent; the transmitter shall include a power output sensor controlling the transmitter to a maximum transmit period of 1 minute for each transmission.

NOTE 6: The preferred channel spacing is 100 kHz allowing for subdivision into 50 kHz or 25 kHz.

NOTE 7: The power density can be increased to +6,2 dBm/100 kHz and -0,8 dBm/100 kHz, if the band is limited to 865 MHz to 868 MHz and 865 MHz to 870 MHz respectively.

NOTE 8: For wideband modulation other than FHSS and DSSS with a bandwidth of 200 kHz to 3 MHz, duty cycle can be increased to 1 % if the band is limited to 865 MHz to 868 MHz and power to ≤10 mW e.r.p.

NOTE 9: Duty cycle may be increased to 1 % if the band is limited to 865 MHz to 868 MHz.

Figure 38 – additional notes

In Sub-Saharan Africa countries, the unlicensed bands usually differ from one country to another, with also various constraints from one country to another.

In the WAZIUP project, the radio transmission will have to adapted to the local regulations. Therefore, range or performance of the long-range platform may vary from one country to another.

For Senegal, we will use the 863-865MHz band with transmission power limited to 10dBm (10mW) [11]. We are in the process of adapting to other African partner's country as well.

5. WAZIUP LOW-COST IOT PLATFORM

5.1. IoT platform's objectives

The WAZIUP low-cost IoT platform will provide an open-source, simple long-range communication library, generic building blocks and tools for building efficient low-cost IoT devices and gateways from “off-the-shelves” consumer-market components. The long-range communication library targets LoRa radio modules as LoRa radio technologies is chosen for the IoT communication part.

For the low-cost end-devices, we show the usage of the long-range communication library and how low-power/duty-cycled applications can be programmed.

For the low-cost gateway, WAZIUP provides a low-level radio bridge program to receive LoRa packets from end-devices and higher-level programs/tools/scripts to further process the received data. A typical simple task would be for instance to push received data from deployed sensors to public IoT cloud platforms.

Training materials such as step-by-step tutorial slides and short tutorial video sequences explain how to build the entire low-cost WAZIUP LoRa IoT ecosystem.

Regarding LoRaWAN adoption, rather than providing large-scale deployment support, WAZIUP targets small size deployment scenarios. In those kind of applications, there will be most likely a single application owner, e.g. a fish farming manager willing to get in real-time various water quality indicators in the fish ponds.

5.1.1. Deployment scenarios

Our LoRa framework focuses on easy integration of low-cost "off-the-shelves" components with simple, open programming libraries and templates for easy appropriation and customization by third-parties applications, communities and ICT companies.

In addition, WAZIUP also takes into account the fact that Internet connectivity can be quite unstable or simply impossible to get in some remote areas. Figure 39 depicts the 2 scenarios (with Internet access and without Internet access) that WAZIUP can handle.

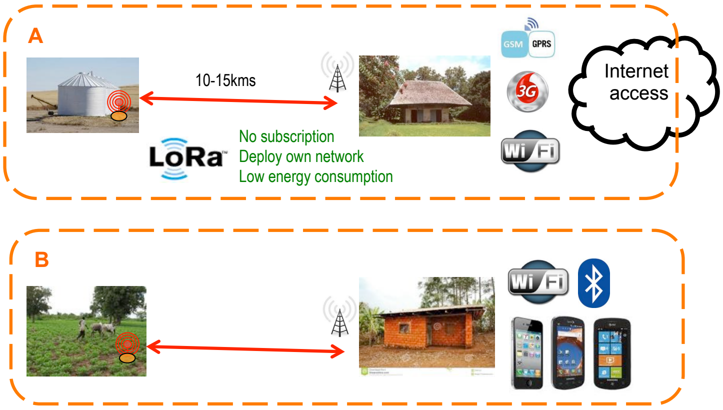


Figure 39 – Traditional Internet access scenario (top), without Internet access (bottom)

Our framework can be deployed in a fully autonomous way without the need of Internet access nor Internet servers (especially avoiding network and application servers as defined by LoRaWAN) to get the sensed data. Therefore, our framework proposes local interaction methods with the end-user using smartphones/tablet/laptop with well-known technologies such as WiFi or Bluetooth.

5.1.2. Why are we not LoRaWAN compliant?

From the orientation and design choice perspective, we already explained our choice above: while LoRaWAN is targeted at large scale deployments, WAZIUP targets small-scale deployments. Furthermore, we want to provide a level of performance and customization that is not available with LoRaWAN.

With the gateway-centric mode of LoRa LPWAN technology, commercial LoRaWAN gateways for large-scale deployment scenarios are able to listen on several channels and LoRa settings simultaneously. They typically use advanced radio concentrators chips capable of scanning up to 8 different channels: the SX1301 concentrator is used instead of the SX127x chip series that are designed for end-devices.

Our gateway uses a different approach in the context of agriculture/micro and small farm/village environments: simpler "single connection/channel" (one combination of BW, CR, SF and one frequency at a time) gateways can be built around an SX1272/76 radio module, much like an end-device would be. This design choice greatly decreases the cost and complexity of the gateway. At the same time, advanced mechanisms to limit interferences can be implemented to compensate for the "single connection/channel" limitation.

Our framework also does not follow the LoRaWAN standard because it wants to propose the possibility to add advanced and ad-hoc mechanisms such as:

- P2P (device-to-device) communications to allow for direct device cooperation schemes;
- LoRa repeaters functionality in end-devices to handle practical deployment issues when some sensing devices are in very remote areas, placed very close to the ground or obstacles;
- Advanced (and ad-hoc) channel access methods to increase transmission reliability;
- Advanced Quality of Service mechanism to handle the duty-cycle limit of sub-GHz transmission, providing some means of guaranteeing transmission latency;

More information are detailed in our FAQ:

[Doc] Low-cost LoRa IoT devices and gateway FAQ
Latest version can be downloaded from:
https://github.com/CongducPham/tutorials/blob/master/FAQ.pdf

5.2. Low-cost gateways

With the simpler "single-connection" gateway approach, a low-cost gateway can be built based on a simpler radio module, much like an end-device would be. Then, by using an embedded Linux platforms such as the Raspberry PI with high price/quality/reliability tradeoff, the cost of such gateway can be less than 45 euro.

Therefore, rather than providing large-scale deployment support, IoT platforms in developing countries need to focus on easy integration of low-cost "off-the-shelves" components with simple, open programming libraries and templates for easy appropriation and customization by third-parties. By taking an adhoc approach, complex and smarter mechanisms, such as advanced radio channel access to overcome the limitations of a low-cost gateway, can even be integrated as long as they remain transparent to the final developers. The low-cost gateway architecture will be described in Section 5.5.

5.3. Long-range communication library

The long-range communication library is based on an open-source library developed by the Libelium company for their Libelium SX1272 radio module. We enhanced it with various mechanisms for WAZIUP as explained in the next paragraphs. Note that the developed library works for end-devices and gateways, therefore simplifying maintenance, updates and future common developments.

5.3.1. Improvements to the Libelium SX1272 library

The following improvements have been performed on the initial SX1272 library:

```
*  Now, 26th, 2016
*      - add preliminary support for ToA limitation
*      - when in "production" mode, uncomment #define LIMIT_TOA
*  Now, 16th, 2016
*      - provide better power management mechanisms
*      - manage PA_BOOST and dBm setting
*  Jan, 23rd, 2016
*      - the packet format at transmission does not use the original Libelium format anymore
*      - the format is now dst(1B) ptype(1B) src(1B) seq(1B) payload(xB)
*          - ptype is decomposed in 2 parts type(4bits) flags(4bits)
*          - type can take current value of DATA=0001 and ACK=0010
*          - the flags are from left to right: ack_requested|encrypted|with_appkey|is_binary
*          - ptype can be set with setPacketType(), see constant defined in SX1272.h
*          - the header length is then 4 instead of 5
*  Jan, 16th, 2016
*       - add support for SX1276, automatic detect
*       - add LF/HF calibaration copied from LoRaMAC-Node.
*       - change various radio settings
*  Dec, 10th, 2015
*       - add SyncWord for test with simple LoRaWAN
*       - add mode 11 that have BW=125, CR=4/5, SF=7 on channel 868.1MHz
*  Nov, 13th, 2015
*       - add CarrierSense() to perform some Listen Before Talk procedure
*       - add dynamic ACK suport
*  Jun, 2015
*       - Add time on air computation and CAD features
```

Notable improvements are:

1. The support of both Arduino-based end-devices as well as Raspberry-based gateway to reduce maintenance complexity and third-party appropriation and training,
2. The support of both SX1272, 1276 and 1278 transceivers which makes the library able to drive most available SPI-based radio modules on the market,
3. a more flexible power management with selection of PA_BOOST or RFO amplifier lines to drive most available SPI-based radio modules on the market,

4. a carrier sense mechanism with customizable back-off procedure,
5. a Time on Air limitation for duty-cycle regulation enforcement,
6. additional frequency bands for Africa countries,
7. a simplification of developper's API

5.3.2. LoRa modules that have been tested

There are many SX1272/76/78-based radio modules available and we currently tested with 7: the Libelium SX1272 LoRa, the HopeRF RFM92W(SX1272) & RFM95W(SX1276), the Modtronix inAir9(SX1276) & inAir9B(SX1276) & inAir4(SX1278 or SX1276) and the NiceRF SX1276. Actually most native SPI-based LoRa modules are supported without modifications as reported by many users. In most cases, only a minimum soldering work is necessary to connect the required SPI pins of the radio (MISO, MOSI, CS, CLK) to the corresponding pins on the microcontroller board.

5.3.3. Radio regulations such as frequency bands and duty-cycle

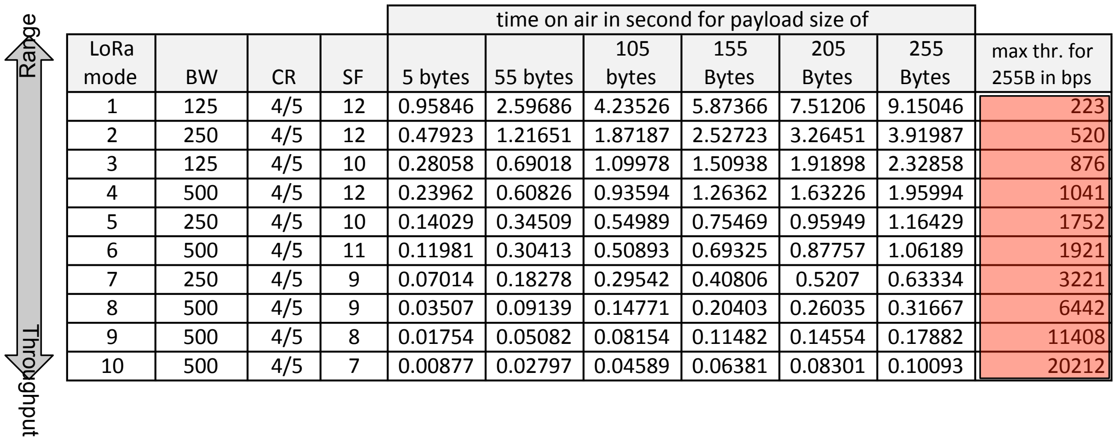
In Europe, electromagnetic transmissions in the 868MHz ISM Band used by Semtech's LoRa technology falls into the Short Range Devices (SRD) category. The ETSI EN300-220-1 and ERC/REC 70-03 documents specify various requirements for SRD devices, especially those on unlicensed frequency bands and radio activity limitation.

Regarding frequency bands, the library has 6 predefined channels (from 4 to 9) in the 863-865MHz band, 8 pre-defined channels (from 10 to 17) in the 865-868MHz band and 13 pre-defined channels (from 0 to 12) in the 903-915MHz band.

Regarding regulation under duty-cycle limitation, generally transmitters are constrained to a maximum of 0.1%, 1% or 10% every hour depending on the transmission power and the frequency band. For instance a 1% duty-cycle means 36s of radio activity time per period of 1 hour. This duty cycle limit applies to the total transmission time, even if the transmitter can change to another channel. The rationale for such constraints is to avoid saturating the radio channel as this is an unlicensed band (free for everybody to use as opposed to most of frequency bands used in commercial mobile phone networks).

5.3.4. LoRa modes

As indicated previously, the 3 main LoRa parameters are BW, CR and SF. BW and SF being the 2 most important. However you do not need to act on these parameters directly. The communication library defines 10 so-called LoRa modes (from 1 to 10) that are various combinations of BW and SF. For instance LoRa mode 1 defines BW=125kHz, CR=4/5 and SF=12. This combination provides the highest sensitivity at the receiver therefore it is suitable to achieve the longest range. However, the transmission time is the highest. Practically a real deployment can use this mode for all deployed devices to be sure to get the larger coverage. For the other modes, the range is generally decreased but transmission time is reduced. Figure 40 shows the various LoRa mode as combination of BW and SF.



LoRa mode	BW	CR	SF	time on air in second for payload size of 5 bytes	55 bytes	105 bytes	155 Bytes	205 Bytes	255 Bytes	max thr. for 255B in bps
1	125	4/5	12	0.95846	2.59686	4.23526	5.87366	7.51206	9.15046	223
2	250	4/5	12	0.47923	1.21651	1.87187	2.52723	3.26451	3.91987	520
3	125	4/5	10	0.28058	0.69018	1.09978	1.50938	1.91898	2.32858	876
4	500	4/5	12	0.23962	0.60826	0.93594	1.26362	1.63226	1.95994	1041
5	250	4/5	10	0.14029	0.34509	0.54989	0.75469	0.95949	1.16429	1752
6	500	4/5	11	0.11981	0.30413	0.50893	0.69325	0.87757	1.06189	1921
7	250	4/5	9	0.07014	0.18278	0.29542	0.40806	0.5207	0.63334	3221
8	500	4/5	9	0.03507	0.09139	0.14771	0.20403	0.26035	0.31667	6442
9	500	4/5	8	0.01754	0.05082	0.08154	0.11482	0.14554	0.17882	11408
10	500	4/5	7	0.00877	0.02797	0.04589	0.06381	0.08301	0.10093	20212

Figure 40 – LoRa mode as combination of BW and SF

5.3.5. Minimum function set to build a long-range end-device

Here is an example of a minimal setting for an end-device to actually send packets to a gateway.

```
sx1272.setMode(1);
sx1272.setChannel(CH_10_868);
sx1272.setPower('M'); // or sx1272.setPowerDBM(14);
sx1272.setNodeAddress(6);
sx1272.setPacketType(PKT_TYPE_DATA);
sx1272.sendPacketTimeout(1,"TC/18.56",8);
```

5.3.6. Packet format

The LoRa PHY layer packet format is unchanged and managed by the radio module. Then, there is a 4-byte header before the real user data. The header is organized as follows:

[DST(1B), PTYPE(4bits), FLAGS(4bits), SRC(1B), SN(1B)] [DATA(nB)]

DST is the destination address. With the gateway-centric topology, the gateway usually has address 1 so DST will most likely be 1. SRC is the source address on 1-byte [2..255]. SN is the packet sequence number. PTYPE has currently 2 values: 0001 for DATA packet and 0010 for an ACK packet. FLAGS is a 4-bit array that defines the following options:

- 1000: ack requested
- 0100: data is encrypted
- 0010: data has application key
- 0001: reserved

This 4-byte header is managed by our communication library and only the n bytes of [DATA] will be provided to the programmer. If the programmer wants to implement application key to perform further filtering, he can set the application key flag and insert in [DATA] a sequence of bytes that can be further checked at the post-processing stage. In general, one can implement its own packet format, variants or new functionalities (such as AES encryption) by defining a specific format in [DATA] and make the appropriate decoding at the post-processing stage.

5.4. Building a low-cost IoT device

As indicated previously, the availability of low-cost, open-source hardware platforms such as Arduino-like boards is clearly an opportunity for building low-cost IoT devices from consumer market components. Our long-range communication library therefore targets such Arduino-like boards: original Arduino boards (Uno, MEGA, Due, Micro, Pro Mini, Nano, M0) but also many other Arduino-compatible boards from Sparkfun, Teensy, Adafruit Feather, RFduino, Ideetron Nexus, Sodaq,... if they have compatibility with the Arduino IDE. One main issue for an easy eligible board being the availability of a built-in 3.3v pin to power the radio to avoid an extra voltage regulator. Figure 41 shows our long-range communication library long-range library supporting various Arduino boards and LoRa radio modules. All of these boards and radio modules have been successfully tested and we are continuously testing new boards and radio modules.

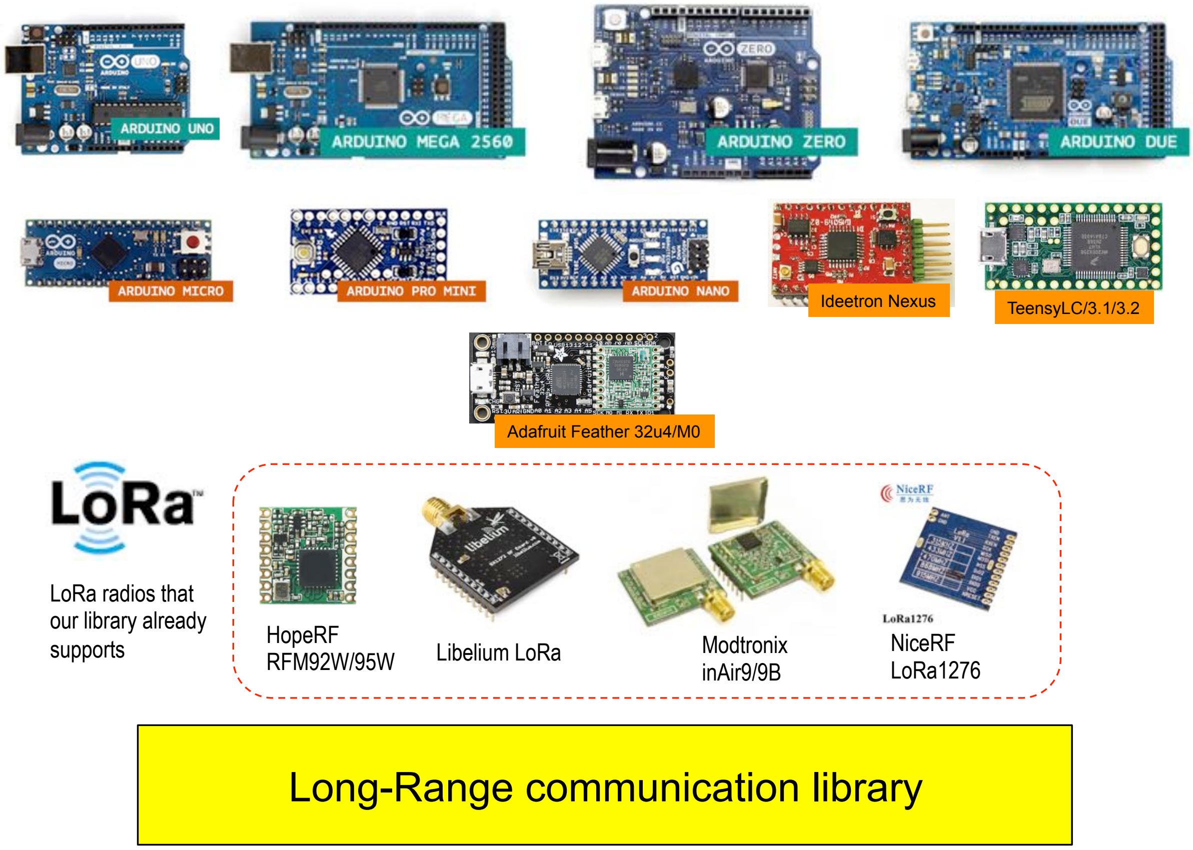


Figure 41 – The long-range library for Arduino-compatible boards and LoRa radio modules

Additionally, some radio shields from the market are actually based on the radio modules supported by WAZIUP. This is the case for the quite interesting Dragino product line which is based on the HopeRF RFM95W : LoRaBee, LoRa/GPS shield, LoRa Shield and Dragino LoRa GPS Hat shield. Figure 42 below, with images taken from Dragino, shows from left to right the aforementioned products.

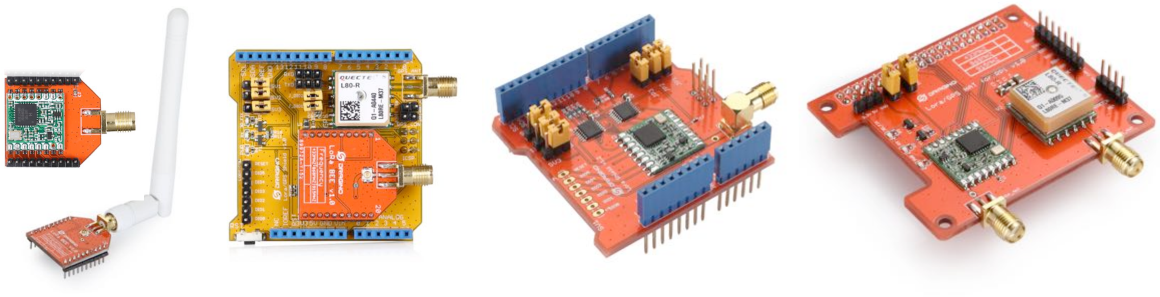


Figure 42 – Dragino LoRa product line based on HopeRF RFM95W

In the context of WAZIUP, we use **the Arduino Pro Mini in its 3.3v and 8MHz version** for simple, small-memory applications such as telemetry applications. Such Arduino Pro Mini can be purchased for about 1.5€ a piece from Chinese manufacturers. We then use **the Teensy31/3.2/LC for more power/memory demanding applications**.

5.4.1. Software integration for long-range IoT device

At the end-device, the main software components are the long-range communication library and the predefined building blocks for realizing generic tasks such as duty-cycled behaviour, physical sensor management, low-power management and encryption. Figure 43 shows the software building blocks for easy integration of long-range IoT devices.

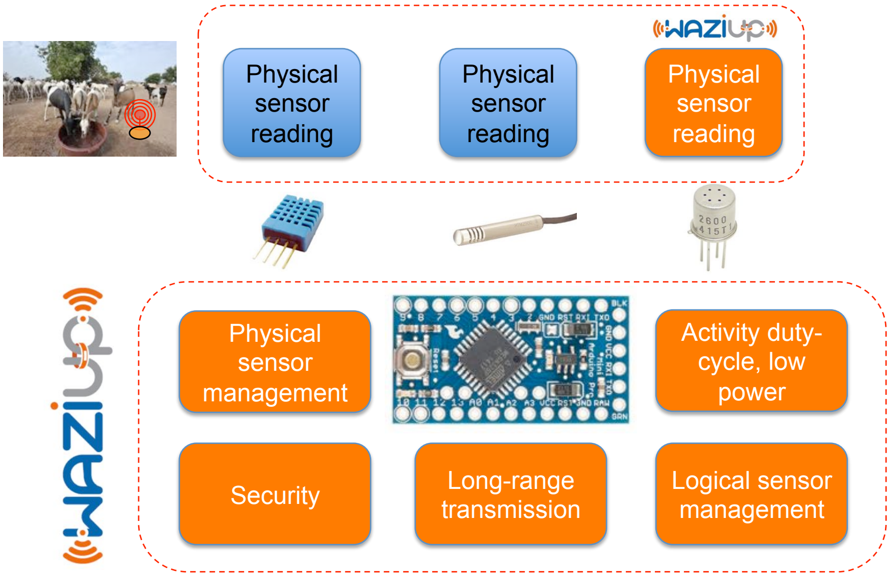


Figure 43 – Software building blocks for easy integration of long-range IoT devices

Low-power is an important feature for WAZIUP and IoT in general. One advantage of using mass-market components is also the availability of a large variety of software libraries. For the generic sensor device based on the Arduino Pro Mini or the Teensy, we use specific low-power libraries that are capable of considerably reducing the power consumption of the device by providing deep sleep or hibernate modes. The security building block is also an important feature that WAZIUP provides: AES 128-bit encryption mode is supported to provide both security and compatibility with LoRaWAN if necessary.

5.4.2. Software templates for quick and easy appropriation

We have developed a number of examples to demonstrate how simple, yet effective, low-cost LoRa IoT device can be programmed. For instance, they show how LoRa radio modules are configured and how a device can send sensed data to a gateway. They actually serve as template for future developments.

Arduino_LoRa_Ping_Pong shows a simple ping-pong communication between a LoRa device and a gateway by requesting an acknowlegment for data messages sent to the gateway.

Arduino_LoRa_Simple_temp illustrates how a simple LoRa device with temperature data can be flashed to an Arduino board. The example illustrates in a simple manner how to implement most of the features of a real IoT device: periodic sensing, transmission to gateway, duty-cycle and low-power mode to run on battery for months.

Arduino_LoRa_temp illustrates a more complex example by adding a custom Carrier Sense mechanism that you can easily modify, AES encryption and the possibility to send LoRaWAN packet. It can serve as a template for a more complex LoRa IoT device.

Arduino_LoRa_Generic_Sensor is a very generic sensor template where a large variety of new physical sensors can be added. All physical sensors must be derived from a base Sensor class (defined in Sensor.cpp and Sensor.h) and should provide a get_value() and get_nomenclature() function. All the periodic task loop with duty-cycle low-power management is already there as in previous examples : the duty-cycle building block can be configured to trigger sensor reading every M minutes. All sensors connected to the board will be polled and the returned values concatenated into a message string for transmission. Some predefined physical sensors are also already defined:

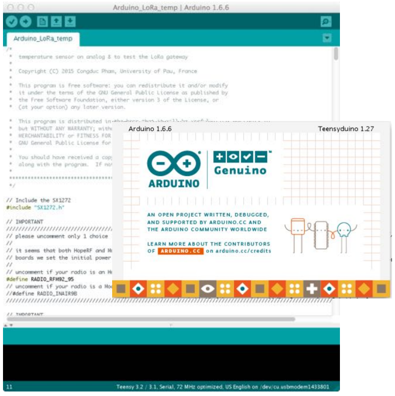
- very simple LM35DZ analog temperature sensor
- digital DHT22 temperature and humidity sensor
- digital DS18B20 temperature sensor
- ultra-sonic HC-SR04 distance sensor
- Davies Leaf Wetness sensor
- general raw analog sensor

Arduino_LoRa_InteractiveDevice is a tool that turns an Arduino board to an interactive device where a user can interactively enter data to be sent to the gateway. AES encryption and the possibility to send LoRaWAN packet is included. There are also many parameters that can dynamically be configured. This example can serve for test and debug purposes as well.

5.4.3. Programming the board

Programming and deploying the low-cost end-device is made very simple thanks to the usage of Arduino boards and Arduino IDE. These steps are described in details in the various tutorials that were developed for WAZIUP. We show here some figures to illustrate the simplicity that we want to provide to end-users or third-parties.

Getting the software

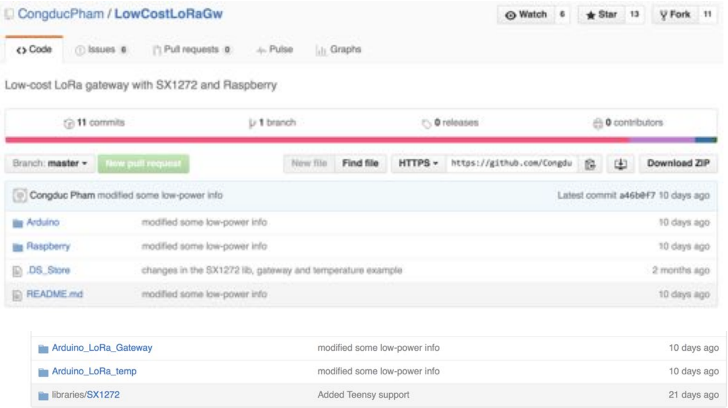


Arduino_LoRa_temp | Arduino 1.6.6

Arduino 1.6.6 Teensyduino 1.27

ARDUINO
Genuino
AN OPEN PROJECT WRITTEN, DEBUGGED, AND SUPPORTED BY ARDUINO.CC AND THE ARDUINO COMMUNITY WORLDWIDE
LEARN MORE ABOUT THE CONTRIBUTORS OF ARDUINO.CC on arduino.cc/credits

11 Teensy 3.2 / 3.1, Serial, 72 MHz optimized, US English on /dev/cu.usbmodem1433801



CongducPham / LowCostLoRaGw

Low-cost LoRa gateway with SX1272 and Raspberry

11 commits 1 branch 0 releases 0 contributors

https://github.com/Congdu

Congduc Pham modified some low-power info Latest commit a46b0f7 10 days ago

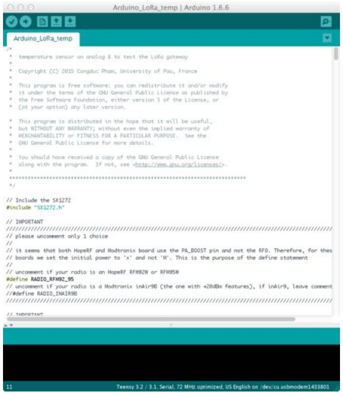
Arduino	modified some low-power info	10 days ago
Raspberry	modified some low-power info	10 days ago
.DS_Store	changes in the SX1272 lib, gateway and temperature example	2 months ago
README.md	modified some low-power info	10 days ago

Arduino_LoRa_Gateway	modified some low-power info	10 days ago
Arduino_LoRa_temp	modified some low-power info	10 days ago
libraries/SX1272	Added Teensy support	21 days ago

Fisrt, you will need the Arduino IDE 1.6.6 or later (left). Then get the LoRa library from our github: https://github.com/CongducPham/LowCostLoRaGw (right).

Get into the Arduino folder and get both Arduino_LoRa_temp and SX1272 folder. Copy Arduino_LoRa_temp into your “sketch” folder and SX1272 into “sketch/libraries”

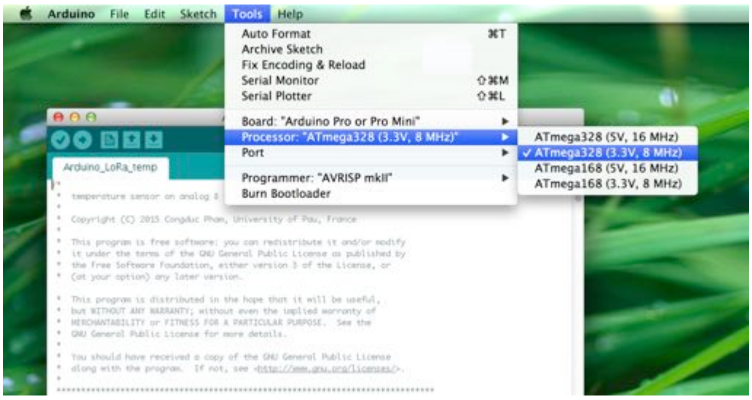
Compiling



Arduino_LoRa_temp | Arduino 1.6.6

```
/*
 *  temperature sensor on analog 8 to test the LoRa gateway
 *
 *  Copyright (C) 2015 Congduc Pham, University of Pau, France
 *
 *  This program is free software: you can redistribute it and/or modify
 *  it under the terms of the GNU General Public License as published by
 *  the Free Software Foundation, either version 3 of the License, or
 *  (at your option) any later version.

 *  This program is distributed in the hope that it will be useful,
 *  but WITHOUT ANY WARRANTY; without even the implied warranty of
 *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *  GNU General Public License for more details.
 *
 *  You should have received a copy of the GNU General Public License
 *  along with the program.  If not, see <http://www.gnu.org/licenses/>.
 *
 *****************************************************************************
 */

// Include the SX1272
#include "SX1272.h"

// IMPORTANT
///////////////////////////////////////////////////////////////////////////////////////////
// please uncomment only 1 choice
//
// it seems that both HopeRF and Modtronix board use the PA_BOOST pin and not the RFO. Therefore, for thes
// boards we set the initial power to 'x' and not 'M'. This is the purpose of the define statement
//
// uncomment if your radio is an HopeRF RFM92W or RFM95W
#define RADIO_RFM92_95
// uncomment if your radio is a Modtronix inAir9B (the one with +20dBm features), if inAir9, leave comment
//#define RADIO_INAIR9B
///////////////////////////////////////////////////////////////////////////////////////////
```

11 Teensy 3.2 / 3.1, Serial, 72 MHz optimized, US English on /dev/cu.usbmodem1433801



Board: "Arduino Pro or Pro Mini"

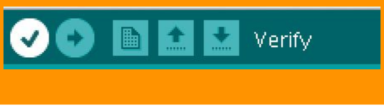
Processor: "ATmega328 (3.3V, 8 MHz)"

✓ ATmega328 (3.3V, 8 MHz)

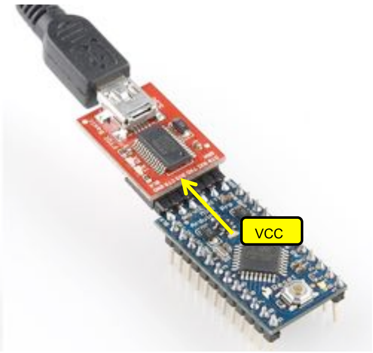
Programmer: "AVRISP mkII"

Open the Arduino_LoRa_temp sketch and select the Arduino Pro Mini board with its 3.3V & 8MHz version.

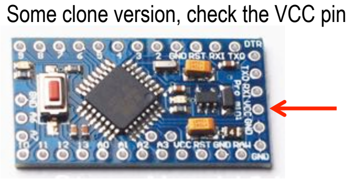
Then, click on the « verify » button

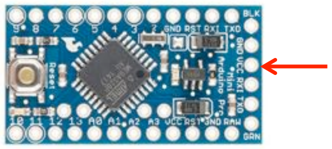


Uploading (1)



Some clone version, check the VCC pin

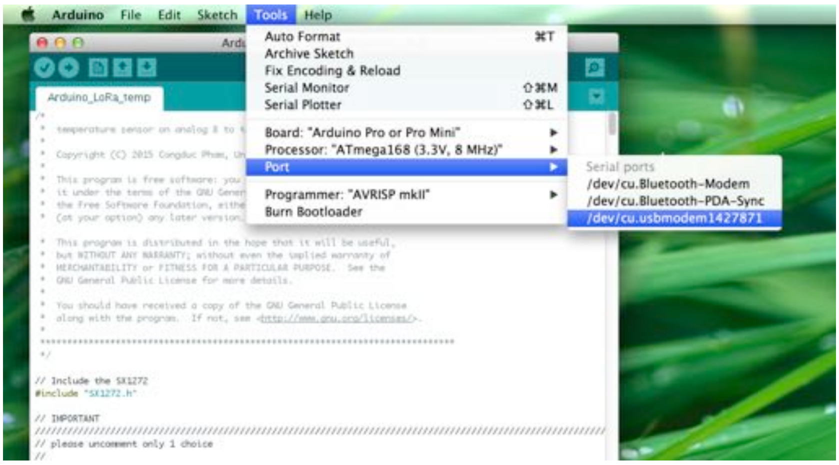




Original Sparkfun version

For the Pro Mini, you need to have an FTDI breakout cable working at 3.3v level (there is also 5v version but our advised Pro Mini version is running at 3.3v to reduce energy consumption). Be careful, on some low-cost Pro Mini version (Chinese manufacturer for instance) the pins may be in reversed order. The simplest way in to check the VCC pin and make it to correspond to the VCC pin of the FTDI breakout.

Uploading (2)

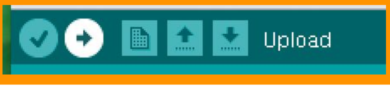


Board: "Arduino Pro or Pro Mini"
Processor: "ATmega168 (3.3V, 8 MHz)"
Programmer: "AVRISP mkII"

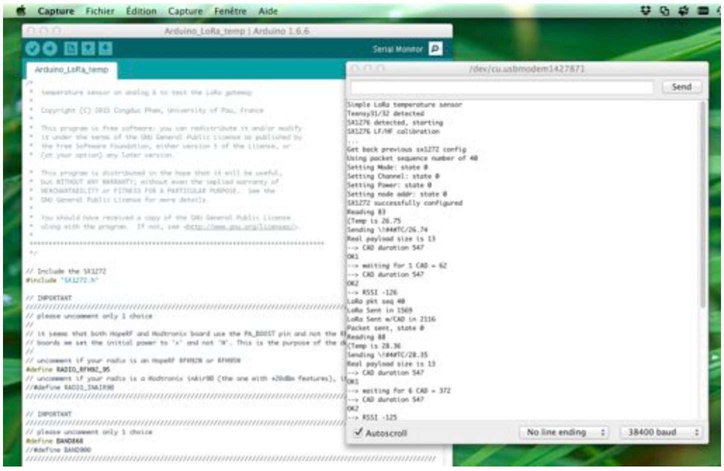
/dev/cu.Bluetooth-Modem
/dev/cu.Bluetooth-PDA-Sync
/dev/cu.usbmodem1427871

```
/*
 *  temperature sensor on analog 8 to t
 *
 *  Copyright (C) 2015 Congduc Pham, Un
 *
 *  This program is free software: you
 *  it under the terms of the GNU Gener
 *  the Free Software Foundation, eithe
 *  (at your option) any later version.
 *
 *  This program is distributed in the hope that it will be useful,
 *  but WITHOUT ANY WARRANTY; without even the implied warranty of
 *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *  GNU General Public License for more details.
 *
 *  You should have received a copy of the GNU General Public License
 *  along with the program.  If not, see <http://www.gnu.org/licenses/>.
 *
 *****************************************************************************
 */

// Include the SX1272
#include "SX1272.h"

// IMPORTANT
/////////////////////////////////////////////////////////////////////////////////////
// please uncomment only 1 choice
//
```

Connect the USB end to your computer and the USB port should be detected in the Arduino IDE. Select the serial port for your device. It may have another name than what is shown in the example. Then click on the « upload » button



Serial monitor

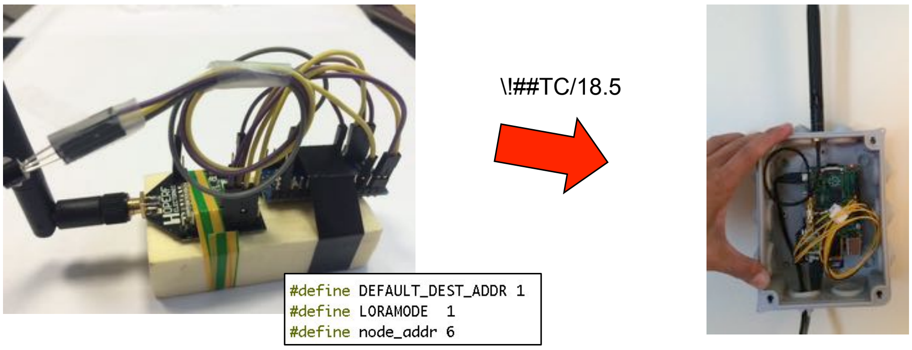


```
/*
 *  temperature sensor on analog 8 to test the LoRa gateway
 *
 *  Copyright (C) 2015 Congduc Pham, University of Pau, France
 *
 *  This program is free software: you can redistribute it and/or modify
 *  it under the terms of the GNU General Public License as published by
 *  the Free Software Foundation, either version 3 of the License, or
 *  (at your option) any later version.

 *  This program is distributed in the hope that it will be useful,
 *  but WITHOUT ANY WARRANTY; without even the implied warranty of
 *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *  GNU General Public License for more details.
 *
 *  You should have received a copy of the GNU General Public License
 *  along with the program.  If not, see <http://www.gnu.org/licenses/>.
 *
 *****************************************************************************
 */

// Include the SX1272
#include "SX1272.h"

// IMPORTANT
///////////////////////////////////////////////////////////////////////////////
// please uncomment only 1 choice
//
// it seems that both HopeRF and Modtronix board use the PA_BOOST pin and not the RF
// boards we set the initial power to 'x' and not 'M'. This is the purpose of the d
//
// uncomment if your radio is an HopeRF RFM92W or RFM95W
#define RADIO_RFM92_95
// uncomment if your radio is a Modtronix inAir9B (the one with +20dBm features), if
//#define RADIO_INAIR9B
///////////////////////////////////////////////////////////////////////////////

// IMPORTANT
///////////////////////////////////////////////////////////////////////////////
// please uncomment only 1 choice
#define BAND868
//#define BAND900
///////////////////////////////////////////////////////////////////////////////
```

```
Simple LoRa temperature sensor
Teensy31/32 detected
SX1276 detected, starting
SX1276 LF/HF calibration
...
Get back previous sx1272 config
Using packet sequence number of 40
Setting Mode: state 0
Setting Channel: state 0
Setting Power: state 0
Setting node addr: state 0
SX1272 successfully configured
Reading 83
(Temp is 26.75
Sending \!##TC/26.74
Real payload size is 13
--> CAD duration 547
OK1
--> waiting for 1 CAD = 62
--> CAD duration 547
OK2
--> RSSI -126
LoRa pkt seq 40
LoRa Sent in 1569
LoRa Sent w/CAD in 2116
Packet sent, state 0
Reading 88
(Temp is 28.36
Sending \!##TC/28.35
Real payload size is 13
--> CAD duration 547
OK1
--> waiting for 6 CAD = 372
--> CAD duration 547
OK2
--> RSSI -125
```

You can see the output from the sensor if it is connected to your computer. Use the Arduino IDE « serial monitor » to get such output, just to verify that the sensor is running fine, or to debug new code. Be sure to use 38400 baud.

Once programmed with for instance the **Arduino_LoRa_Simple_temp** template, the end-device is fully operational and start the periodic sensing and transmission. In Figure 44, the temperature is 18.5 °C. The next section will describe the low-cost gateway part.



```
#define DEFAULT_DEST_ADDR 1
#define LORAMODE  1
#define node_addr 6
```

Figure 44 – Simple temperature sensor with periodic sensing and transmission

5.4.4. Customization of templates

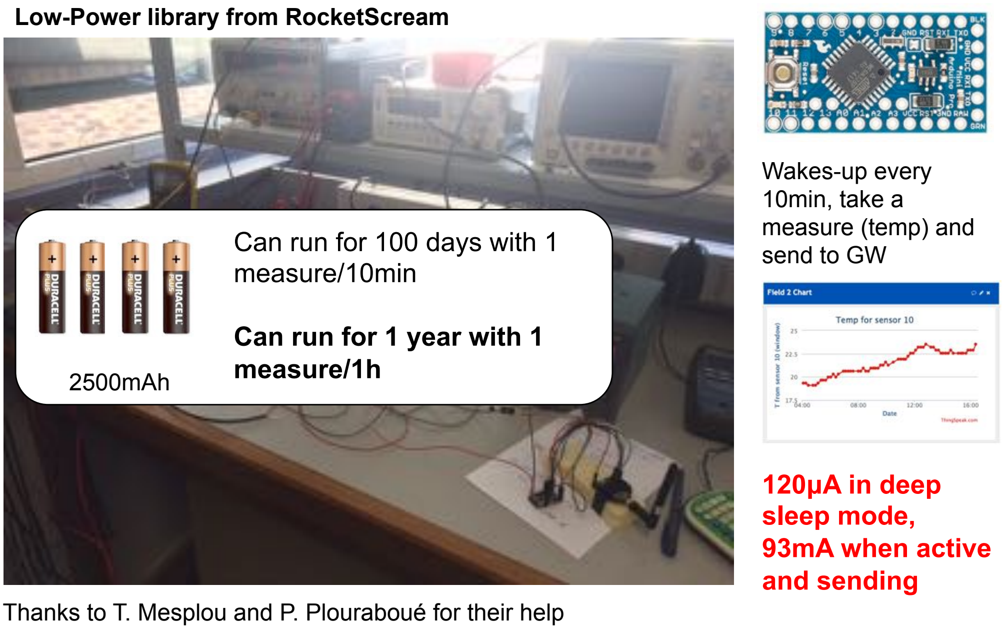
The templates can easily be customized by changing some define statements and variables that are clearly identified in the template code. We summarize below the main parts that can be customized for adaptation to use cases.

```
///////////////////////////////////////////////////////////////////
// please uncomment only 1 choice
//
#define ETSI_EUROPE_REGULATION
//#define FCC_US_REGULATION
//#define SENEGAL_REGULATION
///////////////////////////////////////////////////////////////////
```

```
///////////////////////////////////////////////////////////////////
// please uncomment only 1 choice
#define BAND868
//#define BAND900
//#define BAND433
///////////////////////////////////////////////////////////////////

///////////////////////////////////////////////////////////////////
//
// uncomment if your radio is an HopeRF RFM92W, HopeRF RFM95W, Modtronix inAir9B, NiceRF1276
// or you known from the circuit diagram that output use the PABOOST line instead of the RFO
//#define PABOOST
///////////////////////////////////////////////////////////////////

///////////////////////////////////////////////////////////////////
// COMMENT OR UNCOMMENT TO CHANGE FEATURES.
// ONLY IF YOU KNOW WHAT YOU ARE DOING!!! OTHERWISE LEAVE AS IT IS
#if not defined _VARIANT_ARDUINO_DUE_X_ && not defined __SAMD21G18A__
#define WITH_EEPROM
#endif
#define WITH_APPKEY
#define FLOAT_TEMP
//#define NEW_DATA_FIELD
#define LOW_POWER
#define LOW_POWER_HIBERNATE
//#define WITH_ACK
///////////////////////////////////////////////////////////////////

///////////////////////////////////////////////////////////////////
// CHANGE HERE THE LORA MODE, NODE ADDRESS
#define LORAMODE  1
#define node_addr 10
//////////////////////////////////////////////////////////////////

///////////////////////////////////////////////////////////////////
// CHANGE HERE THE TIME IN MINUTES BETWEEN 2 READING & TRANSMISSION
unsigned int idlePeriodInMin = 10;
///////////////////////////////////////////////////////////////////
```

5.4.5. IoT device power consumption tests

We did some power consumption measures using the **Arduino_LoRa_Simple_temp** template and the Modtronix inAi9 radio module where the transmission power is set to 14dBm. Figure 45 shows our energy consumption tests for an Arduino Pro Mini.



Low-Power library from RocketScream

2500mAh

Can run for 100 days with 1 measure/10min

Can run for 1 year with 1 measure/1h

Wakes-up every 10min, take a measure (temp) and send to GW

120µA in deep sleep mode, 93mA when active and sending

Thanks to T. Mesplou and P. Plouraboué for their help

Figure 45 – power consumption and low-power tests

Using the very low-cost Pro Mini, **energy consumption in deep sleep mode can be as low as 120uA** with very small and easy modification to the board (mainly removing the power led). **When transmitting, the power consumption jumps to 93mA**. We did the same tests for the Teensy boards and with the Teensy LC the preliminary test shows an **energy consumption of about 90uA in hibernate mode** for a board that is much more powerful than the Pro Mini.

Table 1 below summarizes the preliminary power consumption tests that we did with the Arduino Pro Mini, the Teensy LC and the Teensy31. All the tests use the Arduino_LoRa_Simple_temp template and the Modtronix inAir9 radio module. The active state is measured when the board is not transmitting.

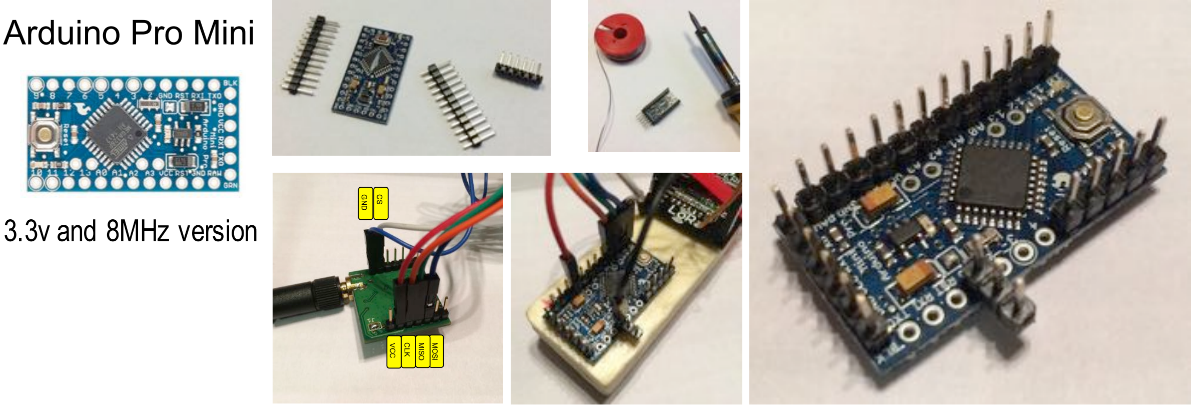
Test setting	Power consumption
Pro Mini 3.3v 8MHz	
Active	20mA
Deep Sleep	120uA
Teensy LC	
Active at 24MHz	12mA
Active at 48MHz	15mA
Hibernate	80uA-90uA
Teensy 3.1	
Active at 24MHz	20mA
Active at 48MHz	30mA
Active at 72MHz	34mA
Hibernate	115uA-120uA

Table 1 – Summary of energy consumption for Pro Mini and Teensy boards

We will describe in more details the energy requirements of the whole system in section 5.6.

5.4.6. IoT device integration for outdoor usage

With all the provided building blocks, realizing IoT device integration for outdoor usage can be done easily. Figure 46 shows a Chinese Pro Mini powered with 4 AA batteries and fitted into a water-proof case. We provide a specific tutorial for building such IoT devices.



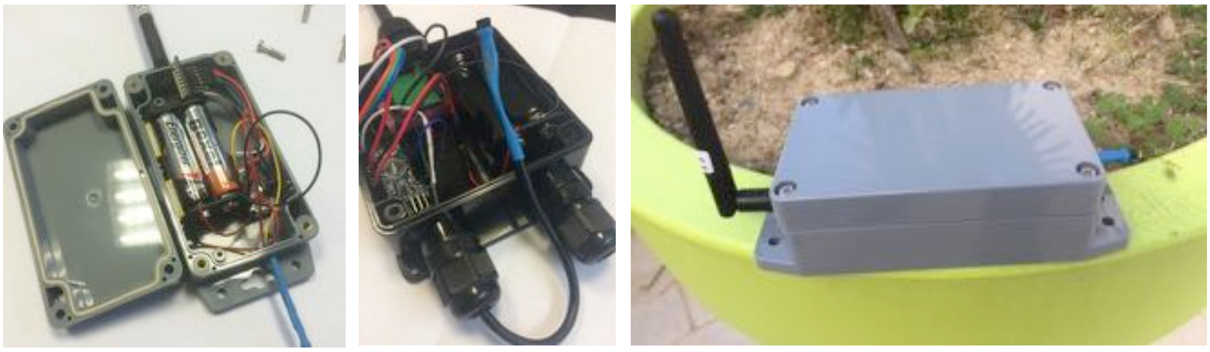


Figure 46 – Low-cost hardware integration

More information are detailed in :

[Doc] Low-cost LoRa IoT platform part list
Latest version can be downloaded from:
https://github.com/CongducPham/tutorials/blob/master/low-cost-iot-hardware-parts.pdf

[Slides] "Low-cost LoRa IoT device: a step-by-step tutorial"
Latest version can be downloaded from:
https://github.com/CongducPham/tutorials/blob/master/Low-cost-LoRa-IoT-step-by-step.pdf

[Slides] "Building IoT device for outdoor usage: a step-by-step tutorial"
Latest version can be downloaded from:
https://github.com/CongducPham/tutorials/blob/master/Low-cost-LoRa-IoT-outdoor-step-by-step.pdf

[Slides] "Low-cost LoRa IoT device: supported physical sensors"
Latest version can be downloaded from:
https://github.com/CongducPham/tutorials/blob/master/Low-cost-LoRa-IoT-supported-sensors.pdf

[Video] Build your low-cost, long-range IoT device with WAZIUP
https://www.youtube.com/watch?v=YsKbJeeav_M

5.5. Building a low-cost gateway

5.5.1. Gateway hardware and architecture

Our LoRa low-cost gateway can be qualified as "single connection" as it is built around an SX1272/76, much like an end-device would be. The low-cost gateway is based on a Raspberry PI (1A+/1B/1B+/2B/3B) which is both a low-cost and a reliable embedded Linux platform. To install the Raspberry, our pre-installed SD card image can be downloaded (on http://cpham.perso.univ-pau.fr/LORA/WAZIUP/raspberrypi-jessie-WAZIUP-demo.dmg.zip). Complete instructions to install from scratch with the Raspbian OS is also provided. When all the software components are installed, a radio module can be connected. Figure 47 shows the various steps for building our low-cost LoRa gateway.

The gateway software architecture has been designed for maximum flexibility and easy third-party appropriation and customization. In our gateway architecture we clearly want to decouple the specific lower level radio bridge program from the higher-level data post-processing stage that must be easily customized by third parties. The data post-processing block (orange block in Figure 48) is written in high-level language (e.g. Python) for simplicity and maximum customization possibilities by third-parties.

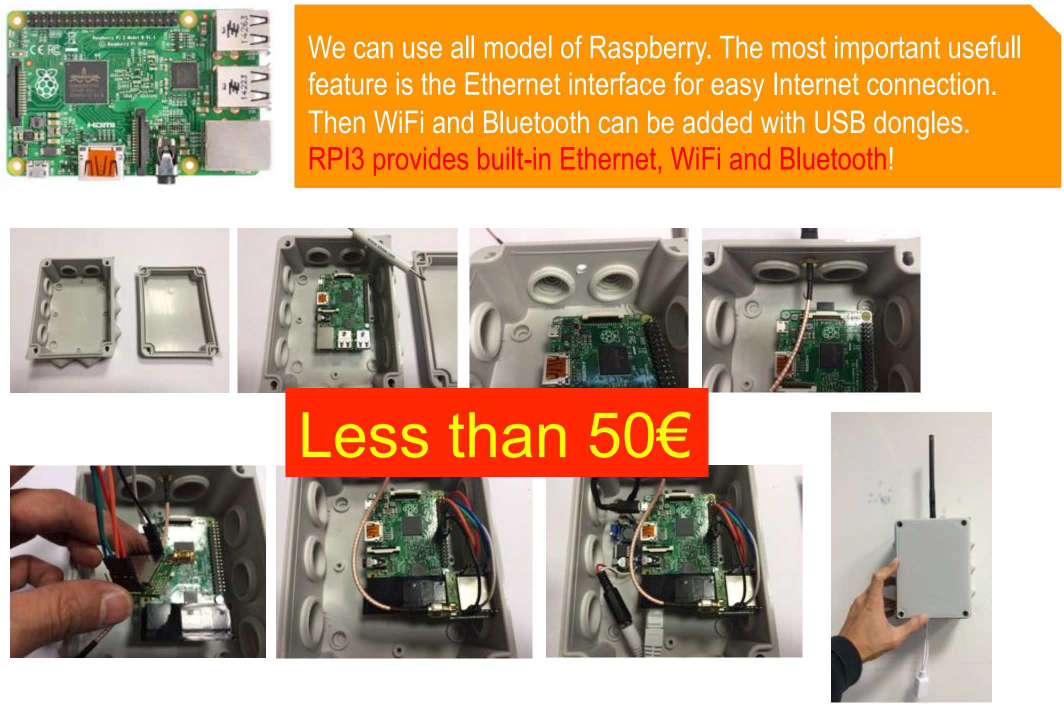

We can use all model of Raspberry. The most important usefull feature is the Ethernet interface for easy Internet connection. Then WiFi and Bluetooth can be added with USB dongles. RPI3 provides built-in Ethernet, WiFi and Bluetooth!

Figure 47 – Low-cost single channel LoRa gateway

As can be seen in the right part of Figure 48, the WAZIUP project will provide most of the gateway software logic, with the top layer being highly customizable for specific application's needs.

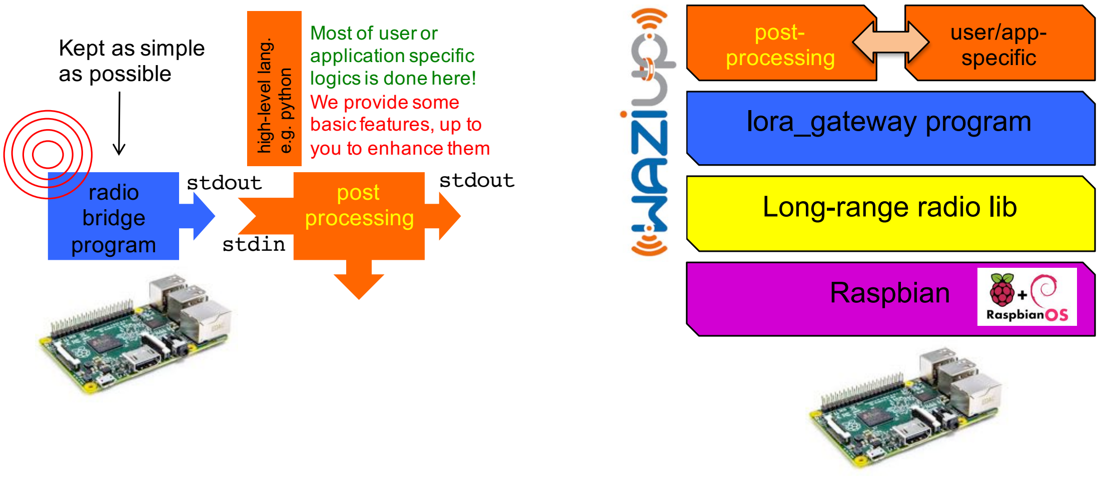

Kept as simple as possible
radio bridge program
stdout
stdin
high-level lang. e.g. python
post processing
stdout
Most of user or application specific logics is done here!
We provide some basic features, up to you to enhance them
WAZIUP
post-processing
user/app-specific
lora_gateway program
Long-range radio lib
Raspbian
RaspbianOS

Figure 48 – Gateway architecture

As can be seen, the post-processing block is the core of all incoming packet data processing tasks. We provide a Python template for the post-processing block (post_processing_gw.py) and Figure 49 shows a detailed view of the proposed post-processing template components and features. **The main component is the « incoming data parsing block » that calls user/app-specific cloud scripts**. Our post-processing template also provides additional features such as gateway temperature monitoring (with a DHT22 sensor connected to the gateway), AES encryption and decryption, management downlink data request, management of data from other radio interfaces and simple LoRaWAN interoperability. These 2 last features will be described in more details in Section 6 and Section 7 respectively.

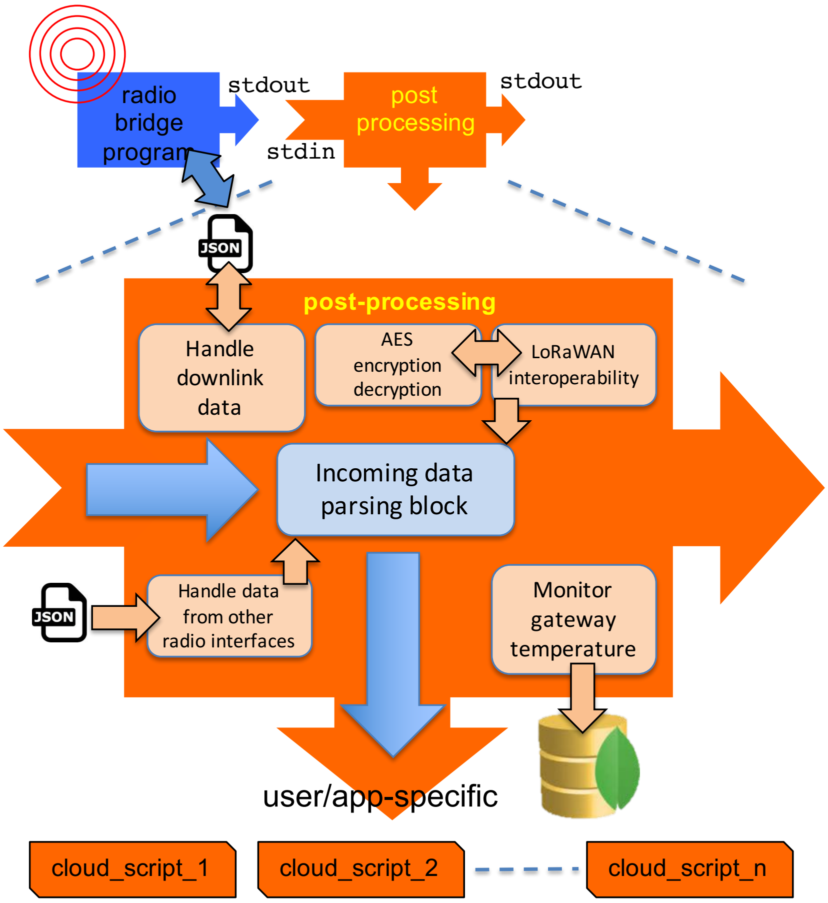


Figure 49 – post-processing block template

5.5.2. Lower level radio bridge and post-processing block interaction

As shown in Figure 49, the lower level radio bridge write formatted received data to stdout and the post-processing block will read from stdin.

For each incoming LoRa packet, the lower level radio bridge will write the 5 following formatted strings:

1. `^p`. Indicates a packet info string formatted as follows : `^pdst(%d),ptype(%d),src(%d),seq(%d),len(%d),SNR(%d),RSSI(%d)` e.g. `^p1,16,3,0,10,8,-45`
2. `^r`. Indicates a radio info string formatted as follows : `^rbw(%d),cr(%d),sf(%d)` e.g. `^r125,5,12`
3. `^t`. Indicate a timestamp info string formatted as follows : `^ttime(%s)` e.g. `^t2016-12-25T01:15:11.264700`
4. `\xFF\xFE`. 2-bytes prefix to indicate incoming packet payload
5. received packet payload

For instance if “hello” from sensor 6 is received at the lower level radio bridge, the following strings will be written to stdout, assuming LoRa mode 1 (see Figure 40 for a list of LoRa modes):

```
^p1,16,6,0,5,8,-45
^r125,5,12
^2016-12-25T01:51:11.058
\xFF\xFEhello
```

The post-processing block will read these strings to get packet info, radio info, timestamp info and finally packet payload content to take further action such as parsing the payload content and eventually upload payload content to IoT clouds. The current post-processing template would show the following output when receiving the previous strings :

```
2016-12-25T00:51:11.059762
rcv ctrl pkt info (^p): 1,16,6,0,5,8,-45
splitted in:  [1, 16, 6, 0, 5, 8, -45]
(dst=1 type=0x10(DATA) src=6 seq=0 len=5 SNR=8 RSSI=-45)
rcv ctrl radio info (^r): 125,5,12
splitted in:  [125, 5, 12]
(BW=500 CR=5 SF=12)
rcv timestamp (^t): 2016-12-25T01:51:11.058
got first framing byte
--> got data prefix
hello
```

5.5.3. What clouds for low-cost IoT?

If the gateway is connected to the Internet as shown in Figure 50, data received on the gateway are usually pushed/uploaded to some Internet/cloud servers such as the WAZIUP Cloud Platform (see WAZIUP D3.1). These tasks are handled by the post-processing stage as previously illustrated in Figure 49.

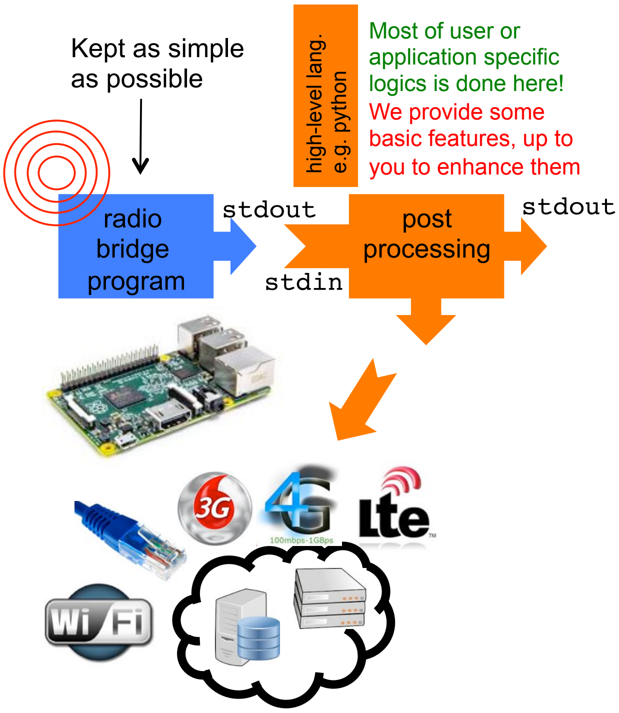

Kept as simple as possible
high-level lang. e.g. python
Most of user or application specific logics is done here!
We provide some basic features, up to you to enhance them
radio bridge program
stdout
stdin
post processing
stdout
3G
4G
Lte
Wi Fi

Figure 50 – Post-processing stage with Internet connectivity.

Additionally, it is important in the context of developing countries to be able to use a wide range of infrastructures and, if possible, at the lowest cost. Fortunately, along with the global IoT uptake, there is also a tremendous availability of sophisticated and public IoT clouds platforms and tools, offering an unprecedented level of diversity which contributes to limit dependency to proprietary infrastructures. Many of these platforms offer free accounts with limited features but that can already satisfy the needs of most agriculture/micro and small farm/village business models. It is therefore desirable to highly decouple the low-level stage gateway functionalities from the high-level stage with data post-processing features, privileging high-level languages for the latter stage (e.g. Python) so that customizing data management tasks can be done in minutes, using standard tools, simple REST API interfaces and available public clouds as depicted in Figure 51 with publicly available IoT clouds such as FireBase, ThingSpeak or GroveStreams.

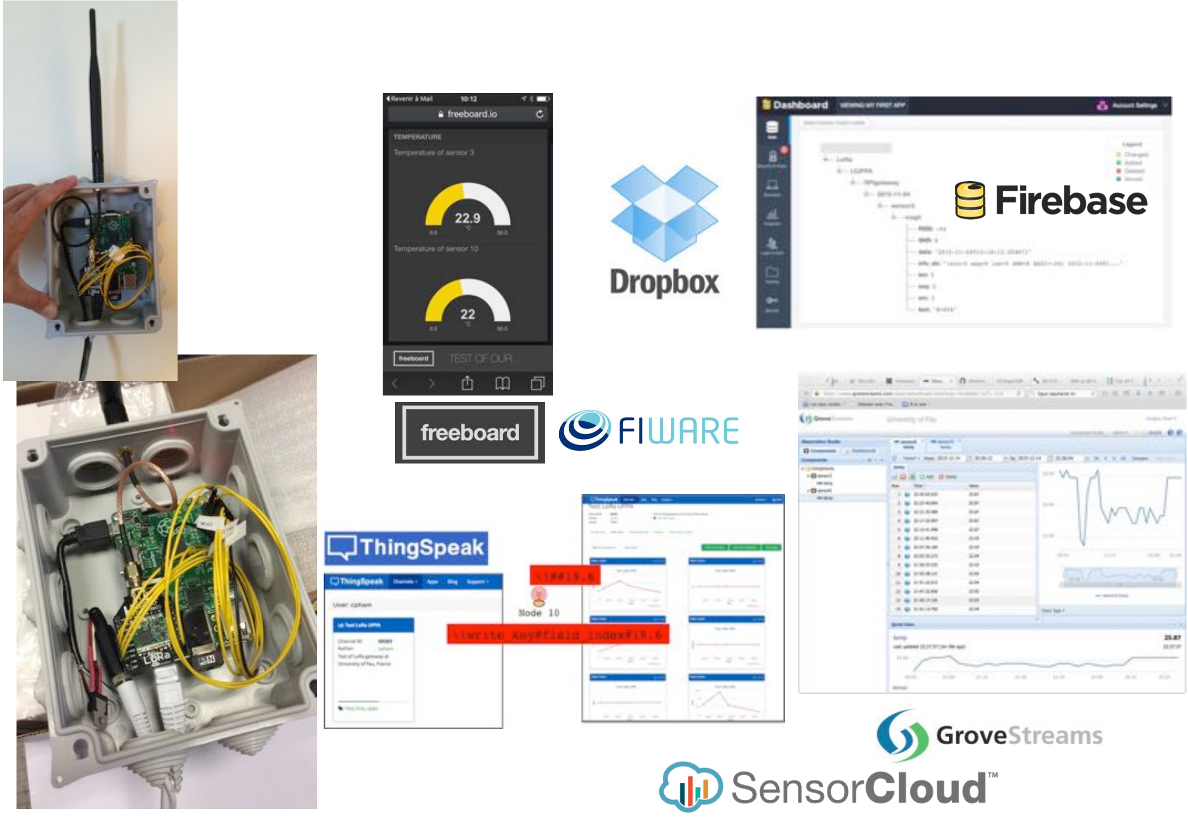


Figure 51 – From gateway to IoT clouds

5.5.4. Uploading to clouds

To indicate that a payload content should be uploaded to IoT clouds, the current post-processing block uses the « \ ! » prefix. For each IoT cloud that is enabled at the gateway, a cloud script should be provided. The cloud script will typically dissect the payload to extract relevant information and will upload the relevant information to the corresponding cloud platform using adequate commands.

The post-processing block will parse a clouds.json file that contains a list of clouds definitions for data to be uploaded. Here is an example with 3 clouds: local MongoDB, ThingSpeak and Grovestreams.

```
{
        "clouds" : [
                {
                        "notice":"do not remove"
                        "name":"Local gateway MongoDB",
                        "script":"python CloudMongoDB.py",
                        "type":"database",
                        "max_months_to_store":2,
                        "enabled":false
                },
                {
                        "name":"ThingSpeak cloud",
                        "script":"python CloudThingSpeak.py",
                        "type":"iotcloud",
                        "write_key":"",
                        "enabled":true
                },
                {
                        "name":"GroveStreams cloud",
                        "script":"python CloudGroveStreams.py",
                        "type":"iotcloud",
                        "write_key":"",
                        "enabled":true
                }
}
```

For each cloud declaration, there are only 2 relevant fields: "script" and "enabled". "script" is used to provide the name of a script. The launcher that will be used must also be indicated. Actually, "script" is more a command line than a file name. In this way, several script languages can be used (including shell scripts and binary executables provided that they read parameters that are passed by their command line). For instance, if the script is a python script, "script" should contain "python my_script_filename".

"enabled", when set to true, indicates that this cloud is active so that the post-processing block will call the associated script to perform upload of the received data. All the other fields are not relevant for the post-processing block but can be used by the associated script to get additional information that the user may want to provide through the clouds.json file.

After parsing clouds.json, the post-processing block has the list of enabled clouds as shown in the following output from the post-processing block :

```
Parsing cloud declarations
[u'python CloudThingSpeak.py']
[u'python CloudThingSpeak.py', u'python CloudGroveStreams.py']
Parsed all cloud declarations
post_processing_gw.py got cloud list:
[u'python CloudThingSpeak.py', u'python CloudGroveStreams.py']
```

Then, assuming that _enabled_clouds contains:

```
['python CloudThingSpeak.py', 'python CloudGroveStreams.py']
```

the main data upload processing loop in the post-processing block is now very simple and looks as follows:

```
#ldata will contain the payload
ldata = getAllLine()
print "number of enabled clouds is %d" % len(_enabled_clouds)

#loop over all enabled clouds to upload data
#it is up to the corresponding cloud script to handle the data format
#
for cloud_index in range(0,len(_enabled_clouds)):
        print "--> cloud[%d]" % cloud_index
        cloud_script=_enabled_clouds[cloud_index]
        print "uploading with "+cloud_script
        cmd_arg=cloud_script+" \""+ldata+"\""+" \""+pdata+"\""+
                " \""+rdata+"\""+" \""+tdata+"\""+" \""+_gwid+"\""
        os.system(cmd_arg)
print "--> cloud end"
```

For instance, we provide the CloudThingSpeak.py template script that accepts the following formatted content: « write_key#field_index#TC/18.5 ». If write_key and field_index are not specified, then the default write key and field index will be used. Therefore, an IoT device sending « \ !##TC/18.5 » as illustrated previously in Figure 44 will trigger at the gateway the execution of the CloudThingSpeak.py script that will upload « 18.5 » to the default ThingSpeak channel at the default field index. With the previous example, the execution of the main data upload processing loop will look as follows, with text in red printed by the cloud script while text in black are those printed by the main processing loop:

```
number of enabled clouds is 2
--> cloud[0]
uploading with python CloudThingSpeak.py
ThingSpeak: uploading
rcv msg to log (\!) on ThingSpeak ( default , 4 ): 18.5
ThingSpeak: will issue curl cmd
curl -s -k -X POST --data field4=18.5 https://api.thingspeak.com/[…]
ThingSpeak: returned code from server is 156
--> cloud[1]
uploading with python CloudGroveStreams.py
GroveStreams: uploading
Grovestreams: Uploading feed to: /api/feed?compId=node_6&TC=18.5
--> cloud end
```

More information are detailed in:

[Doc] Low-cost LoRa IoT platform part list
Latest version can be downloaded from:
https://github.com/CongducPham/tutorials/blob/master/low-cost-iot-hardware-parts.pdf

[Slides] "Low-cost LoRa gateway: a step-by-step tutorial"
Latest version can be downloaded from:
https://github.com/CongducPham/tutorials/blob/master/Low-cost-LoRa-GW-step-by-step.pdf

[Slides] "Low-cost LoRa IoT antenna tutorial for gateway"
Latest version can be downloaded from:
https://github.com/CongducPham/tutorials/blob/master/Low-cost-LoRa-IoT-antennaCable.pdf

[Slides] "Low-cost LoRa IoT: using the WAZIUP demo kit"
Latest version can be downloaded from:
https://github.com/CongducPham/tutorials/blob/master/Low-cost-LoRa-IoT-using-demo-kit.pdf

[Video] Build your low-cost LoRa gateway with WAZIUP
https://www.youtube.com/watch?v=peHkDhiH3lE

5.6. Energy requirements of the whole system

5.6.1. Measured energy consumption

As shown previously in Figure 45, the energy consumption of an Arduino Pro Mini in its 3.3v and 8MHz version is about 120uA in deep sleep mode. With a scenario of periodic sensing and transmission once every hour, the sensor can be active for sensing and transmission between 2.5s and 6s depending on the number of connected sensors. The transmission of the sensed data using the LoRa mode that provides the longest range (i.e., mode 1) takes about 1.5s, leaving about 1s to 4.5s for polling and reading the physical sensor. In the polling/reading state, the consumption is about 20mA. During transmission, the consumption is about 93mA. Therefore the mean consumption/hour can be estimated as:

$$E_{2.5} = [0.120 \text{ x } 3597.5 + 93 \text{ x } 1.5 + 20 \text{ x } 1] / 3600 = 0.164\text{mA}$$

$$E_{6} = [0.120 \text{ x } 3594 + 93 \text{ x } 1.5 + 20 \text{ x } 4.5] / 3600 = 0.183\text{mA}$$

In both scenarios, using traditional 2500mAh batteries, the expected lifetime can theoretically be longer than a year. We have a simple temperature sensor running on 4 AA batteries since April 6th, 2016 for real tests on autonomy. Real time data are pushed to a ThingSpeak channel : https://thingspeak.com/channels/66583, see data for 'Sensor 6'.

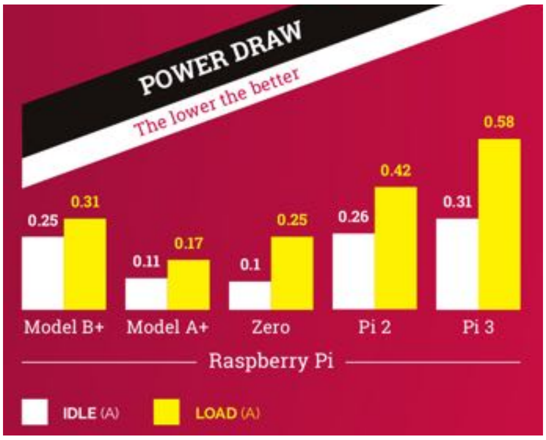


Figure 52 – Raspberry power consumption (left), 10000mAh USB charging pack (right)

At the gateway side running on a Raspberry PI, we measured the consumption to be between 100mA and 330mA depending on the Raspberry version: from version 1A+ to version 3. Our RPI3-based gateway with Internet via Ethernet and both WiFi and Bluetooth interfaces enabled consumes about 320mA in continuous reception mode. This consumption level roughly corresponds to the idle case shown in Figure 52 left (taken from https://www.raspberrypi.org/magpi/raspberry-pi-3-specs-benchmarks/) as the radio module's consumption is very small. The 'load' power consumption values shown in Figure 52 correspond to heavy CPU computation benchmarks.

The usual setting for such a gateway is to run with traditional wall-plug power supply. There are however some bigger battery packs such as those used for smartphone recharging that can be used to power the Raspberry gateway for more than 10 hours.

5.6.2. Investigating usage of solar panels

Nowadays, it is quite easy to power small electronic devices from rechargeable batteries plus solar panels. There are some mass-market integrated products such as those dedicated for on-the-go smartphone recharging. One sample is illustrated in Figure 53.

Figure 53 – Mass-market integrated USB solar charging pack

For instance, if a Raspberry A+ is used for the gateway, a 10000mAh battery can power the gateway for several hours. If the battery pack can be recharged from the Sun at the same time then the idea of a fully autonomous gateway can be reality. Such gateway can be used in a standalone manner as it will be described later on.

More elaborated and dedicated charging circuit can also be found easily to handle solar panel and rechargeable batteries (e.g. LiPo). There are also experiments addressing deployment of solar-powered LoRa gateways (such as the one reported at http://embeddedexperience.blogspot.fr/2015/05/first-lora-network-now-open.html) and these issues will be further investigated.

5.6.3. Internet connectivity with 2G/3G shield/dongle

Cellular Internet connectivity can be provided to the Raspberry gateway using either shields or dongles. Figure 54 shows a 3G shield and a USB 3G dongle that can be connected to the Raspberry gateway.

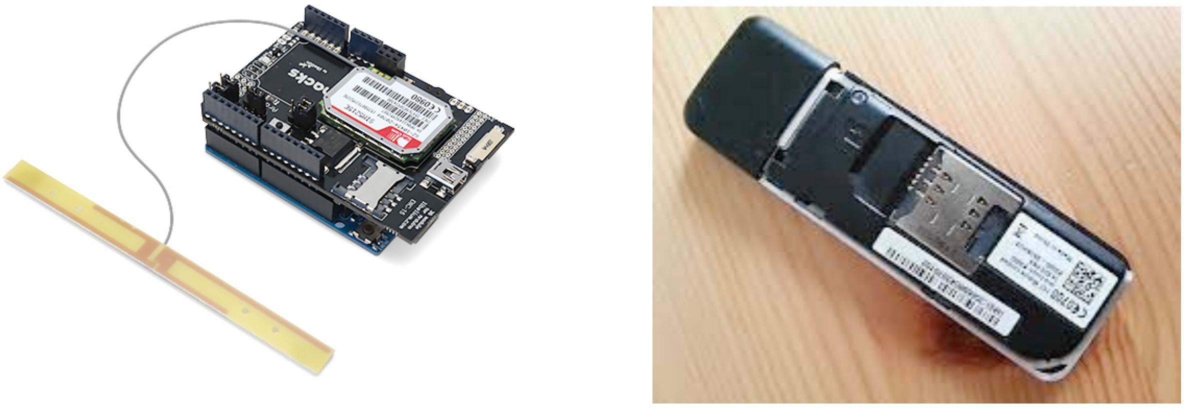


Figure 54 – 3G shield (left) and 3G dongle (right)

The energy consumption of these devices is quite high and further tests will be investigated in WAZIUP.

5.7. Running without Internet connectivity

Our gateway can also handle cases where Internet connectivity is not available as data can be locally stored on the gateway in a NoSQL MongoDB database as illustrated in Figure 55.

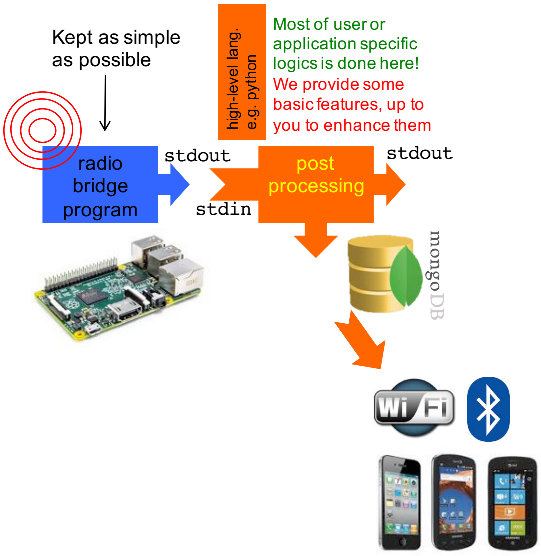
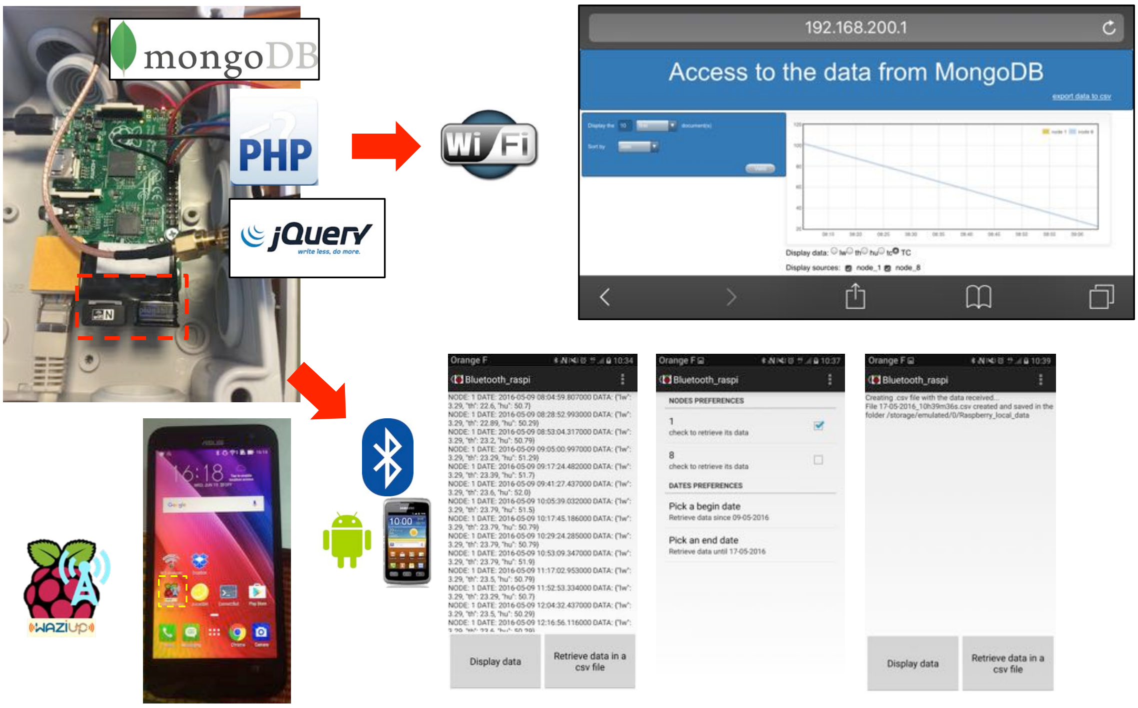

Kept as simple
as possible
high-level lang.
e.g. python
Most of user or application specific logics is done here!
We provide some basic features, up to you to enhance them
radio bridge program
stdout
stdin
post processing
stdout
mongoDB

Figure 55 – Gateway without Internet access

The gateway can either be used as an end-computer by just attaching a keyboard and a display, or it can also interact with the end-users' computing device (smartphone, tablet). Figure 56 shows the gateway's embedded web server and a specific Android smartphone application displaying the content of the local MongoDB database.



192.168.200.1

Access to the data from MongoDB

export data to csv

Display data: ○ lw ○ th ○ hu ○ tc ● TC

Display sources: ☑ node_1 ☑ node_8

Orange F 10:34

Bluetooth_raspi

```
NODE: 1 DATE: 2016-05-09 08:04:59.807000 DATA: {"lw":
3.29, "th": 22.6, "hu": 50.7}
NODE: 1 DATE: 2016-05-09 08:28:52.993000 DATA: {"lw":
3.29, "th": 22.89, "hu": 50.29}
NODE: 1 DATE: 2016-05-09 08:53:04.317000 DATA: {"lw":
3.29, "th": 23.2, "hu": 50.79}
NODE: 1 DATE: 2016-05-09 09:05:00.997000 DATA: {"lw":
3.29, "th": 23.29, "hu": 51.29}
NODE: 1 DATE: 2016-05-09 09:17:24.482000 DATA: {"lw":
3.29, "th": 23.39, "hu": 51.7}
NODE: 1 DATE: 2016-05-09 09:41:27.437000 DATA: {"lw":
3.29, "th": 23.6, "hu": 52.0}
NODE: 1 DATE: 2016-05-09 10:05:39.032000 DATA: {"lw":
3.29, "th": 23.79, "hu": 51.5}
NODE: 1 DATE: 2016-05-09 10:17:45.186000 DATA: {"lw":
3.29, "th": 23.79, "hu": 50.79}
NODE: 1 DATE: 2016-05-09 10:29:24.285000 DATA: {"lw":
3.29, "th": 23.79, "hu": 50.79}
NODE: 1 DATE: 2016-05-09 10:53:09.347000 DATA: {"lw":
3.29, "th": 23.79, "hu": 51.9}
NODE: 1 DATE: 2016-05-09 11:17:02.953000 DATA: {"lw":
3.29, "th": 23.5, "hu": 50.79}
NODE: 1 DATE: 2016-05-09 11:52:53.334000 DATA: {"lw":
3.29, "th": 23.29, "hu": 50.7}
NODE: 1 DATE: 2016-05-09 12:04:32.437000 DATA: {"lw":
3.29, "th": 23.5, "hu": 50.29}
NODE: 1 DATE: 2016-05-09 12:16:56.116000 DATA: {"lw":
3.29, "th": 23.6, "hu": 50.29}
```

Display data | Retrieve data in a csv file

Orange F 10:37

Bluetooth_raspi

NODES PREFERENCES

1
check to retrieve its data ☑

8
check to retrieve its data □

DATES PREFERENCES

Pick a begin date
Retrieve data since 09-05-2016

Pick an end date
Retrieve data until 17-05-2016

Orange F 10:39

Bluetooth_raspi

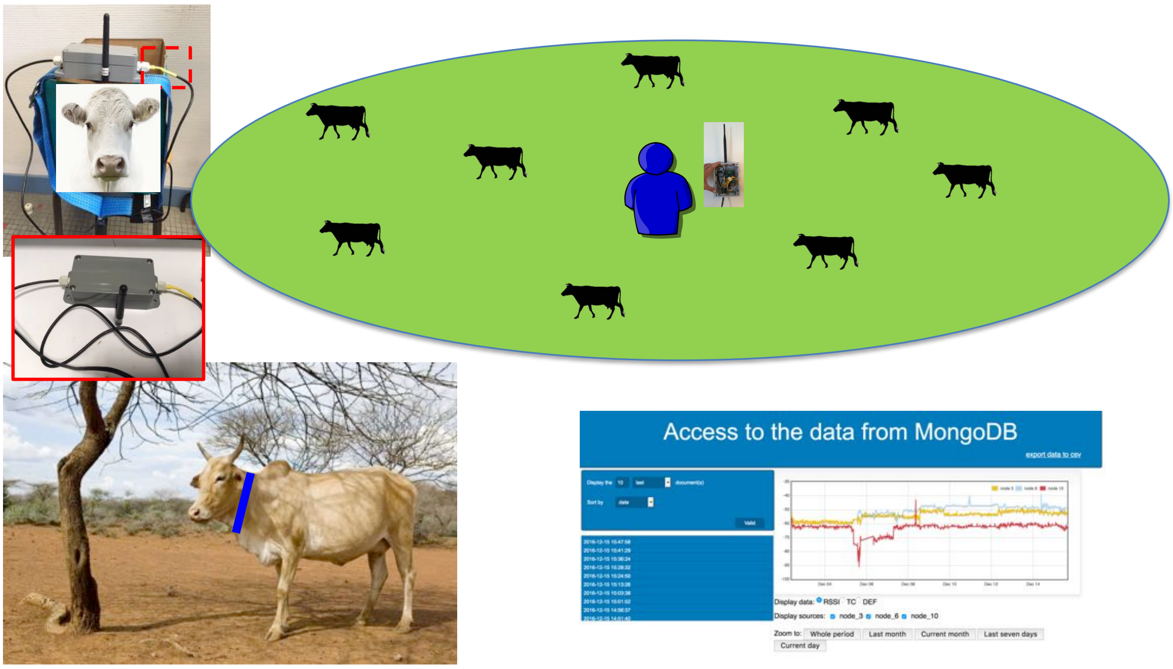
Creating .csv file with the data received...
File 17-05-2016_10h39m36s.csv created and saved in the folder /storage/emulated/0/Raspberry_local_data

Display data | Retrieve data in a csv file

Figure 56 – Autonomous gateway for no-Internet access deployment scenarios

With Raspberry 1 or 2, WiFi or Bluetooth dongles for Raspberry can be found at really low-cost but the Raspberry PI3 already comes with embedded WiFi and Bluetooth and is therefore more interesting. A smartphone can be used to display captured data and notify users about important events without the need of Internet access as this situation can clearly happen in very isolated areas.

If such a gateway is operated with a high-capacity battery (see Figure 53) that can power the gateway for about 10 hours, one can then imagine fully autonomous long-range sensing applications such as a cattle rustling application using beacon-collar devices as illustrated in Figure 57. The figure shows an autonomous gateway powered with a high-capacity battery pack/solar panel that is used by a farmer to collect beacons from collars placed on cows of the herd. The embedded web page provided by the gateway is accessed on the farmer’s smartphone and alerts can be indicated if some beacons are not received. The RSSI can also give some indications on the distance of the cows (the RSSI issues will be investigated in more detailed in the future for the Cattle Rustling Use Case).

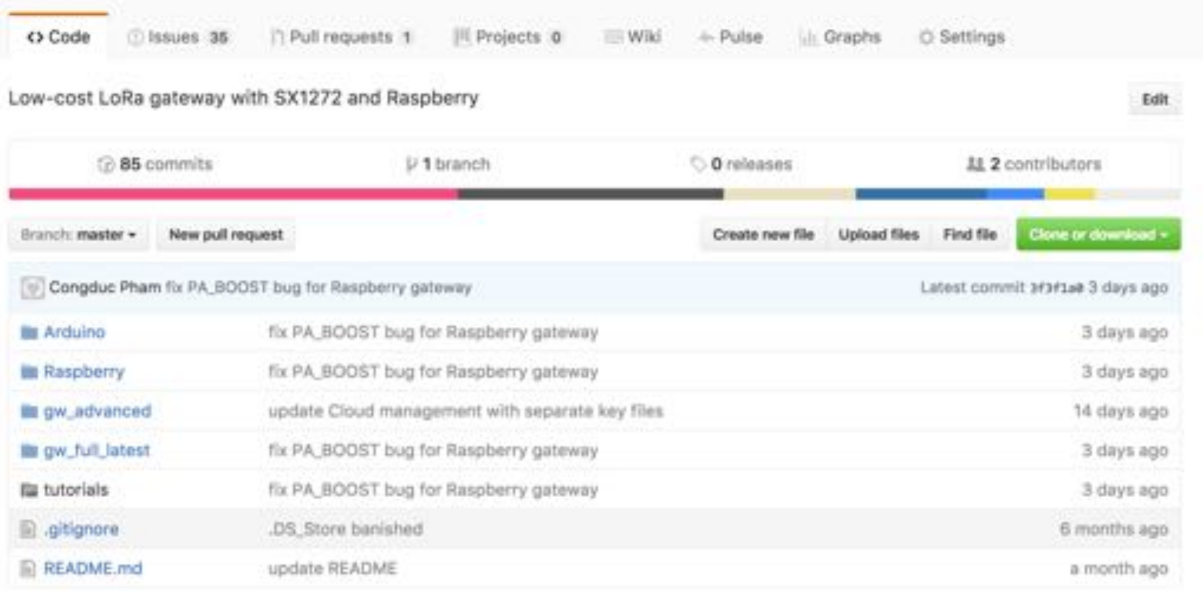

Access to the data from MongoDB
export data to csv
Display the 10 last document(s)
Sort by date
Valid
node 3
node 6
node 10
Dec 04
Dec 06
Dec 08
Dec 10
Dec 12
Dec 14
Display data: ● RSSI ○ TC ○ DEF
Display sources: ☑ node_3 ☑ node_6 ☑ node_10
Zoom to: Whole period Last month Current month Last seven days Current day

2016-12-15 15:47:58
2016-12-15 15:41:29
2016-12-15 15:36:24
2016-12-15 15:28:32
2016-12-15 15:24:50
2016-12-15 15:13:26
2016-12-15 15:03:38
2016-12-15 15:01:52
2016-12-15 14:56:37
2016-12-15 14:51:40

Figure 57 – Fully autonomous cattle rustling application

5.8. Software and tutorial materials available on the github

All the software for both IoT devices and gateway are distributed through a github repository: https://github.com/CongducPham/LowCostLoRaGw with extensive REAME files for explanations. The gateway part has already been be integrated and structured into the WAZIUP repository as the IoT Bridge component: https://github.com/Waziup/Platform/tree/master/gateway/IoTBridge. The IoT Bridge component in the WAZIUP platform is composed of a low-level part and of a high-level part which is the post-processing stage described previously in Figure 49.



Low-cost LoRa gateway with SX1272 and Raspberry

85 commits | 1 branch | 0 releases | 2 contributors

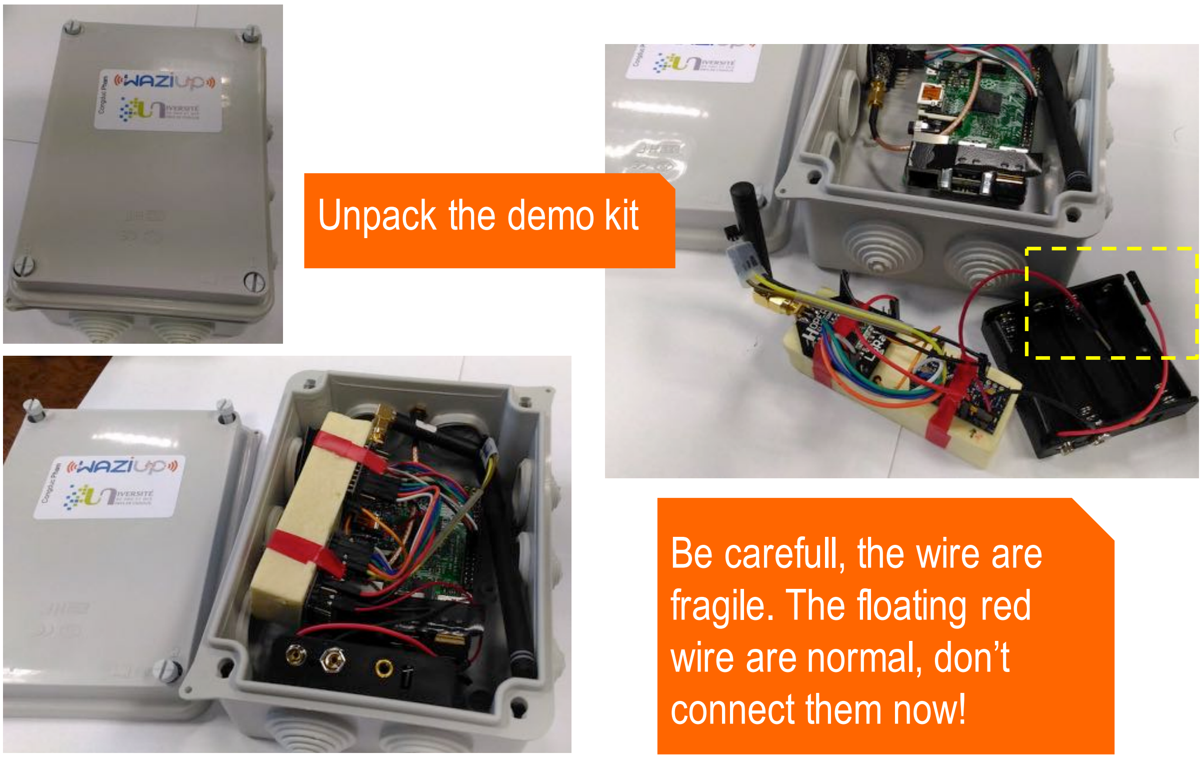
Congduc Pham fix PA_BOOST bug for Raspberry gateway — Latest commit 3f3f1a8 3 days ago

Arduino	fix PA_BOOST bug for Raspberry gateway	3 days ago
Raspberry	fix PA_BOOST bug for Raspberry gateway	3 days ago
gw_advanced	update Cloud management with separate key files	14 days ago
gw_full_latest	fix PA_BOOST bug for Raspberry gateway	3 days ago
tutorials	fix PA_BOOST bug for Raspberry gateway	3 days ago
.gitignore	.DS_Store banished	6 months ago
README.md	update README	a month ago

Figure 58 – github repository for IoT devices and gateway

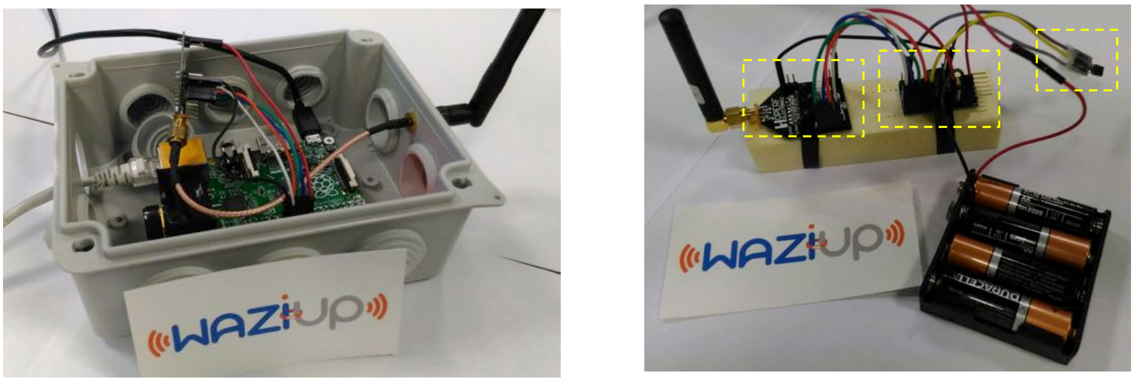
5.9. WAZIUP demo kit

As part of WP2, we also packaged a WAZIUP demo kit consisting in a low-cost gateway and a simple temperature IoT device, see Figure 59 and Figure 60. The gateway is pre-configured to upload received data to the WAZIUP LoRa ThingSpeak demo channel : https://thingspeak.com/channels/123986.


Unpack the demo kit

Be carefull, the wire are fragile. The floating red wire are normal, don't connect them now!

Figure 59 – WAZIUP LoRa demo kit



Raspberry PI with LoRa radio module and WiFi

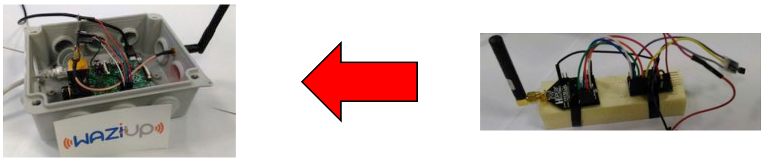
Arduino Pro Mini with LoRa radio module & temperature sensor

Figure 60 – demo gateway (left) and demo IoT device (right)

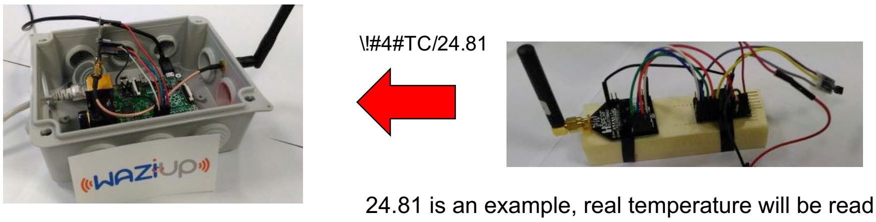
The IoT end-device runs the Arduino_LoRa_Simple_temp template and works « out-of-the-box » with the preconfigured gateway as illustrated in Figure 61. It also acts as a WiFi access-point and content of the local MongoDB database can be accessed through the embedded web server as depicted in Figure 62. **This behavior is highly appreciated in demo sessions especially when using participants' own smartphones to connect to the gateway.**

- ❑ Once switched on the end-device will
  1. Initialize the radio
  2. Take a measure (temperature)
  3. Send the measure to gateway
  4. Go to sleep for 10 minutes and repeat from step 2

Takes about 4s



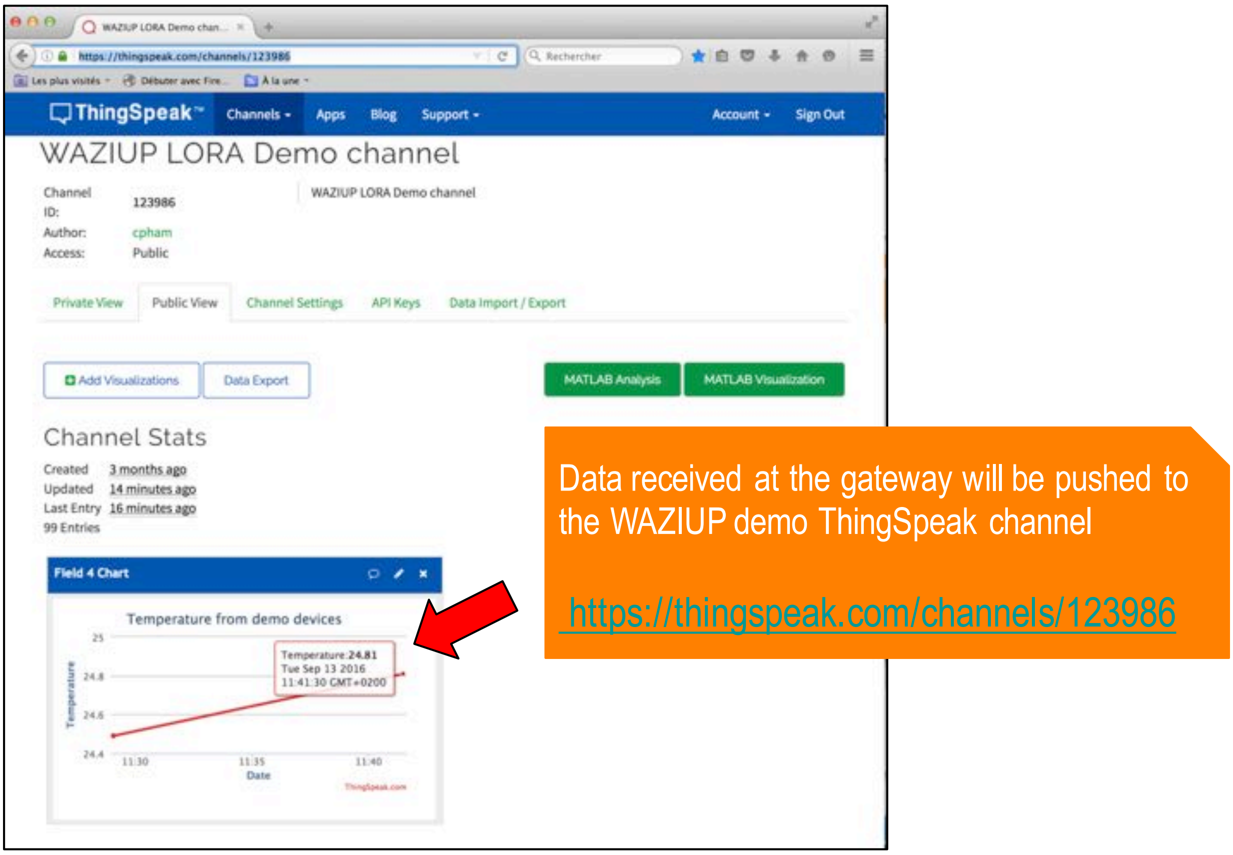
- ❑ After demonstration, just disconnect the VCC wires (red)
- ❑ No need to remove the batteries



24.81 is an example, real temperature will be read by temperature sensor

The default configuration in the Arduino_LoRa_Simple_temp example is:

Send packets to the gateway (one or many if in range)
Use LoRa mode 1
Node short address is 8



WAZIUP LORA Demo channel

Channel ID: 123986

Author: cpham

Access: Public

WAZIUP LORA Demo channel

Channel Stats

Created 3 months ago

Updated 14 minutes ago

Last Entry 16 minutes ago

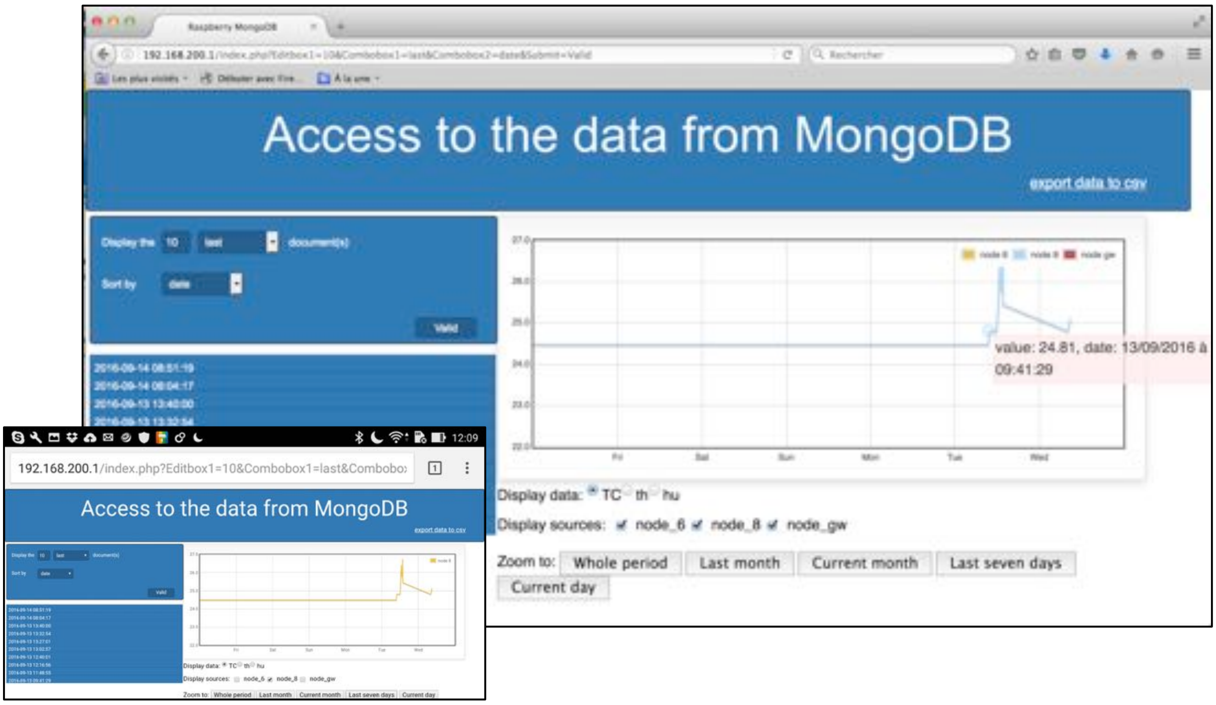
99 Entries

Temperature: 24.81
Tue Sep 13 2016
11:41:30 GMT+0200

Data received at the gateway will be pushed to the WAZIUP demo ThingSpeak channel

https://thingspeak.com/channels/123986

Figure 61 – Using the demo kit



Access to the data from MongoDB

export data to csv

Display the 10 last documents(s)

Sort by date

2016-09-14 08:51:19
2016-09-14 08:04:17
2016-09-13 13:40:00
2016-09-13 13:32:54
2016-09-13 13:27:01
2016-09-13 13:02:57
2016-09-13 12:40:01
2016-09-13 12:16:56
2016-09-13 11:48:55
2016-09-13 09:41:29

value: 24.81, date: 13/09/2016 à 09:41:29

Display data: ● TC ○ th ○ hu

Display sources: ✓ node_6 ✓ node_8 ✓ node_gw

Zoom to: Whole period | Last month | Current month | Last seven days | Current day

192.168.200.1/index.php?Editbox1=10&Combobox1=last&Combobox2=date&Submit=Valid

Figure 62 – Visualizing real-time data with the embedded web server

The demo kit has been distributed to 3 African partners for demo and training purposes during the WAZIUP mid-review meeting that took place in September 2016.

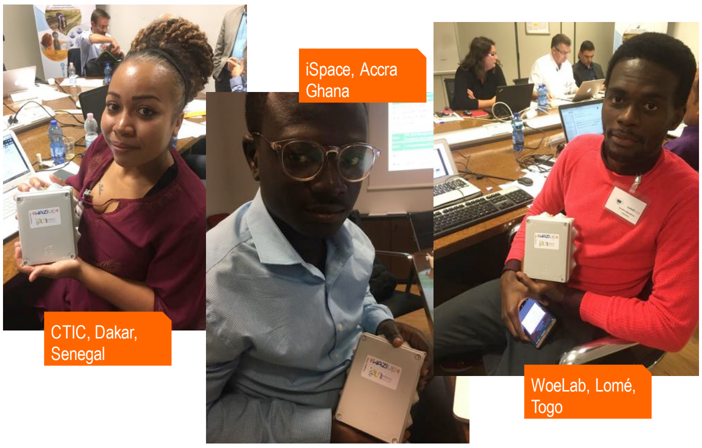


Figure 63 – distribution of the WAZIUP demo kit

More information are detailed in :

[Slides] "Low-cost LoRa IoT: using the WAZIUP demo kit"
Latest version can be downloaded from:
https://github.com/CongducPham/tutorials/blob/master/Low-cost-LoRa-IoT-using-demo-kit.pdf

5.10. Minimum Viable Products based on the IoT platform

The generic IoT device will serve for so-called Minimum Viable Product (MVP) implementation and prototyping. Depending on the application domain, dedicated physical sensors will be connected and the device's behavior adapted from the generic template as illustrated in Figure 64.

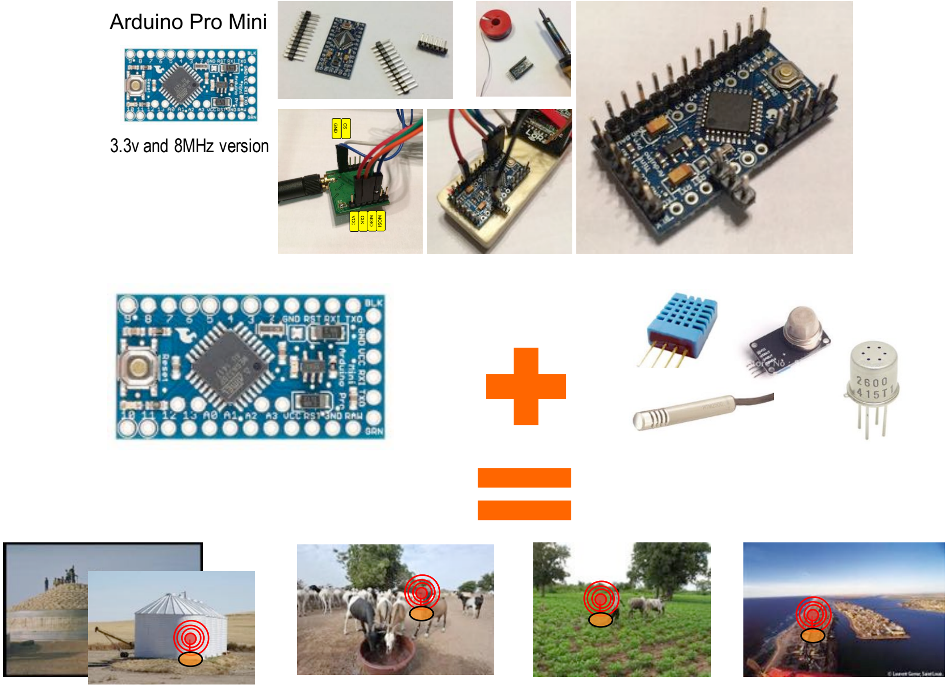
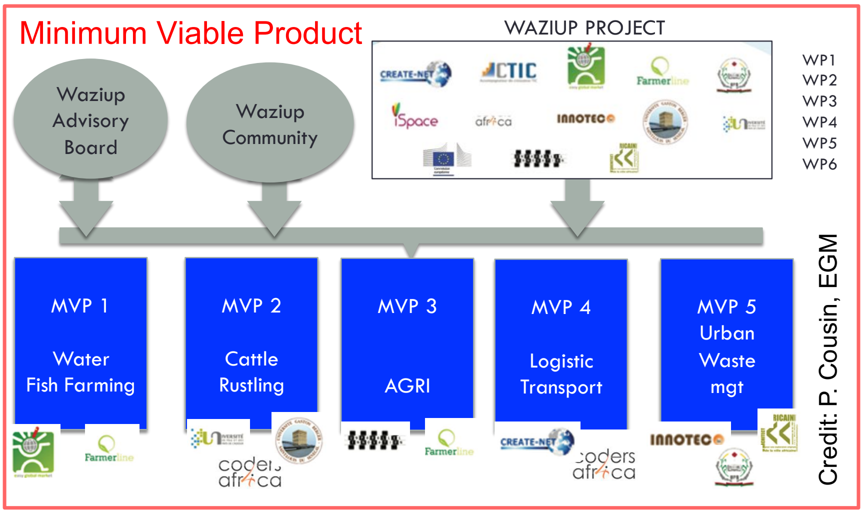


Figure 64 – Generic sensor IoT device platform

WAZIUP proposes 5 MVP in the domain of Water/Fish farming, Cattle Rustling, Agriculture, Logistic and Urban Waste Management as illustrated in Figure 65.



Minimum Viable Product

Waziup Advisory Board

Waziup Community

WAZIUP PROJECT

WP1
WP2
WP3
WP4
WP5
WP6

MVP 1 Water Fish Farming

MVP 2 Cattle Rustling

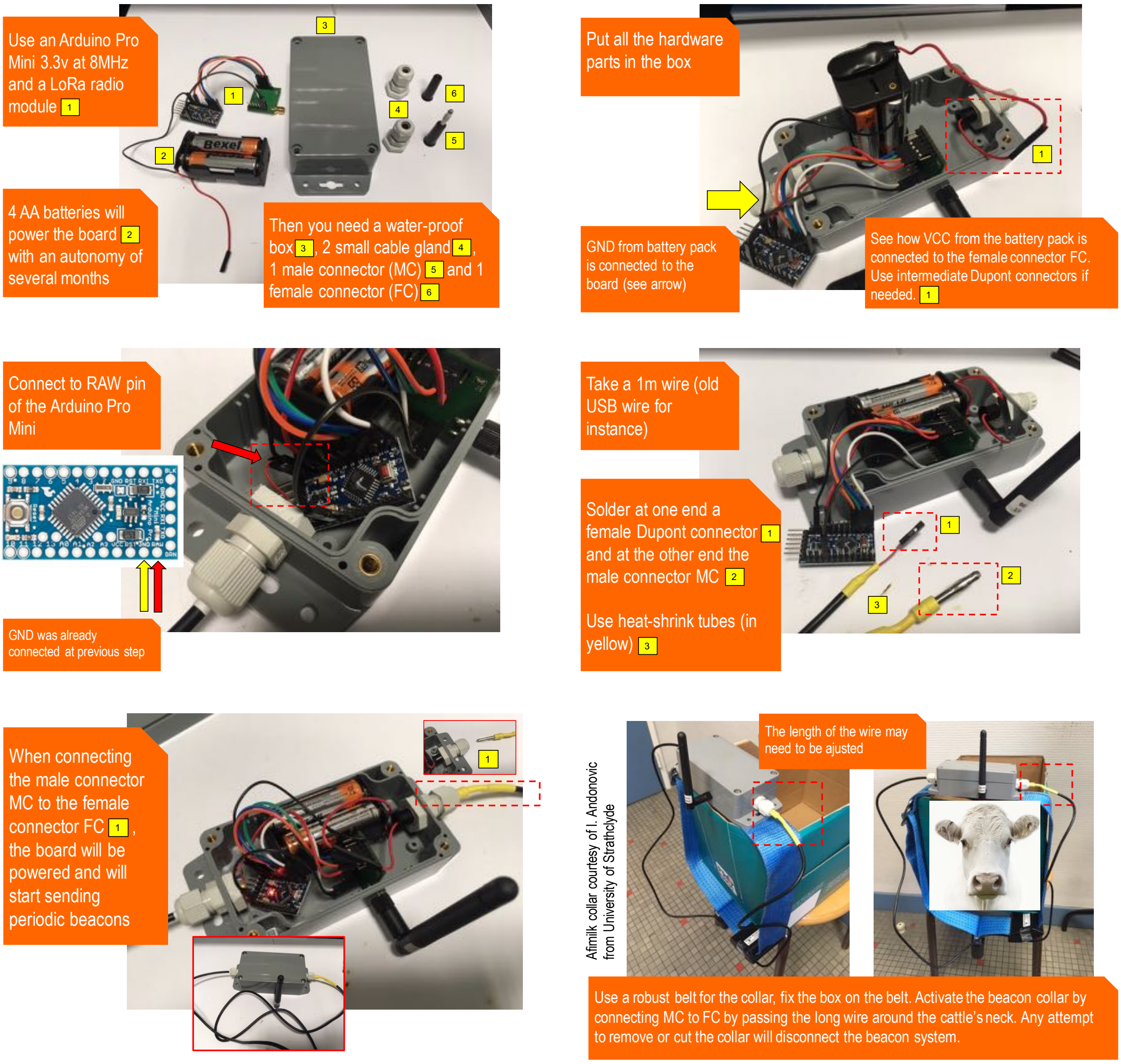
MVP 3 AGRI

MVP 4 Logistic Transport

MVP 5 Urban Waste mgt

Credit: P. Cousin, EGM

Figure 65 – Minimum Viable Products defined in WAZIUP



Use an Arduino Pro Mini 3.3v at 8MHz and a LoRa radio module 1

4 AA batteries will power the board 2 with an autonomy of several months

Then you need a water-proof box 3, 2 small cable gland 4, 1 male connector (MC) 5 and 1 female connector (FC) 6

Put all the hardware parts in the box

GND from battery pack is connected to the board (see arrow)

See how VCC from the battery pack is connected to the female connector FC. Use intermediate Dupont connectors if needed. 1

Connect to RAW pin of the Arduino Pro Mini

GND was already connected at previous step

Take a 1m wire (old USB wire for instance)

Solder at one end a female Dupont connector 1 and at the other end the male connector MC 2

Use heat-shrink tubes (in yellow) 3

When connecting the male connector MC to the female connector FC 1, the board will be powered and will start sending periodic beacons

Afimilk collar courtesy of I. Andonovic from University of Strathclyde

The length of the wire may need to be ajusted

Use a robust belt for the collar, fix the box on the belt. Activate the beacon collar by connecting MC to FC by passing the long wire around the cattle's neck. Any attempt to remove or cut the collar will disconnect the beacon system.

Figure 66 – Example of an active beacon collar system for cattle rustling applications

For instance, the Cattle Rustling MVP that proposes an active beacon system that was illustrated previously in Figure 57, adapts the Arduino_LoRa_Simple_temp template to implement the beacon system. Figure 66 shows the various steps for building the long-range collar beacon system from the generic IoT platform.

5.11. Overview of the WAZIUP platform architecture

Figure 67 illustrates the proposed IoT platform architecture. The full LPWAN architecture is represented by the top part taken from a Semtech illustration where end-nodes are sending data to concentrator/gateways that are controlled by network servers that, in turn will provide data to application servers.

WAZIUP platform can implement the full LPWAN architecture with low-cost end-devices, low-cost gateway, the WAZIUP FIWare-based data cloud and WAZIUP data analytic platform targeting a multi-users scenario. This is represented by box A in Figure 67.

However, the main objective of WAZIUP is to adapt this architecture to developing countries where it is desirable to provide simpler architectures, that take into account the issue of very small deployment scenario for single user (e.g. farmer) and the issue of Internet connectivity. For very simple deployment scenario, the Network Server stage can easily be replaced by a public & free IoT cloud account such as ThingSpeak or GroveStreams, removing the dependency on a single network server. Box B represents the « no Internet » scenario where the low-cost gateway will both store data and directly interact with the end-user with WiFi and Bluetooth devices (e.g. smartphones, tablet,…).

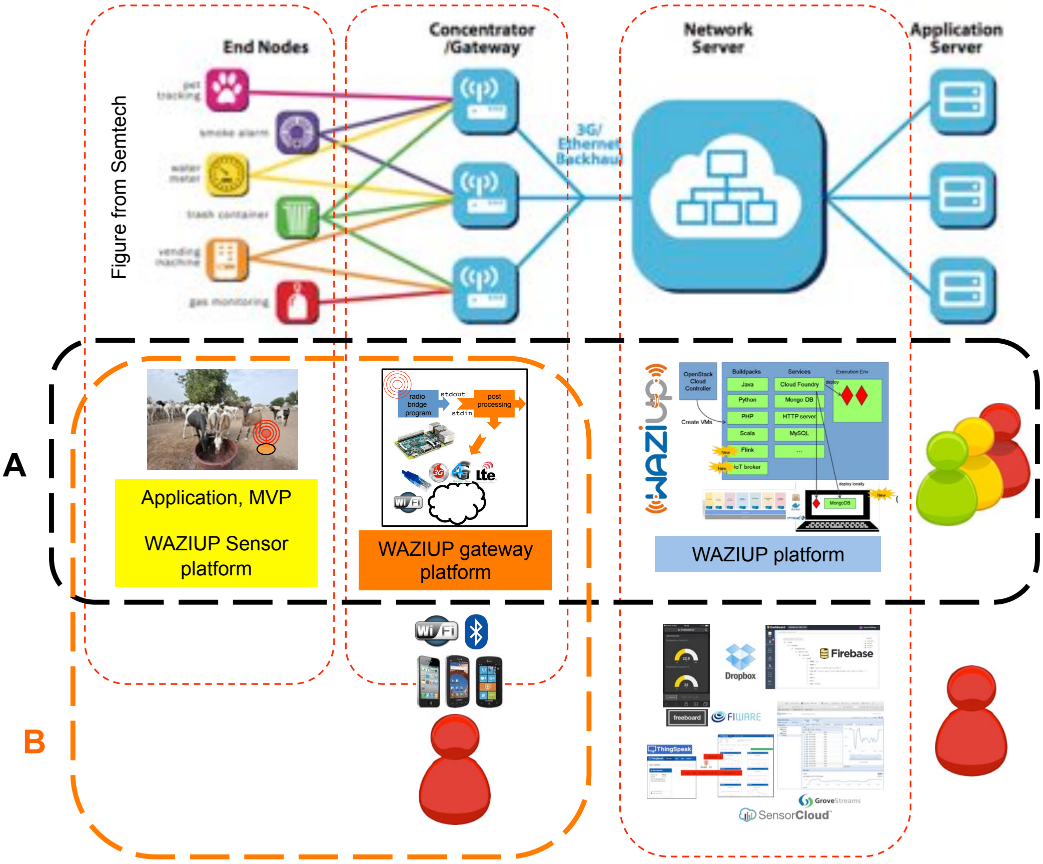


Figure 67 – IoT platform general architecture

6. Heterogeneous networking

We present in this Section how heterogeneous networking is handled in the WAZIUP IoT platform. First of all, we will not address the case where sensor nodes can have direct access to the Internet with a GPRS/3G or WiFi connection, referred to as sensor<->Internet connectivity. In these cases traditional TCP/IP protocol stack running on the sensor node will hide the lower communication layers. However, a WiFi sensor can be connected to the gateway's WiFi and therefore can not have direct access to the Internet. In this case, the proposed framework that will be described in the following subsections can be used although a traditional « Wireless-to-Wired Ethernet » routing/forwarding method is probably better (see for instance « How To: Wifi to Ethernet Bridge » at https://www.raspberrypi.org/forums/viewtopic.php?t=132674).

So, by heterogeneity we mean sensor<->gateway radio technology heterogeneity. Gateway<->Internet heterogeneity is not really an issue as this type of Internet connectivity is usually provided thought a dedicated network interface with, once again, traditional TCP/IP protocol stack that completly hides the lower communication layers.

This sensor<->gateway heterogeneity will be managed at the gateway level where data coming from other radio interfaces can be incorporated into the post-processing block as briefly illustrated previously in Figure 49. We will explain our proposed approach in the next sections.

6.1. Heterogeneity with legacy short-range radio technologies

Sensor<->gateway heterogeneity therefore mainly comes from legacy short-range radio technologies such as IEEE 802.15.4. In the last decade, before long-range and low-power LPWAN technologies became reality, short-range sensors mostly based on IEEE 802.15.4 radios were the « de facto » standard for building sensing infrastructures. IEEE 802.15.4 is used in various technologies as the PHY/MAC layer as illustrated in Figure 68.

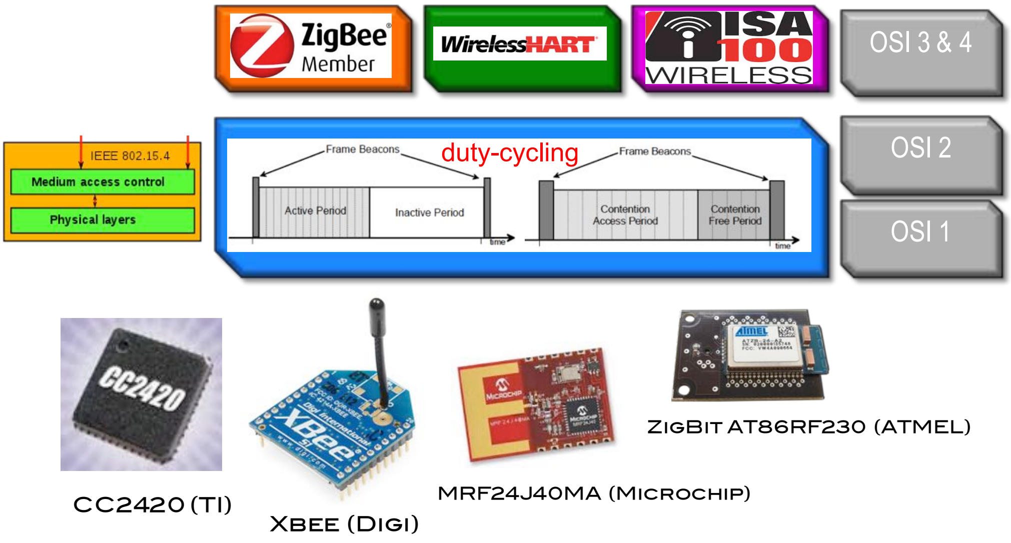


Figure 68 – IEEE 802.15.4 as PHY/MAC layer in various short-range technologies

IEEE 802.15.4 radios can be added to Arduino boards (e.g. with XBee modules) but there are also a large variety of existing integrated sensors with CC2420 radios as shown in Figure 69.

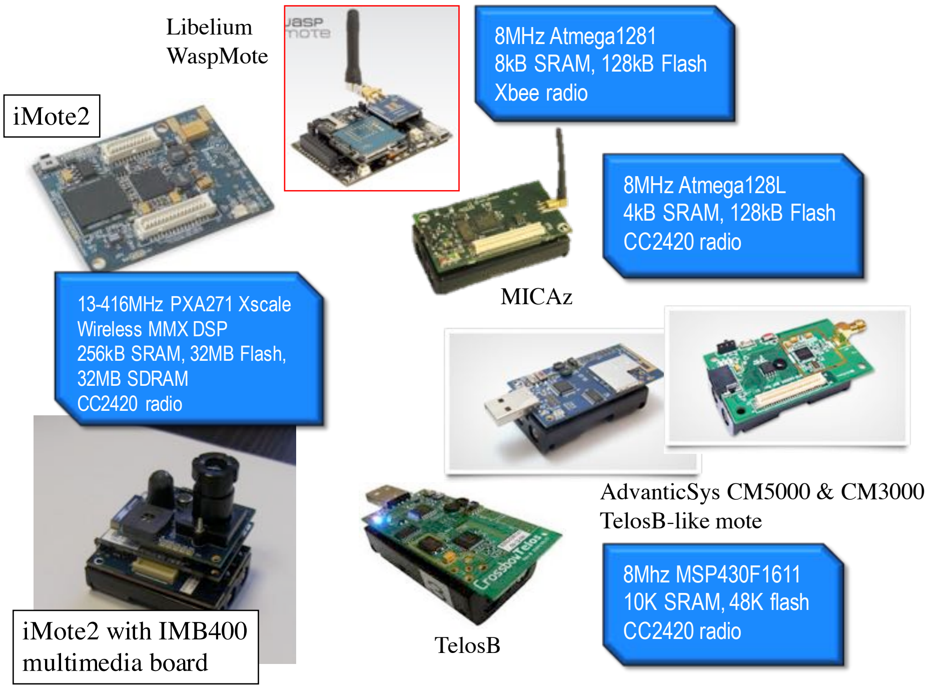


Figure 69 – Various integrated sensors with embedded IEEE 802.15.5 radios

Even if this heterogeneity issue has little importance in developing countries because the number legacy short-range infrastructures is likely to be very small, WAZIUP provides support for heterogeneous networking scenarios as shown in Figure 70 where legacy IEEE 802.15.4 sensors can be re-used to complement a long-range infrastructure. The communication pattern must be from sensors to gateway because the WAZIUP IoT platform does not provide communication between end-devices although this can be done.

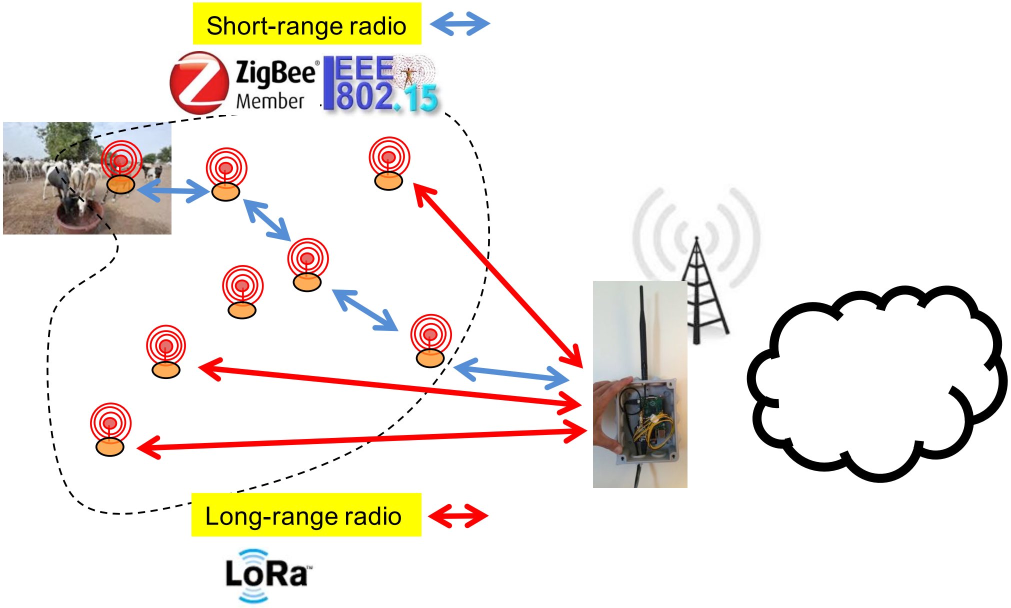


Figure 70 – Short-range (IEEE 802.15.4) for legacy sensors mixed with long-range LoRa

6.2. Taking into account data from other radio interfaces

Our approach is illustrated in Figure 71 : for each radio technology X, a USB-serial bridge is used as a transparent X gateway. Then a software component block, implementing specific packet format dissector/interpretor for technology X, will format packet information into a JSON record that will be written into a text file. The text file containing JSON records is common to all radio interface block. Our gateway will then periodically read new JSON records from the text file and inject the payload into the main incoming data parsing block, thus using all the cloud facilities already defined for long-range IoT devices.

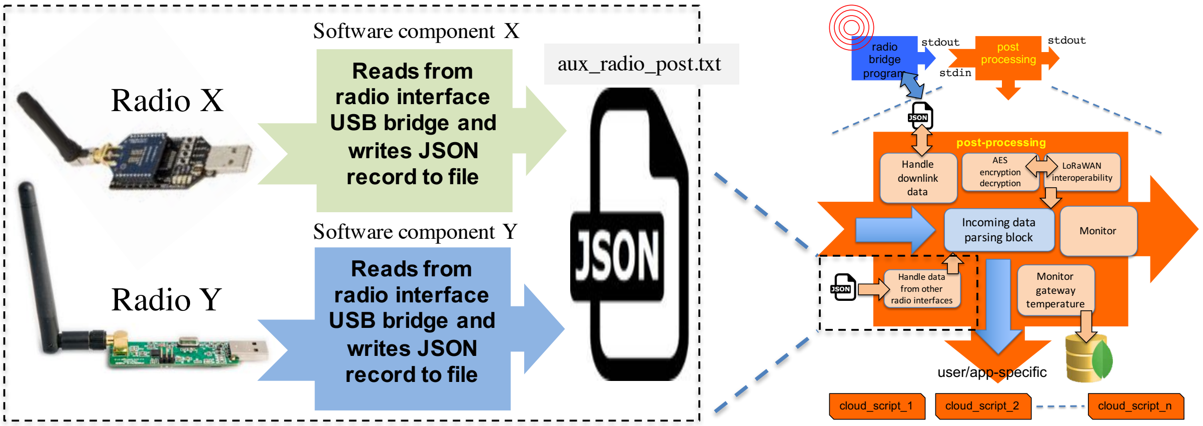


Figure 71 – General approach for heterogeneous networking

The shared text file contains lines of JSON records as shown below. There are 2 JSON records, each record is for one data packet :

```
{"^p":"^p1,16,3,0,55,8,-45","^r":"^r802154,0C,3332", \
"^t":"^t2016-10-12T09:00:20.956902","data":"\\!TC/22.5"}

{"^p":"^p1,16,6,0,128,9,-55","^r":"^r802154,0C,3332", \
"^t":"^t2016-10-12T09:01:22.453498","data":"\\!HU/85"}
```

Each JSON record must have at least 3 mandatory fields : "^p", "^t" and "data". The first 2 fields have the same meaning than previously explained : packet information and timestamp information. Note that an "^r" field can be specified to keep the same formalism than with the LoRa radio as shown in the example (e.g. "^r":"^r802154,0C,3332") to provide radio specific information. However these information will not be parsed by the post-processing block as these radio information are often very specific to the radio technology. Actually, it is possible to use another field name as this field is only for keeping track of what packet has been received on what interface.

By default, the post-processing block will periodically search for the « aux_radio/aux_radio_post.txt » file that will contain the new JSON records.

```
2016-11-08 23:13:31.476448
post aux_radio: checking for aux_radio/aux_radio_post.txt
post aux_radio: no aux_radio messages
post aux_radio: list of aux_radio messages
None
```

When new JSON records are created for incoming packets from other radio interfaces, the post-processing block will read all records from the aux_radio/aux_radio_post.txt file into memory and deletes the file.

```
2016-11-08 23:35:32.808260
post aux_radio: checking for aux_radio/aux_radio_post.txt
post aux_radio: reading aux_radio/aux_radio_post.txt
{u'^p': u'^p1,16,3,0,55,8,-45', u'data': u'\\!TC/22.5', u'^r':
u'^r802154,0C,3332', u'^t': u'^t2016-10-12T09:00:20.956902'}
post aux_radio: list of aux_radio messages
{"^p":"^p1,16,3,0,55,8,-45","^r":"^r802154,0C,3332","^t":"^t2016-10-
12T09:00:20.956902","data":"\\!TC/22.5"}
```

Then, when the gateway is idle from processing LoRa packets, it will start processing pending data packets from the other radio interfaces :

```
post aux_radio: process aux_radio message
{"^p":"^p1,16,3,0,55,8,-45","^r":"^r802154,0C,3332","^t":"^t2016-10-
12T09:00:20.956902","data":"\\!TC/22.5"}

post aux_radio: inserting in input buffer
^p1,16,3,0,55,8,-45
^t2016-10-12T09:00:20.956902
\!TC/22.5

2016-11-08T23:36:49.820980
rcv ctrl pkt info (^p): 1,16,3,0,55,8,-45
splitted in:  [1, 16, 3, 0, 55, 8, -45]
(dst=1 type=0x10(DATA) src=3 seq=0 len=55 SNR=8 RSSI=-45)
rcv timestamp (^t): 2016-10-12T09:00:20.956902
number of enabled clouds is 1
--> cloud[0]
uploading with python CloudThingSpeak.py
ThingSpeak: uploading
rcv msg to log (\!) on ThingSpeak ( default , 4 ): 22.5
ThingSpeak: will issue curl cmd
curl -s -k -X POST --data field4=22.5 https://api.thingspeak.com/[…]
ThingSpeak: returned code from server is 156
--> cloud end
```

As can be seen, when a pending data packet is processed, the post-processing block shows the entire JSON record (text in red). When the output of the post-processing block is logged into a file then one can trace what packet has been received on what radio interface and when it has been processed.

6.2.1. Use case with IEEE 802.15.4

In order to receive IEEE 802.15.4 packets, we use an USB-serial bridge where an XBee802 radio module can be plugged in, see Figure 72. With the USB-serial bridge, an IEEE 802.15.4 packet that is received by the radio module will be written to a serial port (e.g. /dev/tty/USB0 for instance). Our software component for the IEEE 802.15.4 USB-serial bridge is written in Python and periodically polls for data on the serial interface. In case of an incoming packet, it will create a JSON record with the required field (e.g. "^p", "^t" and "data") and, if necessary, optional fields such as "^r":"^r802154,0C,3332" to indicate that the packet comes from an « 802154 » radio, listening on channel 0x0C with PANID 0x3332.

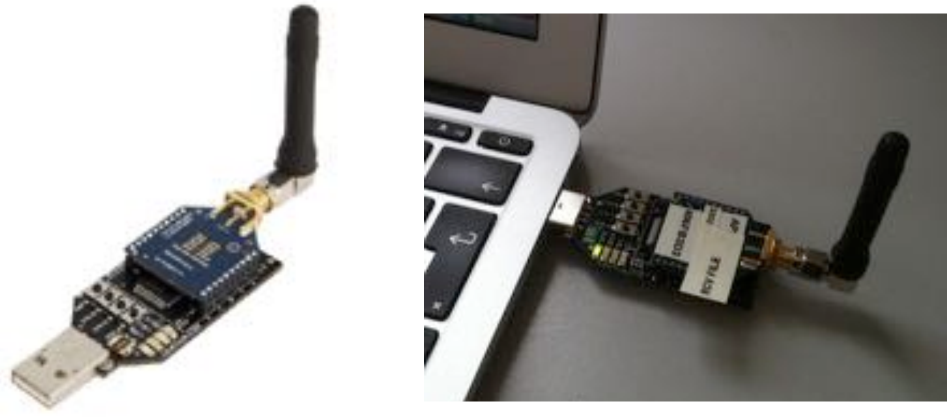


Figure 72 – XBee serial gateway

Each software component X for a radio X appends their JSON records into the aux_radio/aux_radio_post.txt file.

6.2.2. Generalization to other type of interfaces

Figure 71 showed the general approach based on radio bridge and dedicated software components blocks. Therefore for a given radio technology, a USB-serial bridge is usually required. In many cases, when a dedicated USB dongle is not available, it is possible to program a microcontroller sensor node to act as a transparent USB-serial bridge. Then, the software component has to be written or adapted to format incoming data into JSON records as explained previously.

For widely used radio technologies such as WiFi or Bluetooth where the network interface can be embedded in to the gateway (for instance the Raspberry 3 has built-in WiFi and Bluetooth), the same approach can be implemented: a software component can be written or adapted to format incoming data on the WiFi or Bluetooth interface into JSON records as explained previously. In these cases, it is most likely that high-level libraries (e.g. Python libraryies) can be used to handle all the connection issues in order to get the packet payload.

7. Tests with LoRaWAN and minimum interoperability

7.1. LoRaWAN packet format and requirements

7.1.1. LoRaWAN packet format

A brief description of LoRaWAN was provided in Section 4.3.2. We describe here more details of the LoRaWAN packet format as it is the core support for a minimum interoperability level. Taken from the LoRaWAN specification document [6], the LoRaWAN packet format is illustrated in Figure 73.

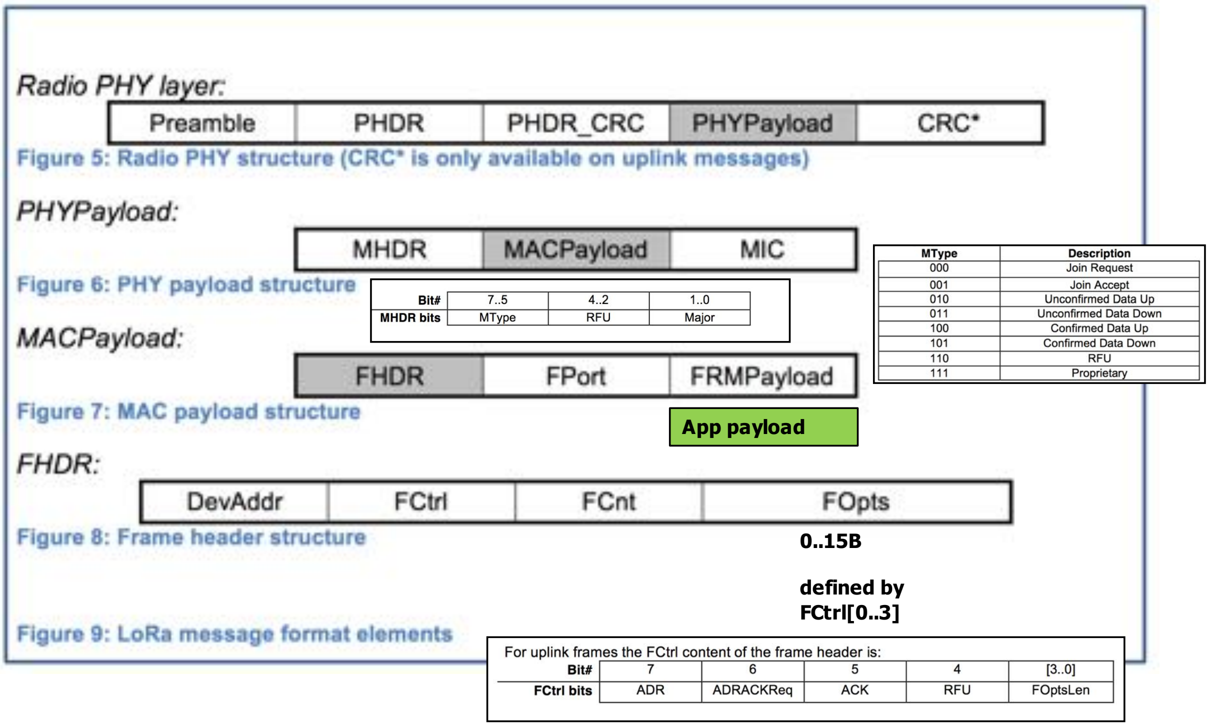


Figure 5: Radio PHY structure (CRC* is only available on uplink messages)

Figure 6: PHY payload structure

Bit#	7..5	4..2	1..0
MHDR bits	MType	RFU	Major

MType	Description
000	Join Request
001	Join Accept
010	Unconfirmed Data Up
011	Unconfirmed Data Down
100	Confirmed Data Up
101	Confirmed Data Down
110	RFU
111	Proprietary

Figure 7: MAC payload structure

Figure 8: Frame header structure

Figure 9: LoRa message format elements

For uplink frames the FCtrl content of the frame header is:

Bit#	7	6	5	4	[3..0]
FCtrl bits	ADR	ADRACKReq	ACK	RFU	FOptsLen

Figure 73 – LoRaWAN packet format

A LoRaWAN packet, as our own LoRa packet format (see Section 5.3.6) starts at the PHYPayload and consists of MHDR, FHDR, DevAddr, FCtrl, FCnt, FOpts, FPort, encrypted application payload and MIC.

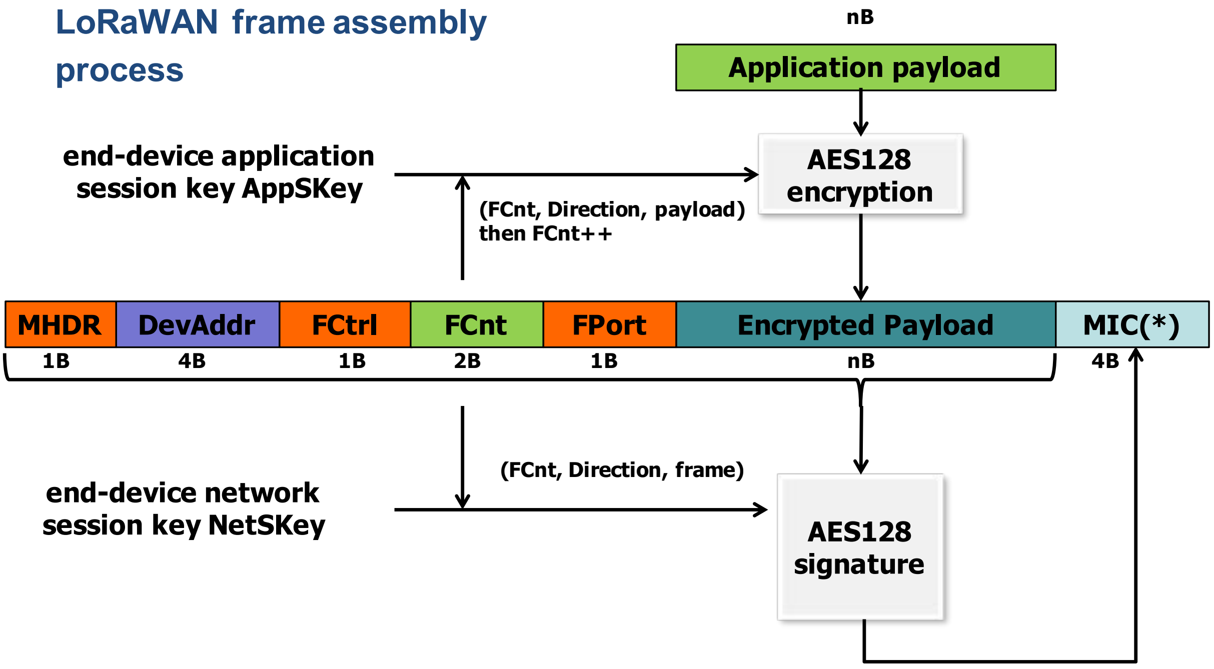
To support LoRaWAN packet, it is necessary to be able to encrypt the application payload and compute the MIC according to the LoRaWAN specifications.

7.1.2. LoRaWAN encryption

The LoRaWAN encryption procedure uses 2 keys : an application key (AppSKey) and a network key (NetSKey). Both are 16-bytes long. The 128-bit AES-CTR encryption mode is then applied with the 2 keys as illustrated in Figure 74 for the frame assembly process.

AppSKey is used to encryt the application payload and NetSKey is used to produce the MIC. In both operations, FCnt, which is a counter that is incremented for each packet, and

Direction, which takes 0x00 value for uplink and 0x01 for downlink, are incorporated into the encryption process.


LoRaWAN frame assembly process
nB
Application payload
AES128 encryption
end-device application session key AppSKey
(FCnt, Direction, payload) then FCnt++
MHDR
DevAddr
FCtrl
FCnt
FPort
Encrypted Payload
MIC(*)
1B
4B
1B
2B
1B
nB
4B
(FCnt, Direction, frame)
end-device network session key NetSKey
AES128 signature

(*) MIC = Message Integrity Check

Figure 74 – LoRaWAN frame assemby process

7.2. Setting up a LoRaWAN gateway : the Multitech mConduit

7.2.1. The Multitech mConduit

"MultiConnect Conduit from Multitech is a programmable gateway that uses an open Linux development environment (mLinux) to enable machine-to-machine (M2M) connectivity using various wireless interfaces. It also provides an online application store as a platform for developers to provision and manage their gateway and associated sensors and devices. Conduit allows you to use either IBM's Node-RED drag-and-drop interface or mLinux Open Embedded/Yocto to develop IoT applications for monitoring and controlling assets. A diverse range of MultiConnect mCard accessory cards provide the local wired or wireless field asset connectivity and plug directly into the rear of the Conduit gateway" .

The Multitech Conduit is shown in Figure 75.

"*The MTAC-LoRa accessory card provides long range RF support using Semtech's LoRa radio technology. The Conduit gateway uses this accessory card to connect MultiTech's mDot products to the cloud*".

Our Conduit model is the MTCDT-210L-US-EU-GB mLinux Programmable Gateway.

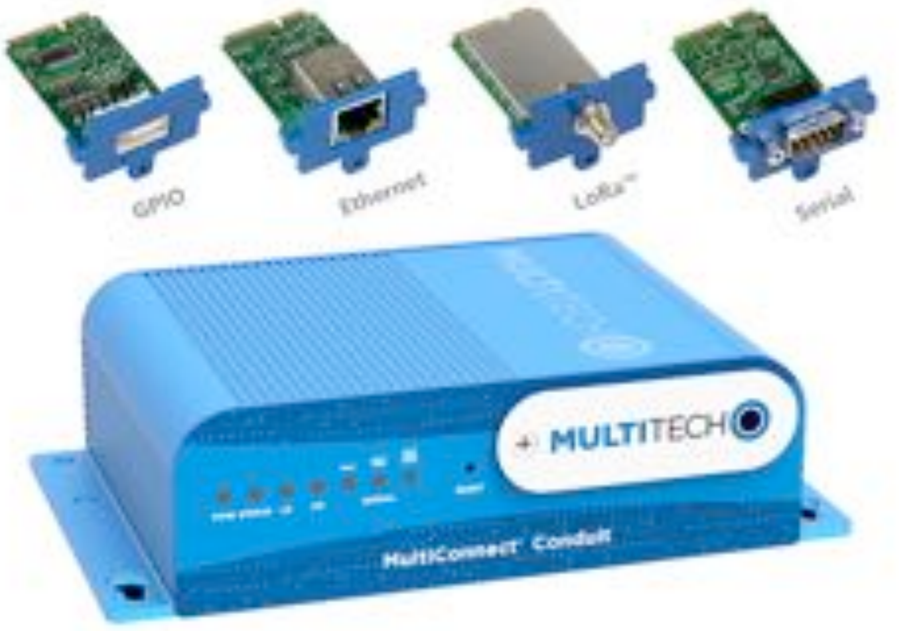


Figure 75 – The Multitech Conduit and some available mCards

The Conduit can be configured in two ways. One is to use the serial port by connecting the gateway to your computer with a micro-USB-cable to the USB device port and opening up a serial monitor tool such as Putty. In our case, we use a second solution which is to access the terminal interface via Ethernet. First, we need to connect the antenna to the MTAC-LoRa module, then the Ethernet cable and finally the power supply. The gateway shoud now boot up. By default the Conduit has an hardcoded IP address : **192.168.2.1** (DHCP is disabled). On your computer, configure the network interface that is connected to the Conduit to be a static IP address within **192.168.2.2 – 192.168.2.254**. Next open an SSH connection using the default factory credential: user is **root** , password is **root**.

On Linux, issue this command in your computer's terminal :

```
ssh root@192.168.2.1
```

When prompted, enter the default password. On Windows, install Putty and open a new session SSH on port 22 using the above defaults.

If the login was successful the Conduit's terminal prompt should appear:

```
root@mtcdt:~#
```

Upgrading LoRa Server & Packet Forwarder

1. Download from http://www.multitech.net/developer/downloads/ the latest **Lora-Packet-forwarder** and **Lora-Network-Server** packages from the Multitech. The files should be similar to this:

    - ***lora-packet-forwarder_1.4.1-r9.1_arm926ejste.ipk***
    - ***lora-network-server_1.0.8-r0.0_mlinux.ipk***

2. Copy both packages to the Conduit using SCP or an alternative (flash drive for example), and install them using the **opkg** package manager :

```
$ opkg install lora-packet-forwarder_*.ipk
$ opkg install lora-network-server_*.ipk
```

Configuring the packet forwarder for TheThingsNetwork

1. Issue these commands on the Conduit :

```
$ mkdir /var/config/lora
$ cp /opt/lora/lora-network-server.conf.sample /var/config/lora/lora-
network-server.conf
```

2. Edit **/var/config/lora/lora-network-server.conf** and modify these settings as needed (use vi or nano).

Field	MTAC-LORA-915 (NA)	MTAC-LORA-868 (EU)
lora["frequencyBand"]:	"915"	"868"
lora["channelPlan"]:	"US915" or "AU915"	"EU868"
lora["frequencySubBand"]:	(integer: 1 to 8)	Not applicable
lora["frequencyEU"]:	Not applicable	default: 869500000 range: [863500000 - 867500000] and [869100000 - 869500000]
network["public"]:	true	

3. Restart the network server. This will generate a configuration file for the packet forwarder.

```
$ /etc/init.d/lora-network-server restart
```

4. Copy the generated packet forwarder configuration to the config partition

```
$ cp /var/run/lora/1/global_conf.json
/var/config/lora/global_conf.json
```

5. Edit **/var/config/lora/global-conf.json** and modify these settings.Enter the server address depending on your region :

router.eu.staging.thethings.network # EU 433 and EU 863-870
router.us.staging.thethings.network # US 902-928
router.cn.staging.thethings.network # China 470-510 and 779-787
router.au.staging.thethings.network # Australia 915-928 MHz

```
"gateway_conf" :
{
     "gateway_ID" : "00000008004A0410",
     "forward_crc_disabled" : true,
     "forward_crc_error" : false,
     "forward_crc_valid" : true,
     "keepalive_interval" : 12,
     "push_timeout_ms" : 120,
     "serv_port_down" : 1700,
     "serv_port_up" : 1700,
     "server_address" : "router.eu.thethings.network",
     "stat_interval" : 20,
     "synch_word" : 52
}
```

6. Edit **/etc/init.d/lora-network-sever** as follows :

   Comment out the lora network server start code (line 57 to 61)

```
# start network server
# start-stop-daemon --start --background --make-pidfile \
#   --pidfile $net_server_pidfile --exec $net_server -- \
#   -c $conf_file --lora-eui $lora_eui --lora-path $run_dir --db $conf_db \
#   --noconsole -l $net_server_log
#sleep 1
```

   On line 65, change the **-c $run_dir** to **-c $conf_dir**

```
# start packet forwarder
start-stop-daemon --start --background --make-pidfile \
     --pidfile $pkt_fwd_pidfile --exec $pkt_fwd -- \
     -c $conf_dir -l $pkt_fwd_log
echo "OK"
```

   Comment out the Lora network server stop code (line 71)

```
#start-stop-daemon --stop --quiet --oknodo --pidfile $net_server_pidfile --retry 15
 start-stop-daemon --stop --quiet --oknodo --pidfile $pkt_fwd_pidfile --retry 5
```

7. Restart the packet forwarder

```
$ /etc/init.d/lora-network-server restart
```

8. For debbugging, you can view the packet forwarder log with this command:

```
$ tail -f /var/log/lora-pkt-fwd.log
```

```
setup_sx125x:678: Note: SX125x #0 clock output enabled
setup_sx125x:721: Note: SX125x #0 PLL start (attempt 1)
setup_sx125x:673: Note: SX125x #1 version register returned 0x21
setup_sx125x:678: Note: SX125x #1 clock output enabled
setup_sx125x:721: Note: SX125x #1 PLL start (attempt 1)
lgw_start:1120: Note: calibration started (time: 3000 ms)
lgw_start:1141: Note: calibration finished (status = 191)
Info: Initialising AGC firmware...
Info: loading custom TX gain table
Info: putting back original RADIO_SELECT value
*** Basic Packet Forwarder for Lora Gateway ***
Version: 1.4.1
*** Lora concentrator HAL library version info ***
Version: 1.7.0; Options: ftdi sx1301 sx1257 full mtac-lora private;
***
INFO: Little endian host
INFO: found global configuration file /var/config/lora/global_conf.json, parsing it
INFO: /var/config/lora/global_conf.json does contain a JSON object named
SX1301_conf, parsing SX1301 parameters
INFO: radio 0 enabled, center frequency 867500000
INFO: radio 1 enabled, center frequency 868500000
INFO: Lora multi-SF channel 0>  radio 1, IF -400000 Hz, 125 kHz bw, SF 7 to 12
INFO: Lora multi-SF channel 1>  radio 1, IF -200000 Hz, 125 kHz bw, SF 7 to 12
INFO: Lora multi-SF channel 2>  radio 1, IF 0 Hz, 125 kHz bw, SF 7 to 12
INFO: Lora multi-SF channel 3>  radio 0, IF -400000 Hz, 125 kHz bw, SF 7 to 12
INFO: Lora multi-SF channel 4>  radio 0, IF -200000 Hz, 125 kHz bw, SF 7 to 12
INFO: Lora multi-SF channel 5>  radio 0, IF 0 Hz, 125 kHz bw, SF 7 to 12
INFO: Lora multi-SF channel 6>  radio 0, IF 200000 Hz, 125 kHz bw, SF 7 to 12
INFO: Lora multi-SF channel 7>  radio 0, IF 400000 Hz, 125 kHz bw, SF 7 to 12
INFO: Lora std channel> radio 1, IF -200000 Hz, 250000 Hz bw, SF 7
INFO: FSK channel> radio 1, IF 300000 Hz, 100000 Hz bw, 50000 bps datarate
INFO: /var/config/lora/global_conf.json does contain a JSON object named
gateway_conf, parsing gateway parameters
INFO: gateway MAC address is configured to 00000008004A0410
INFO: server hostname or IP address is configured to "router.eu.thethings.network"
INFO: upstream port is configured to "1700"
INFO: downstream port is configured to "1700"
INFO: synch word is configured to 34
INFO: downstream keep-alive interval is configured to 12 seconds
INFO: statistics display interval is configured to 20 seconds
INFO: upstream PUSH_DATA time-out is configured to 120 ms
INFO: packets received with a valid CRC will be forwarded
INFO: packets received with a CRC error will NOT be forwarded
INFO: packets received with no CRC will be forwarded
INFO: found local configuration file /var/config/lora/local_conf.json, parsing it
INFO: redefined parameters will overwrite global parameters
```

```
INFO: /var/config/lora/local_conf.json does not contain a JSON object named
SX1301_conf
INFO: /var/config/lora/local_conf.json does contain a JSON object named
gateway_conf, parsing gateway parameters
INFO: gateway MAC address is configured to 00000008004A0410
INFO: packets received with a valid CRC will be forwarded
INFO: packets received with a CRC error will NOT be forwarded
INFO: packets received with no CRC will be forwarded
INFO: [main] concentrator started, packet can now be received

INFO: Start of downstream thread
INFO: [down] PULL_ACK received in 29 ms
INFO: [down] PULL_ACK received in 28 ms

##### 2016-10-13 22:34:30 GMT #####
### [UPSTREAM] ###
# RF packets received by concentrator: 0
# CRC_OK: 0.00%, CRC_FAIL: 0.00%, NO_CRC: 0.00%
# RF packets forwarded: 0 (0 bytes)
# PUSH_DATA datagrams sent: 0 (0 bytes)
# PUSH_DATA acknowledged: 0.00%
### [DOWNSTREAM] ###
# PULL_DATA sent: 2 (100.00% acknowledged)
# PULL_RESP(onse) datagrams received: 0 (0 bytes)
# RF packets sent to concentrator: 0 (0 bytes)
# TX errors: 0
```

For Upstream :

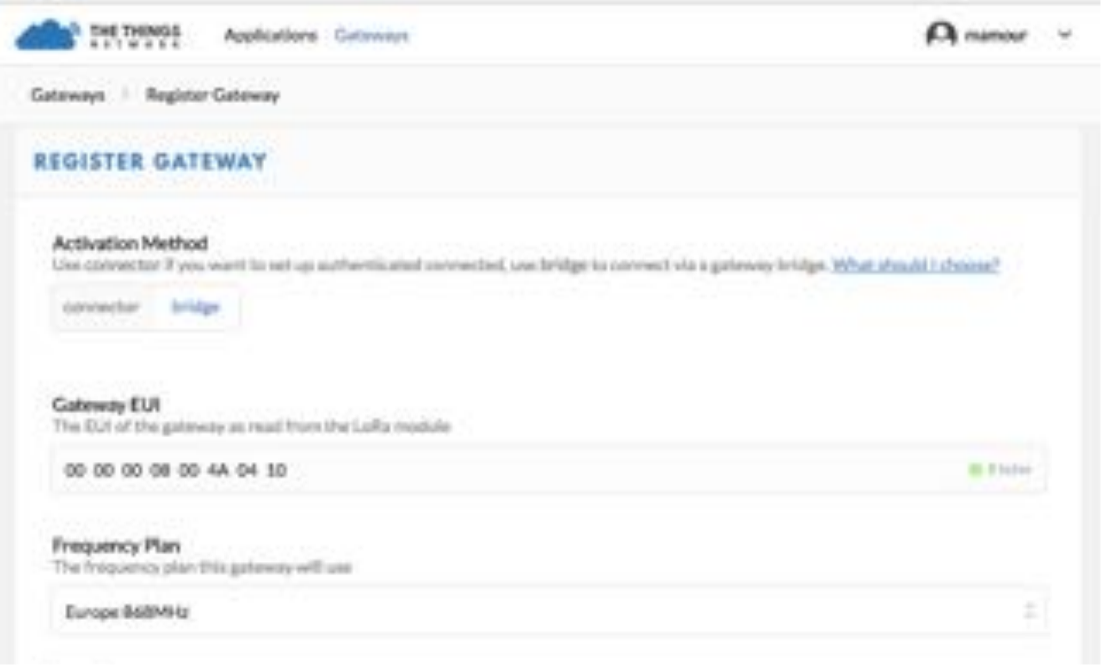
- **RF packets received by concentrator** : packets are stored in concentrator)
- **RF packets forwarded** : packets are forwarded to The Things Network server
- **PUSH_DATA acknowledged** : forwarding acknowledgement
    - ✓ 0% : no active connection to TTN backend
    - ✓ 100% : active connection to TTN backend

For Downstream :

- **PULL_DATA** sent : requesting available downstream packets
    - ✓ 0% : no active connection to TTN backend
    - ✓ 100% : active connection to TTN backend
- **PULL_RESP(onse) datagrams received** : downstream packets
- **RF packets sent to concentrator** : concentrator sends downstream packets

The gateway is now operational.

Gateway registration on The Things Network (see Figure 76)



REGISTER GATEWAY

Activation Method
Use connector if you want to set up authenticated connected, use bridge to connect via a gateway bridge. What should I choose?

Gateway EUI
The EUI of the gateway as read from the LoRa module
00 00 00 08 00 4A 04 10

Frequency Plan
The frequency plan this gateway will use
Europe 868MHz

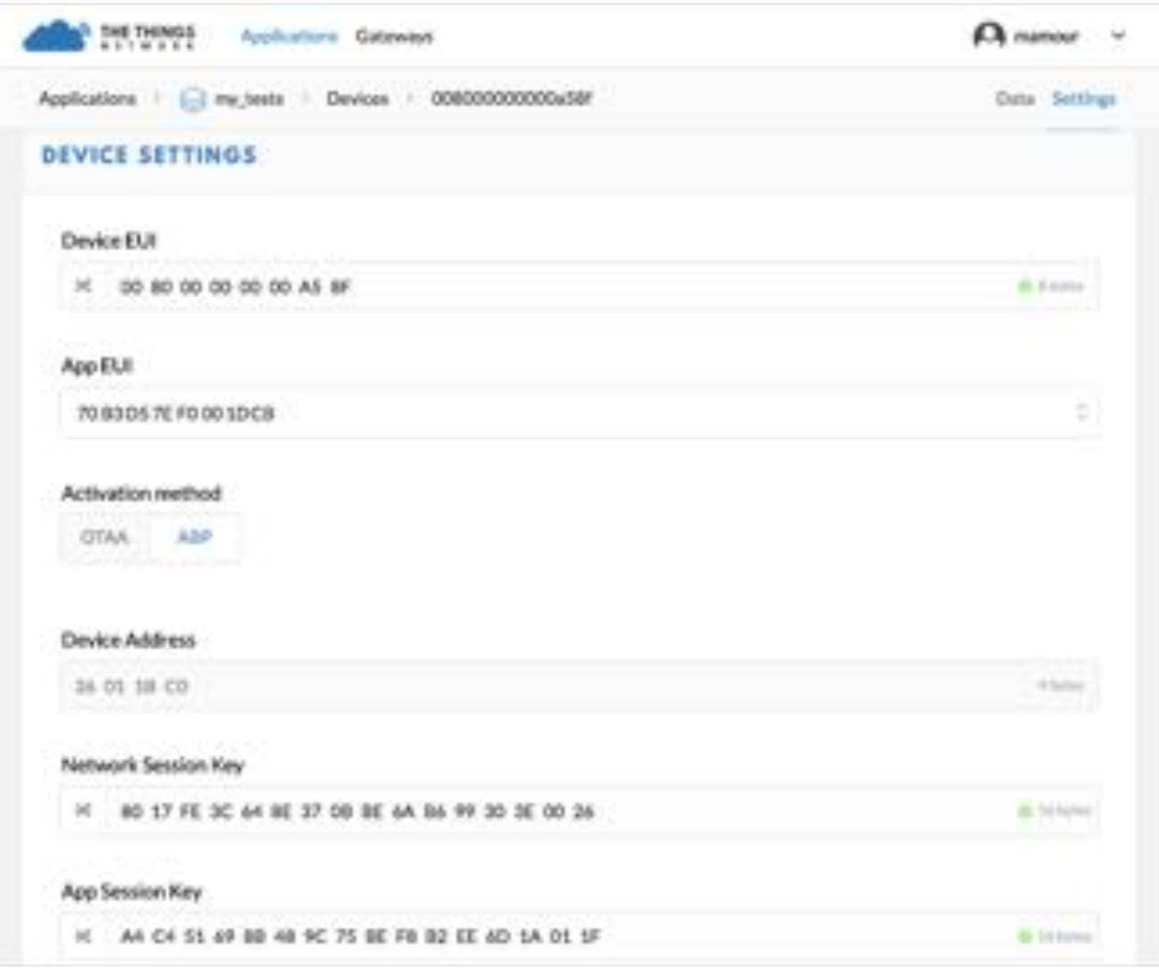
Figure 76 – Gateway settings

7.2.2. The mDot

"*The MultiConnect mDot is a secure, programmable, long-range and low-power RF module that provides data connectivity to sensors, industrial equipment, and remote appliances*" (see Figure 77).

Figure 77 – MultiConnect mDot

Before our mDot device can communicate via The Things Network we need to register it with an application. Figure 78 depicts these settings. We use ABP activation method that is useful for workshops: no waiting for a downlink window to become available to confirm the activation. In production, the default Over The Air Activation (OTTA) is recommended. This is more reliable because the activation will be confirmed and more secure because the session keys will be negotiated with every activation.



DEVICE SETTINGS

Device EUI
00 80 00 00 00 00 A5 8F
8 bytes

App EUI
70 B3 D5 7E F0 00 1D CB

Activation method
OTAA ABP

Device Address
26 01 1B C0
4 bytes

Network Session Key
80 17 FE 3C 64 8E 37 0B BE 6A B6 99 30 3E 00 26
16 bytes

App Session Key
A4 C4 51 69 8B 48 9C 75 BE F8 B2 EE 6D 1A 01 1F
16 bytes

Figure 78 – Device settings on TTN

Setting up the mDot to join the Conduit LoRa network

Setting up the mDot to join the Conduit LoRa network, we need to connect the mDot to our computer using Serial to USB connector and with an hyper terminal to send AT commands. These AT commands are as follows :

- Manual join mode
  `AT+NJM=0`
- Public network mode
  `AT+PN=1`
- Set Network Session Key
  `AT+NSK=8017FE3C648E370BBE6AB699303E0026`
- Set Data Session Key
  `AT+DSK=A4C451698B489C75BEF8B2EE6D1A011F`
- Set Network ID
  `AT+NI=0,70B3D57EF0001DCB`
- Set Network Address
  `AT+NA=26011BC0`
- Save settings
  `AT&W`
- Restart
  `ATZ`

For checking, we can display current settings and status:

```
AT&V
```

```
Device ID: 00:80:00:00:00:00:a5:8f
Frequency Band: FB_868
Frequency Sub Band: 0
Public Network: on
Start Up Mode: COMMAND
Network Address: 26011bc0
Network ID: 70:b3:d5:7e:f0:00:1d:cb
Network ID Passphrase:
Network Key:
Network Key Passphrase:
Network Session Key: 80.17.fe.3c.64.8e.37.0b.be.6a.b6.99.30.3e.00.26
Data Session Key: a4.c4.51.69.8b.48.9c.75.be.f8.b2.ee.6d.1a.01.1f
Network Join Mode: MANUAL
Network Join Retries: 2
Join Byte Order: LSB
Link Check Threshold: off
Link Check Count: off
Error Correction: 1 bytes
ACK Retries: off
Encryption: on
CRC: on
Adaptive Data Rate: off
Command Echo: on
Verbose Response: off
Tx Frequency: 0
Tx Data Rate: SF_9
Tx Power: 11
Tx Wait: on
Tx Inverted Signal: off
Rx Frequency: 869525000
Rx Data Rate: SF_9
Rx Inverted Signal: on
Rx Output Style: HEXADECIMAL
Debug Baud Rate: 115200
Serial Baud Rate: 115200
Wake Mode: INTERVAL
Wake Interval: 10 s
Wake Delay: 100 ms
Wake Timeout: 20 ms
Log Level: 0
```

Now, we can join the Conduit LoRa network by sending a message without requesting an ACK:

```
AT+ACK=0
AT+SEND=Hello from Dakar
```

Check if the Conduit is receiving packages

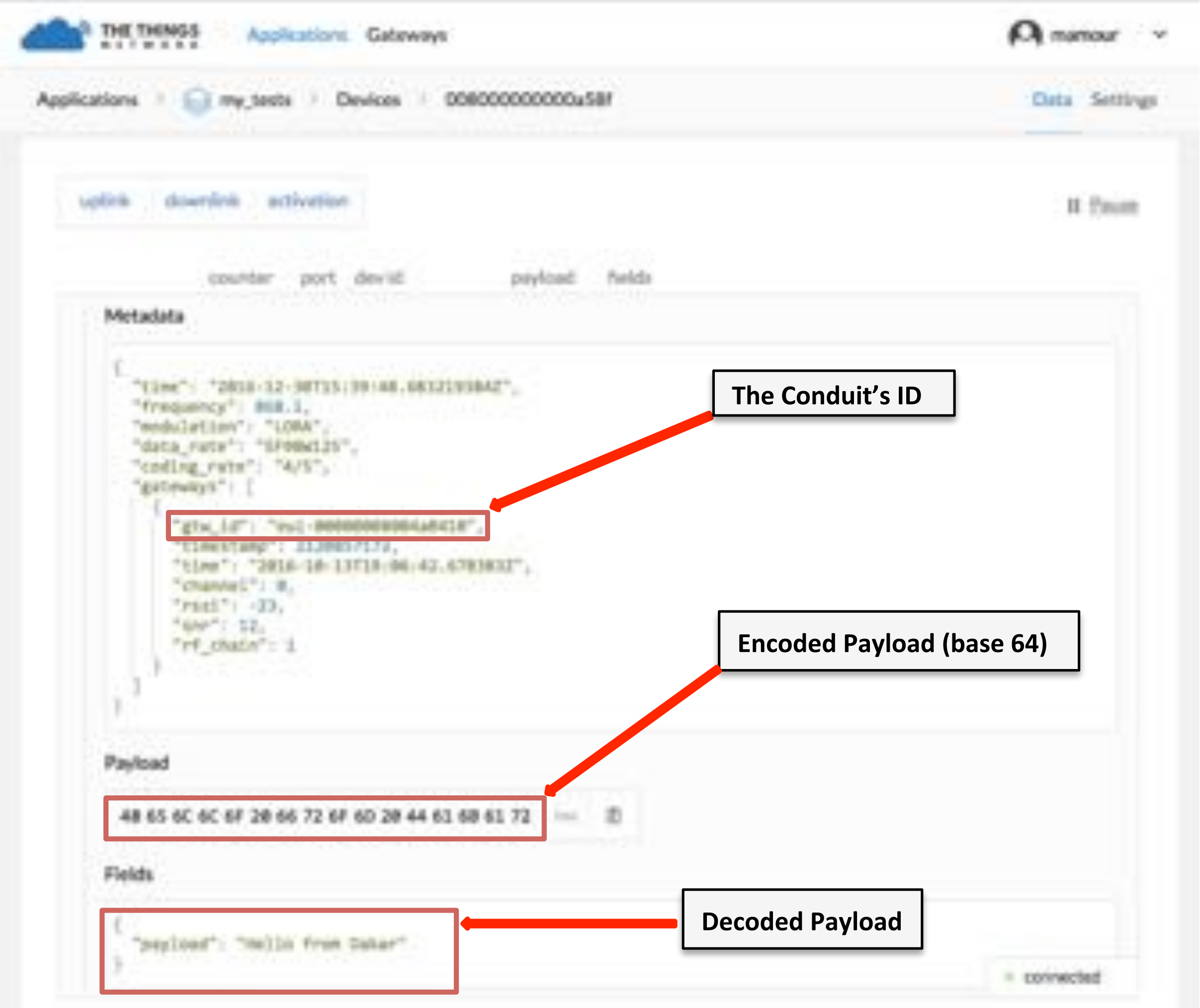
We can run

```
tail -f /var/log/lora-pkt-fwd.log
```

on the Conduit terminal session as described above and view the log. Here is a successful result :

```
##### 2016-12-30 16:39:48 GMT #####
### [UPSTREAM] ###
# RF packets received by concentrator: 1
# CRC_OK: 100.00%, CRC_FAIL: 0.00%, NO_CRC: 0.00%
# RF packets forwarded: 1 (29 bytes)
# PUSH_DATA datagrams sent: 1 (252 bytes)
# PUSH_DATA acknowledged: 100.00%
### [DOWNSTREAM] ###
# PULL_DATA sent: 1 (100.00% acknowledged)
# PULL_RESP(onse) datagrams received: 0 (0 bytes)
# RF packets sent to concentrator: 0 (0 bytes)
# TX errors: 0
##### END #####
INFO: [down] PULL_ACK received in 28 ms
INFO: [down] PULL_ACK received in 28 ms
```

Visualize packages on The Things Network



```
{
  "time": "2016-12-30T15:39:48.683219384Z",
  "frequency": 868.1,
  "modulation": "LORA",
  "data_rate": "SF9BW125",
  "coding_rate": "4/5",
  "gateways": [
    {
      "gtw_id": "eui-00000000004a0418",
      "timestamp": 2120057172,
      "time": "2016-10-13T18:06:42.678383Z",
      "channel": 0,
      "rssi": -33,
      "snr": 12,
      "rf_chain": 1
    }
  ]
}
```

```
48 65 6C 6C 6F 20 66 72 6F 6D 20 44 61 6B 61 72
```

```
{
  "payload": "Hello from Dakar"
}
```

Figure 79 – Receiving data on TTN

Sources : www.multitech.net, www.thethingsnetwork.org

7.3. Providing a simple LoRaWAN interoperability level

The LoRaWAN interoperability level that the WAZIUP platform proposes is as follows:

- Uplink message only: a low-cost IoT device can send LoRaWAN-formatted packet that can be received both by WAZIUP low-cost and commercial LoRaWAN gateway;
- The WAZIUP low-cost gateway can received and process LoRaWAN packets from a LoRaWAN device (either with a LoRaWAN radio module, or using a LoRaWAN protocol stack such as several "IBM LoRaWAN-in-C" [12] (LMIC) ports for Arduino boards; local decryption can be enabled if AppSKey and NetSKey are both known by the gateway;
- For the IoT device, the knowledge of AppSKey and NetSKey are realized following the Activation By Personalization mechanism (ABP) which means that AppSKey and NetSKey are stored initially in the IoT device.

7.3.1. Low-cost IoT device

At the low-cost IoT device, both AppSKey and NetSkey are stored as follows :

```
unsigned char AppSkey[16] = {
  0x2B, 0x7E, 0x15, 0x16, 0x28, 0xAE, 0xD2, 0xA6,
  0xAB, 0xF7, 0x15, 0x88, 0x09, 0xCF, 0x4F, 0x3C
};

unsigned char MwkSkey[16] = {
  0x2B, 0x7E, 0x15, 0x16, 0x28, 0xAE, 0xD2, 0xA6,
  0xAB, 0xF7, 0x15, 0x88, 0x09, 0xCF, 0x4F, 0x3C
};
```

Here we use the standard keys for test purposes. Then, a 4-byte device address is defined, possibly using the 1-byte address used by our long-range communication library (see Sections 5.3.6 and 5.4.4). In the following examples, we used:

```
unsigned char DevAddr[4] = {
  0x00, 0x00, 0x00, 0x06
};
```

LoRaWAN AES-CTR encryption functionalities are provided by the great light-weight library written by Gerben den Hartog, initially for the Ideetron Nexus board, but further used in several LMIC ports for Arduino boards because of its simplicity, efficiency and small memory footprint.

As indicated previously, **Arduino_LoRa_temp** is a template with AES encryption and the possibility to send LoRaWAN packet. Assuming that the payload is \!#3#TC/22.50, we show below an example output of an IoT device running the Arduino_LoRa_temp program:

```
LoRa temperature sensor
Arduino Pro Mini detected
SX1276 detected, starting
SX1276 LF/HF calibration
...
Get back previous sx1272 config
Using packet sequence number of 100
Setting Mode: state 0
Setting Channel: state 0
Setting Power: state 0
Setting node addr: state 0
SX1272 successfully configured
Reading 70
(Temp is 22.55
Sending \!#3#TC/22.50
Real payload size is 13
\!#3#TC/22.50
plain payload hex
5C 21 23 33 23 54 43 2F 32 32 2E 35 30
Encrypting
encrypted payload
4F 29 E7 49 77 A1 C2 E7 81 E6 16 A6 68
calculate MIC with NwkSKey
transmitted LoRaWAN-like packet:
MHDR[1] | DevAddr[4] | FCtrl[1] | FCnt[2] | FPort[1] | EncryptedPayload | MIC[4]
40 06 00 00 00 00 00 00 01 4F 29 E7 49 77 A1 C2 E7 81 E6 16 A6 68 FD 75 68 74
end-device uses native LoRaWAN packet format
--> CAD duration 547
OK1
--> waiting for 1 CAD = 62
--> CAD duration 547
OK2
--> RSSI -126
LoRa pkt size 26
LoRa pkt seq 100
LoRa Sent in 1733
LoRa Sent w/CAD in 9091
Packet sent, state 0
```

7.3.2. Low-cost gateway

The post-processing block of our low-cost gateway can receive LoRaWAN packet and can either locally decrypt the packet if both AppSKey and NetSKey are available or push the encrypted payload into the cloud. The LoRaWAN decryption capabilities are provided by the LoRaWAN python package from https://github.com/jeroennijhof/LoRaWAN. We wrapped the LoRaWAN support using the LoRaWAN encryption/decryption python package into a loraWAN.py python script.

In the following example, we show our low-cost gateway locally decrypting the LoRaWAN packet and pushing data to the ThingSpeak cloud as explained in Section 5.5.4.

```
Parsing cloud declarations
[u'python CloudThingSpeak.py']
Parsed all cloud declarations
post_processing_gw.py got cloud list:
[u'python CloudThingSpeak.py']
raw output from gateway. post_processing_gw will handle packet format
enable local AES decryption

Current working directory: /home/pi/lora_gateway
SX1276 detected, starting.
SX1276 LF/HF calibration
...
**********Power ON: state 0
Default sync word: 0x12
LoRa mode 1
Setting mode: state 0
Channel CH_10_868: state 0
Set LoRa Power to M
Power: state 0
Get Preamble Length: state 0
Preamble Length: 8
LoRa addr 1: state 0
Raw format, not assuming any header in reception
SX1272/76 configured as LR-BS. Waiting RF input for transparent RF-serial bridge
```

```
--- rxlora. dst=0 type=0x00 src=0 seq=0 len=26 SNR=-16 RSSIpkt=-110 BW=125
CR=4/5 SF=12
2016-10-20T12:34:22.420292
rcv ctrl pkt info (^p): 0,0,0,0,26,-16,-110
splitted in:  [0, 0, 0, 0, 26, -16, -110]
rawFormat(len=26 SNR=-16 RSSI=-110)
rcv ctrl radio info (^r): 125,5,12
splitted in:  [125, 5, 12]
(BW=125 CR=5 SF=12)
rcv timestamp (^t): 2016-10-20T12:09:25.281

got first framing byte
--> got data prefix
raw format from gateway
LoRaWAN?
loraWAN: valid MIC
loraWAN: plain payload is \!#3#TC/22.50
plain payload is : \!#3#TC/22.50
number of enabled clouds is 1
--> cloud[0]
uploading with python CloudThingSpeak.py
ThingSpeak: uploading
rcv msg to log (\!) on ThingSpeak ( default , 3 ): 22.50
ThingSpeak: will issue curl cmd
curl -s -k -X POST --data field3=22.50&field7=0 https://api.thingspeak.com/[…]
ThingSpeak: returned code from server is 210
--> cloud end
```

When the gateway does not know AppSKey nor NetSkey as it is the case when several organizations share the same gateway, the following example shows how the gateway can push the encrypted payload into the cloud where decryption will be performed in a later step. The example uses FireBase to store in JSON format the LoRaWAN packet. Encrypted payload and the entire LoRaWAN frame are stored in Base64 format.

```
Parsing cloud declarations
[u'python CloudThingSpeak.py']
Parsed all cloud declarations
post_processing_gw.py got cloud list:
[u'python CloudThingSpeak.py']
Parsing cloud declarations
[u'python CloudFireBaseLWAES.py']
Parsed all cloud declarations
post_processing_gw.py got LoRaWAN encrypted cloud list:
[u'python CloudFireBaseLWAES.py']
raw output from gateway. post_processing_gw will handle packet format

Current working directory: /home/pi/lora_gateway
2016-10-26T22:14:12.985467
rcv ctrl pkt info (^p): 0,0,0,0,26,8,-45
splitted in:  [0, 0, 0, 0, 26, 8, -45]
rawFormat(len=26 SNR=8 RSSI=-45)
rcv ctrl radio info (^r): 125,5,12
splitted in:  [125, 5, 12]
(BW=125 CR=5 SF=12)
got first framing byte
--> got data prefix
raw format from gateway
LoRaWAN?
--> DATA encrypted: local aes not activated
--> FYI base64 of LoRaWAN frame w/MIC: QAYAAAAAAAABTynnSXehwueB5hamaP11aHQ=
--> number of enabled clouds is 1
--> LoRaWAN encrypted cloud[0]
uploading with python CloudFireBaseLWAES.py
FireBase: uploading
Firebase: upload success
--> LoRaWAN encrypted cloud end
```

On FireBase, the following record is stored:

```
sensor0x00000006
      msg0x0000
            FCnt: "0x0000"
            FPort: "0x01"
            MIC: "0xfd756874"
            cr: 5
            data_b64: "TynnSXehwueB5hamaA=="
            datarate: "SF12BW125"
            devAddr: "0x00000006"
            frame_b64: "QAYAAAAAAAABTynnSXehwueB5hamaP11aHQ="
            gateway_eui: "00000027EB795C47"
            len: 26
            pdata:"0,0,0,0,26,8,-45"
            ptype: "0x40"
            ptypestr: "unconfirmed data up"
            rdata:"125,5,12"
            rssi: -45
            snr: 8
            time: "2016-10-26T20:14:13.110056"
```

The previous loraWAN.py script can be used on a computer to decrypt the encrypted payload.

```
> python loraWAN.py "QAYAAAAAAAABTynnSXehwueB5hamaP11aHQ=" "1,20,6,0,26,8,-45"
"125,5,12"
?loraWAN: valid MIC
?loraWAN: plain payload is \!#3#TC/22.50
?plain payload is : \!#3#TC/22.50
^p1,16,6,0,13,8,-45
^r125,5,12
??\!#3#TC/22.50
```

Note the 2 '?' in front of the plain data. These are normally the data prefix \xFF\xFE inserted by the lora_gateway program and emulated by loraWAN.py. The '?' in front of the other lines indicate to a post-processing block that these lines should be ignored.

Once the application has the plain payload, it can further push the plain data into other IoT clouds. For instance, a final app running on a Linux machine can simply use our post-processing block written in Python (post_processing_gw.py) with the plain data to upload data to IoT clouds just as a gateway with NetSKey and AppSKey would do.

```
> python loraWAN.py "QAYAAAAAAAABTynnSXehwueB5hamaP11aHQ=" "1,20,6,0,26,8,-45"
"125,5,12" | python post_processing_gw.py
Parsing cloud declarations
[u'python CloudThingSpeak.py']
Parsed all cloud declarations
post_processing_gw.py got cloud list:
[u'python CloudThingSpeak.py']
Parsing cloud declarations
[u'python CloudFireBaseLWAES.py']
Parsed all cloud declarations
post_processing_gw.py got LoRaWAN encrypted cloud list:
[u'python CloudFireBaseLWAES.py']

Current working directory: /home/pi/my_final_app
2016-10-27T22:18:45.079058
rcv ctrl pkt info (^p): 1,16,6,0,13,8,-45
splitted in:  [1, 16, 6, 0, 13, 8, -45]
```

```
    (dst=1 type=0x10(DATA) src=6 seq=0 len=13 SNR=8 RSSI=-45)
    rcv ctrl radio info (^r): 125,5,12
    splitted in:  [125, 5, 12]
    (BW=125 CR=5 SF=12)
    got first framing byte
    --> got data prefix
    number of enabled clouds is 1
    --> cloud[0]
    uploading with python CloudThingSpeak.py
    ThingSpeak: uploading
    rcv msg to log (\!) on ThingSpeak ( default , 3 ): 22.50
    ThingSpeak: will issue curl cmd
    curl -s -k -X POST --data field3=22.50&field7=0 https://api.thingspeak.com/ […]
    ThingSpeak: returned code from server is 218
    --> cloud end
```

If you use this feature where post_processing_gw.py takes its input from loraWAN.py, you have to pass at least 2 arguments: the base64-encoded string and the packet info string (e.g. ^p string). loraWAN.py set the packet type and the data length to the appropriate value for post_processing_gw.py. post_processing_gw.py can use exactely the same cloud configuration than a normal gateway would have.

7.3.3. From LoRaWAN mDot to low-cost gateway

As described previously, the mDot is a LoRaWAN radio module from MultiTech. The mdot will be configured by ABP, with the AppSKey and NetSKey matching those declared in loraWAN.py. The frequency will be 868.1MHz and the LoRa parameters are : bw=125kHz, cr=4/5 and sf=12 which correspond to our LoRa mode 1.

```
        # Data session encryption key (16 bytes)
        AT+DSK=2B7E151628AED2A6ABF7158809CF4F3C
        OK

        # Network session encryption key (16 bytes)
        AT+NSK=2B7E151628AED2A6ABF7158809CF4F3C
        OK
```

Then, sending with the mDot :

```
        AT+SEND=HelloWorld
        OK

        AT+SEND=HelloWorld_1
        OK
```

The low-cost gateway is started as follows, note the sync word of 0x34 to match the one of LoRaWAN.

```
    > sudo ./lora_gateway --mode 1 --raw --sw 34 --freq 868.1 | python
    post_processing_gw.py --raw -aes

    Parsing cloud declarations
    Parsed all cloud declarations
    post_processing_gw.py got cloud list:
    []
    raw output from gateway. post_processing_gw will handle packet format
    enable local AES decryption

    Current working directory: /home/pi/lora_gateway
    SX1276 detected, starting.
    SX1276 LF/HF calibration
    ...
    **********Power ON: state 0
```

```
Default sync word: 0x12
Set sync word to 0x34
LoRa sync word: state 0
LoRa mode 1
Setting mode: state 0
Frequency 868.100000: state 0
Set LoRa Power to M
Power: state 0
Get Preamble Length: state 0
Preamble Length: 8
LoRa addr 1: state 0
Raw format, not assuming any header in reception
SX1272/76 configured as LR-BS. Waiting RF input for transparent RF-serial bridge
```

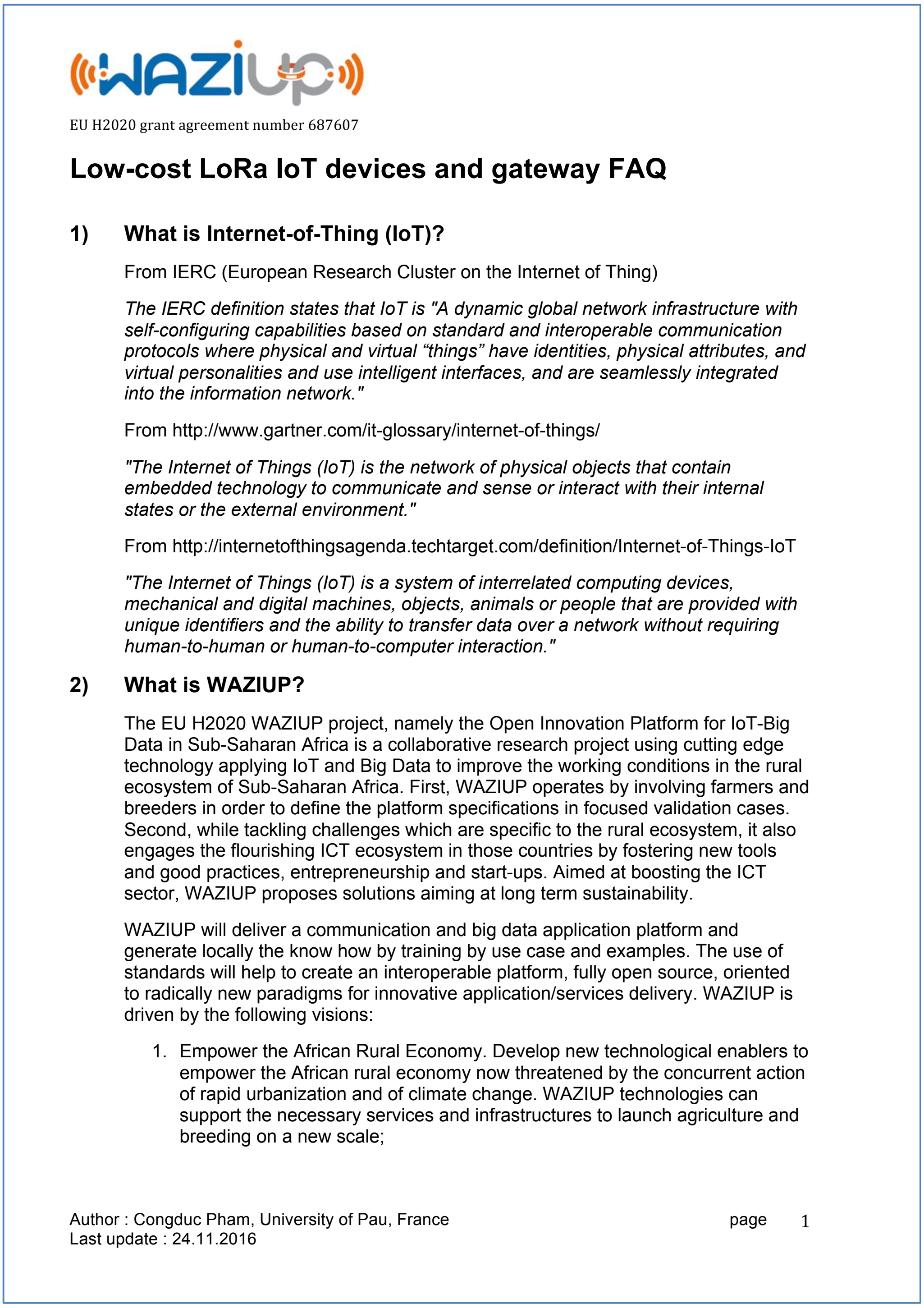
Here is the reception on our low-cost gateway :

```
...
--- rxlora. dst=0 type=0x00 src=0 seq=0 len=23 SNR=7 RSSIpkt=-68 BW=125
CR=4/5 SF=12
2016-10-24T11:13:51.119549
rcv ctrl pkt info (^p): 0,0,0,0,23,7,-68
splitted in:  [0, 0, 0, 0, 23, 7, -68]
rawFormat(len=23 SNR=7 RSSI=-68)
rcv ctrl radio info (^r): 125,5,12
splitted in:  [125, 5, 12]
(BW=125 CR=5 SF=12)
rcv timestamp (^t): 2016-10-24T11:13:51.118

got first framing byte
--> got data prefix
raw format from gateway
LoRaWAN?
loraWAN: valid MIC
loraWAN: plain payload is HelloWorld
plain payload is : HelloWorld
HelloWorld
--- rxlora. dst=0 type=0x00 src=0 seq=0 len=25 SNR=6 RSSIpkt=-80 BW=125
CR=4/5 SF=12
2016-10-24T11:30:34.333976
rcv ctrl pkt info (^p): 0,0,0,0,25,6,-80
splitted in:  [0, 0, 0, 0, 25, 6, -80]
rawFormat(len=25 SNR=6 RSSI=-80)
rcv ctrl radio info (^r): 125,5,12
splitted in:  [125, 5, 12]
(BW=125 CR=5 SF=12)
rcv timestamp (^t): 2016-10-24T11:30:34.333

got first framing byte
--> got data prefix
raw format from gateway
LoRaWAN?
loraWAN: valid MIC
loraWAN: plain payload is HelloWorld_1
plain payload is : HelloWorld_1
HelloWorld_1
```

8. ANNEXES

8.1. Our LoRa FAQs

[Doc] Low-cost LoRa IoT devices and gateway FAQ
Latest version can be downloaded from:
https://github.com/CongducPham/tutorials/blob/master/FAQ.pdf



EU H2020 grant agreement number 687607

Low-cost LoRa IoT devices and gateway FAQ

1) What is Internet-of-Thing (IoT)?

From IERC (European Research Cluster on the Internet of Thing)

The IERC definition states that IoT is "A dynamic global network infrastructure with self-configuring capabilities based on standard and interoperable communication protocols where physical and virtual "things" have identities, physical attributes, and virtual personalities and use intelligent interfaces, and are seamlessly integrated into the information network."

From http://www.gartner.com/it-glossary/internet-of-things/

"The Internet of Things (IoT) is the network of physical objects that contain embedded technology to communicate and sense or interact with their internal states or the external environment."

From http://internetofthingsagenda.techtarget.com/definition/Internet-of-Things-IoT

"The Internet of Things (IoT) is a system of interrelated computing devices, mechanical and digital machines, objects, animals or people that are provided with unique identifiers and the ability to transfer data over a network without requiring human-to-human or human-to-computer interaction."

2) What is WAZIUP?

The EU H2020 WAZIUP project, namely the Open Innovation Platform for IoT-Big Data in Sub-Saharan Africa is a collaborative research project using cutting edge technology applying IoT and Big Data to improve the working conditions in the rural ecosystem of Sub-Saharan Africa. First, WAZIUP operates by involving farmers and breeders in order to define the platform specifications in focused validation cases. Second, while tackling challenges which are specific to the rural ecosystem, it also engages the flourishing ICT ecosystem in those countries by fostering new tools and good practices, entrepreneurship and start-ups. Aimed at boosting the ICT sector, WAZIUP proposes solutions aiming at long term sustainability.

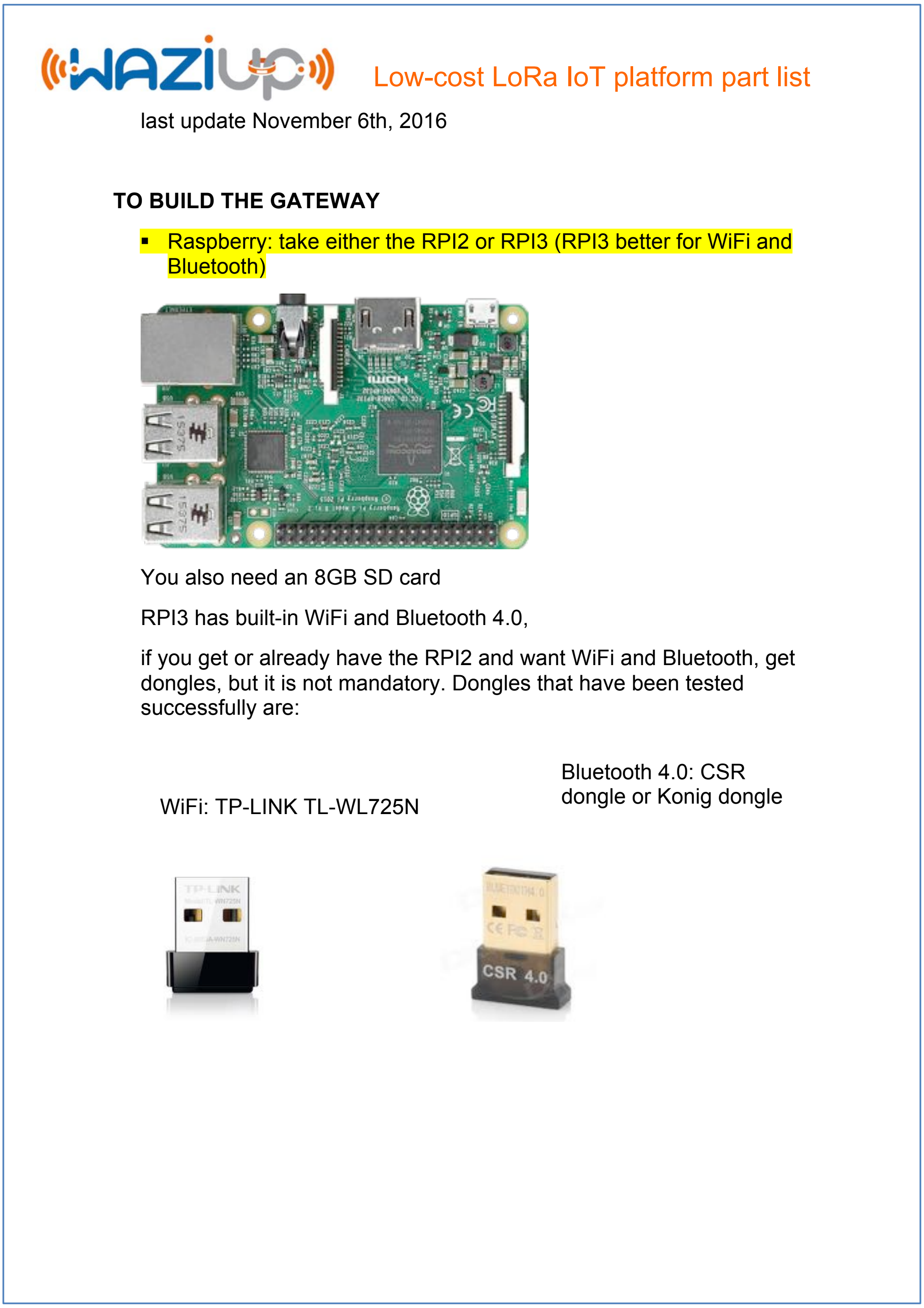
WAZIUP will deliver a communication and big data application platform and generate locally the know how by training by use case and examples. The use of standards will help to create an interoperable platform, fully open source, oriented to radically new paradigms for innovative application/services delivery. WAZIUP is driven by the following visions:

1. Empower the African Rural Economy. Develop new technological enablers to empower the African rural economy now threatened by the concurrent action of rapid urbanization and of climate change. WAZIUP technologies can support the necessary services and infrastructures to launch agriculture and breeding on a new scale;

Author : Congduc Pham, University of Pau, France page 1
Last update : 24.11.2016

8.2. Low-cost LoRa IoT platform part list

[Doc] Low-cost LoRa IoT platform part list
Latest version can be downloaded from:
https://github.com/CongducPham/tutorials/blob/master/low-cost-iot-hardware-parts.pdf



Low-cost LoRa IoT platform part list

last update November 6th, 2016

TO BUILD THE GATEWAY

- Raspberry: take either the RPI2 or RPI3 (RPI3 better for WiFi and Bluetooth)

You also need an 8GB SD card

RPI3 has built-in WiFi and Bluetooth 4.0,

if you get or already have the RPI2 and want WiFi and Bluetooth, get dongles, but it is not mandatory. Dongles that have been tested successfully are:

WiFi: TP-LINK TL-WL725N

Bluetooth 4.0: CSR dongle or Konig dongle

8.3. Tutorials materials

[Slides] "Tutorial on hardware & software for low-cost long-range IoT"
Latest version can be downloaded from:
https://github.com/CongducPham/tutorials/blob/master/tutorial-SWHW-LoRa-WAZIUP.pdf

[Slides] "Low-cost LoRa IoT device: a step-by-step tutorial"
Latest version can be downloaded from:
https://github.com/CongducPham/tutorials/blob/master/Low-cost-LoRa-IoT-step-by-step.pdf

[Slides] "Building IoT device for outdoor usage: a step-by-step tutorial"
Latest version can be downloaded from:
https://github.com/CongducPham/tutorials/blob/master/Low-cost-LoRa-IoT-outdoor-step-by-step.pdf

[Slides] "Low-cost LoRa IoT device: supported physical sensors"
Latest version can be downloaded from:
https://github.com/CongducPham/tutorials/blob/master/Low-cost-LoRa-IoT-supported-sensors.pdf

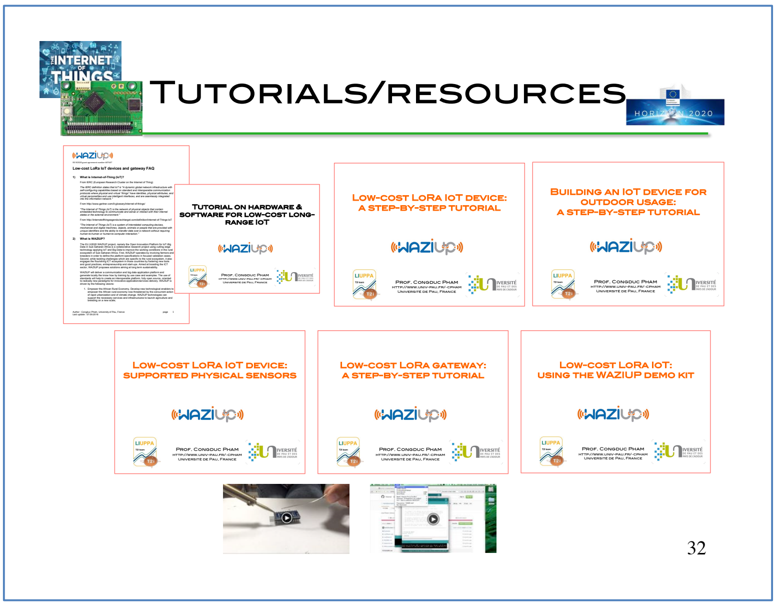
[Slides] "Low-cost LoRa gateway: a step-by-step tutorial"
Latest version can be downloaded from:
https://github.com/CongducPham/tutorials/blob/master/Low-cost-LoRa-GW-step-by-step.pdf

[Slides] "Low-cost LoRa IoT: using the WAZIUP demo kit"
Latest version can be downloaded from:
https://github.com/CongducPham/tutorials/blob/master/Low-cost-LoRa-IoT-using-demo-kit.pdf

[Slides] "Low-cost LoRa IoT antenna tutorial for gateway"
Latest version can be downloaded from:
https://github.com/CongducPham/tutorials/blob/master/Low-cost-LoRa-IoT-antennaCable.pdf

[Video] Build your low-cost, long-range IoT device with WAZIUP
https://www.youtube.com/watch?v=YsKbJeeav_M

[Video] Build your low-cost LoRa gateway with WAZIUP
https://www.youtube.com/watch?v=peHkDhiH3lE



Tutorials/resources

Low-cost LoRa IoT devices and gateway FAQ
Tutorial on hardware & software for low-cost long-range IoT
Low-cost LoRa IoT device: a step-by-step tutorial
Building an IoT device for outdoor usage: a step-by-step tutorial
Low-cost LoRa IoT device: supported physical sensors
Low-cost LoRa gateway: a step-by-step tutorial
Low-cost LoRa IoT: using the WAZIUP demo kit
Prof. Congduc Pham
http://www.univ-pau.fr/~cpham
Université de Pau, France
HORIZON 2020

32

8.4. Keynote talks

We provide here some links to selected keynote talks in international events where the WAZIUP project and its approach towards low-cost long-range IoT developped in WP2 has been presented.

"Internet-of-Thing for All"
Keynote at FDSE'2016/ACOMP'2016
Can Tho University of Technology, Can Tho, Vietnam
November 24th, 2016
Latest version can be downloaded from:
http://cpham.perso.univ-pau.fr/LORA/WAZIUP/IoT4all.pdf

"Internet-of-Thing and reasons it is becoming a reality"
Keynote at BDAW'2016
American University of Bulgaria, Sofia, Bulgaria
November 10th, 2016
Latest version can be downloaded from:
http://cpham.perso.univ-pau.fr/LORA/WAZIUP/BDAW16-IoT-reality.pdf

"Deploying low-cost and long-range Internet of Things in developing countries: the challenges of the WAZIUP H2020 project"
Invited talk at SMYLE event on "Understand the issues and challenges of the connected world" Neuchatel, Switzerland, September 23rd, 2016
Latest version can be downloaded from:
http://cpham.perso.univ-pau.fr/LORA/WAZIUP/Talk-iot-challenges-smyle.pdf

"Low-power, long-range WAN for IoT: a technology overview"
Invited talk at RESSACS'2016
IRD Bondy-Paris
May 10th 2016
Latest version can be downloaded from:
http://cpham.perso.univ-pau.fr/LORA/WAZIUP/RESSACS16-LPWAN-review.pdf

"Internet-of-Thing and reasons it is becoming a reality"
Keynote at ICCSA 2016
University of Oum El Bouaghi, Algeria
April 12th 2016
Latest version can be downloaded from:
http://cpham.perso.univ-pau.fr/LORA/WAZIUP/IoT-reality.pdf

" Lower Cost, Longer Range Sensing Systems for Surveillance Infrastructures"
Seminar at MAPCI institute
Lund University, Lund, Sweden
March 17th 2016.
Latest version can be downloaded from:
http://cpham.perso.univ-pau.fr/LORA/WAZIUP/LowerCostLongerRange.pdf

8.5. Scientific publications

We provide here some links to selected scientific publications in international peer-reviewed journals and conferences where the WAZIUP project and its advanced mechanisms proposed as part of WP2 have been presented.

1. C. Pham, A. Rahim, P. Cousin, "WAZIUP: A low-cost infrastructure for deploying IoT in developing countries". Accepted for AFRICOMM'2016, Ouagadougou, Burkina Faso, Dec. 6-7, 2016.

2. C. Pham, "QoS for Long-Range Wireless Sensors under Duty-Cycle Regulations with Shared Activity Time Usage". ACM Transactions on Sensor Networks. Vol 12(4), September 2016.

3. C. Pham, "Low-cost, Low-Power and Long-range Image Sensor for Visual Surveillance". Proceedings of the 2nd Workshop on Experiences with Design and Implementation of Smart Objects (SMARTOBJECTS'16). Co-located with ACM MobiCom'2016, New-York, USA, October 3-7, 2016.
   Can be downloaded at:
   http://cpham.perso.univ-pau.fr/Paper/SmartObjects16.pdf

4. C. Pham, A. Rahim, P. Cousin, "Low-cost, Long-range Open IoT for Smarter Rural African Villages". Proceedings of the IEEE International Smart Cities Conference (ISC2), Trento, Italy, Sep. 12-15, 2016.
   Can be downloaded at:
   http://cpham.perso.univ-pau.fr/Paper/ISC2-16.pdf

5. C. Pham, "Building low-cost gateways and devices for open LoRa IoT test-beds". Proceedings of the 11th EAI International Conference on Testbeds and Research Infrastructures for the Development of Networks & Communities (TridentCom'2016), Hangzhou, China, June 13-15, 2016.
   Can be downloaded at:
   http://cpham.perso.univ-pau.fr/Paper/TridentCom2016.pdf

6. C. Pham, "Towards Quality of Service for Long-range IoT in Unlicensed Radio Spectrum". Proceedings of Wireless Days (WD'2016), Toulouse, France, March 2016.
   Can be downloaded at:
   http://cpham.perso.univ-pau.fr/Paper/WD16.pdf

REFERENCES

[1] LinkLab. A comprehensive look at low-power, wide are networks. 2015.

[2] EDN networks. Low power wide-area networking alternatives for the IoT. [http://www.edn.com/design/systems-design/4440343/1/Low-power-wide-area-networking-alternatives-for-the-IoT].

[3] Semtech. AN1200.22. LoRa modulation basics. Rev. 2. May 2015.

[4] RevSpace. Decoding LoRa. [https://revspace.nl/DecodingLora].

[5] Matt Knight. Reversing LoRa. [https://static1.squarespace.com/static/54cecce7e4b054df1848b5f9/t/57489e6e07eaa0105215dc6c/1464376943218/Reversing-Lora-Knight.pdf]

[6] LoRaAlliance. LoRaWAN specification v1.02. [https://www.lora-alliance.org/What-Is-LoRa/Technology]. 2016.

[7] LoRaAlliance. LoRaWAN white papers. [https://www.lora-alliance.org/What-Is-LoRa/LoRaWAN-White-Papers]

[8] Semtech. Wireless RF solutions. [http://www.semtech.com/wireless-rf]

[9] Semtech. AN1200.19. SX127x Reference design overview. Rev. 1. May 2014.

[10] ETSI. Electromagnetic compatibility and radio spectrum matters (ERM); short range devices (SRD); radio equipment to be used in the 25 MHz to 1 000 MHz frequency range with power levels ranging up to 500 mw; part 1. 2012.

[11] Agence de Régulation des Télécoms du Sénégal, DECISION DETERMINANT LES CARACTERISTIQUES ET LES CONDITIONS TECHNIQUES D'UTILISATION DES RESEAUX ET DES INSTALLATIONS RADIOELECTRIQUES EXCLUSIVEMENT COMPOSES D'APPAREILS DE FAIBLE PUISSANCE ET DE FAIBLE PORTEE. 2004-005 ART/DG/DRC/D.Rég.

[12] IBM. LoraWAN in C. [https://www.research.ibm.com/labs/zurich/ics/lrsc/lmic.html]

ACRONYMS LIST

Acronym	Explanation
ABP	Activation by Personalization
AES	Advanced Encryption Standard
AFA	Adaptive Frequency Agility
API	Application Programming Interface
BW	Bandwidth
CSS	Chirp Spread Spectrum
DIY	Do-It-Yourself
DSSS	Direct Sequence Spread Spectrum
ETSI	European Telecommunications Standards Institute
FAQ	Frequently Asked Questions
FEC	Forward Error Correction
FHSS	Frequency Hopping Spread Spectrum
FSK	Frequency Shift Keying
GPRS	General Radio Packet Service
GPS	Global Positioning System
GSM	Global System for Mobile communications
IDE	Integrated Development Environment
IoT	Internet-of-Thing
IP	Internet Protocol
ISM band	Industrial Scientific Medical band
JSON	JavaScript Object Notation
LBT	Listen Before Talk
LOS	Line of Sight
LPWAN	Low Power Wide Area Networks
LTE	Long-Term Evolution
M2M	Machine-to-Machine
MAC	Medium Access Control
MIC	Message Integrity Check
MVP	Minimum Viable Product

NB	Narrow Band
NLOS	Non Line of Sight
OTTA	Over The Air Activation
PAN ID	Personal Area Network ID
PER	Packet Error Rate
PHY layer	Physical layer
QoS	Quality of Service
REST API	REpresentational State Transfer API
RSSI	Received Signal Strength Indicator
SF	Spreading Factor
SNR	Signal to Noise Ratio
SRD	Short Range Device
TTN	The Thing Network
USB	Universal Serial Bus
WiFi	Wireless Fidelity

PROJECT CO-ORDINATOR CONTACT

Dr. Abdur Rahim

CREATE-NET

Via alla Cascata 56/D

Povo- 38123 Trento, Italy

Tel: (+39) 0461 408400

Fax: (+39) 0461421157

Email: abdur.rahim@create-net.org

ACKNOWLEDGEMENT

This document has been produced in the context of the H2020 WAZIUP project. The WAZIUP project consortium would like to acknowledge that the research leading to these results has received funding from the European Union’s H2020 Research and Innovation Program under the Grant Agreement H2020-ICT-687607.