Sensor Networks

LoRaWAN communications for a plant monitoring IoT system using the
B-L072Z-LRWAN1 ARM mbed-based platform

Project 1: Report template
v 1.0.2

Daniel Rodríguez Moya
Date: 2025-01-13

Table of contents

Table of figures

Table of tables

1 OVERVIEW AND INTRODUCTION

This document focuses on the description of the objectives and requirements stablished for the development of the Project 1 [1] of the course "Sensor Networks", which consists of the implementation of the LoRaWAN connectivity to the IoT system [2] developed in "Embedded platforms and connections for IoT" course.

The device is designed to be capable of monitoring the crucial environmental conditions and health status of a plant throughout its life and send that data to an online server to study its evolution over time.

This way, sellers could benefit from a gadget that would offer a warranty to customers and/or allow the adjustment of prices according to possible misfortunes happening to sensitive plants, like bonsais. Another potential application is monitoring the plants inside a smart greenhouse to improve vegetables quality and production numbers. Figure 1 illustrates how the device performs in practice.
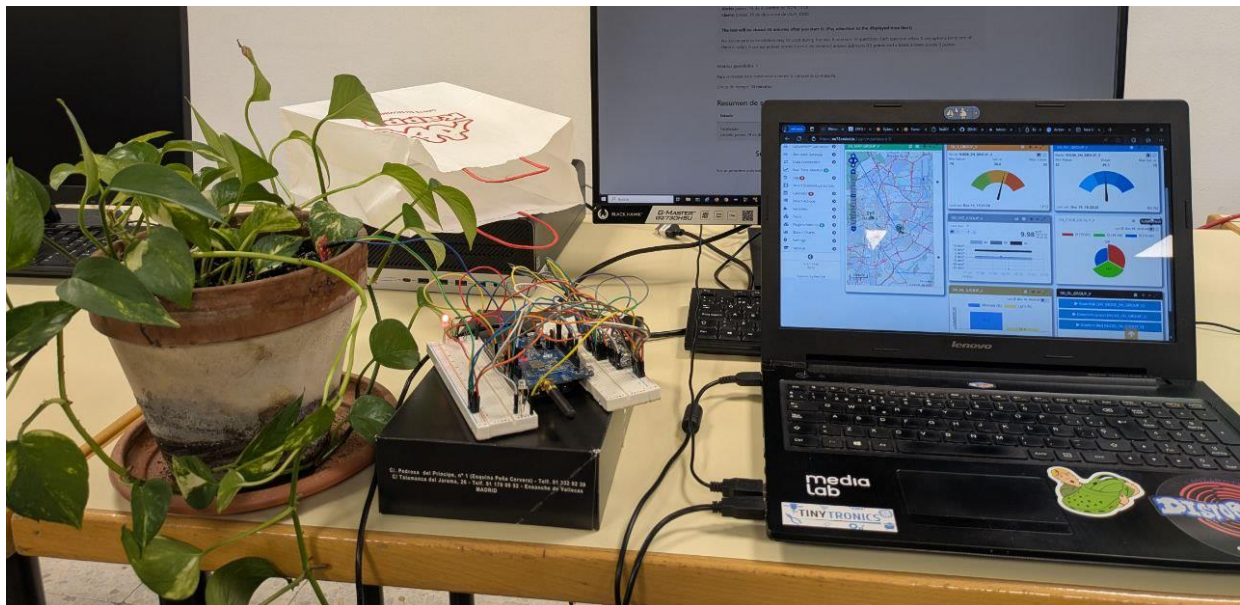

Figure 1: Demo of the project

The variables of interest to be measured are listed in

Table 1.

Table 1: Variables of interest and units

Field	Magnitude	Units
Ambient	Relative humidity	%
	Temperature	$^{\circ}$ C
	Light	%
Soil	Humidity	%
Geolocation	Latitude	$^{\circ}$, ' , "
	Longitude	$^{\circ}$, ' , "
Accelerations	Gravity	m/s²
Colour	Intensity	cd

This way, the election of the hardware to be used has been proposed by the professors [2], bearing in mind a justified dimensioning related to the input/output peripherals and number of computations per execution cycle required for the application.

To begin with, the microcontroller unit (MCU) is the STM32 based B-L072Z-LRWAN1 LoRa/Sigfox Discovery Kit [3], Figure 2. This development board is powered by an STM32L072ZCZ, based on ARM Cortex M0+ core, whose specifications are 192 kb of flash memory, 20 kb of RAM and 20 kb of EEPROM. The module also includes a SX1276 LoRa radio chip to provide the board with long-range and low power consumption data transmission/reception. Moreover, the MCU can access peripherals via 16-bit ADC, LP-UART, SPI, I2C and USB. To complete the pack, the board includes an ST-LINK embedded debug tool interface, built-in LEDs, built-in buttons, antenna for LoRa and Arduino UNO shields connectors, among other features.
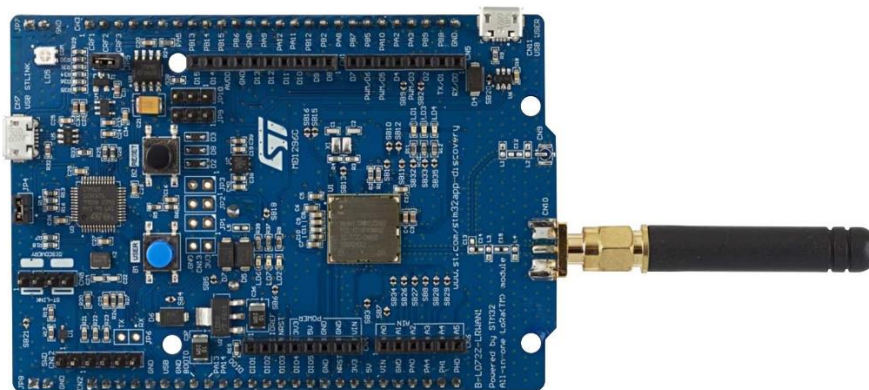

Figure 2: B-L072Z-LRWAN1 LoRa/Sigfox Discovery Kit

On the input devices side,

Table 2 describes the sensors used to measure each variable.

Table 2: Input devices main features

Name	Variable	Range and accuracy	Connection method	Picture
Si7021 [4]	Relative humidity	0 – 80 % (±3 %)	I2C	
	Temperature	-10 – 50 ºC (±0.4 ºC)		
HW5P-1 [5]	Ambient light	0 – 100 %	ADC	
SparkFun Soil Moisture Sensor [4]	Soil moisture	0 – 100 %	ADC	
TCS34725 [7]	Leaf colour	ADC range (bits)	I2C	
MMA8451Q [8]	Accelerations	-8 – 8 g	I2C	
Adafruit Ultimate GPS [9]	Longitude	-180 º – 180 º	Serial line	
	Latitude	-90 º – 90 º		

On the output devices side, an external common anode RGB LED is connected to three digital pins on the board to adjust each channel intensity. Common anode results in an inverse logic programming. Limiting current resistors are required to protect each channel as well.

To be able to implement all the required logic, Mbed OS [10] has been used as it provides an accessible API to develop C++ applications for ARM based microcontrollers and includes a wide collection of handbooks with code examples.

Keil Studio [11] has been chosen to compile and flash the programs into the board. This online IDE is part of the Mbed tool kit and allows developers to easily debug code, WebUSB flash and do version control.

The last piece of software to be used is ResIOT [12], an Italian platform that provides all the necessary services to manage LoRaWAN networks with an abundant number of nodes and gateways. This includes, not only the ability to register and setup devices, but also process and represent data graphically in a friendly and customizable interface.

2 SUMMARY OF THE SPECIFICATIONS IMPLEMENTED VERSUS SPECIFICATIONS REQUIRED

This section must begin with the statement that every specification required [1] in the course has been finally implemented so, rather than presenting a versus, this section will summarize the functionalities.

To begin with, the system must fulfil the system requirements from the second phase, *adding the plant's health monitoring application*, and should fulfil those from the third one, *improving the plant's health monitoring application*. The first phase served as a training to learn how the LoRaWAN library for Mbed [13] worked, register a node in ResIOT, set it up and send a demo variable to populate a line chart with its values. All the requirements are listed in

Table 3.

Table 3: System requirements of the project

Functionality	Reference code	Phase
Add to your mbed-os-example-lorawan project the code (from your previous plant's health monitoring project) to get the GPS location from the GPS module and to send it to the network server.	SR1	Second
Go to your node configuration in ResIOT and modify the LUA code to retrieve the values of the latitude and longitude coordinates.	SR2	
Go to your dashboard tab in the ResIOT server and add a new widget to see the location of your node in a map.	SR3	
Add to your mbed-os-example-lorawan project the code to get at least 3 environmental parameters from the sensors and send them to the network server. You can choose which parameters to use: temperature, light, soil moisture, etc.	SR4	
Go to your node configuration in ResIOT and modify the LUA code to retrieve the parameter values.	SR5	
Go to your dashboard tab in the ResIOT server and add new widgets to show the parameter values.	SR6	
Modify the Mbed code to allow changing the colour of the RGB LED according to the command received from the ResIOT network server. The available commands must be: "OFF", "Green", and "Red".	SR7	
Using the ResIOT mobile application or the Webpage, send any of the defined commands and test the system.	SR8	
Add to the current project all the code from your previous project developed in the course "Embedded platforms and communications for IoT" to get all the parameters from all sensors.	SR9	Third
Convert the data format of the system parameters from string to the most appropriate type to reduce their length. For instance, you can convert the latitude and longitude from string to float to use only 4 bytes per parameter. Configure the data to be able to send the maximum number of parameters.	SR10	
Go to your node configuration in ResIOT and modify the LUA code to retrieve the values of the parameters defined in the previous point.	SR11	
Go to your dashboard tab in the ResIOT server and add new widgets to see the new parameter values.	SR12	

All these requirements are materialized based on the LoRaWAN network depicted in Figure 3 [1].
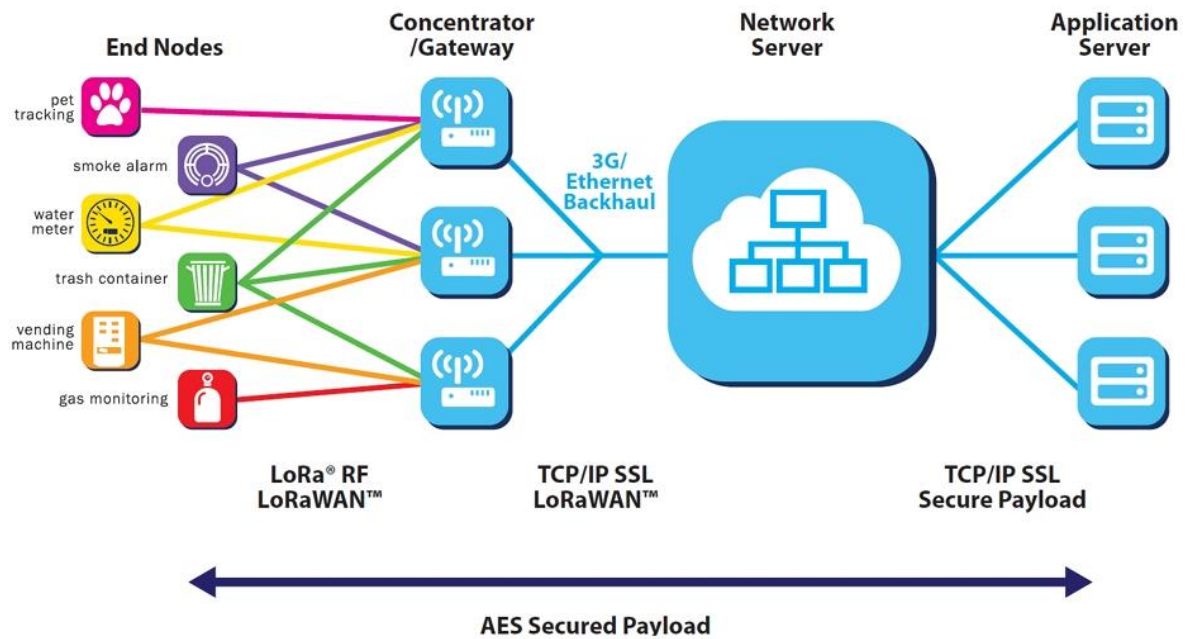
Figure 3: LoRaWAN architecture

For this project, the main areas of work are the full development of a LoRaWAN node, Figure 4, send the data to an already installed gateway by the UPM, Figure 5, and the setup of the node on the network server and application.
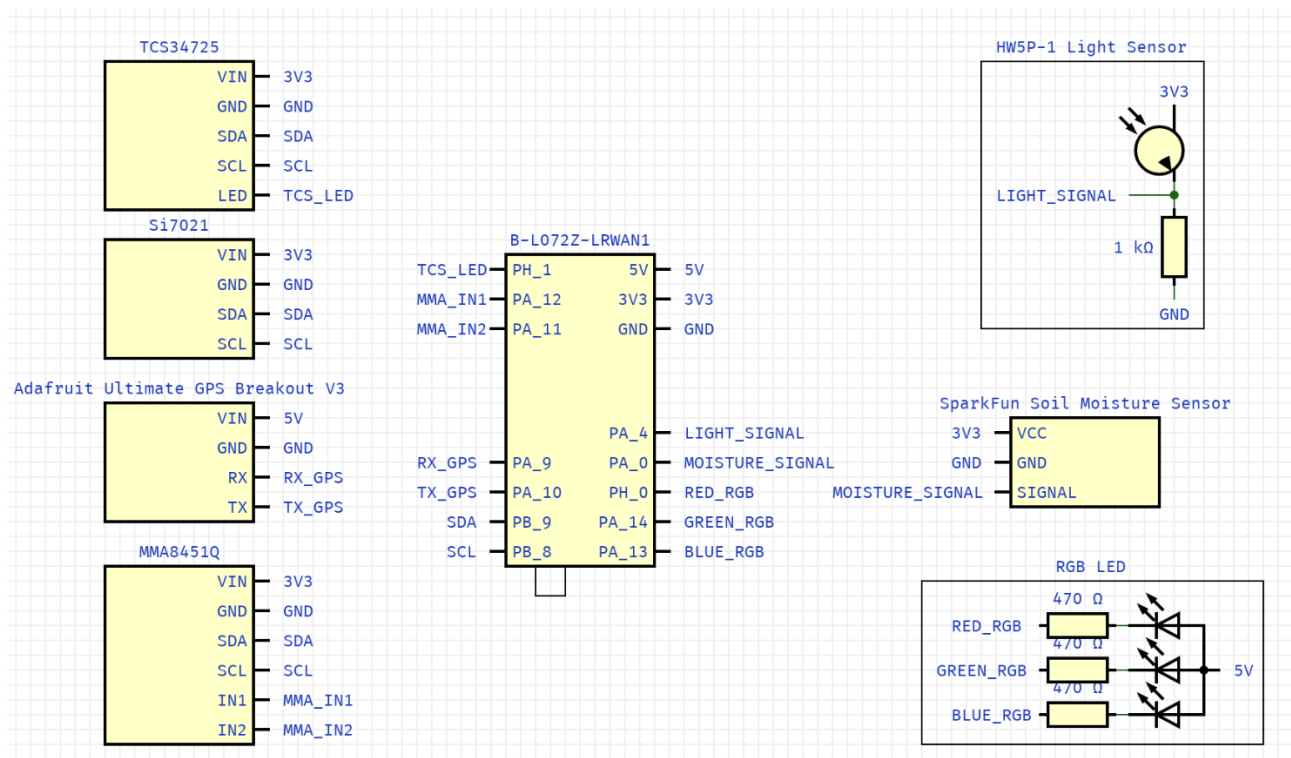
Figure 4: LoRaWAN node developed for Project 1

Figure 5: UPM Campus Sur LoRaWAN gateway location

3 NODE SOFTWARE ORGANIZATION

3.1.1 Description of the implementation

To illustrate the implementation of the developed program, the flowchart in Figure 6 will be used to go into further details.
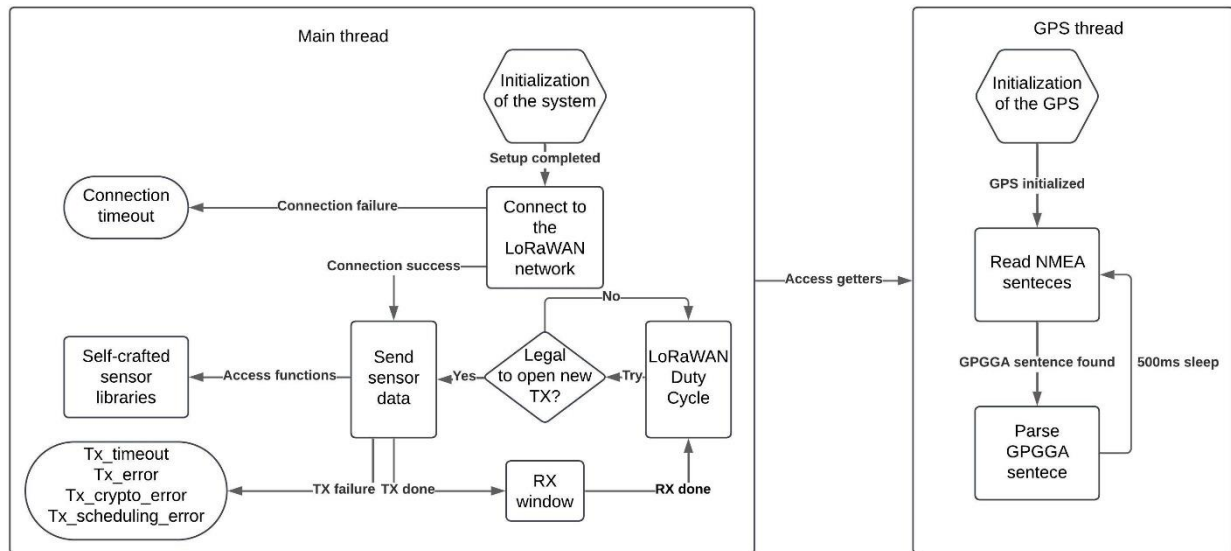
Figure 6: Mbed program flowchart of the project

Essentially, the code consists of the implementation of the *Mbed LoRaWAN state machine* with the addition of self-crafted libraries to read the sensors and a secondary thread to make the GPS operate in the most optimal way without being interrupted by the logic in the main.

If more detailed information is required for the comprehension of the implementation, it is strongly recommended to check the delivered code, as it includes many comments to explain key concepts.

3.1.2 Code size

The application uses **113 KB** of flash memory and **20 KB** of RAM. On the other hand, we have configured the thread sizes with the following memory stack space:

1. The main thread occupies **4096 bytes**.
2. The GPS thread occupies **1024 bytes**.

Figure 7 that has been extracted from the compilation process showcases the main information regarding application's memory.

Figure 7: Application code size

3.1.3 Detected problems and implemented solutions

The main problem developing this LoRaWAN IoT application has been understanding the LUA functions provided by ResIOT [14]. This is a consequence of a step back from the previous iteration of the project that only displayed the variables using Keil Studio's serial monitor.

Now, the libraries that returned the values of the variables in their corresponding magnitudes are just returning the values in bits. The reason behind has been the way the data is packed to be send as a LoRa uplink message. The format is now an array of 8-bit unsigned integers converted to hexadecimal with a maximum allowed size of thirty positions.

By having the sensor readings in bits for this project it is simple to fit them into the TX buffer. *uint8_t* variables just need a conversion to hexadecimal, while *uint16_t* can be decomposed in two 8-bit parts. The low byte goes as a *uint8_t* variable, while the high byte is shifted eight positions to the right. Explained as a decimal number it means multiplying the high byte by 2^8. In the case of *int16_t* variables, like the accelerometer readings, the procedure is the same, however there is an important comment when decoding. Figure 8 displays how the 16-bit decomposition is done in Mbed.

Figure 8: 16-bit variable decomposed into two 8-bit pieces

In the case of floating-point variables, the approach is completely different. As floats are 32-bit variables, the strategy includes the use of pointers to access the binary value of the variable, so that the data sent for latitude and longitude is decomposed in 4 bytes, beginning with the LSB, then shifting the first intermediate byte eight positions to the right, then the second intermediate byte sixteen positions to the right and, the MSB, twenty-four positions to the right. Figure 9 shows the steps to cast a float into a 32-bit unsigned variable.

```
float current_lat, current_lon;
```

```
uint32_t lat_u32 = *(uint32_t *) &current_lat;
```

```
tx_buffer[pos++] = lat_u32 & 0xff;
tx_buffer[pos++] = (lat_u32 >> 8) & 0xff;
tx_buffer[pos++] = (lat_u32 >> 16) & 0xff;
tx_buffer[pos++] = (lat_u32 >> 24) & 0xff;
```

Figure 9: Code lines to transform a float variable into a uint32_t one

This explains the reason behind the decision of sending the raw data from the I2C and analogic sensors instead of sending the floating-point magnitudes. Most of these numbers can be reduced to 2 bytes instead of 4 if this approach is used, and that is a precious save of resources in LoRaWAN applications.

Once the measurement decomposition has been made in Mbed to populate the TX buffer, it is time to decode the byte array in ResIOT using a LUA script. There is a differentiation in how to decompose three types of variables: *uint16_t* or *uint32_t* and *int16_t*.

To begin with unsigned variables, the approach is the one in Figure 10, where the bytes are regrouped in an array, shifted again using the functions provided by ResIOT if they were shifted by the Mbed program and added to deliver the measurement in bits. Then, to get the values in the corresponding magnitudes, the formula provided in each sensor datasheet is used [15] [16] [17]. In addition, measurements have been rounded to the same number of decimal figures that were set for the "Embedded Platforms and Communications for IoT" course.

```
    -- Si7021 --
local temperature_array = {payload[7], payload[8]}
    local temperature_16bit = resiot_ba2intLE16(temperature_array)
    local temperature = ((175.72 * temperature_16bit) / 65536) - 46.85
    temperature = tonumber(string.format("%.2f", temperature))
```

Figure 10: Process of decoding uint16_t variables in LUA

For the signed variables, like the accelerometer readings, a function must be written, Figure 11, to receive the converted to unsigned variables in the TX buffer and change the range from 0 – 65535 to -32768 – 32767.

```lua
-- Define a function to convert unsigned 16-bit to signed 16-bit (uint16_t [0 - 65635], int16_t [-32768 - 32767])
function unsignedToSigned16bit(value)
    if value >= 32768 then
        return value - 65536
    else
        return value
    end
end
```

Figure 11: LUA function to turn uint16_t variables into int16_t

Another challenge to overcome in this project was the rearrangement of the code that came from "Embedded Platforms and communications for IoT" course as the threads and message queues' philosophy that was implemented did not work in this scenario, overloading the MCU capabilities.

The decision that has been taken is to implement all sensors in the main thread while keeping the GPS in its own thread to improve its accuracy and performance. In addition, the message queue to send variables among threads was discarded in favour of using the much lightweight volatile variables.

4 NETWORK SERVER CONFIGURATION

4.1.1 Node definition

As part of the Group V, the OTAA credentials have been created as follows in Figure 12.

```
static uint8_t DEV_EUI[] = {0x86, 0x39, 0x32, 0x35, 0x59, 0x37, 0x91, 0x94};
static uint8_t APP_EUI[] = {0x70, 0xb3, 0xd5, 0x7e, 0xd0, 0x00, 0xac, 0x4a};
static uint8_t APP_KEY[] = {0x86, 0x39, 0x32, 0x35, 0x59, 0x37, 0x91, 0x94, 0x86, 0x39, 0x32, 0x35, 0x59, 0x37, 0x91, 0x94};
```

Figure 12: OTAA credentials for Group V

The *AppEUI* has been provided by the lecturers' team, while *DevEUI* and *AppKey* have been created following the requirements.

4.1.2 LUA Scripting

The scene for Group V is *SN_TEST_V*, where the following steps have been followed to received, decode and set the values on the dashboard.

1. Payload reception and conversion into a hexadecimal byte array, Figure 13.

```lua
-- Define a function to parse payload and decode sensor data
function parsePayload(appeui, deveui, payloadIn)
    -- Decode the payload into individual variables
    payload, Error = resiot_hexdecode(payloadIn)
        if Error ~= "" then
        -- error
        --resiot_debug(Error)
    else
        -- value read correctly
        --resiot_debug(ArrByte)
    end
```

Figure 13: Payload reception and conversion to a hexadecimal byte array

2. Variable bytes regrouping and magnitude formatting. It is remarkable that the function *resiot_ba2intLE16* [14] was used for *uint16_t* variables so that they could be restored from the two-byte little endian hexadecimal values into the unsigned 16-bit ones, Figure 14. For the floating-point numbers, the function used is *resiot_ba2float32LE* [14], which converts four-byte arrays into 32-bit floats. As previously mentioned, there is also implementation to

convert bits to magnitudes and rounding of decimal figures according to the specification of the project.

```lua
local humidity_array = {payload[9], payload[10]}
    local humidity_16bit = resiot_ba2intLE16(humidity_array)
    local humidity = ((125 * humidity_16bit) / 65536) - 6
    humidity = tonumber(string.format("%.2f", humidity))
```

Figure 14: Byte array to magnitudes process

3. Value setting to the corresponding node field and error handling, Figure 15.

```lua
-- Set values to ResIOT tags
local worked, err

-- --------------------------------------------------------------------------------

worked, err = resiot_setnodevalue(appeui, deveui, "ax", ax)
if not worked then
    --resiot_debug(string.format("Error setting ax: %s", err))
end
```

Figure 15: Values setting to the corresponding node fields and error handling

4. Main code to select whether to generate a fake payload when the script is manually run, where bytes are generated randomly to be tested, or receive a real payload from the node itself.

```
-- Main script execution
Origin = resiot_startfrom() -- Scene process starts here

if Origin == "Manual" then -- Manual script execution for testing
    -- Generate a random payload
    math.randomseed(os.time())
    payload = ""
    for i = 1, 28 do
        payload = payload .. string.format("%02X", math.random(0, 255))
    end


    appeui = "70b3d57ed000ac4a"
    deveui = "8639323559379194"
else -- Normal execution, get payload received from device
    appeui = resiot_comm_getparam("appeui")
    deveui = resiot_comm_getparam("deveui")
    payload, err = resiot_getlastpayload(appeui, deveui)
    if not payload then
        --resiot_debug(string.format("Error getting payload: %s", err))
        return
    end
end


-- Process the payload
parsePayload(appeui, deveui, payload)
```

Figure 16: Main code to process payloads depending on the reception mode

4.1.3 Dashboard

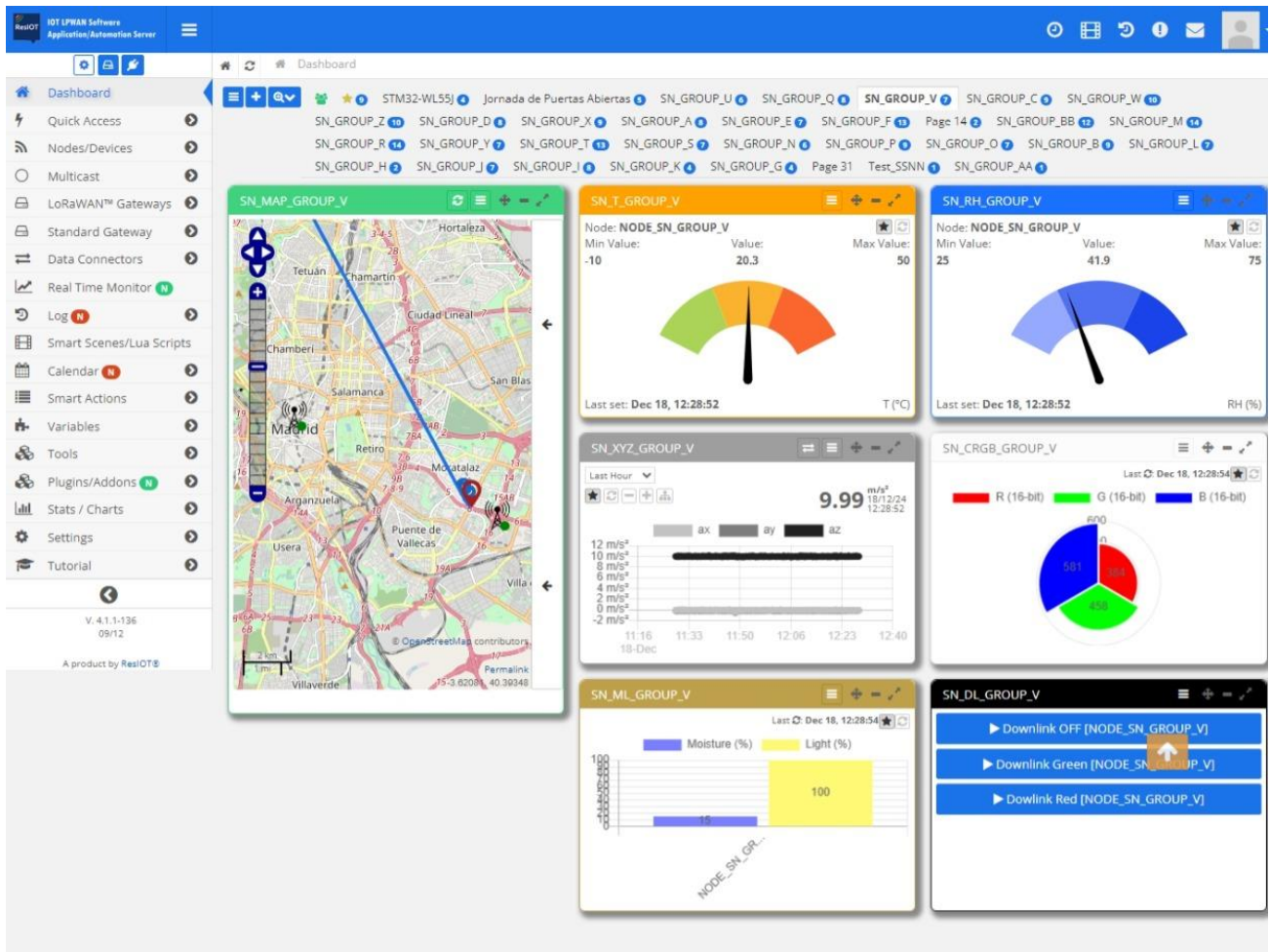Figure 17 displays how the ResIOT dashboard is designed for the application.

Figure 17: ResIOT dashboard for Group V

It is conformed by a set of seven widgets, which are:

1. SN_MAP_GROUP_V: it is a map to display the node's current position provided by the Adafruit Ultimate GPS Breakout v3. The tracking option has been enabled so that the route is visible. In addition, the gateways position has been also added to help users find coverage range.

2. SN_T_GROUP_V: it is a gauge to display temperature of the Si7021 in the required range. Some thresholds have been added to make it more visual.

3. SN_RH_GROUP_V: another gauge for the second variable of the Si7021, relative humidity. Same philosophy as the previous widget.

4. SN_XYZ_GROUP_Z: it is a line chart that groups the three acceleration components from the MMA8451Q. The decision to group them is the shared magnitude and usefulness of be able to compare them in real-time over the same graph.

5. SN_CRGB_GROUP_V: it is a polar area chart where the RGB channels are compared.

6. SN_ML_GROUP_V: it is a bar chart in which both analogic sensor readings are grouped as they are required to be in percentage.

7. SN_DL_GROUP_V: it is a buttons widget where the commands to send downlinks are listed. In the following section, a review of the implementation is taking place as this approach was not compulsory.

5 ADVANCED SPECIFICATIONS IMPLEMENTED

The two advanced specifications implemented have been the addition of all the sensors from "Embedded Platforms and Communications for IoT" using 28 bytes out of the 30 bytes available and the use of commands in LUA format to send the downlinks using dashboard buttons instead of using the Node downlink menu.

The command code is the following, Figure 18.

```lua
Data = "4F4646"              -- Message, "OFF" in hex
Port = "15"                  -- The port number
AppEUI = "70b3d57ed000ac4a"
DevEUI = "8639323559379194"
HexID = "636f6e32"           -- The Connector Id that identifies the Broker/WebSocket/LoRaServer you want to use to send your message
Reference = ""               -- When this particular transmission has been received. Can be left empty
Confirm = false              -- Allows the reference of the comunication
Command = ""                 -- A string that is used to filter determinated messages. It can be left empty

resiot_debug("Sending command to node\n")

Error = resiot_tx(Data, Port, DevEUI, AppEUI, HexID, Reference, Confirm, Command)
if Error ~= "" then
      -- error
      resiot_debug(Error)
end
```

Figure 18: LUA code to send downlinks using commands

Basically, it is just a reinterpretation of the fields that must be filled in the Node downlink tab but using the function *resiot_tx* [14]. The data to be sent in the downlink must be in hexadecimal in this case instead of being the literal string with the word.

6 IMPROVEMENTS AND FUTURE WORK

Listing some areas of improvement for this project, two clear tasks rapidly come to mind.

The first one is to understand the case of use of the application and adjust the data transmission interval to a realistic range. This application uses a Class A LoRaWAN device, Figure 19 [18], which are intended to transmit every set of minutes or hours instead of seconds as it has been currently implemented.
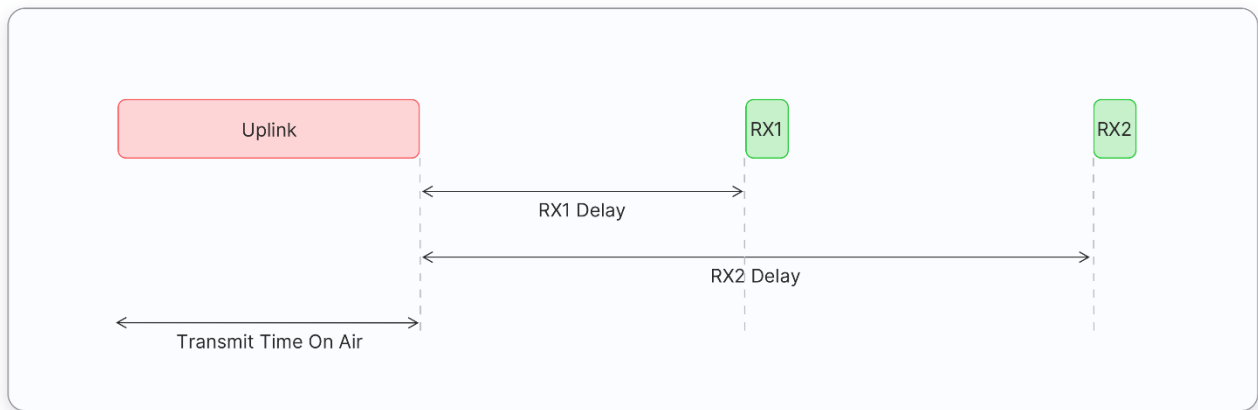

Figure 19: Class A device uplink/downlink windows

A useful tool to adjust these intervals is the GitHub project *Airtime calculator for LoRaWAN* [19], where, according to the payload size, data rates and spreading factors, the maximum number of messages per time interval is calculated considering the 1% duty cycle and the fair access policy.

Secondly, and having chosen the interval, an extremely important consideration is to implement some of the MCU low-power modes [20], as the working times of the application are drastically smaller compared to the waiting times until the next transmission. This way, the device could be powered over batteries for elongated periods of time.

7 REFERENCES

[1] M. Ruiz, E. Barrera, S. Esquembri and P. J. Lobo, "Moodle UPM," November 2024. [Online]. Available: https://moodle.upm.es/titulaciones/oficiales/pluginfile.php/12337101/mod_resource/content/9/SensorNetworks_Project_1_specifications_v1.2.4_2024.pdf. [Accessed 23 November 2024].

[2] G. Azuara, M. Chavarrías, S. Esquembri, E. Juárez, G. Rosa, M. Ruiz and J. Sancho, "Moodle UPM," September 2024. [Online]. Available: https://moodle.upm.es/titulaciones/oficiales/pluginfile.php/12148705/mod_resource/content/10/EmbPlaComIoT_FinalProject_2024_25.pdf. [Accessed 14 October 2024].

[3] ST, "ST," 6 June 2019. [Online]. Available: https://www.st.com/en/evaluation-tools/b-l072z-lrwan1.html. [Accessed 11 October 2024].

[4] S. Labs, "silabs," 2022. [Online]. Available: https://www.silabs.com/documents/public/data-sheets/Si7021-A20.pdf. [Accessed 10 October 2024].

[5] Arduino, "arduino," [Online]. Available: https://wiki-content.arduino.cc/documents/datasheets/HW5P-1.pdf. [Accessed 11 October 2024].

[6] SparkFun, "sparkfun," [Online]. Available: https://cdn.sparkfun.com/datasheets/Sensors/Biometric/SparkFun_Soil_Moisture_Sensor.pdf. [Accessed 20 October 2024].

[7] TAOS, "adafruit," August 2012. [Online]. Available: https://cdn-shop.adafruit.com/datasheets/TCS34725.pdf. [Accessed 22 October 2024].

[8] N. Semiconductors, "nxp," February 2017. [Online]. Available: https://www.nxp.com/docs/en/data-sheet/MMA8451Q.pdf. [Accessed 14 October 2024].

[9] Adafruit, "adafruit," 23 August 2012. [Online]. Available: https://cdn-learn.adafruit.com/downloads/pdf/adafruit-ultimate-gps.pdf. [Accessed 15 October 2024].

[10] a. MBED, "MBED," 2024. [Online]. Available: https://os.mbed.com/mbed-os/. [Accessed 18 September 2024].

[11] arm, "arm KEIL Studio," 2024.

[12] ResIOT, "ResIOT," 2024. [Online]. Available: https://www.resiot.io/en/features/#:~:text=ResIoT%20%C2%AE%20IoT%20Platform%20provides,all%2Din%2Done%20gateway.. [Accessed 29 November 2024].

[13] S. Networks, "UPM drive," 2024. [Online]. Available: https://drive.upm.es/s/7bXCrhakM4oavRO. [Accessed 4 December 2024].

[14] ResIOT, "Docs ResIOT," 2024. [Online]. Available: https://docs.resiot.io/Lua_FunctionList/. [Accessed 5 December 2024].

[15] Adafruit, "Adafruit Si7021 Temperature & Humidity Sensor Breakout Board," [Online]. Available: https://www.adafruit.com/product/3251. [Accessed 24 September 2018].

[16] Adafruit, "Adafruit Triple-Axis Accelerometer - ±2/4/8g @ 14-bit - MMA8451," [Online]. Available: https://www.adafruit.com/product/2019. [Accessed 24 September 2018].

[17] Texas Advanced Optoelectronic Solutions, "TCS3472 COLOR LIGHT-TO-DIGITAL CONVERTER with IR FILTER," [Online]. Available: https://cdn-shop.adafruit.com/datasheets/TCS34725.pdf. [Accessed 24 September 2018].

[18] T. T. Network, "The Things Network," 2024. [Online]. Available: https://www.thethingsnetwork.org/docs/lorawan/classes/. [Accessed 7 December 2024].

[19] avbentem, "GitHub," 5 September 2020. [Online]. Available: https://avbentem.github.io/airtime-calculator/ttn/eu868. [Accessed 13 December 2024].

[20] a. MBED, "Mbed," 2024. [Online]. Available: https://os.mbed.com/docs/mbed-os/v6.15/apis/power-optimization.html. [Accessed 17 October 2024].