LoRaWAN communications for a plant monitoring IoT System

Project report

Diego Aceituno Seoane

Sensor Networks
December 2024

Contents

List of Figures

List of Tables

GLOSARY

ADR Adaptative Data Rate 14

BLE Bluetooth Low Energy 7

FSK Frecuency Shift Keying 6

GFSK Gaussian Frecuency Shift Keying 6

GMSK Gaussian Minimum-Shift Keying 6

GNSS Global Navigation Satellite System 7

I2C Inter-Integrated Circuit 17

IoT Internet of Things 4, 21

LoRaWAN Long Range Wide Area Network 1, 4, 6, 7, 8, 9, 10, 11, 13, 17, 19, 21

MCU MicroController Unit 12, 17

MSK Minimum-Shift Keying 6

NMEA National Marine Electronics Association 17

OTAA Over The Air Activation 13

RAM Random Access Memory 17

RF Radio Frecuency 7

RTOS Real Time Operative System 17

SPI Serial Peripheral Interface 6

1 INTRODUCTION

This document presents the report of the first project developed for the "Sensor Networks" course.

The project is based around the study of wireless sensors design and the implementation of a use case in order to obtain data from remote sensors, applying knowledge in communication protocols and in low capabilities platforms.

The system is centered around a solution designed in a previous course (Embedded platforms and communications for IoT (Internet of Things)). In this course, a solution to obtain biological data from a plant and other parameters such as the GPS location was designed[1]. This solution is presented in the Figure 1 and uses the `DISCO-L072CZ-LRWAN1` board.

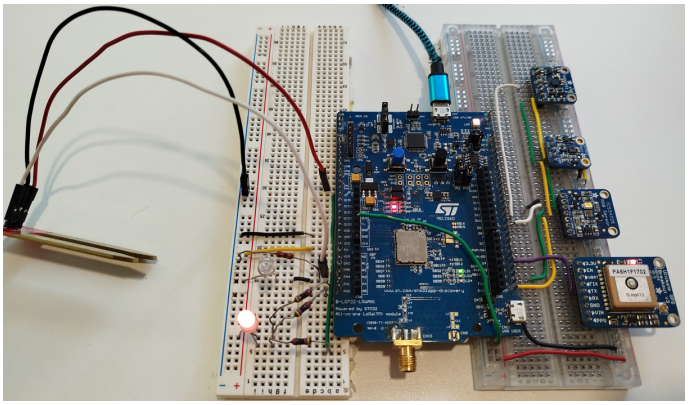

Figure 1: Solution developed in the previous course

To further expand on this idea, the task presented is to modify the system and send the data with wireless communications to allow the final user to see all the information needed regarding the status of the plant remotely. To achieve this, the LoRaWAN (Long Range Wide Area Network) module of the board is used. This module combined with a event queue based solution will send the information of the plant to a remote gateway that will route the data to a network server that will process this into a dashboard.

The final dashboard that presents the information obtained is is presented in Figure 2.

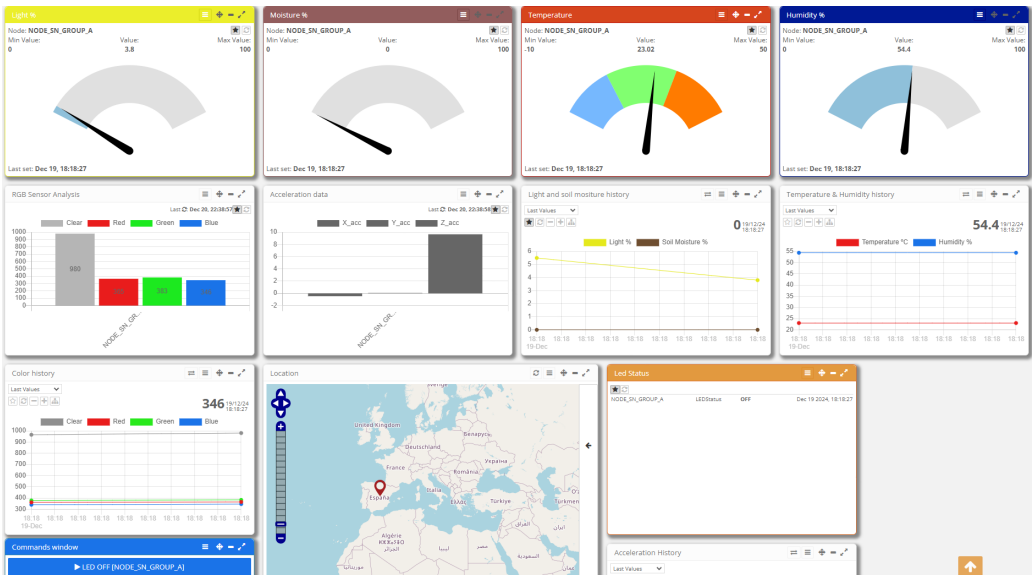

Figure 2: Final dashboard of the system

2 REQUIREMENTS ANALYSIS

2.1 Specifications required and implemented

This section presents in the next tables the list of requirements needed to implement, as well as the implementation status for each specification.

Req. ID	Requirement description	Implemented
Phase2.1	The target must be able to send the location.	Yes
Phase2.2	The target must be able to send, apart from the location, 3 environmental parameters.	Yes
Phase2.3	The dashboard must have a widget to represent the previous parameters values.	Yes
Phase2.4	The dashboard must allow the user to send commands to change the RGB Led. The commands should be: “OFF”, “Green” and “Red”.	Yes

Table 1: Phase 2 requirements implementation status

Req. ID	Requirement description	Implemented
Phase3.1	Get all the parameters from the sensors.	Yes
Phase3.2	Convert the data format of the system parameters from string to the most appropriate type to reduce their length.	Yes
Phase3.3	Modify the LUA code to retrieve the values of all the parameters.	Yes

Table 2: Optional requirements implementation status

2.2 Extra specifications implemented

To further expand the communication capabilities of the system, a frame based communication was designed. To achieve this, the communication was optimized for `30 Bytes`. This design is based around a header byte that allows for:

- Frame version.
- Control and data separation.
- Scalability.

The design for this is described in a dedicated chapter of the document.

3 SYSTEM ARCHITECTURE

The global architecture for the system is presented in Figure 3.

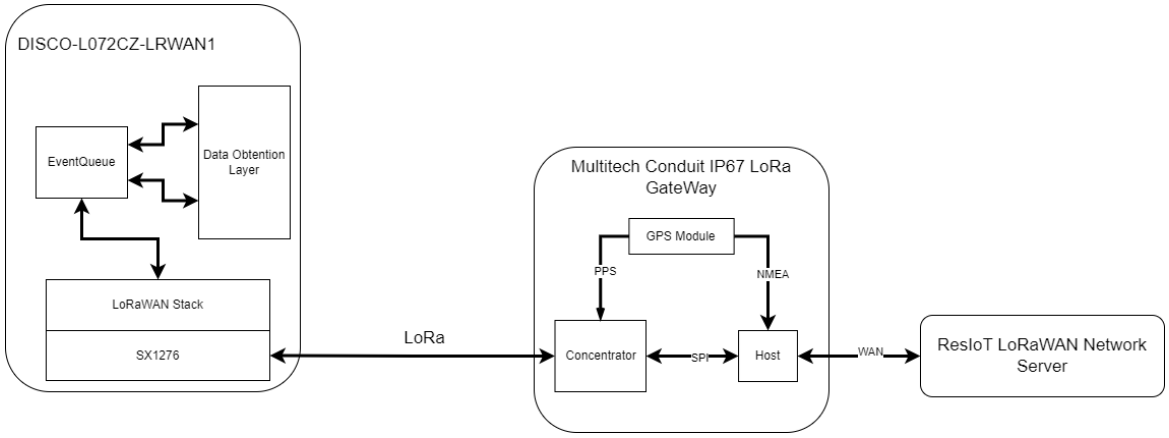

Figure 3: Final system architecture

3.1 DISCO-L072CZ-LRWAN1 and sensors

The board used is the same as the previous project[2]. The sensors connected to this board are as follows:

- The accelerometer is an Adafruit breakout board for the MMA8451Q[3].
- The color sensor is the Adafruit breakout board for the TCS34725[4].
- The temperature and humidity sensor is an Adafruit breakout board for the Si7021A20[5].
- The GPS module is the Adafruit Ultimate GPS Breakout v3[6].

3.2 LoRaWAN module

To communicate the data obtention platform, the board includes the `SX1276`[7] SPI (Serial Peripheral Interface) module in the packaging.

As the radio communication subsystem of LoRaWAN is proprietary, the module is developed by Semtech.

This module implements a LoRa Modem, with some specific characteristics that are going to be transparent for the development of this project but are important to understand:

- Sensibility down to `-148 dBm`.
- An integrated `+20 dBm` power amplifier.
- A bitrate up to `300 kbps`.
- A low RX current of `9.9 mA`.
- Apart from the LoRa modulation, it also supports FSK (Frecuency Shift Keying), GFSK (Gaussian Frecuency Shift Keying), MSK (Minimum-Shift Keying) and GMSK (Gaussian Minimum-Shift Keying).
- Compatibility with another wireless system such as Sigfox.

All this information was obtained from the reference datasheet.

Finally, the high level protocol stack is provided by Mbed-OS and by examples.

3.3 Gateway

To intercommunicate the different protocols stacks of the system, a Gateway is used. The gateway is shown in Figure 4.

Figure 4: Multitech conduit 210/220 series

The gateway is powered by an `32bit ARM9` processor. It's composed by 3 main parts:

- A concentrator that focuses on obtaining data from the LoRaWAN signals. In this case, it uses the Europe frequency band, `868 MHz`.
- A host that processes the RF (Radio Frecuency) messages and communicates with the network serve, in this case, the ResIoT instance. The interface used for this communication is an Ethernet `10/100 Base T`.
- Finally, a GNSS (Global Navigation Satellite System) module to provide location. This module supports GPS, Galileo or QZSS systems.

Some extra features of the gateway that are not used for this project are:

- Wi-Fi communication capabilities.
- BT Classic and BLE (Bluetooth Low Energy).
- Backhaul can also be connected through cellular networks.

All this information was taken from the datasheet of the gateway[8].

The gateway is located in the roof of the Building 8.

4 NODE SOFTWARE

4.1 Description of the implementation

The system extends the functionality of the Mbed-OS LoRaWAN example provided by the teachers, in which an event-queue is provided. This example is developed using Mbed-OS 6.16.0.

This event-queue provides an asynchronous event dispatcher, in this case for LoRaWAN events. And, for any event that happens in the system, a callback will be done to process that situation. This working can be seen in the next figure.

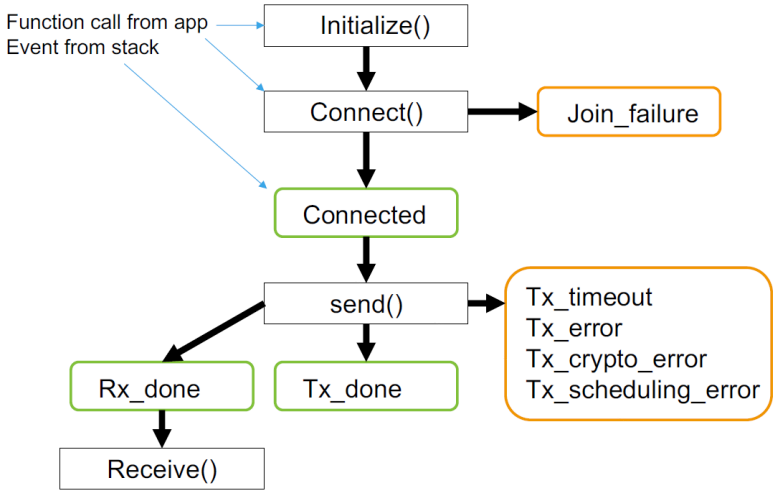

Figure 5: EventQueue process[9]

The solution is implemented by extending the previous event-queue. This event queue will detect possible TX windows in the LoRaWAN PHY layer and create an event to send a new message with data regarding the measurements of the plant sensors. These sensors are: GPS, brightness, soil moisture, humidity, temperature, linear acceleration, RGB data and led status.

To obtain this data, measurement threads are changed from the previous project, these threads are now independent and read new values each 5 seconds, storing the data in a mutex-controlled structure. This structure is later used when the LoRaWAN stack detects a possible transmitting window.

The threads defined for the solution are:

- Main thread with the event-queue.
- I2C thread.
- GPS thread.

4.2 Modules

The modules used for this project are mainly the same from the previous course project, with the only exception being a new module to allocate the shared data for the communication between threads.

The module are also used by the same threads:

- The I2C thread communicates with the accelerometer, temperature & humidity and the color module.
- The GPS thread controls only the GPS module.

4.3 Threads and communication design

LoRaWAN is used, which analyzes a possible transmitting window for the next message. Because of this, a method is needed that ensures the maximum utilization of these windows without depending on the main thread. The final design for this can be seen in the next figure.

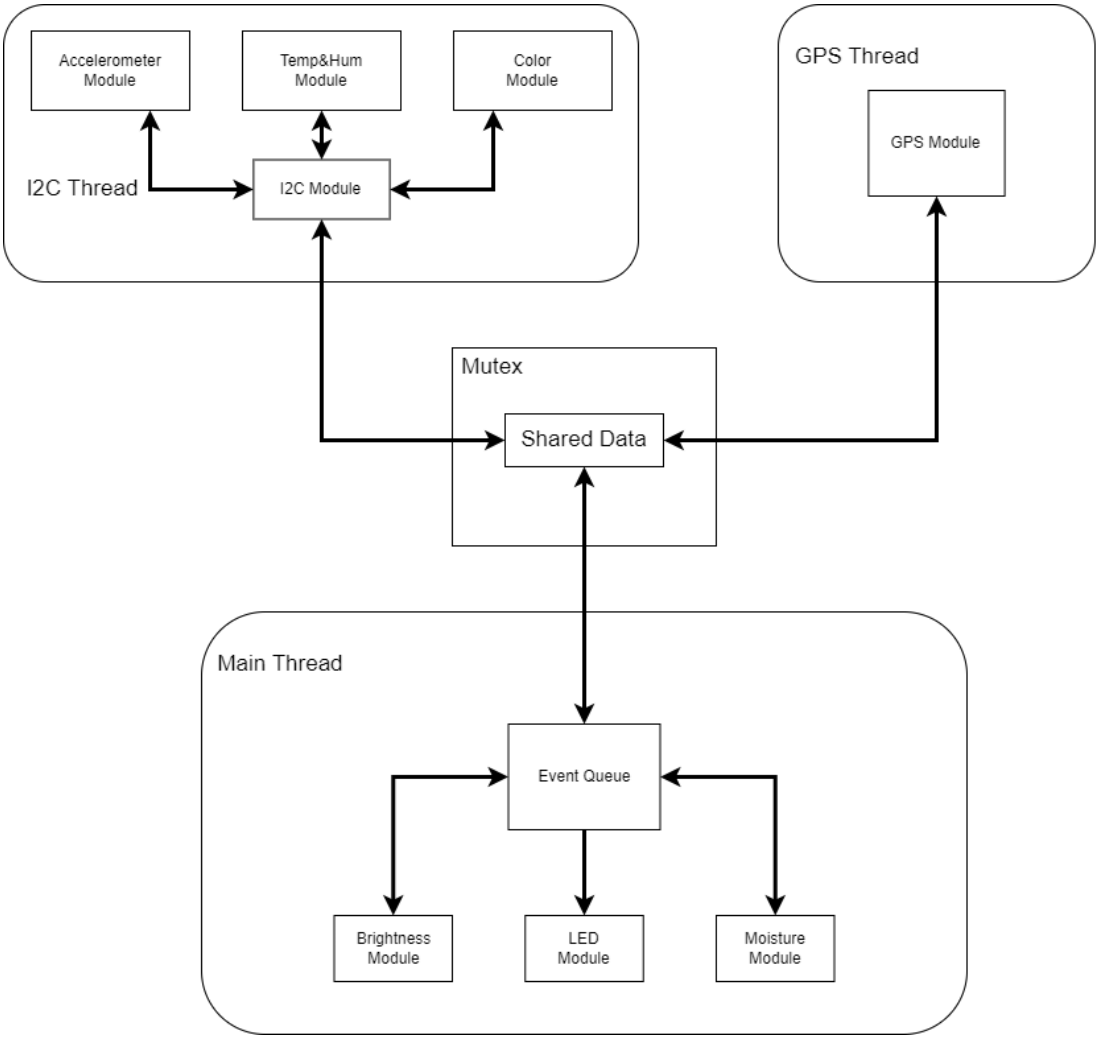

Figure 6: Final module design for the system

In the previous project, the approach was having the main thread controlling the execution of the measuring threads with signals. In this project, that won't work because the main thread is controlled by the event-queue. To solve this and taking into account the best-effort nature of the system, the decision was to make the measuring threads run independently of the main thread, waking themself up every `N` seconds.

The communication between the measuring threads and the main thread had to be redone as well, as in the previous project used an intermediate queue. In this system that could create problems in the next events:

- The system doesn't control the time between readings of the data, so there's a possibility of filling of the queue.
- The system only focuses on sending the last measurements possible for each sensor, so there's no need for previous data that isn't processed.

To solve all this situation, the communication was reduced to a shared structure that contains all the necessary data that is going to be sent through LoRaWAN. The access to this common structure is done using a Mbed-OS mutex, that ensures the protection of the data, allowing access to one thread at a time.

This structure is designed to be directly sent to the LoRaWAN software stack, so it's designed as a network frame. The structure is presented in the appendix and it's explained in the next chapters.

4.4 Frame design

The frame that is sent as payload in the application level of LoRaWAN is presented in the next table.

0	1 … 29
HDR	Message payload

Table 3: Frame structure in bytes, with the header and the message payload

It contains two main parts:

- **The header**: this byte is defined as an advanced implementation, and is explained in the next chapter, but it provides support for message types, versions and led state.
- **The payload**: this structure of 29 bytes contains the data. And the content varies depending on the message type. But the system mainly uses the measurement report type, which contains all the sensor data to cover the requirements of the project. This type is presented in the next table.

0 … 31			
Latitude			
Longitude			
Altitude		Clear	
Red		Green	
Blue		Temperature	
Humidity (`10b`)	X_Acc (`14b`)		Light (`MS 8b`)
Light(2b)	Y_Acc (`14b`)	Moisture (`10b`)	Z_Acc (`MS 6b`)
Z_Acc (`LS 8b`)			

Table 4: Measurement report structure

All the data is sent as *LITTLE ENDIAN*, and for each data the format was defined taking into account the constraint of the `29 Bytes`. The format for the measurement report data is:

1. **Latitude & Longitude** : The latitude and the longitude are transmitted as the whole signed float value (`4 Bytes` each), as any trunking of the value could potentially mean errors of several kilometers.
2. **Altitude**: For the altitude, an unsigned `16 bit` integer value is sent to represent the altitude in meters above the sea level. This is because the minimum for this project is around $700m$ above sea level (Madrid city).

3. **RGB Values**: The RGB Values are sent as raw data from the sensor, as unsigned `16 bit` values. This is done for all the sensor values which are: clear, read, green and blue, that means `8 Bytes` for the color data.
4. **Temperature**: The Temperature is still sent as raw data from the sensor, as a signed `16 bit` integer. This means that the processing to obtain the degrees value must be done in the network server.
5. **Humidity**: Up to this point, all the previous values where divisible by a whole `Byte`, but the message had already used `20 Bytes`, and there was still the acceleration, humidity, light and soil moisture values that needed to be send in `9 Bytes`. To solve this, a encoding was defined for the humidity, light and moisture values to only use `10 bits` on each. This design is part of the problems found chapter.
6. **X_Acc**: The acceleration values are sent as they come from the sensor, transmitting the whole `14 bits` of each axis.
7. **Light**: The percentage value is transmitted using the same encoding for the humidity and soil moisture values in `10 bits`.
8. **Y_Acc**: The Y-axis acceleration `14 bit` signed value.
9. **Soil Moisture**: The percentage value is transmitted using the same encoding for the humidity and light values in `10 bits`.
10. **Z_Acc**: The Z-axis acceleration `14 bit` signed value.

As the whole message with the header uses the `30 Byte` limit of the LoRaWAN stack, no padding was used.

5 ADVANCED HEADER IMPLEMENTATION

To give extensibility to the frame structure, the decision was to add a `header` to the payload content. This gave the possibility to:

- Have different payloads.
- Configure the next payload in reaction to a `DownLink` message.
- Give extensibility to add future payloads.

The `header` implementation is presented in the next table:

0 1 2	3 4 5	6 7
MSG_TYPE	LED_STATE	VERSION

Table 5: Header byte of the frame structure

Each field has a different meaning a use case:

- **MSG_TYPE**: These three `bits` allow for the extensibility of the system, giving the possibility for 8 message types. For now, the next types are defined:
  - 0 - `MEASUREMENT_REPORT`: This type of message is used in `UpLink` to send measurements.
  - 1 - `LED_CHANGE`: This type of message is used in `DownLink` from the server to the node to change the led color. If this type of message is used, the first `Byte` of the payload contains one of the next values: `0x01` for clear, `0x02` for red, `0x04` for green, `0x10` for blue.
- **LED_STATE**: These three `bits` are used whenever the system sends information `UpLink` to add the current state of the LED connected to the MCU. Each bit correspond to the next component:
  - `Bit` 0: Red bit, if set to 1, red component is `ON`.
  - `Bit` 1: Green bit, if set to 1, green component is `ON`.
  - `Bit` 2: Blue bit, if set to 1, blue component is `ON`.
- **VERSION**: These two `bits` allow for future versions without needing to change older codes deployed. For now, these are not used and set to 0.

6 NETWORK SERVER CONFIGURATION

In this case, the Network Server is an instance of `ResIoT`[10]. This instance needs to be configured for each node created.

6.1 Node configuration

For this case, the node follows the next configuration and definitions:

- **Type of activation**: In this project, the node uses OTAA (Over The Air Activation), which gives the possibility of changing the network on startup. This method needs some configuration in the node in the form of some identifiers and an `APP_KEY`.The configuration for the device is done in the main module and can be seen in Figure 7.

```
/**
 * Default and configured device EUI, application EUI and application key
 */
static const uint8_t DEFAULT_DEV_EUI[] = {0x40, 0x39, 0x32, 0x35, 0x59, 0x37, 0x91, 0x94};
static uint8_t DEV_EUI[] = {0x71, 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[] = {0xf3, 0x1c, 0x2e, 0x8b, 0xc6, 0x71, 0x28, 0x1d,
                            0x51, 0x16, 0xf0, 0x8f, 0xf0, 0xb7, 0x92, 0x8f};
```

Figure 7: Configuration for the `OTAA` in the node

  - `DEV_EUI`: The unique identifier specified by the teachers, in this project, the first byte is $0x71$.
  - `APP_EUI`: The unique identifier for the end application in the network server.
  - `APP_KEY`: This key is used to generate the two keys for the MAC layer and the payload.
- **Type of device**: the node works in class A of LoRaWAN. It will open 2 specific RX windows after a TX.

6.2 Node definition in the network server

In the network server, a node was defined with the next characteristics:

- **Name**: `NODE_SN_GROUP_A`.
- **Node AUTH**: `LoRaWAN OTAA Class A`.
- **Device EUI**: The same defined in the node configuration in Figure 7.
- **Application**: The app that shares the same `APP_EUI` as in the Figure 7. In this case, `RRSS2024_AppEUI`, that has the configuration of Figure 8.

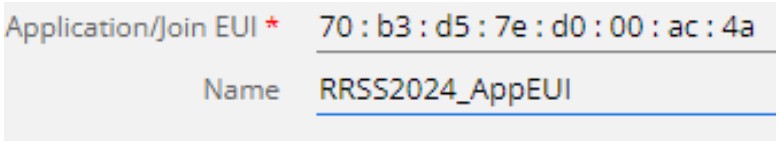

Figure 8: App configuration in the network server

- **APP_KEY**: The same defined in the node configuration in Figure 7.
- **LoRaWAN Network Server**: This field points to the server “eu72.resiot.io 7677”, configured as a LoRaWAN Europe server (`868 MHz`) for Class A+C nodes.

- **Advanced LoRaWAN configuration**: the most important one is that it has ADR (Adaptative Data Rate). The reception windows offset are also by default.
- **Node Fields**: These elements allow for dynamic updates of the dashboard. The ones defined and used are as follows:

Field Name	Content Type	Example value	Predefined in the node?
Latitude	Numeric	40.233921051	Yes
Longitude	Numeric	−3.377102145	Yes
Altitude	Numeric	665	Yes
X_Acc	Numeric	−0.4692	No
Y_Acc	Numeric	0.0574608	No
Z_Acc	Numeric	9.65342109375	No
Temperature	Numeric	23.02	No
Humidity	Numeric	54.4	No
Clear	Numeric	980	No
Red	Numeric	365	No
Green	Numeric	383	No
Blue	Numeric	346	No
Light	Numeric	3.8	No
Moisture	Numeric	15.8	No
LEDStatus	String	OFF	No

Table 6: Configuration of the node fields in the ResIoT application.

Finally, for the node, 4 commands were defined to control the RGB Led connected to the node.

- LED OFF
- LED RED
- LED GREEN
- LED BLUE

6.3 LUA Scripting

On RX

For the automations of the node in the ResIoT application, a smart scene was defined to run on RX. This scene processes all the messages that the board sends.

The scene in this case is the `SN_TEST_A`, with the hex id 73636531313632. It is configured with a maximum number of instances of 10 and a timeout of 600 seconds.

The script allows for the automatic obtention of the “appeui”, “deveui” and payload parameters, both in automatic mode and in manual mode. When used in the manual mode, it injects predefined values to validate the functionality without the connection to the node being required.

When the previous data is obtained, the only function of the script (called *parsePayload*) is called to parse the payload and set the specific node fields for the “appeui” and “‘deveui”’ parameters. This function follows the next structure:

1. The data is decoded as hexadecimal bytes.
2. The header is processed to obtain mainly the `LEDState`, to assign the specific node field.

3. The GPS data such as the latitude, longitude and altitude is obtained using conversions to integers from *LITTLE_ENDIAN* byte arrays. No conversion is done.
4. The color values for the Clear, Red, Green and blue components are obtained and assign. No conversions are done.
5. The Temperature signed `16 bit` value is obtained and converted as the datasheet of the sensor specifies[5]:
$$\text{Temperature} = \frac{175.72 \cdot \text{raw\_temp}}{65536.0} - 46.85$$
6. All the previous data was align to bytes, but the next data is align every `3 bytes`, and for each combination of these `3 bytes`, the lowest `10 bits` represent a percentage value, and the highest `14 bits` represent one of the accelerometer axis. This is done only for the 9 last bytes of the payload.
7. The first `3 bytes` have the humidity(from 0 to 1000) and the X axis acceleration. The humidity is divided by 10.
8. The second `3 bytes` have the light percentage(from 0 to 1000) and the Y axis acceleration. The light is divided by 10.
9. The last `3 bytes` have the soil moisture percentage(from 0 to 10000) and the Z axis acceleration. The soil moisture is divided by 10.

As LUA is a programming language that is not well suited for low level applications, this approach of bit manipulation caused a lot of problems. These problems and the solutions design are documented in a dedicated chapter.

On TX

For any of the commands defined in the ResIoT application for the node, an associated LUA script with few lines is defined. As all the defined commands have to do with the RGB Led, the script follows the same pattern across the commands:

1. Configuration of the port and confirmation in the communication to the node.
2. Configuration of the `DevEUI` and `AppEUI` parameters.
3. Payload specification and sending to the node.

6.4 Dashboard design

The general look of the dashboard is presented in the Figure 9.

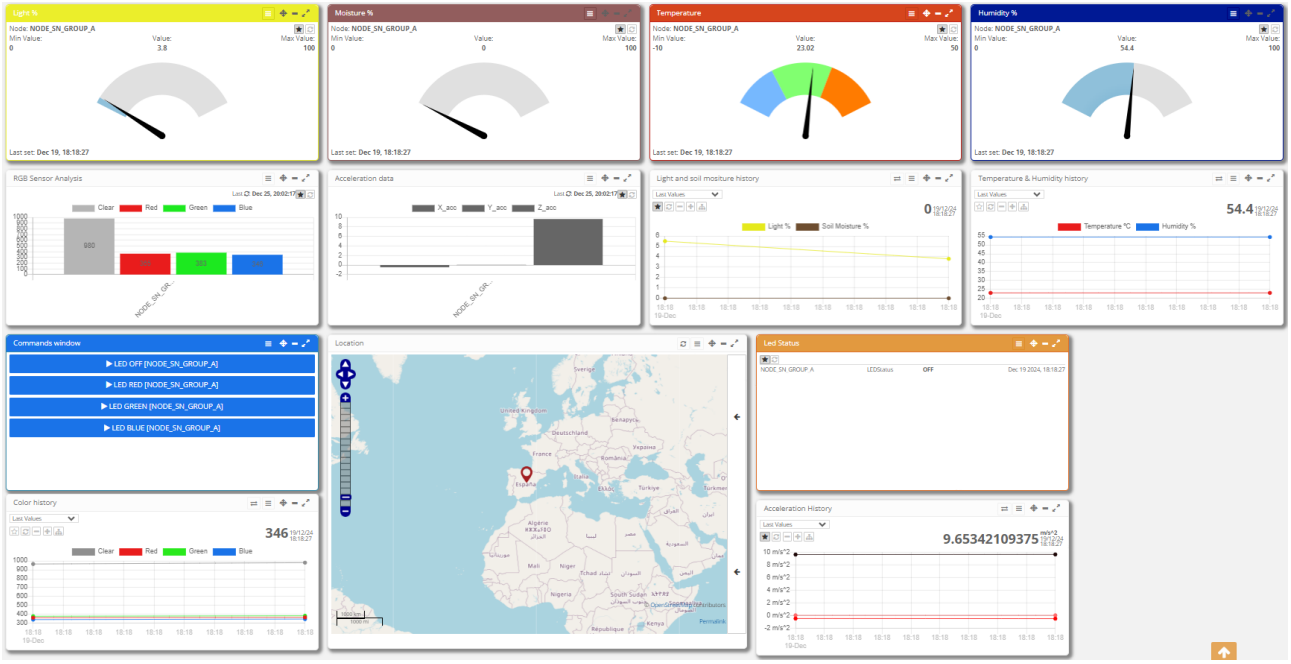

Figure 9: General look of the whole dashboard

Each widget of the above is describing, following the order from the left to the right:

- **Light %**: A gauge widget is used to show the percentage of brightness. Goes from 0 to 100 with 1 decimal.
- **Moisture %**: A gauge widget is used to show the percentage of moisture in the soil from the node. Goes from 0 to 100 with 1 decimal.
- **Temperature**: A gauge widget is used to show the temperature in Celsius degrees with 2 decimals, going from -10ºC to 50ºC, with 3 intervals: blue (-10 to 7), green (7 to 30) and orange (30 to 50).
- **Humidity %**: A gauge widget is used to show the percentage of relative humidity in the node. Goes from 0 to 100 with 1 decimal.
- **RGB Sensor Analysis**: A bars widget is used to represent the next data from the node: clear, red, green and blue values.
- **Acceleration data**: Another bars widget is used to represent the 3 axis of the accelerometer.
- **History widgets**: For the previous data, several widgets are created to represent time series of the changing values, to let the user see problems.
- **Location**: This widget uses the Latitude, Longitude and Altitude values to represent the location of the node in a map by `OpenStreetMap`.
- **Led Status**: A list widget is used to represent the led status, which comes from each message received in the ResIoT server.
- **Commands Window**: A buttons widget is used to let the user send the defined commands for the node from the dashboard.

7 PROBLEMS DETECTED AND SOLUTIONS IMPLEMENTED

7.1 Memory limits

As several software resources for the LoRaWAN stack were used, The amount of RAM (Random Access Memory) used by the system was more than the previous project without the measurement threads.

When the system was being tested, and the threads included in the project, the call to start the threads returned an `osStatus==osErrorNoMemory(0x85)`. This indicated that the RTOS (Real Time Operative System) could not allocate the stack size indicated in the constructor call for the thread.

Another factor that worsen the issue was the queue developed for the communication of the previous project. This queue needed the allocation of more bytes to be able to manage more structures. In reaction to this, the communication decision to use only 1 structure with a mutex was done.

Following the problems with the threads, to further understand the requirements of memory to adjust the stack size for each thread, a stack tracing tool was used[11].

With this tool, the results in Figure 10 were achieved in terms on stack analysis.

```
GPS thread ID: 200022e4
I2C thread ID: 200024c4
Main thread ID: 200028f8
Thread: 0x200028f8, Stack size: 2048, Max stack: 488
Thread: 0x200022e4, Stack size: 1024, Max stack: 656
Thread: 0x20002870, Stack size: 896, Max stack: 352
Thread: 0x200024c4, Stack size: 512, Max stack: 248
Thread: 0x200028b4, Stack size: 768, Max stack: 96
Thread: 0x20003550, Stack size: 1024, Max stack: 200
```

Figure 10: Threads stack usage for problem detection

As can be seen in the figure, ignoring the main thread that is not supposed to be changed, the I2C (Inter-Integrated Circuit) thread only uses around `248 Bytes`, while the GPS thread uses around `656 Bytes`(this is the result of the parsing of NMEA (National Marine Electronics Association) data).

With this in mind, the stack were defined as follows: `512 Bytes` for the I2C thread and `1024 Bytes` for the GPS thread.

7.2 Frame size limits

The LoRaWAN technology used in this project only allows around 1% duty cycle in Europe. Because of this, it is very important for the application to use all the bits of the application message as much as possible. This allows for higher relation between transmitting windows allocation and information exchanged between the node and the network server.

In the case of the project, the frame size is `30 Bytes`, and at the start of the project, sending all the data in one single frame seemed impossible without trunking float values, which results in loss of precision.

As the user sees a lot of values that are only important as percentage, a encoding to metrics such as humidity, light and moisture was designed:

1. The value is processed in the MCU (MicroController Unit), which gives a float % value from `0.0` to `100.0` .

2. Then, the value is multiplied by 10 and converted to an unsigned integer.
3. Finally, as the value goes from `0` to `1000`, it can be expressed with only `10 bits`.
4. When received in the network server, the value can be divided by 10 and we have a percentage value with at least 1 decimal, which is perfect for this use case. This approach only reduces precision percentage values without trunking, while leaving full precision for the acceleration.

This encoding gave enough room to send all the data in `29 Bytes`, and allowed the inclusion of a header byte in the frame.

7.3 Usage of LUA 5.1

The design solution for the frame described above created another problem in the Network Server. The Network Server script uses `LUA 5.1,` which doesn't support the most important bitwise operators[12] such as shifting.

To solve this, the next steps were taken:

1. All the values that were align were send first on the frame structure.
2. As the frame had a moment in which the byte alignment was lost, it was decided to separate acceleration values to make `3 Byte` combinations between one percentage value and one acceleration axis. This meant that the bit masking and shifting operation needed to be designed once.
3. Then, the values were obtained using the function that the ResIoT library provides and simple divisions.

8 TESTING AND VALIDATION

8.1 Incremental integration and testing

In the previous project, GIT was used to keep tracking of integration. This was also used for this project.

The workflow used was the same, each functionality, integration or implementation change was done in a separate branch and tested until it was working as the requirements specified it. When the functionality was working and was tested, it was merged to the main branch.

The milestones for this project were:

1. Definition of the shared data for thread communications.
2. Integration of the threads with the shared data.
3. Refactoring of the shared data to work as a frame structure for LoRaWAN.
4. Final testing, documentation and refactoring.

8.2 ResIoT testing

Sometimes, the connection with the Network Server could not be achieved. In order to keep developing from home, the manual injection of RX events in the smart scripts(see Figure 11) was used to test:

- Correct decoding of not align parts of the payload.
- Correct format sent from the node itself.

```
95 if Origin == "Manual" then
96   payload = "12488b21420c0367c0bc02780197008f0083009464f365ff17d8ff00c00f"
97   appeui = "70b3d57ed000ac4a"
98   deveui = "7139323559379194"
```

Figure 11: Manual injection to test the smart script

8.3 Code Size

As well as the previous project, the results from the compilation in Mbed Studio is presented in the next figures. No changes to the default compilation profiles were done.

Module	.text	.data	.bss
[lib]\c_p.l	12250(+0)	16(+0)	576(+0)
[lib]\fz_ps.l	5730(+0)	0(+0)	0(+0)
[lib]\libcpp_p.l	1(+0)	0(+0)	0(+0)
[lib]\libcppabi_p.l	44(+0)	0(+0)	0(+0)
[lib]\m_ps.l	738(+0)	0(+0)	0(+0)
anon$$obj.o	32(+0)	0(+0)	1024(+0)
main.o	2687(+0)	32(+0)	4102(+0)
mbed-os\cmsis	12491(+0)	168(+0)	4945(+0)
mbed-os\connectivity	42807(+0)	0(+0)	0(+0)
mbed-os\drivers	6776(+0)	0(+0)	636(+0)
mbed-os\events	1924(+0)	0(+0)	0(+0)
mbed-os\features	18(+0)	0(+0)	0(+0)
mbed-os\hal	2386(+0)	8(+0)	114(+0)
mbed-os\platform	9367(+0)	64(+0)	420(+0)
mbed-os\rtos	1704(+0)	0(+0)	8(+0)
mbed-os\targets	22877(+0)	8(+0)	1369(+0)
modules\accelerometer_sensor	408(+0)	0(+0)	8(+0)
modules\brightness_sensor	72(+0)	0(+0)	104(+0)
modules\color_sensor	420(+0)	0(+0)	38(+0)
modules\gps_sensor	869(+0)	7(+0)	1128(+0)
modules\i2c	266(+0)	0(+0)	456(+0)
modules\led	152(+0)	0(+0)	84(+0)
modules\moisture_sensor	68(+0)	0(+0)	104(+0)
modules\temp_hum_sensor	240(+0)	0(+0)	3(+0)
trace_helper.o	2(+0)	0(+0)	0(+0)
Subtotals	124329(+0)	303(+0)	15119(+0)

Total Static RAM memory (data + bss): 15422(+0) bytes
Total Flash memory (text + data): 124632(+0) bytes

(a) Debug compilation

Module	.text	.data	.bss
[lib]\c_p.l	11736(-514)	16(+0)	576(+0)
[lib]\fz_ps.l	5730(+0)	0(+0)	0(+0)
[lib]\libcpp_p.l	1(+0)	0(+0)	0(+0)
[lib]\libcppabi_p.l	44(+0)	0(+0)	0(+0)
[lib]\m_ps.l	738(+0)	0(+0)	0(+0)
anon$$obj.o	32(+0)	0(+0)	1024(+0)
main.o	2315(-372)	32(+0)	4102(+0)
mbed-os\cmsis	11359(-1132)	168(+0)	4817(-128)
mbed-os\connectivity	39548(-3259)	0(+0)	0(+0)
mbed-os\drivers	6057(-719)	0(+0)	636(+0)
mbed-os\events	1636(-288)	0(+0)	0(+0)
mbed-os\features	0(-18)	0(+0)	0(+0)
mbed-os\hal	2246(-140)	8(+0)	114(+0)
mbed-os\platform	8000(-1367)	64(+0)	420(+0)
mbed-os\rtos	1703(-1)	0(+0)	8(+0)
mbed-os\targets	21921(-956)	8(+0)	1381(+12)
modules\accelerometer_sensor	368(-40)	0(+0)	8(+0)
modules\brightness_sensor	60(-12)	0(+0)	104(+0)
modules\color_sensor	380(-40)	0(+0)	38(+0)
modules\gps_sensor	841(-28)	7(+0)	1128(+0)
modules\i2c	236(-30)	0(+0)	456(+0)
modules\led	184(+32)	0(+0)	84(+0)
modules\moisture_sensor	60(-8)	0(+0)	104(+0)
modules\temp_hum_sensor	232(-8)	0(+0)	3(+0)
trace_helper.o	0(-2)	0(+0)	0(+0)
Subtotals	115427(-8902)	303(+0)	15003(-116)

Total Static RAM memory (data + bss): 15306(-116) bytes
Total Flash memory (text + data): 115730(-8902) bytes

(b) Develop compilation

Module	.text	.data	.bss
[lib]\c_p.l	11700(-36)	16(+0)	576(+0)
[lib]\fz_ps.l	5730(+0)	0(+0)	0(+0)
[lib]\libcpp_p.l	1(+0)	0(+0)	0(+0)
[lib]\libcppabi_p.l	44(+0)	0(+0)	0(+0)
[lib]\m_ps.l	738(+0)	0(+0)	0(+0)
anon$$obj.o	32(+0)	0(+0)	1024(+0)
main.o	2319(+4)	32(+0)	4102(+0)
mbed-os\cmsis	9984(-1375)	168(+0)	4817(+0)
mbed-os\connectivity	36292(-3256)	0(+0)	0(+0)
mbed-os\drivers	5586(-471)	0(+0)	636(+0)
mbed-os\events	1482(-154)	0(+0)	0(+0)
mbed-os\features	18(+18)	0(+0)	0(+0)
mbed-os\hal	1662(-584)	8(+0)	114(+0)
mbed-os\platform	6168(-1832)	64(+0)	456(+36)
mbed-os\rtos	1050(-653)	0(+0)	8(+0)
mbed-os\targets	20617(-1304)	12(+4)	1388(+7)
modules\accelerometer_sensor	376(+8)	0(+0)	8(+0)
modules\brightness_sensor	60(+0)	0(+0)	104(+0)
modules\color_sensor	336(-44)	0(+0)	40(+2)
modules\gps_sensor	769(-72)	8(+1)	1131(+3)
modules\i2c	228(-8)	0(+0)	456(+0)
modules\led	136(-48)	0(+0)	84(+0)
modules\moisture_sensor	60(+0)	0(+0)	104(+0)
modules\temp_hum_sensor	232(+0)	0(+0)	3(+0)
Subtotals	105620(-9807)	308(+5)	15051(+48)

Total Static RAM memory (data + bss): 15359(+53) bytes
Total Flash memory (text + data): 105928(-9802) bytes

(c) Release compilation

Figure 12: Code size and module size for every compilation profile

9 RESULTS

A LoRaWAN based system was implemented from the case use study to the implementation, following all the specific requirements told by the teachers of the subject. To further expand the capabilities of the communications, a high amount of the efforts allocated were applied to design an advanced header and frame structure to utilize all the `30 Bytes` limits of LoRaWAN.

To achieve this in the time limit, a GitHub based continuous design and integration was followed. In relation to this, the project was documented with Doxygen , and deployed as a static page with GitHub actions, this information is available at `https://ryvenkappa.github.io/SensorNetworksP1/`, with the final repository of the project being at: `https://github.com/RyvenKappa/SensorNetworksP1`.

Finally, the project allowed the student to understand the limitations and capabilities of LoRaWAN, understanding as well the basis of the standard model design for IoT systems that integrate gateways for intercommunications.

10 REFERENCES

[1] *RyvenKappa/EPCIoT_Final_Project*. [Online]. Available: https://github.com/RyvenKappa/EPCIoT_Final_Project (visited on 12/26/2024).

[2] *DISCO-L072CZ-LRWAN1 — Mbed*. [Online]. Available: https://os.mbed.com/platforms/ST-Discovery-LRWAN1/ (visited on 11/28/2024).

[3] *MMA8451Q-1*. [Online]. Available: https://cdn-shop.adafruit.com/datasheets/MMA8451Q-1.pdf (visited on 11/28/2024).

[4] *TCS34725*. [Online]. Available: https://cdn-shop.adafruit.com/datasheets/TCS34725.pdf (visited on 10/14/2024).

[5] *Support_Documents_TechnicalDocs_Si7021-A20*. [Online]. Available: https://cdn-learn.adafruit.com/assets/assets/000/035/931/original/Support_Documents_TechnicalDocs_Si7021-A20.pdf (visited on 10/14/2024).

[6] *GlobalTop-FGPMMOPA6H-Datasheet-V0A*. [Online]. Available: https://cdn-shop.adafruit.com/datasheets/GlobalTop-FGPMMOPA6H-Datasheet-V0A.pdf (visited on 11/28/2024).

[7] *SX1276*. [Online]. Available: https://www.semtech.com/products/wireless-rf/lora-connect/sx1276 (visited on 12/21/2024).

[8] "MultiTech Conduit IP67 200 Series IP67 Base Station for Outdoor LoRa® Deployments EU868 for Europe", [Online]. Available: https://productos-iot.com/wp-content/uploads/2022/03/MultitechIP67200.pdf.

[9] P. J. Lobo Perea and G. Azuara, *Sensor-Networks-Project1_slides_2024*. [Online]. Available: https://moodle.upm.es/titulaciones/oficiales/pluginfile.php/12337100/mod_resource/content/11/Sensor-Networks-Project1_slides_2024.pdf (visited on 12/22/2024).

[10] *ResIOT® - LoRaWAN® Network Server and IoT Platform*, Oct. 2024. [Online]. Available: https://www.resiot.io/en/home/ (visited on 12/23/2024).

[11] *Runtime memory statistics - API references and tutorials — Mbed OS 6 Documentation*. [Online]. Available: https://os.mbed.com/docs/mbed-os/v6.16/apis/runtime-memory-statistics.html (visited on 12/12/2024).

[12] *Lua-users wiki: Bitwise Operators*. [Online]. Available: http://lua-users.org/wiki/BitwiseOperators (visited on 12/25/2024).

11 APPENDIX

A. Shared data structures for the frame.

```
#ifndef SHARED_DATA_H
#define SHARED_DATA_H
#include "mbed.h"
#include <cstdint>
/**
* @file shared_data.h
*
* @brief Shared data module to define the structures that are sent to the lorawan modules.
*
* @author Diego Aceituno Seoane
*
* @date 16-12-2024
*
*/


/**
* @brief Type of message for the 3lsb of the Header
*/
typedef enum{
    MEASUREMENT_REPORT,
    LED_CHANGE,
} msg_type_t;

typedef struct PACKED{
    uint32_t    latitude  : 32;   /*!< 4B full float latitude */
    uint32_t    longitude : 32;   /*!< 4B full float longitude */
    uint16_t    altitude  : 16;   /*!< 2B full altitude from the gps */
    uint16_t    clear     : 16;   /*!< 2B full clear data from the color sensor */
    uint16_t    red       : 16;   /*!< 2B full red data from the color sensor */
    uint16_t    green     : 16;   /*!< 2B full green data from the color sensor */
    uint16_t    blue      : 16;   /*!< 2B full blue data from the color sensor */
    int16_t     temp      : 16;   /*!< 2B full temp data from the temperature sensor */
    uint16_t    hum       : 10;   /*!< 10b humidity as a percentage with 1 digit * 10 */
    int16_t     x_acc     : 14;   /*!< 14b acceleration in the x axis from the sensor */
    uint16_t    light     : 10;   /*!< 10b light as a percentage with 1 digit * 10 */
    int16_t     y_acc     : 14;   /*!< 14b acceleration in the y axis from the sensor */
```

```
38      uint16_t     moisture  : 10;   /*!< 10b moisture as a percentage
            with 1 digit * 10 */
39      int16_t      z_acc     : 14;   /*!< 14b acceleration in the z axis
            from the sensor */
40  } measurement_report_t;
41
42  /**
43  * @brief Structure for the frame sent to the lorawan application
44  */
45  typedef struct PACKED {
46      uint8_t      version   : 2;    /*!< Header 2b version, used in both
             DL and UL */
47      uint8_t      led_state : 3;    /*!< Header 3b for the led status or
              unused in DownLink */
48      msg_type_t     msg_type  : 3; /*!< Header 3b For message type,
            used in both DL and UL */
49      union{
50          measurement_report_t measurement_report;
51      };
52  } frame_data_t;
53
54  extern frame_data_t frame_data;   /*!< Frame data element */
55
56  extern Mutex frame_data_mutex;    /*!< Mutex to access the structure
        */
57
58  #endif
```

Listing 1: Shared_data.h

B. LUA Script on RX.

```
function parsePayload(appeui,deveui,payload)
    raw_payload = payload
    payload = resiot_hexdecode(payload)
    header = payload[1]
    --El primer byte del payload es un header
    --2 bits de version
    --3 bits para el led, RGB
    --3 de message type
    version = resiot_and64(0x03,header)

    header_led_raw = resiot_and64(0x1C,header)
    header_led = header_led_raw/2^2
    resiot_debug(header_led_raw)
    if header_led==0 then
        worked, err = resiot_setnodevalue(appeui, deveui, "LEDStatus",
            "OFF")
    elseif header_led==1 then
        worked, err = resiot_setnodevalue(appeui, deveui, "LEDStatus",
            "RED")
    elseif header_led == 2 then
        worked, err = resiot_setnodevalue(appeui, deveui, "LEDStatus",
            "GREEN")
    elseif header_led == 4 then
        worked, err = resiot_setnodevalue(appeui, deveui, "LEDStatus",
            "BLUE")
    end

    msg_type = resiot_and64(0xE0,header)


    latitude = resiot_ba2float32LE({payload[2],payload[3],payload[4],
       payload[5]})
    worked, err = resiot_setnodevalue(appeui, deveui, "Latitude",
       latitude)
    longitude = resiot_ba2float32LE({payload[6],payload[7],payload[8],
       payload[9]})
    worked, err = resiot_setnodevalue(appeui, deveui, "Longitude",
       longitude)
    altitude = resiot_ba2intLE16({payload[10],payload[11]})
    worked, err = resiot_setnodevalue(appeui, deveui, "Altitude",
       altitude)



    clear = resiot_ba2intLE16({payload[12],payload[13]})
    worked, err = resiot_setnodevalue(appeui, deveui, "Clear", clear)

    red = resiot_ba2intLE16({payload[14],payload[15]})
    worked, err = resiot_setnodevalue(appeui, deveui, "Red", red)

```

```
42      green = resiot_ba2intLE16({payload[16],payload[17]})
43      worked, err = resiot_setnodevalue(appeui, deveui, "Green", green)
44
45      blue = resiot_ba2intLE16({payload[18],payload[19]})
46      worked, err = resiot_setnodevalue(appeui, deveui, "Blue", blue)
47
48      raw_temp = resiot_int16(resiot_ba2intLE16({payload[20],payload
           [21]}))
49      temp = (175.72*raw_temp/65536.0)-46.85
50      temp = tonumber(string.format("%.2f", temp))
51      worked, err = resiot_setnodevalue(appeui, deveui, "Temperature",
           temp)
52
53      --Empieza la parte NO alineada
54      --la humedad esta en porcentaje multiplicado por 10, usando 10
           bits
55      raw_humidity_bytes = resiot_ba2intLE16({payload[22],payload[23]})
56      masked_humidity = resiot_and64(0x03FF,raw_humidity_bytes)--Mascara
            de 10 bits
57      humidity = tonumber(string.format("%.2f", (masked_humidity/10.0)))
58      worked, err = resiot_setnodevalue(appeui, deveui, "Humidity",
           humidity)
59
60      --Accelerometer
61      raw_Xacc_bytes = resiot_int16(resiot_ba2intLE16({payload[23],
           payload[24]}))
62      masked_Xacc = (resiot_int16(resiot_and64(0xFFFC,raw_Xacc_bytes))
           /4)
63      x_acc = masked_Xacc*9.80665/1024.0
64      worked, err = resiot_setnodevalue(appeui, deveui, "X_acc", x_acc)
65      --resiot_debug(x_acc)
66
67      --Nueva parte-------
68      --Luz esta en porcentaje multiplicado por 10, usando 10 bits
69      raw_light_bytes = resiot_ba2intLE16({payload[25],payload[26]})
70      masked_light = resiot_and64(0x03FF,raw_light_bytes)--Mascara de 10
            bits
71      light = tonumber(string.format("%.2f", (masked_light/10.0)))
72      worked, err = resiot_setnodevalue(appeui, deveui, "Light", light)
73
74      raw_Yacc_bytes = resiot_int16(resiot_ba2intLE16({payload[26],
           payload[27]}))
75      masked_Yacc = (resiot_int16(resiot_and64(0xFFFC,raw_Yacc_bytes))
           /4)
76      y_acc = masked_Yacc*9.80665/1024.0
77      worked, err = resiot_setnodevalue(appeui, deveui, "Y_acc", y_acc)
78      --resiot_debug(y_acc)
79
80      --Moisture esta en porcentaje multiplicado por 10, usando 10 bits
81      raw_moisture_bytes = resiot_ba2intLE16({payload[28],payload[29]})
```

```
82      masked_moisture = resiot_and64(0x03FF,raw_moisture_bytes)--Mascara
            de 10 bits
83      moisture = tonumber(string.format("%.2f", (masked_moisture/10.0)))
84      worked, err = resiot_setnodevalue(appeui, deveui, "Moisture",
            moisture)
85
86      raw_Zacc_bytes = resiot_int16(resiot_ba2intLE16({payload[29],
            payload[30]}))
87      masked_Zacc = (resiot_int16(resiot_and64(0xFFFC,raw_Zacc_bytes))
            /4)
88      z_acc = masked_Zacc*9.80665/1024.0
89      worked, err = resiot_setnodevalue(appeui, deveui, "Z_acc", z_acc)
90      --resiot_debug(y_acc)
91  end
92
93  Origin = resiot_startfrom()
94
95  if Origin == "Manual" then
96    payload = "12488
          b21420c0367c0bc02780197008f0083009464f365ff17d8ff00c00f"
97    appeui = "70b3d57ed000ac4a"
98    deveui = "7139323559379194"
99  else
100   appeui = resiot_comm_getparam("appeui")
101   deveui = resiot_comm_getparam("deveui")
102   payload, err = resiot_getlastpayload(appeui, deveui)
103   resiot_debug("SN_TEST_GG Test Auto Mode\n")
104 end
105 parsePayload(appeui,deveui,payload)
```

Listing 2: rx_.lua