How you can make and set up a “Mini Rete LoRa”

.. and monitor (almost) anything

Paolo Bonelli

Version 5: 28/08/2023

Table of Contents

Preface

Today we talk a lot about IOT (Internet Of Things), meaning objects scattered around the house, the factory or even on the open field, that send data to the Internet in order to monitor environmental parameters and eventually activate alarm messages.

Such systems are already on the market, but, in addition to the their price, the main obstacle for many people looking for a particular application is their lack of flexibility. In other words, a lot of people need to build their IOT system by putting together individual pieces of hardware and software like a LEGO.

Think of a farmer who needs weather variables on the own field, a small town community who needs to monitor rainfall, river levels or landslides in order to have alarm on their cell phone. OK, these guys don't know the basic of electronics but the Arduino revolution taught us that many people can learn very fast to manage these stuff. Furthermore, makerspaces and fablabs are almost everywhere and their communities often make themselves available to ordinary people to help them.

But what technologies can a guy, who wants to make and set up an IOT system, use?

For a private person with little familiarity with electronics, a data transmission technology is needed that uses cheap hardware and possibly easy-to-use open source software; it may also be useful for these systems to work outside the walls of the house, outdoors, where there is no electricity or WiFi facilities. In these cases LoRa transmission technology is an good opportunity and it can be used everywhere.

My job is just that: building pieces of IOT network with inexpensive materials and open source software that should be easy to modify to adapt it to your needs. Minimal Arduino experience and little DIY skills are required.

In this article I want to explain how to build a network of sensors that transmit data over the Internet at low cost, taking the first steps towards an exciting experience, perhaps on a par with the one experienced by Guglielmo Marconi at the end of the XIX century, when, in the park of his villa, he saw, with great jubilation on his rudimentary receiver, a signal transmitted via radio and sent by a transmitter few kilometers away!

What is a MiniReteLoRa and what is it for?

Let's start with some examples of practical applications:

• in a cultivated field there is the need to measure the temperature of the air and the ground, the water content in the soil and so on, but we are far from a house with WiFi and electric energy;

• still in the countryside, we have to control a pump to lift water from a well and fill a tank when it is about to empty, but this is located quite far away; we also want the pump to turn on automatically and follow the operation of the whole system on the Internet;

• we have a greenhouse where it is important to keep the air temperature under control at various points to manage the heating systems or the opening of the air vents, but we don't want to put a lot of electric wires in the middle of the plants;

• we want to install on a hill a small network of rain gauges and vibration sensors to alert the population of a small village in the event of a flood or landslide;

monitoring air and water quality can help to prevent environmental disasters alerting people when the level of pollutants is too high

All of these applications require sensors, data acquisition and transmission electronics, and a receiver node located far away.

What system can be used to transmit the data? The GSM-4G network? A WiFi network? In the first case we would have to pay a fee and equip each sensor-card with a SIM with an energy consumption that does not allow us to power the node with a battery for months; in the second case we would need sensor-boards equipped with WiFi, again with a non-negligible battery consumption and a WiFi/GSM router for Internet with a coverage that could not exceed 100-200 m.

For some years a new transmission technology has become available with the relative Hw and Sw: it is LoRa (Long Range). The acronym indicates a radio signal modulation technique which allows for great sensitivity at the receiving point, resulting in a coverage of kilometers, as the name implies, with the transmitter's output power reduced to a minimum and short transmission times for each sensor-board.

LoRa, in Europe, uses the free 868 MHz (915 MHz in USA) band provided you don't occupy it for more than 1% of the time. Imagine a sensor that transmits small quantity of data every 5 minutes with a transmission time of 70 ms (milliseconds), the bandwidth occupation percentage is about 0.02% which would allow us to have up to 50 sensor boards scattered everywhere!!

The typical consumption of a sensor board is 140 mA in transmission and 0.30 mA at rest, (with the CPU in sleep), therefore the average consumption, always for the same example, is 0.32mA. In these evaluations I have not taken into account the consumption of the sensors, which in any case can be negligible if these are activated only shortly before the transmission of their data.

“The egg of Columbus” for our needs?

Yes, but what difficulties are encountered in building such a network? Is there something “turnkey” on the market? Do you have to be super expert in Hw/Sw?

Here the world of open source helps us: that universe of knowledge available on the Net, free and often made easy by skilled disseminators. In addition, there is a low-cost hardware market that is expanding more and more, offering us sensors, microcontrollers and easy-to-wire solutions. If we are already able to connect a sensor to a microcontroller like Arduino, we can with little effort enter the world of the Internet of Things (IOT) that uses LoRa and build our own MiniReteLoRa.

Now I will try to describe the architecture of what I call "MiniReteLoRa" (literally in english “mini network LoRa”), a completely autonomous network of sensors-transmitters and concentrator-receiver, which does not need to rely on particular external servers and submit to complex protocols.

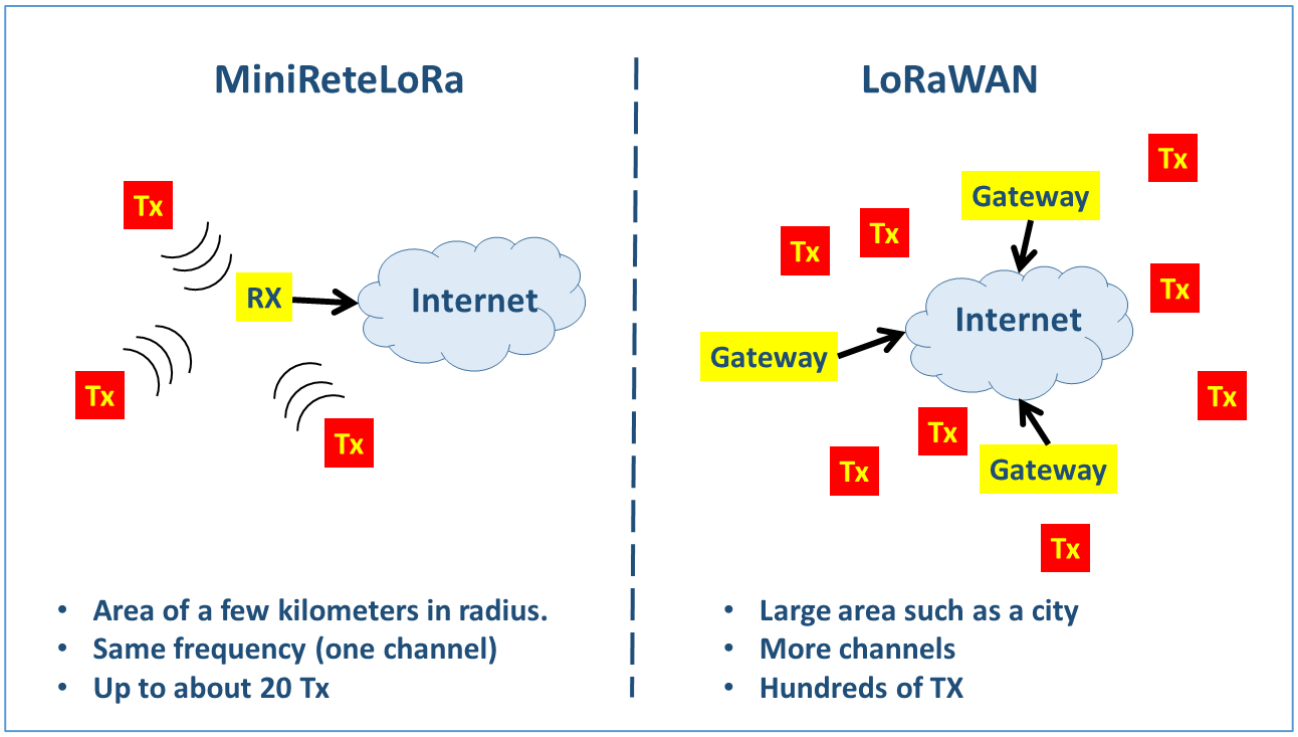


Figure 1 - Difference between MiniReteLoRa and LoRaWAN

A MiniReteLoRa is a technology that allows the acquisition and transmission of quantities detected by sensors, located in mainly outdoor locations, therefore without WiFi connectivity, requiring little energy and low costs. However, nothing prevents us from using this technique even in a closed environment, such as our home or factory.

While the term LoRa indicates the modulation technology of a radio carrier, patented by the Semtech company, the terms MiniReteLoRa and LoRaWAN are used to define a type of device architecture, with their software, which exploit LoRa technology to transmit and receive data .

The MiniReteLoRa, which I describe here, should not be confused with the LoRaWAN protocol, created for a more complex network infrastructure and more suitable for commercial use. LoRaWAN is made for many transmitter nodes and some receivers (gateways) connected on the Internet to a network server managed by the service provider. The LoRaWAN architecture, having multiple gateways in the network, offers greater spatial coverage.

In the Figure 1 I try to illustrate the difference between these two approaches in a simple way, while the Figure 2 shows the detailed scheme of a LoRaWAN network.

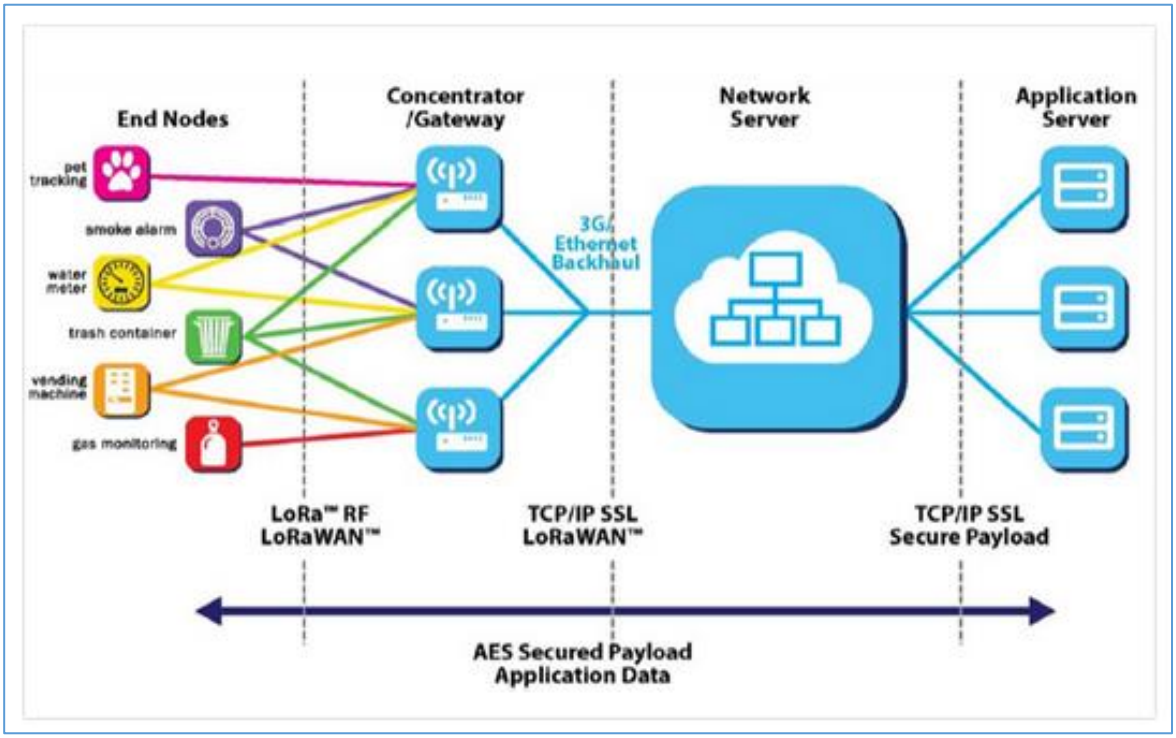


Figure 2 – Detailed scheme of LoRaWAN (from www.internet4things.it)

I designed the MiniReteLoRa to work without third-party infrastructures, so it can be used anywhere and implemented by anyone. The MiniReteLoRa has an architecture that includes:

• several detector/transmitter nodes (TX) which perform the function of acquiring data from the sensors, packaging them in a message which also reports some information on the node and transmitting these messages in broadcast mode with LoRa technology;

• one receiver node (RX) capable of receiving and selecting the messages sent by the TX nodes, extracting the data and presenting them in various ways, such as: display, LED, data logger. Another possibility of the RX node is that of being able to connect to the Internet via a WiFi hotspot and access a server on which to store the received data.

The TX and RX nodes consist of hardware boards equipped with a microcontroller, programmable with the Arduino IDE, and a LoRa radio module, which can be managed with special libraries. These boards can perform both the TX and RX functions, only by changing the software.

Data transmission can take place periodically, independently controlled by each TX node program, or following an external event recorded by the sensors. In any case it is a one-way transmission, the TX is not expected to send data upon request of an RX. This last feature is necessary if the TX node is to consume as little energy as possible, in fact the permanent "listening" state consumes more energy than a periodic transmission.

Typically a TX, which transmits a short 30 byte message for 70 milliseconds every 5 minutes and then goes into a “sleep” state for the rest of the time, can also consume only 0.3 -0.4 mA, while an RX always listening for messages consumes even 100 times as much.

For this reason, the RX nodes must be powered with an unlimited energy source or large-capacity batteries.

This need is even more evident if the RX node is connected to other boards that perform data logger functions or send data over the Internet with WiFi.

In certain particular applications, the TX node, after its switch-on, can listen to a "start" message sent by RX, for example to command the start of several TXs at different times, however, the need always remains for the TX to perform exclusively the data transmission.

Despite the possibility of setting small frequency variations near the main frequency of 868.0 MHz by software, all the TX nodes and the RX node of a MiniReteLoRa must operate on the same frequency. Possible interferences are minimized if the time between two transmissions of the TX nodes is long compared to the frequency occupation time at the moment of transmission. Being the latter time of the order of fractions of a second, a transmission periodicity of a few tens of seconds can be sufficient to guarantee a very low probability of interference between some TX nodes. Even better would be to set a different transmission periodicity for each node, perhaps equal to a sequence of prime numbers of seconds.

At this point in my explanation, someone will be asking a question:

Is a network where data travels one way reliable? In fact there is no feedback between the receiver and the transmitter, if a data is lost the transmitter will never know it.

I try to answer with two arguments:

• The LoRa transmission system, managed by the library that we will use in our applications, has its own internal data integrity control. So if an incoming message to the RX is corrupted, it will not be processed.

• Reliability must therefore be evaluated on the basis of any lost messages compared to the total (messages transmitted but not received or not processed due to internal errors). But the redundancy of messages will increase the reliability of the mini network. Let me explain better: if, for example, one datum every 10 minutes is enough to know the trend over time of an environmental quantity, the TX node will transmit it three or four times every 10 minutes, in order to be sure that at least one arrives at the correct destination.

Why should I choose a MiniReteLoRa?

• Inexpensive hardware;

• Low energy consumption;

• Simple and easy to understand software;

• Consolidated libraries;

• Compatibility with Arduino and its programming environment;

• Reception and archiving of data locally without the need for external servers;

• Possible re-transmission of data over the Internet to different servers.

How do you make it?

Now let's talk first about the hardware needed to make a MiniReteLoRa and then about the software. For simplicity I present only three microcontroller boards, successfully tested by me; but it is good to know that MiniReteLoRa can also be built with other boards by making only very small changes to the software that I will introduce later.

Hardware

There are several boards on the market that can perform the function of TX and RX nodes or both simultaneously, in a MiniReteLoRa. We can also build one of these boards ourselves, simply by putting together a microcontroller and a LoRa radio module purchased separately.

In the following, for reasons of simplicity, I will only speak of 4 boards: 2 suitable for TX nodes and the 2 more powerful for the RX node, also capable of connecting to a WiFi network.

The board: LILYGO®TTGO T-Deer Pro Mini, as a sensor node

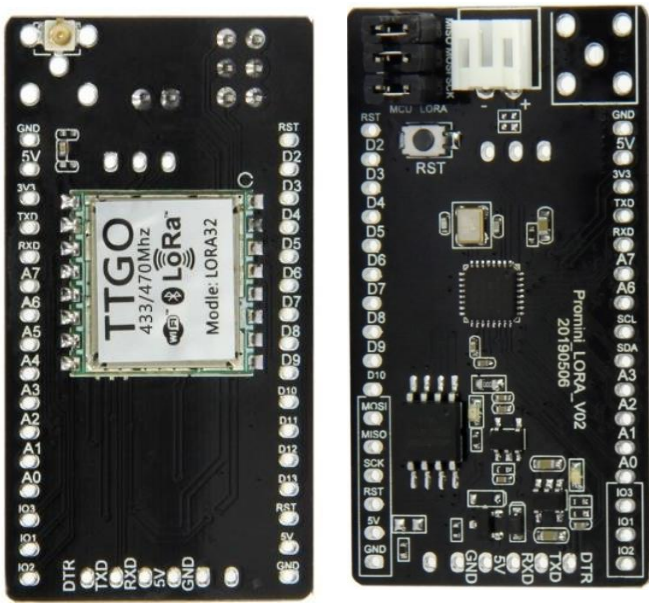


Figure 3- the board T-Deer

Figure 3 shows the LILYGO-TTGO board formed by an ATmega328P CPU clocked at 16 MHz similar to that of Arduino UNO and with a LoRa radio module. This board, like the Arduino Pro Mini, must be programmed with a USB/FTDI cable at 3.3V for the signals and 5V for the power supply, as it does not have a USB socket on board. The FTDI pins are the 6 at the bottom of the figure.

The board is equipped with a JST connector for a possible 3.7 V LiPo battery which is recharged by connecting the board with the FTDI cable to the PC.

Care must be taken when connecting the USB/FTDI cable: the black wire (GND), among the 6 that come out of the FTDI connector, must go to the last one on the left, looking at the first image of the board in the Figure 3. Also be careful to respect the polarities of the LiPo battery: follow those written on the PCB and don't be fooled by the red-black colors of the wires.

Of all the accessible pins on the board, it must be taken into account that many are used for radio functions, more precisely:

D13, D12, D11, D10, D9, D2 intended for communication with the radio module; D8 is free, but in my applications, I use it to connect an LED that signals the transmission or reception of a message; A0 must be connected to D6 if we want to monitor the battery voltage.

So the remaining free pins are: D0 (RX), D1(TX), D3, D5, D7, A1, A2, A3, A4, A5, A6, A7

Keep in mind that if you use D0 and D1 you risk interfering with the serial monitor when we use it during testing.

A6, A7 are analog inputs only, they cannot be used as digital outputs.

The red Power LED, present on the card, can be interrupted, destroying it with tweezers, in order to reduce the current consumption. To notice if the card is working, it is possible to connect a green LED to pin D8 which we will make flash at each message sent.

The LoRa antenna is connected via a "pig tail": a cable which on one side has the SMA female socket on which an antenna suitable for the 868 MHz frequency must be screwed and on the other the U.FL plug which must be connected to the small socket on the PCB.

For more details on this card, I refer to the link:

http://www.lilygo.cn/prod_view.aspx?TypeId=50060&Id=1140&FId=t3:50060:3

The Board ProMiniLoRa as a sensor node

You can assemble a ProMiniLoRa by properly connecting an 8 MHz, 3.3 V Arduino Pro Mini board with a RFM95 radio module, as shown in Figure 4. In Figure 5 you can see the pin diagram. In Figure 6 you can see the board made by me using a PCB for the ProMicroLoRa board modified.

The Arduino Pro Mini board has the advantage to consume less power than an Arduino 16 MHz, 5V.

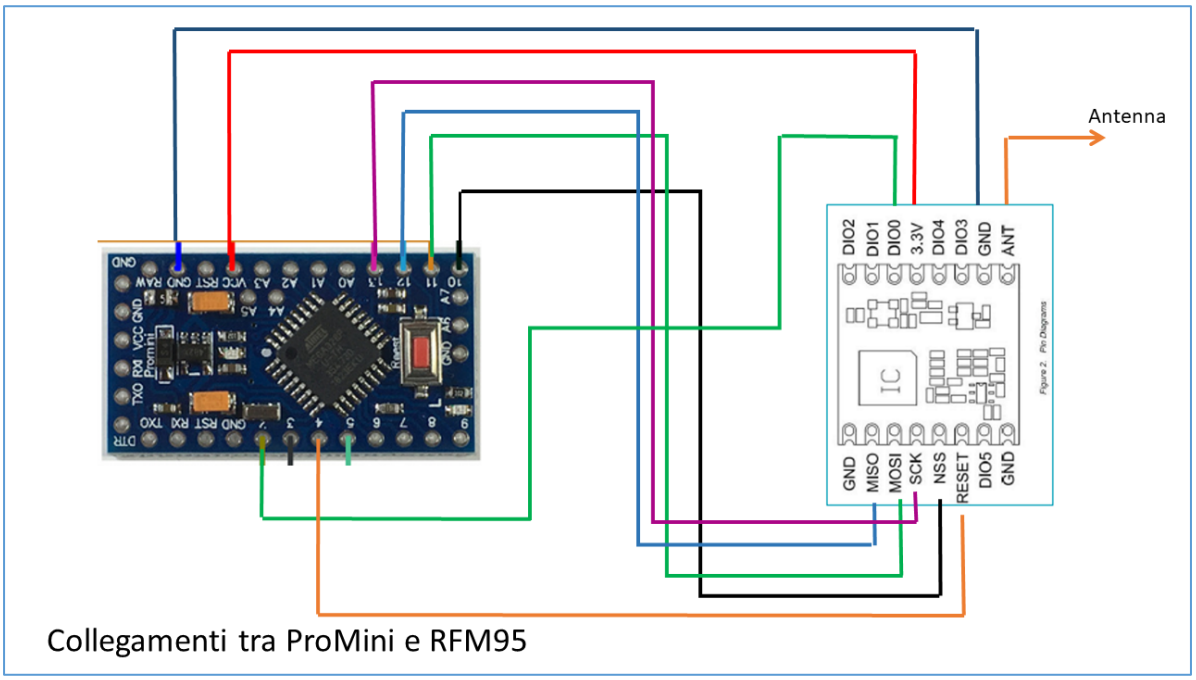


Figure 4 – ProMiniLoRa, wiring between Arduino Pro Mini and RFM95

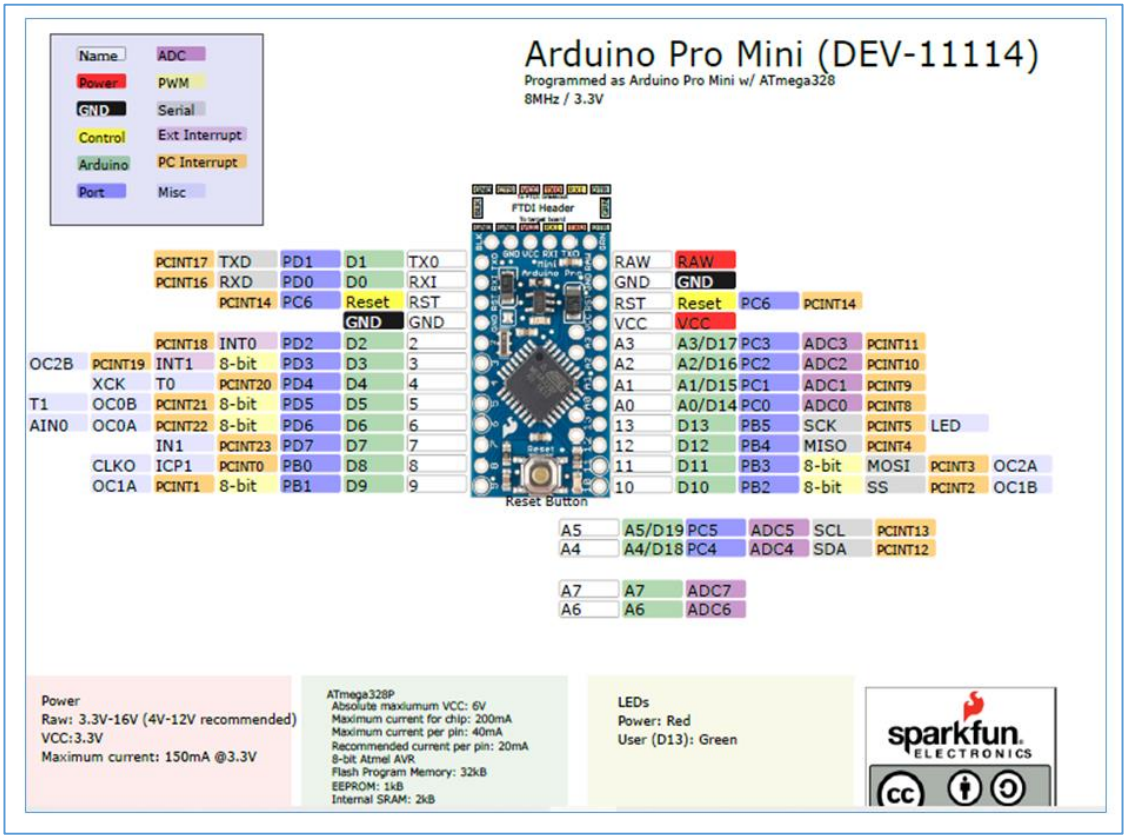


Figure 5 – Arduino Pro Mini pin scheme

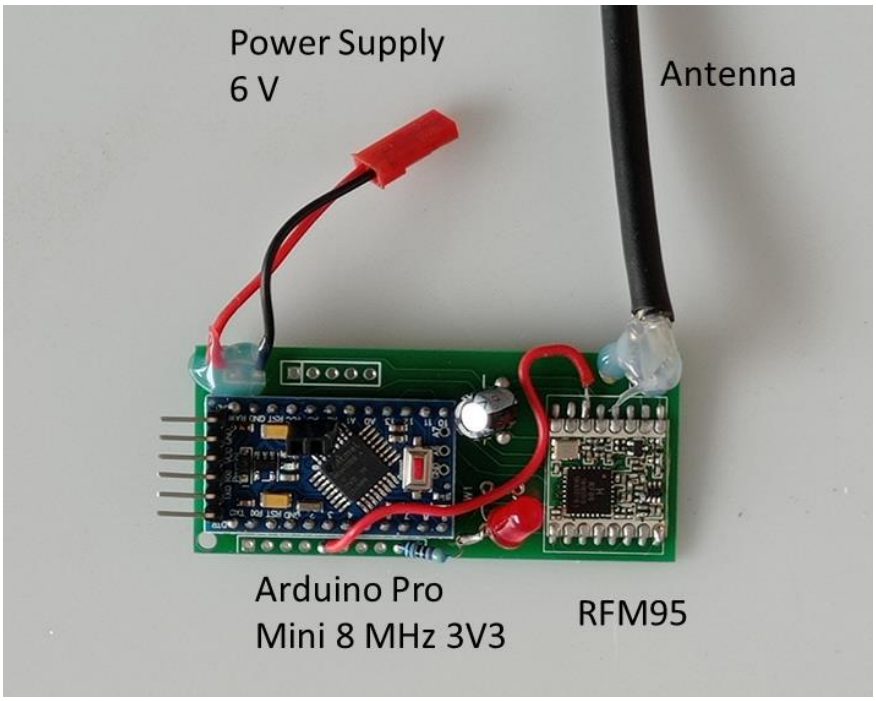


Figure 6 – ProMiniLoRa, the board I made

The connection between the board and the PC must be done with a mini USB - FTDI board set at 3.3V, as in Figure 7.

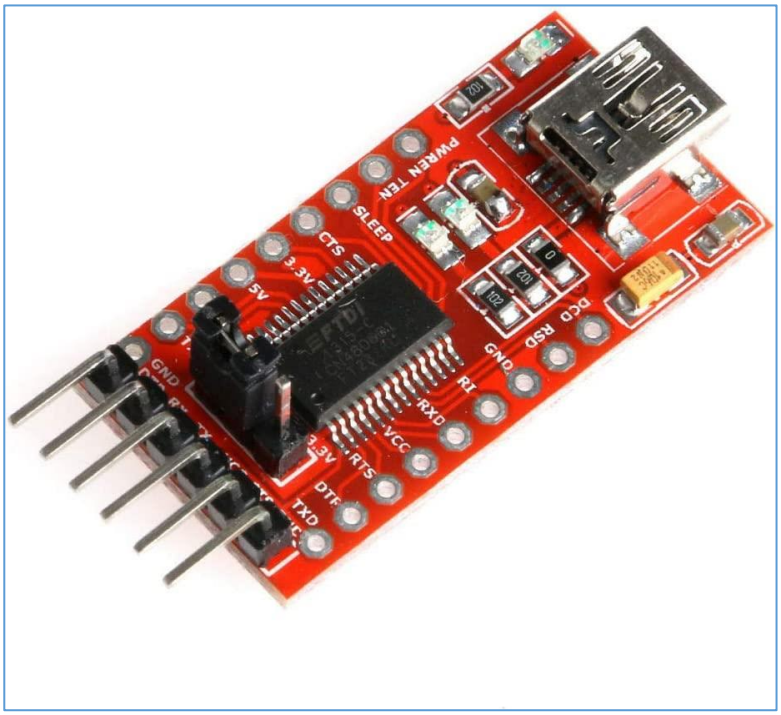


Figure 7 – The FTDI – USB adapter, used for connecting a PC to the Arduino Mini board

The board: LILYGO®TTGO LoRa32 T3_V1.6.1 (LoRa32 V2.1_1.6), as RX node

In brackets is the new name assigned by Lilygo.

https://www.lilygo.cc/products/lora3

The board (Figure 8) can be programmed with the Arduino IDE, using a micro-USB cable. The ESP32 CPU is very powerful, it also has a flash memory, the one that hosts the program, of 4 MB against the 32KB of the Arduino.

The protocol used for the MiniReteLoRa, which we will see later, allows the RX card to receive the messages transmitted by the various TX nodes, decode them and send the data from the TX sensors to an internet server that we can choose at will among those available for free or for a fee.

Figure 9 shows some hardware details of the card, such as the two switches, the LEDs. The board can be powered via the micro USB socket connected to a 5V source, also used for connection to the PC, or with a LiPo battery to be connected to the JST connector. The battery can be recharged by connecting the USB cable to a 5V source.

Figure 8 – The board LoRa32 T3_V1.6.1

The board in Figure 8 is a real "bomb"! It has so many things on board: A powerful ESP32 CPU, a LoRa radio module, an OLED display, a slot for an SD card and above all a WiFi module. The presence of LoRa and WiFi modules allows you to send the data received from the TX nodes via LoRa to the Internet, via WiFi.

The antenna must be connected to the SMA socket on the PCB.

For more details on this card, refer to the link:

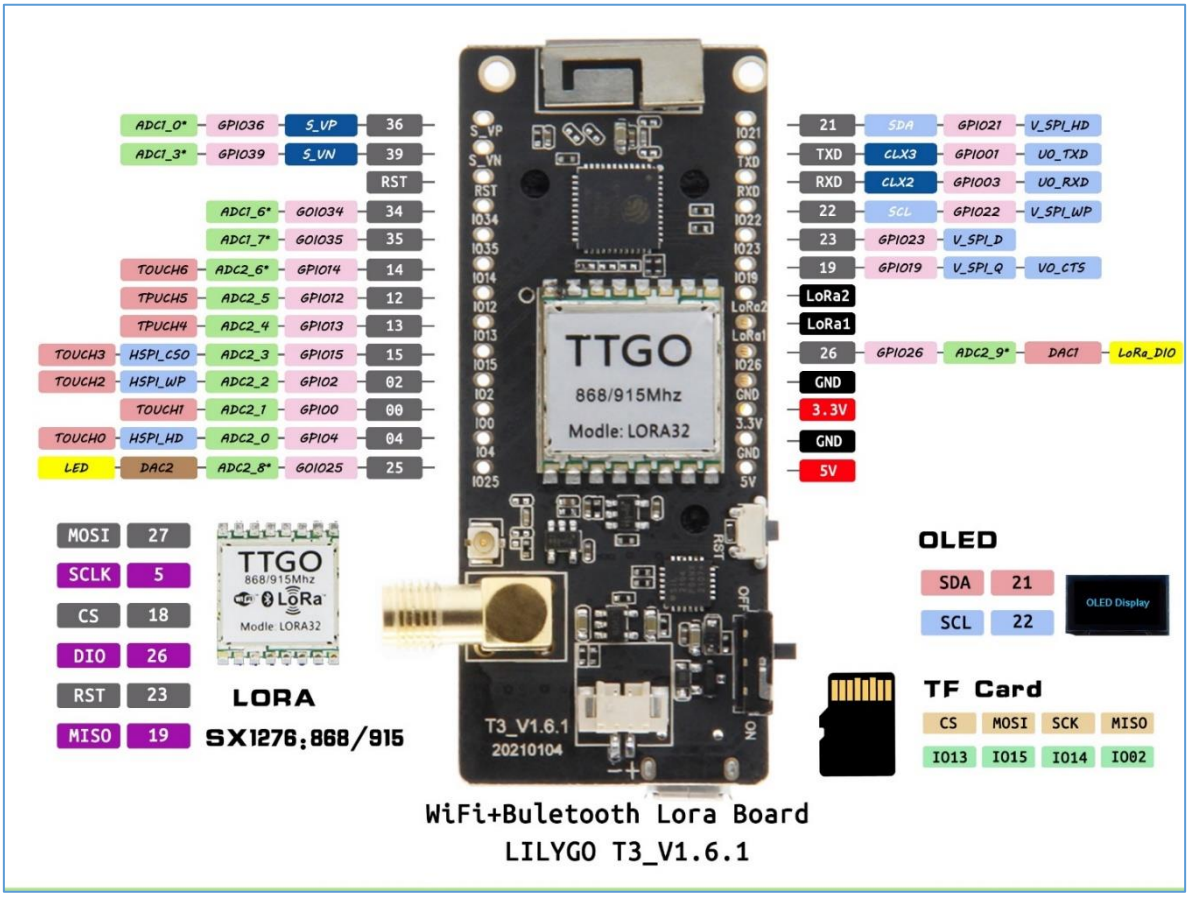
https://www.lilygo.cc/products/lora32-v1-0

Pins used for various devices on the board:

- For LoRa: 5, 14, 18, 19, 23, 26, 27
- For OLEDs (I2C): 21, 22
- For SD card (currently not tested): 2, 4, 12, 13, 14, 15
- For green LED (used by LoRa): 25
- For VBAT: 35

Free pins

12-bit ADC (0 – 4095): 34, 36, 39, 00



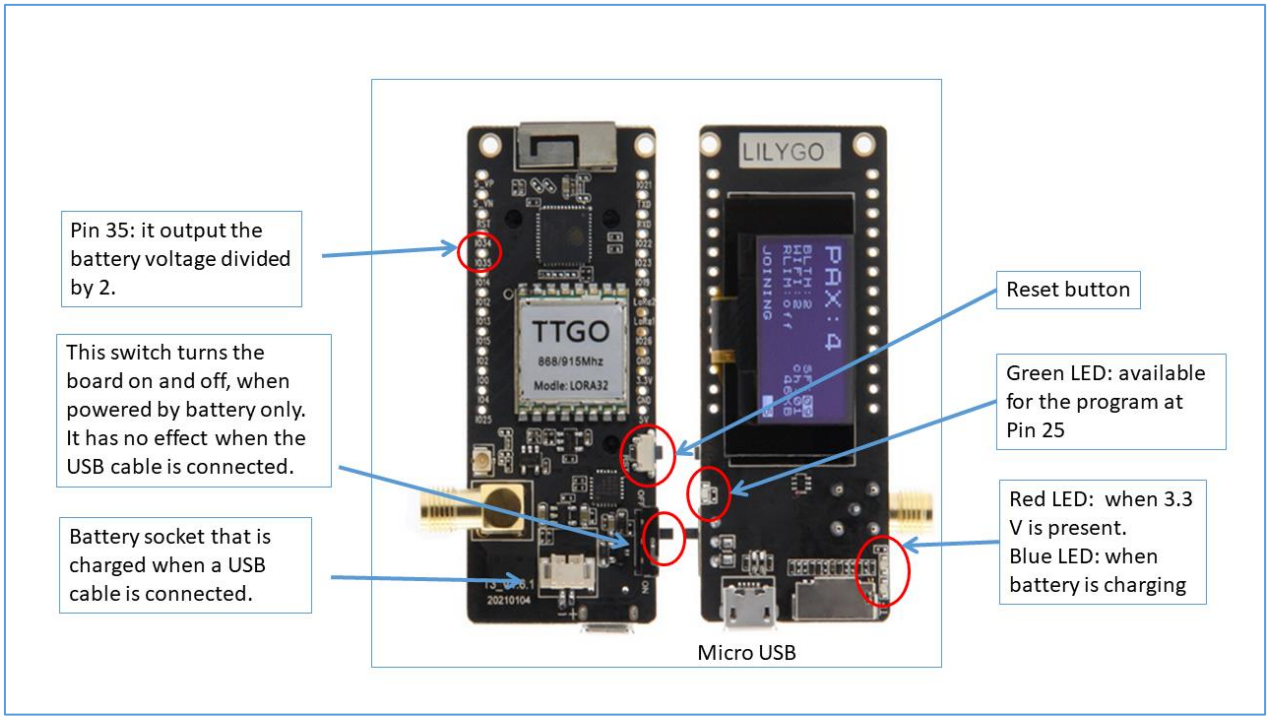


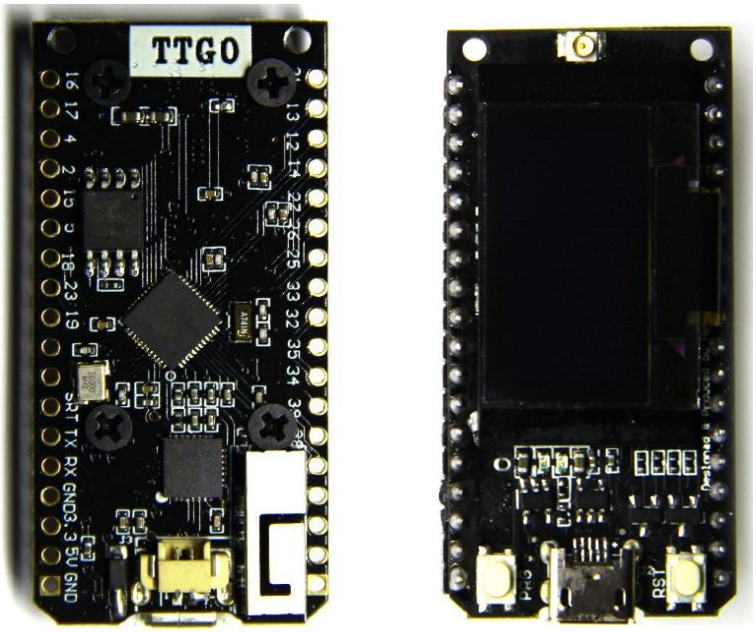
Figure 9 - Some hardware details of the ESP32 board

The board : LILYGO® TTGO LORA32 868/915Mhz (LoRa32 V1.0), as RX node

In brackets is the new name assigned by Lilygo.

This board, even if it has less things on board than the previous one, is equally valid for building a receiver for a MiniReteLoRa. In fact it can be loaded with the same program by changing only a few lines that define the pins of the OLED screen.

I built a complete receiver with this board and a few other components.



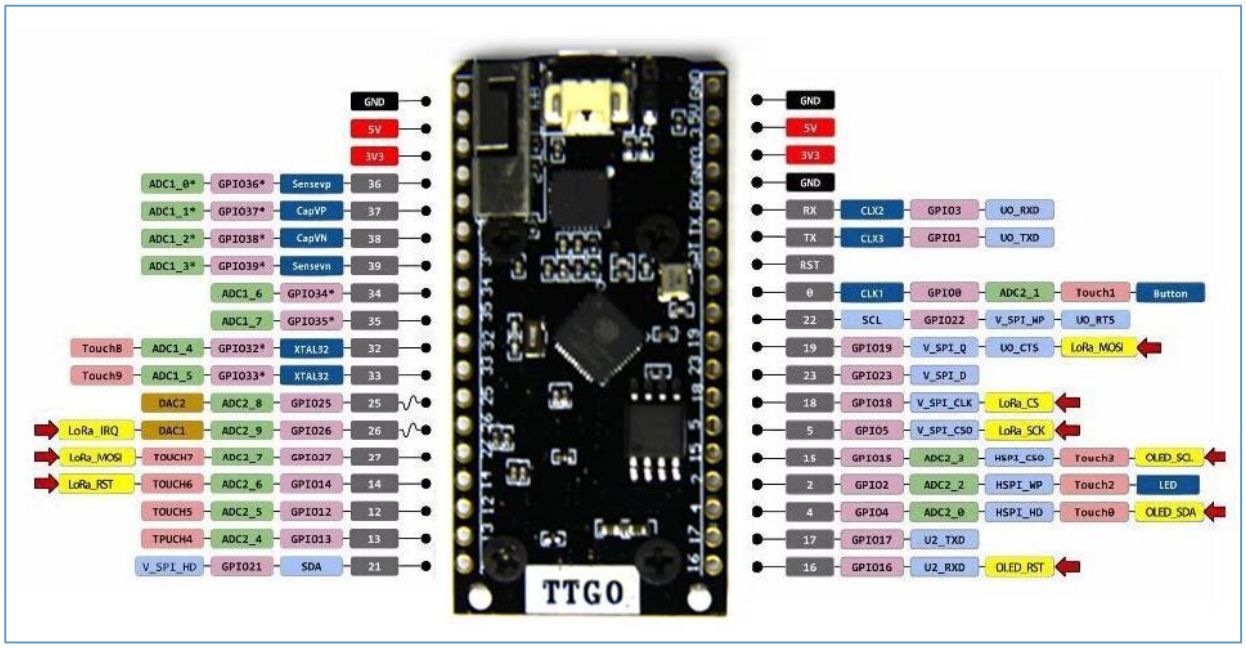


Figure 10 - The board LoRa32 V1.0 (errata: pin 19 = LoRa MISO, not MOSI)

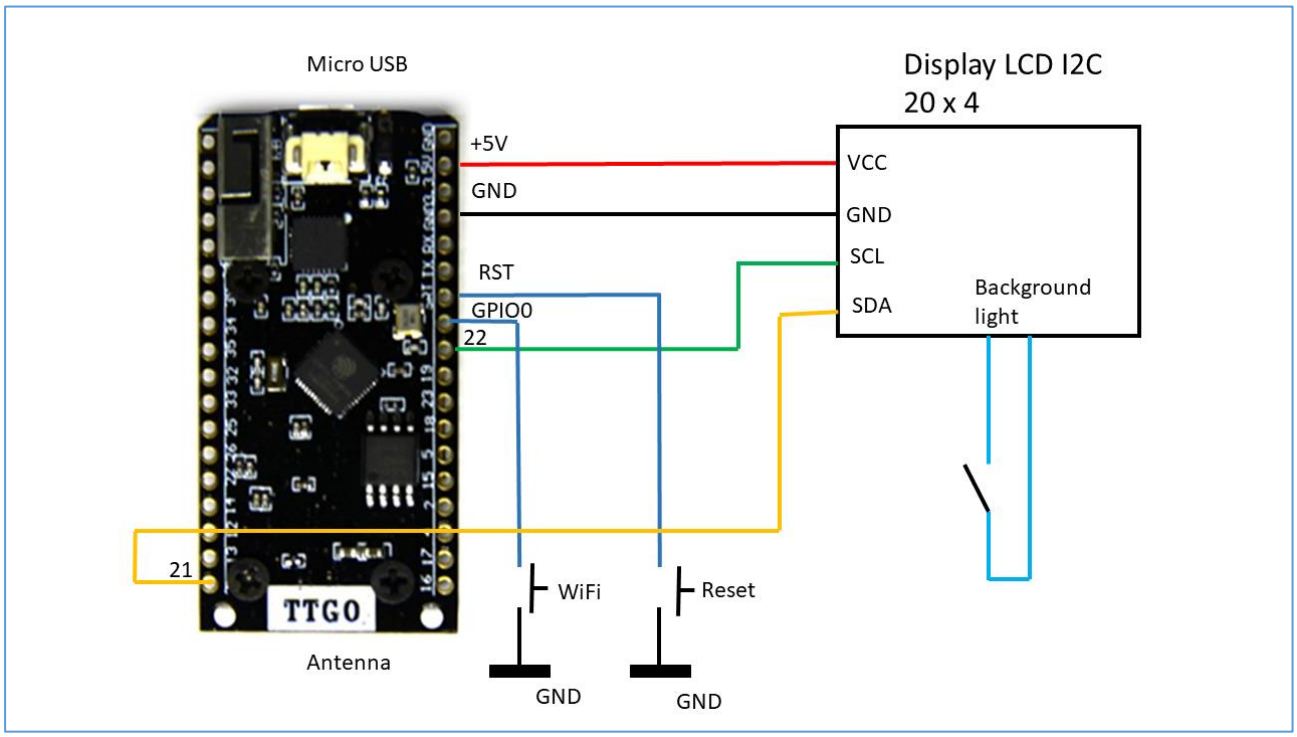


Figure 11 – Wiring for the receiver made with the LoRa32 V1.0 board and explained further

Figure 12 – The Receiver with its LCD display and the two button

Software

This chapter assumes that the reader has minimal C++ programming knowledge such as is needed to program the Arduino. If you've already written a few lines of code with the IDE, it shouldn't be difficult to understand the following. In any case, later you will find a paragraph where I will explain the use of two "Template" programs for both TX and RX, with which the reader will be able to build his applications, adapted to his MiniReteLoRa, changing only some parts of the code without the need to understand all its meaning.

The structure of the sent message

Before examining the software to be loaded on the TX and RX nodes, I will explain how the message transmitted by the TX nodes in a MiniReteLoRa is made.

Following Figure 13 we see that the transmitted message is composed of a series of bytes: some constitute the "preamble" and the terminator of the message, others, those that interest us, are the so-called "payload" which will contain our precious data.

For those who want to know more about the entire message format used by the `RadioHead.h` library, which manages the radio module (Transceiver), the composition of the transmitted packet is shown in Figure 13 .

Packet Format

All messages sent and received by this **RH_RF95** Driver conform to this packet format:

- LoRa mode:
- 8 symbol PREAMBLE
- Explicit header with header CRC (default CCITT, handled internally by the radio)
- 4 octets HEADER: (TO, FROM, ID, FLAGS)
- 0 to 251 octets DATA
- CRC (default CCITT, handled internally by the radio)

from: https://www.airspayce.com/mikem/arduino/RadioHead/classRH_RF95.html

Figure 13 - Complete structure of a LoRa message

We can put whatever we want in the payload and its length can be variable (up to 251 bytes), but it is here that it is important to design a message structure useful for their management during reception. What I have successfully experimented with is the structure visible in Figure 14. Here the payload is made up of a fixed

length part (8 bytes), which I call the header, and a variable length part where the numerical values of the quantities, measured by the the various nodes, will go. The latter is of variable length because I could have different sensors for each TX node.

The numerical values of the quantities measured by the sensors, in the message, are encoded in internal binary format and not in character. Let me explain better with an example: let's imagine the number 25.43 which represents the degrees Celsius of a temperature sensor, its representation could be in ASCII character format (character "2", character "5", character ".", character "4", character "3") for a total of 5 bytes, or in internal format as a 4 byte float variable, or, always in internal format, as a 2 byte int variable if I multiply the value by 100 bringing it to 2543.

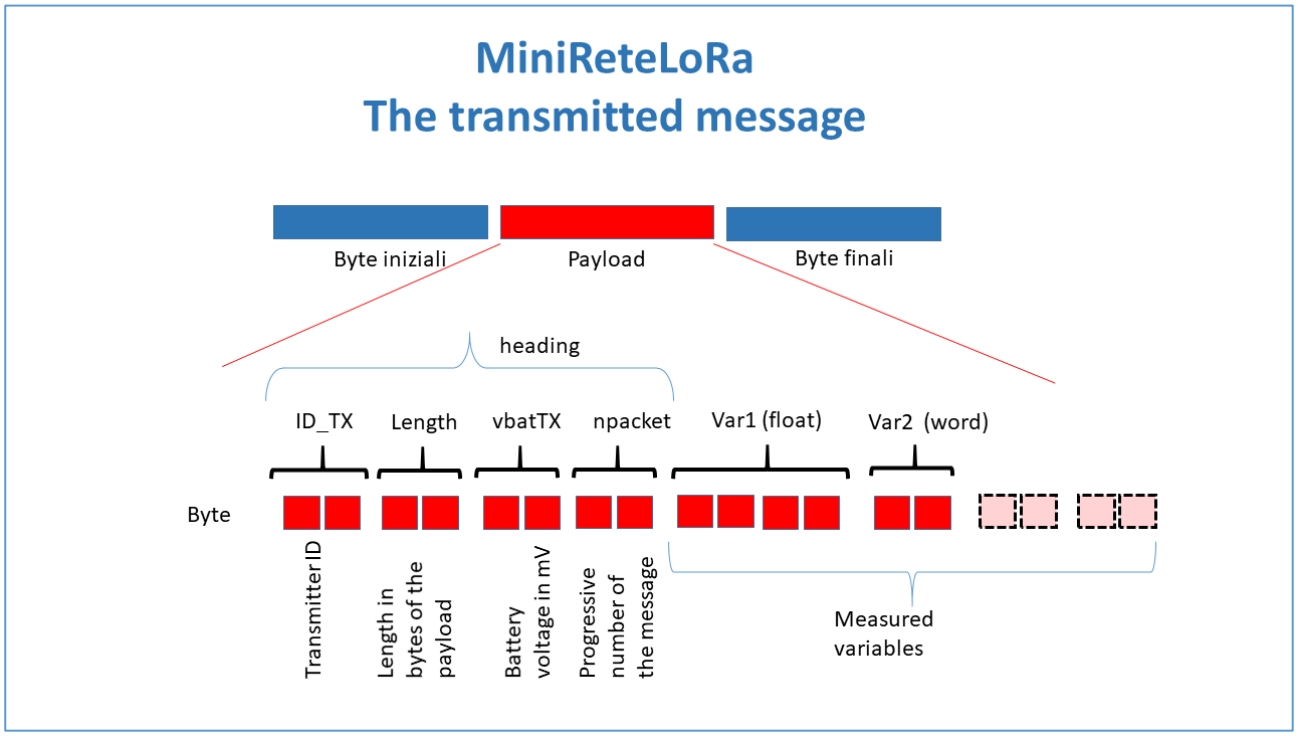


Figure 14 – PayLoad structure of a message for a MiniReteLoRa

Now let's see the advantages and disadvantages of the two encodings. The ASCII character format has the advantage of being accepted by any computer system, being a universally used code; the disadvantage lies in the need to occupy a byte for each digit or symbol, for example if the quantity to be transmitted is a pressure expressed in millibars, 1020.5, it takes 6 bytes. Furthermore, the CPUs of the microcontrollers perform the calculations using internal coding, for example to obtain a pressure at sea level I have to insert what the sensor measures into a formula; once the result has been found, it will have to be converted into ASCII characters and the number of bytes required will depend on the result of these counts.

The internal format is more compact: in fact, think that in a 4-byte float variable there can be very large numbers, with sign, with a precision of 5 or 6 significant digits.

If I have to calculate a quantities inside the TX node and I don't know the result variability range beforehand, using float variables in the program I can feel comfortable. But C programming, the one used with these microcontrollers, also provides for the use of 2-byte or single-byte integer variables, useful for compacting

our message even more. Keep in mind that the maximum recommended length for a LoRa message is 30-40 bytes as a payload.

Therefore the quantities measured by our sensors will occupy the part of bytes that is after the header of the message, while this is composed of integer numbers in internal format as described in Table 1.

Table 1- Heading format

Byte	**content**	**Variables types in internal numeric format**		**Variable names used**
		C++	**Arduino**	
0 - 1	Transmitter ID	uint16_t	word	ID_TX
2 - 3	Numbers of bytes of the whole payload	uint16_t	word	Length
4 - 5	Battery voltage in mV	uint16_t	word	vBatTX
6 - 7	Progressive number of the message	uint16_t	word	npacket

The node identifier (ID_TX) can be a number from 1 to 65535, but, to be consistent with the programs presented here, it is always advisable to use 5-digit numbers: from 10,000 to 65,000.

Now let's see the declaratives for the variables that contain the measured quantities. Let's imagine a TX node with number 12345, which must transmit the content of a float variable, an int and a byte, for a total of 7 bytes, quantities taken from one or more sensors, the declarations will be:

```
float temp;    // air temperature as number of degrees Celsius with one decimal place (°C)
int humidity; // humidity as integer (%)
bytes it rains;  // rain = 1 or no rain = 0
```

Transmitting the values of these variables, whether with LoRa or some other serial mode, requires that one byte be transmitted at a time. Then the values must be converted into a byte array.

This conversion is illustrated in Figure 16.

In this way, the CPU can access the memory areas that contain the measured values, either as float, int, byte variables, or as single bytes of an array.

The variables defined in the `packet12345` structure will be assigned a value during the execution of the program. The only thing to remember is that, having been defined in a structure, then placed in union with an array, they will be called like this:

```
minni.Data12345.temp    = …
minni.Data12345.umidita = …
minni.Data12345.piove   = …
```

The elements of bufVar will also be named like this:

```
minni.bufVar[i]  = …
```

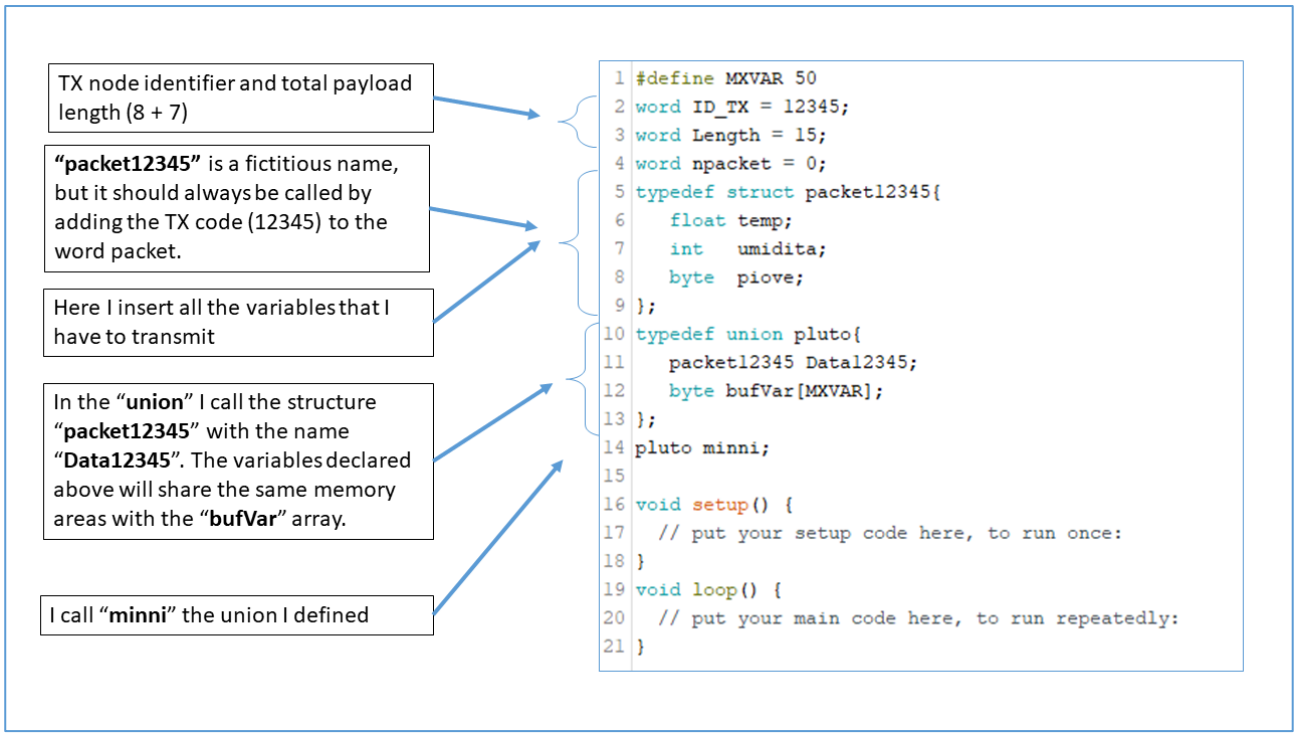


Figure 15 - The part of the program with the declarations

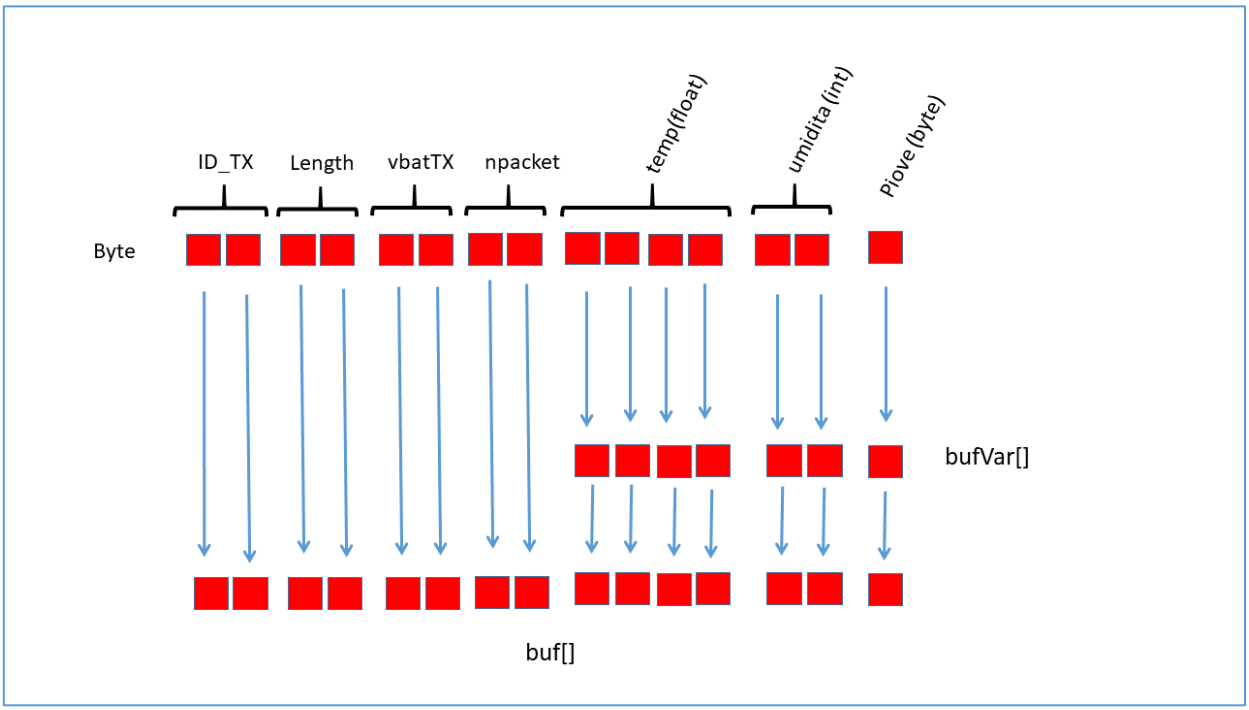


Figure 16 - Correspondence between the bytes of the variables and those of the arrays

It is good practice to declare variables in a structure in decreasing order of number of bytes, i.e. first the 4-byte ones, then the two-byte ones, then the single-byte variables. This is to avoid array matching errors in the union.

The advantage of this way of proceeding also lies in the possibility of carrying out the reverse passage. For example receiving a byte array via LoRa or a serial link and decoding it into variables of different types, declared in a structure.

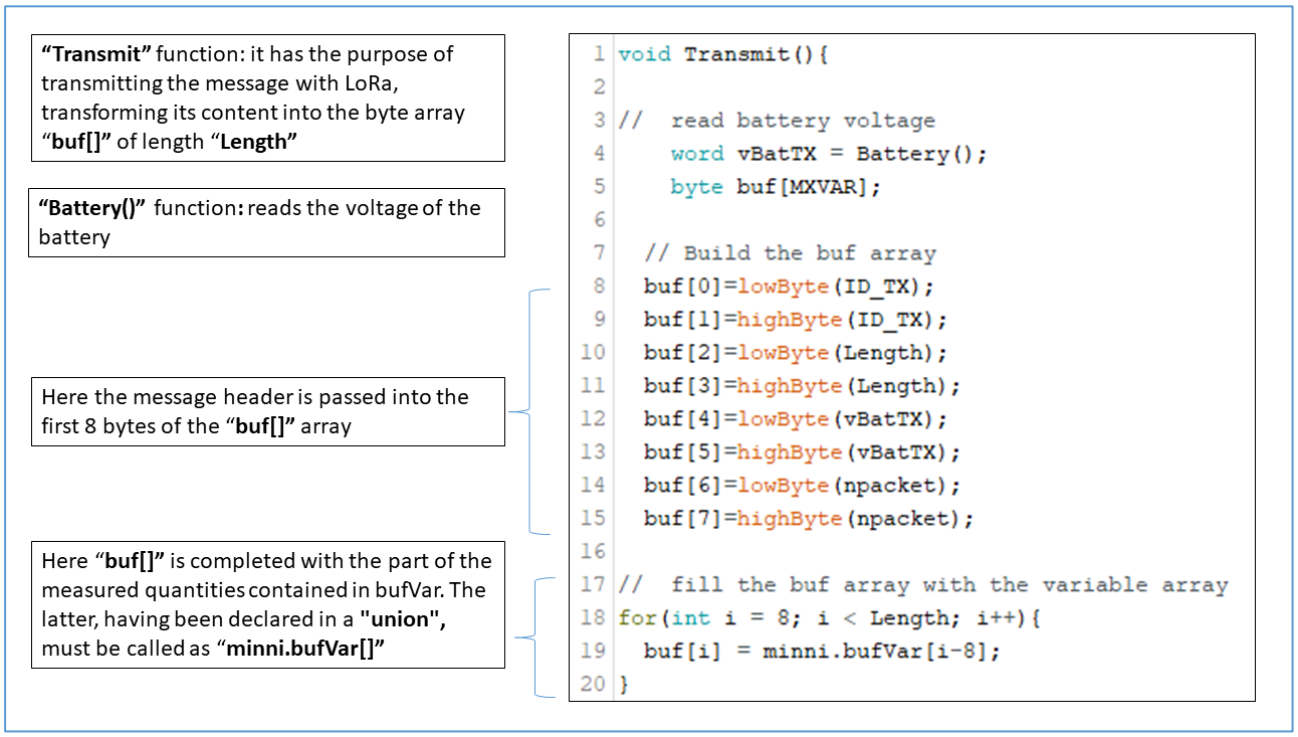


Figure 17 – Transmit function, first part

Figure 17 shows the first part of the Transmit function, which is always the same for all applications, and which prepares the `buf`[] array by loading it with both the 8 bytes of the header and the bufVar array containing the measured quantities.

The second part of the Transmit function can be seen in Figure 18 and includes: the calls to the functions of the RadioHead library responsible for transmitting the message, the npacket increment that will be inserted in the next message and the call to the function which causes the LED to flash as a signal diagnostic of successful transmission.

The purpose of the progressive number of npacket message is to be able to diagnose, in reception, any lost messages.

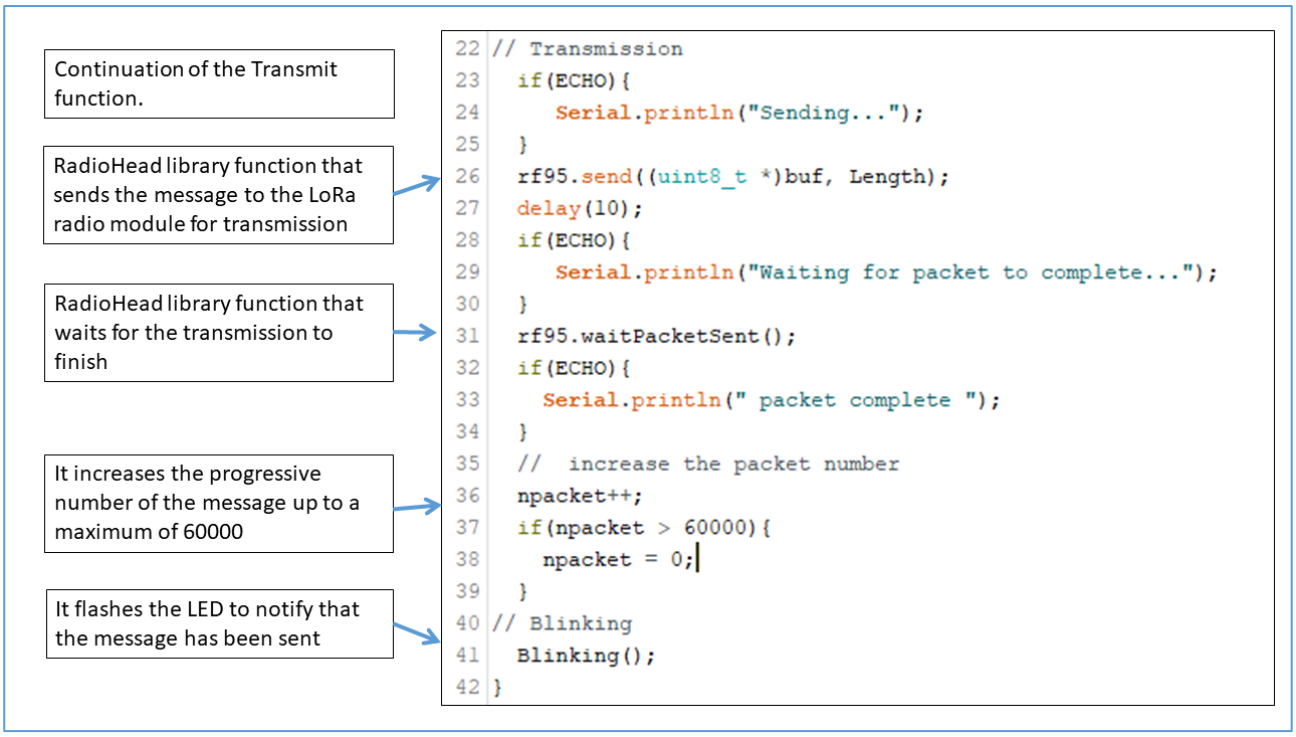


Figure 18– Transmit function, second part

The program for a generic TX node: "TX_Generic"

I will explain in this paragraph a complete program for a real TX node, consisting of a TTGO board, presented earlier, and a DHT22 temperature and humidity sensor.

The source of this program, written through the Arduino IDE, is divided into folder, also called "Tab"; in a Tab there can be a list of declaratives (type of variables, #define, #include of libraries) or a function. This practice is used for greater clarity, but above all to create a "Template", i.e. a generic program for a TX of a MiniReteLoRa, where there will be only some Tab to modify, adapting them to our needs.

In Table 2, I present the Tabs of the TX_Generic program, which measures temperature and humidity through a DHT22 sensor. This program is the "Template" I was talking about earlier.

As you can see from Table 2, some folders have a name with a final ".h". These are used for global variable declarations and library includes, which are normally placed above the `void setup()`. In the main program these folders are included with the `#include` statement:

```
#include Sens.h
#include Packet.h
…
```

Table 2 – Tab list in the TX_Generic

Tab name	What it contains	To be modified
`TX_Generic`	Main program	**Yes** Modify the variables if necessary: `ncicliMax, RF95_FREQ,RF95_POW,` ECHO, and reset the message variables in setup()
`Battery`	Function to read the battery voltage	**No**
`Blinking`	LED flash function	**No**
`Nword`	Function to approximate to the nearest integer	**No**
`Packet.h`	Declarations of the variables measured by the sensor and to be transmitted	**Yes** Enter here the variables read by the various sensors that you want to transmit, the TX identifier and payload length
`Radio.h`	Radio statements	**Yes** only if you change the board. The #define of all the board are written and commented.
`RadioInit`	Radio initialization function	**No** Only If you use the ESP32 board, you have to scomment the `SPI.begin`
`ReadSensor`	Sensor reading function	**Yes** Insert here the reading instructions of the sensors.
`Sens.h`	Sensor declarations	**Yes** Enter here the necessary declarations for the sensors including their libraries, if any.
`SensInit`	Sensor initialization function	**Yes** Enter here the initialisations necessary for reading the sensors
`Transmit`	Function to send the data packet	**No**
`altDelay`	Delay function	**No**

The Tab that contain functions can be called whatever you want, but for clarity it is better to call them with the same name as the function, so the function

```
void RadioInit(){    …    }
```

it will be inside the RadioInit Tab.

The TX_Generic Tab contains the main program, visible in Figure 19 and Figure 20, which is quite short. Let's take a closer look at it.

At the beginning of the program, lines 17 – 19, you can see the Tabs included:

`Radio.h`, which contains the declarations regarding the radio module;

`Sens.h`, which contains the declarations regarding the sensors used, together with the libraries necessary for their reading;

`Packet.h`, which contains the global variables which have to be transmitted, as explained above.

Lines 20 and 21 define the frequency values in MHz and the transmission power, the maximum of which is 23 dBm. The transmit frequency can be set from 868.0 up to 868.6 MHz in steps of 0.1.

Lines 22 and 23 define a counter variable of the number of loop during the run and its maximum number, once reached, the function that transmits the message will be called.

The ECHO variable is used, if set to 1, to activate the control writings on the serial monitor; for some cards, but not described here, it is necessary to set ECHO to 0 when the serial monitor is no longer used.

During the operation phase, ECHO must be set to 0, in order to activate the sleeping mode.

All the instructions needed to initialize radio and sensors are included in the setup:

• the radio (line 27);

• frequency and power (lines 29 and 34);

• sensors (line 38).

The subsequent lines 41 and 42 initialize the two variables to be transmitted, declared in `Packet.h`, which will then be transmitted on line 43. This is important at the beginning for testing the transmission function.

The altDelay function on line 44 has the same purpose as the delay, but is created with specially constructed delay instructions. This is useful if the watchdog procedure is used during the delay.

The RadioHead library, used for radio transmission, allows you to set parameters other than frequency and power; these are the typical parameters of LoRa transmission, such as: Bandwidth, Code rate, Spreading factor. My tests have always been made with the default settings for these parameters and I have always found them good, for those who want to "have fun" changing these parameters, I recommend reading the documentation of the library, of which I have reported the link below.

The loop part of the program is made so that at each cycle the CPU and the radio are put to sleep (lines 58 – 61), consuming as little energy as possible, for eight seconds (this is the maximum acceptable by the Watchdog.sleep function) .

To lengthen the sleep time, the ncicli counter is used, which, having reached its maximum value `ncicliMax`, passes control to the `ReadSensor` and `Transmit` functions (lines 50 – 52).

During the test phase of the program, by setting ECHO = 1, it is possible to have the desired delay without resorting to sleep, by acting on the values of `altDelay` and `ncicliMax`.

The other Tabs are extensively commented within the source and therefore I believe it is not useful to report them here in the text. Instead, I invite the reader to first read and understand the Tabs that need to be changed (see Table 2) which are quite simple.

TX_Generic § | Battery | Blinking | Nword | Packet.h | Radio.h | RadioInit | ReadS

```
#include "Radio.h"
#include "Sens.h"
#include "Packet.h"
#define RF95_FREQ 868.0  // LoRa Frequency
#define RF95_POW   23    // Transmitting power dBm
word ncicli =    0;
word ncicliMax = 10;     // Number of loop cycles after which it transmits
byte ECHO  = 1 ;
void setup() {
  //  Initializing the radio
    RadioInit();
    Serial.println(F("TX_Generic.ino"));
    rf95.setFrequency(RF95_FREQ);
    if (ECHO) {
      Serial.print(F("Set Freq to: ")); Serial.println(RF95_FREQ);
    }
//  Set transmission Power
    rf95.setTxPower(RF95_POW, false);
    if (ECHO) {
    Serial.print(F("Set Power to: ")); Serial.println(RF95_POW);
  }
  SensInit();
```

Figure 19 - Main program di TX_Generic (first part)

```
//   First transmission with all the variables at 0
    minni.Data30003.Humi = 0;
    minni.Data30003.Temp = 0;
    Transmit();
    altDelay(5000);
}

void loop() {
  ncicli++;
  if(ncicli >= ncicliMax){
   ReadSensor();
// Transmit data
    Transmit();
    ncicli = 0;
  }
  if(ECHO){
     altDelay(5000);
  }else{
//  The radio goes to sleep
     rf95.sleep();
// It goes to sleep for 8 seconds.
     Watchdog.sleep(8000);
  }
}
```

Figure 20 - Main program di TX_Generic (secondo part)

The program for the receiving node "RX_Generic_autoconnect"

In this section I will explain the program that allows the two ESP32 boards presented before, to receive via LoRa the data of many TX nodes (transmitting with the same carrier), even different from each other, and transmit them to a server on the Internet.

The server I chose is that of AdafruitIO (https://io.adafruit.com/) which offers free space and the ability to create interactive graphics with your browser.

The RX board can also do other things, such as activating the LEDs, if some thresholds of the received quantities are exceeded, alarms and various actuators, all this changing by adding a few lines to the program.

Here too, as for the TX node, the program is divided into Tabs, some not to be modified and others to be adapted to one's needs.

The `RX_Generic_autoconnect` program, explained here, was written to receive three different TX nodes (30002, 30003, 50000) of which 30002 is the one referred to by the previously presented TX node (`TX_Generic`).

Table 3 – Tab list in the RX_Generic_autoconnect

Tab name	What contains	To be modified
`RX_Generic_autoconnect`	Main program	**No**
`Action`	Function that filters received messages and calls the PrintXXXXX functions	**Yes** if you add other message
`Blinking`	Function for LED blinking when a message is received	**No**
`Packet.h`	It contains the structures of the variables measured by the various TX sensors that we want to receive. The structures are the same as those of the individual TX sketches.	**Yes** if you add other message
`Print30002 ; Print30003; Print50000; …`	Functions with the name of the TX. They are used to print the data on the serial monitor and on the display. There is one for each TX. There are also instructions for sending data to the server	You must write a similar function for each different message to receive and process
`PushButt`	Function that manages the button for entering the WiFi credentials	**No**
`Radio.h`	Contains declarations for LoRa radio	**No**
`Receive`	Function that decodes the data received via LoRa radio	**No**
`ServerConfig.h`	Contains login credentials to the AdafruitIO server	**Yes** only if you change the login to AdafruitIO
`board_defs.h`	Contains the mapping of some pins of the board	**Yes** if you change the board

The program uses special libraries that allow auto-connection to a WiFi network and to an AdafruitIO server account. This function offers the advantage of not having to write the access credentials in the program and thus having to reload it every time a network or server account change is made.

The advantage of this solution is to set up a mini LoRa network composed of sensor nodes and one gateway receiver that you can transport anywhere without the need to modify and reload the program.

The program will have to be modified only if the TX nodes to be received and/or their messages change. The following explains how to make these changes.

`Action`

This function is called immediately after receiving a message to check if the transmitter ID is among those listed in `Packet.h` . Only in that case are the functions `PrintXXXXX` called

`Packet.h, PrintXXXXX, ServerConfig.h`

The Figure 21, Figure 22, Figure 23, show the content of these tabs.

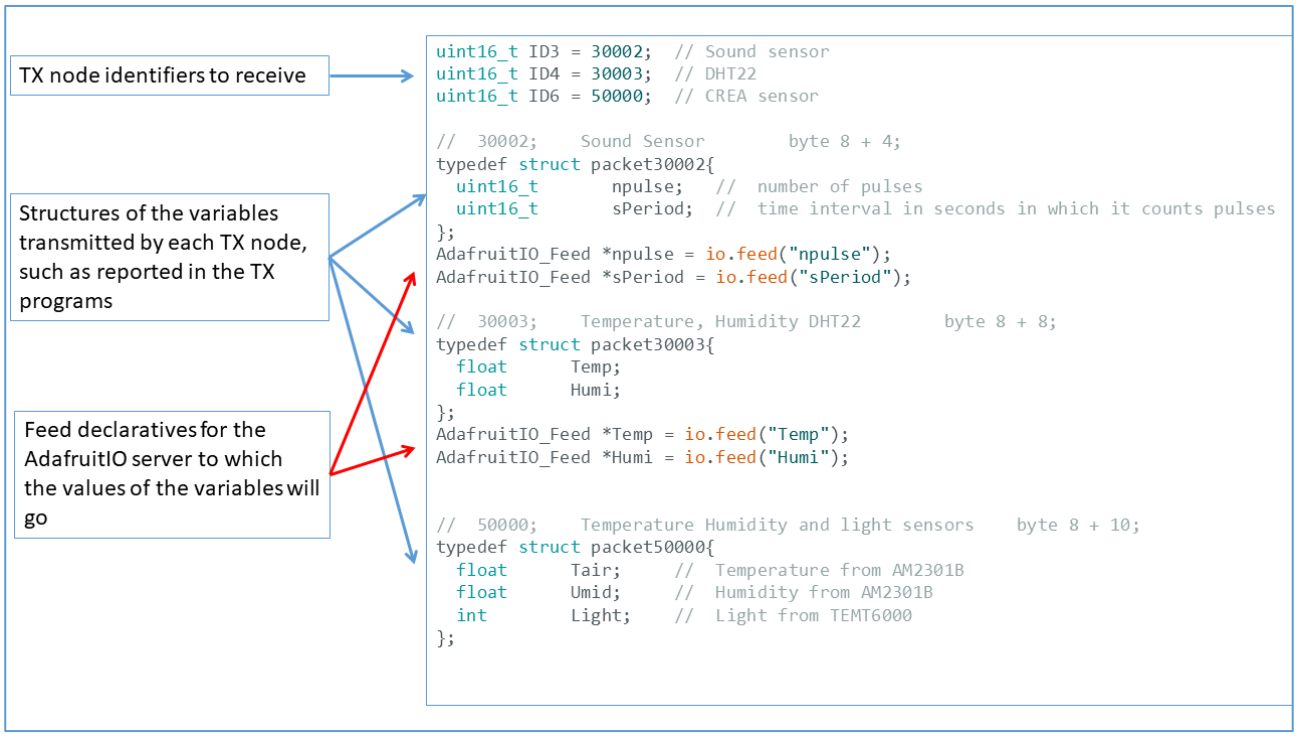


Figure 21 - Packet.h content, first part

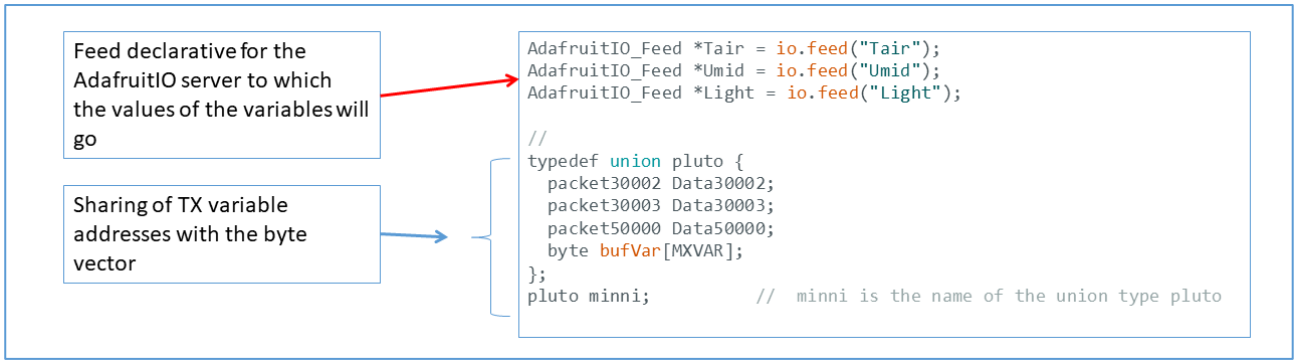


Figure 22 - Packet.h content, second part

Instructions for printing on Serial Monitor

Instructions for LCD display or other

Instructions for sending data to AdafruitIO

```
//  Sound Sensor
void Print30002(){
  Serial.print(" npulse ");
  Serial.print(minni.Data30002.npulse);
  Serial.print("  sPeriod ");
  Serial.print(minni.Data30002.sPeriod);
  Serial.println();
/*
  display.drawString(0,22,"npulse: " +
String(minni.Data30002.npulse));
  display.drawString(0,33,"sPeriod: " +
String(minni.Data30002.sPeriod));
  display.clear();
*/
//  Transmission to the Internet Server
      if(TrasmettiAda){
           npulse->    save(minni.Data30002.npulse);
           sPeriod->    save(minni.Data30002.sPeriod);
           Serial.println("Sent to Adafruit");
      }
}
```

Figure 23 – Print30002

```
/************************ Adafruit IO Config
//  WiFi network
// These constants  must be uppercase

//  Connection to the WiFi
#define WIFI_SSID   ""
#define WIFI_PASS   ""

//  Adafruit access
// visit io.adafruit.com if you need to create an account,
// or if you need your Adafruit IO key.

#define IO_USERNAME  "xxxxxx"
#define IO_KEY       "xxxxxx"
```

Figure 24 - ServerConfig.h

board_defs.h

In this Tab you can find pins declarations for the board.

RSSI signal level and antennas

The RSSI (Receive Signal Strength Indicator) parameter, returned by a specific function of the LoRa library, indicates the strength of the received LoRa radio signal. It is expressed in negative dBm, therefore, if we view it as an absolute value, this number can range from a few tens, with TX and RX close together, to about 110 with TX and RX at a distance of 1 km or more. The relationship between distance and RSSI is not unique and linear, it depends on many external factors that attenuate the signal, such as orography, vegetation and weather. In the RX_Generic program this parameter is shown on the serial monitor and on the display.

The type of antenna used for TX and RX also affects RSSI. Based on my experience, I recommend using either whip antennas or the more expensive but better performing "Ground Plane" antennas, as in Figure 25.

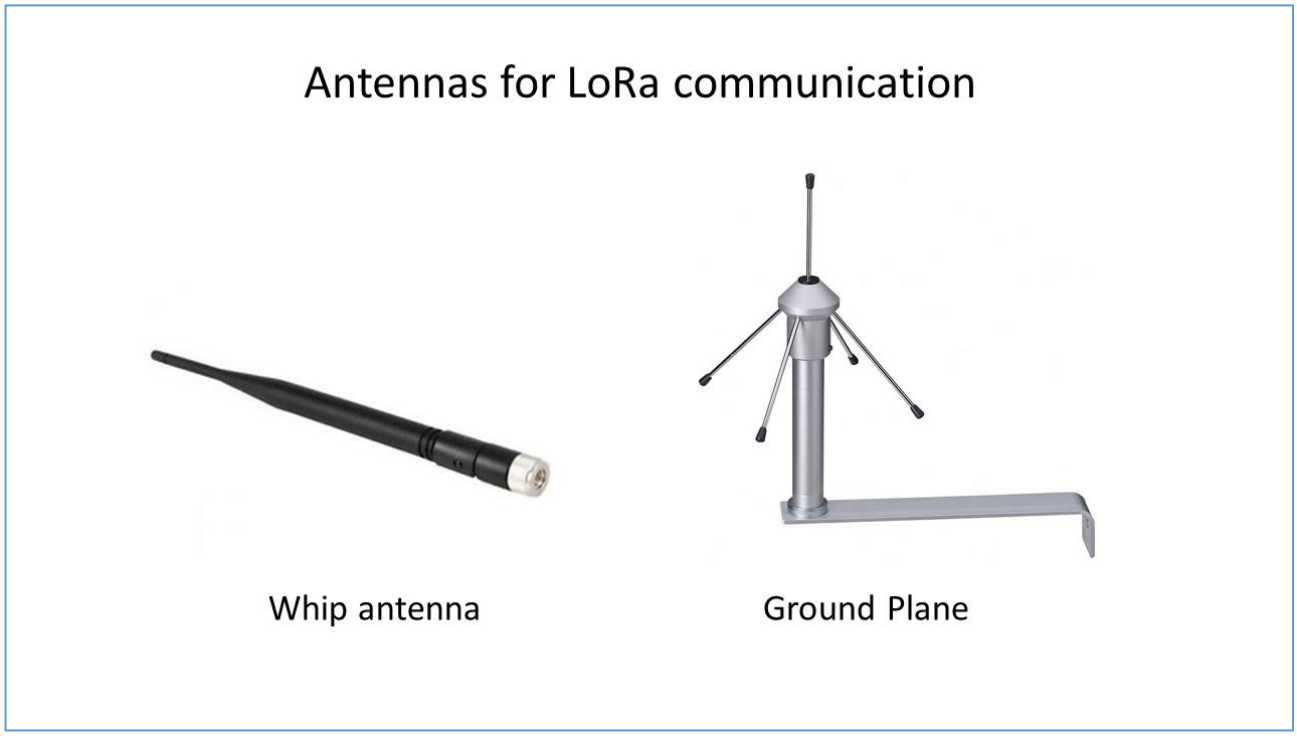


Figure 25- some antennas for TX and RX

Libraries

In the `TX_Generic` the following libraries are used:

```
#include <RH_RF95.h>
#include "Adafruit_SleepyDog.h"
```

the former is included in the RadioHead package:

https://www.airspayce.com/mikem/arduino/RadioHead/classRH__RF95.html

In `Sens.h` we can put the libraries for sensors

In the `RX_Generic_autoconnect` the following libraries are used:

```
#include <SPI.h>
#include <LoRa.h>
#include <Wire.h>
#include <WiFiManager.h>
#include "AdafruitIO_WiFi.h"
#include <LiquidCrystal_I2C.h>
```

All libraries can be included in the IDE using the "Library Management" function of the Arduino IDE itself. You should also include the following links in the "Additional URLs for Board Manager" window in the "Settings" item of the "File" menu.

https://dl.espressif.com/dl/package_esp32_index.json

https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_index.json

Data Platform

Looking for a cloud platform where to send data and then see it graphically, I opted for the Adafruit IO service. In fact, this provides example programs to load on our receiver board, it is easy to use in recovering and display data.

https://io.adafruit.com/

Here you can create a free account only with your own e-mail address. This service allows you to send data to a database with the RX_Generic_autoconnect program, described above, and create graphical representations of various types.

Once you have the account, there is no need to set up anything on the Adafruit IO server for it to acquire and store the data we send it. Instead, you need to take note of the access credentials found under the "Key" item in the top menu that appears once connected. By clicking on the "Key" item, the window in Figure 26 appears, where you just need to copy the two lines below, under the word Arduino.

Once the first data has been sent to the Adafruit server, a "Dashboard" can be created which will contain the graphs that we will set up. The creation of graphics is very simple and is done using our browser and the menus on the site. Figure 27 shows a time graph of the npulse variable transmitted by the TX 30002 node.

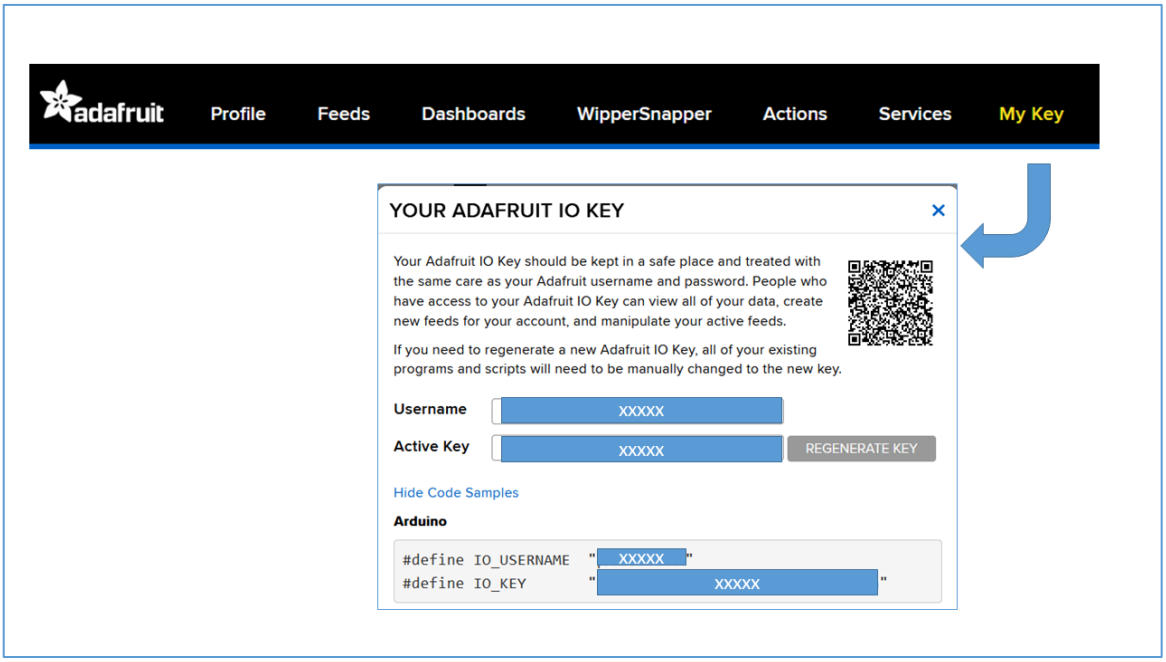


Figure 26 – AdafruitIO login credentials

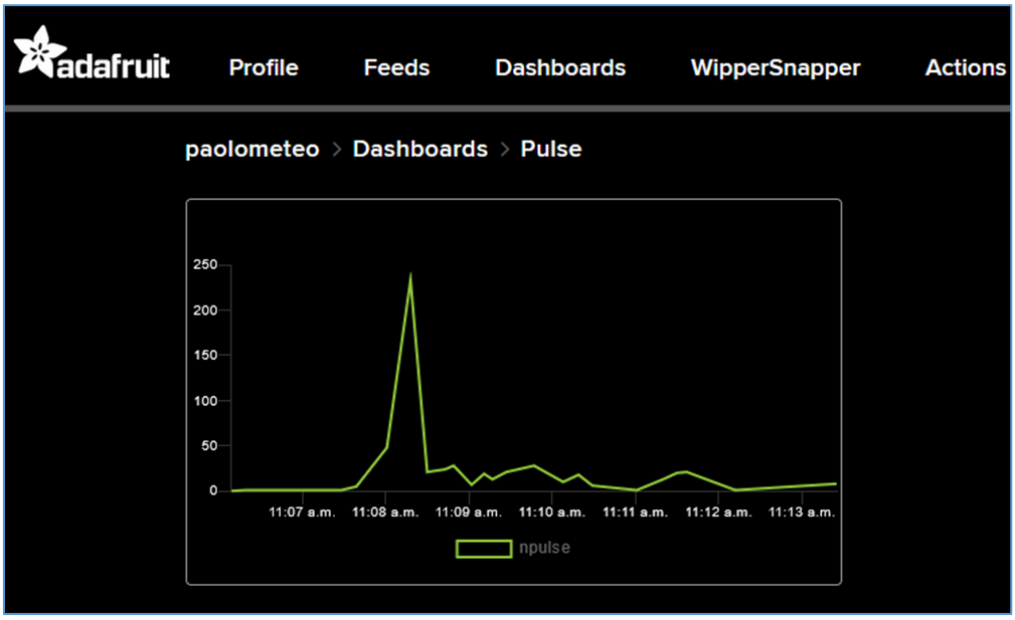


Figure 27 – Example graph made on the dashboard of AdafruitIO

RX Node Operation

In this chapter I will explain how to operate with the RX node, built according to the scheme in Figure 11 and with the `RX_Generic_autoconnect` uploaded.

When the RX node is powered, it generates its own WiFi network with the name (SSID):

ESP32AP.

• This SSID appears in the list of active networks on our PC or mobile phone.

• Select this network and wait for notification of connection; no password is required.

• We are immediately notified with a notification and the menu, as that on Figure 28, will appear on the browser. If this does not happen, we can connect to the URL: http://192.168.4.1.

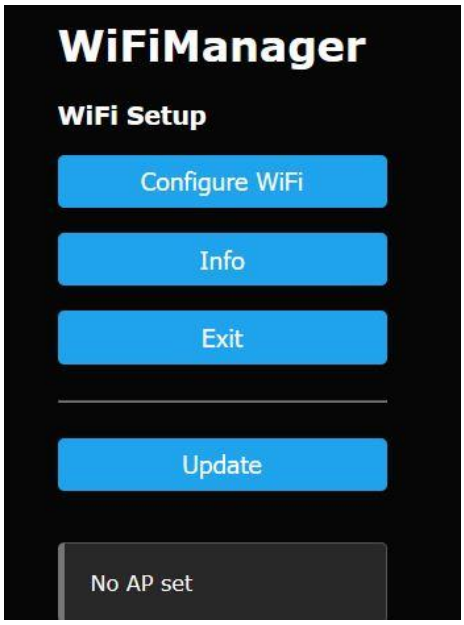


Figure 28 Connection menu

By clicking the first blue button "Configure WiFi" you go to another page where you can write the SSID and password of the WiFi network to which the RX board is to connect (the program will present you a list of active networks, chose one).

Finally, press the "Save" key to close this short dialogue with the board. The receiver connects to the chosen WiFi network and to the Adafruit IO server, the successful connection is shown on the LCD display. At this point each LoRa message received, if among those declared, is displayed on the LCD and forwarded to the server.

The next time I turn on the receiver, it will ask me if I need to change the WiFi network, in this case I have to press the “WiFi” button on RX within 5 seconds and repeat the procedure explained above. If I don't press the button, the receiver will connect to the previously set network.

The “Reset” button will reset the program if I need it.

Reliability test of a MiniReteLoRa

Someone might have doubts about the reliability of a MiniReteLora especially when several TX nodes transmit to a receiver all on the same frequency. Well, an assessment of this reliability comes from a practical application carried out by me at Cascina S. Ambrogio in Milan with a weather-environmental station equipped with three TX nodes which transmit the measurements of many variables, 27 to be exact, towards an RX node which stores them on an SD card using a datalogger.

The data that was collected and analyzed ranges from March 12, 2022 to April 13, 2022, just over a month.

The three TX nodes send data with three different periodicities:

Node	Period (seconds)

12120 87
12121 278
12122 179

The TX nodes were about 500 meters away from the RX receiving station.
Having available the date and time of arrival of the messages of each node and the progressive number of the message, the number of messages arrived was calculated with the interval of a period, of two periods, i.e. one message skipped, and more than two. The table shows the results as a percentage.

Table 4 – percentage of messages arriving at the receiver

TX node	Total messages collected	Messages arrived after 1 period	Messages arrived after 2 period	Messages arrived after 3 period
12120	31669	99,37 %	0,62 %	0,01 %
12121	9900	99,31 %	0,67 %	0,01 %
12122	15436	99,40 %	0,60 %	0

As can be seen, the percentage of messages lost by the three nodes is very low, despite the fact that no synchronization system has been set up on sending messages and the LoRa frequency used has been the same for all.

This test demonstrates that a MiniReteLoRa equipped with 3 TX nodes that send messages of different lengths and with different periodicity to a single RX node, has a respectable reliability and promises interesting results even with a greater number of nodes.

How to set up a repeater to extend the range

If the distance between transmitter and receiver is too long or there are obstacles between them, a repeater can be built. A LoRa repeater (RIP) can be a board with a LoRa tranceiver, controlled by the same libraries as the other of the MiniReteLoRa, which receives a message from a transmitter (TX) and retransmits it on a different channel. The RIP can be placed at an intermediate distance.

The logic is simple:

If the TX sending periodicity is say 100s, on the 868.0 channel, the RIP sleeps for 70s and then listens on that channel until it receives a message from TX. It immediately retransmits it on the 868.1 channel with an ID different from the TX.

But who synchronizes the two?

The RIP in the startup phase, after it is switched on, listens until it receives a message coming from TX and soon retransmits it. Then it enters into the loop phase and proceeds as described above in the logic. Even if the clocks of the two cards were slightly different, the time RIP is listening is long enough to cover these discrepancies.

Naturally there are a TimeOut to prevent RIP from listening forever if TX no longer transmits. In this case RIP transmits its own message with an error code. After a certain time, however RIP transmits its status and its battery voltage.

Why do I use different channels and IDs?

This to avoid that the RX acquires messages from TX and RIP together, in fact it could happen that the TX message is sometimes received by RX under certain particular conditions.

Software

RepeaterGeneric is the program to be uploaded on RIP.

Conclusions

We have reached the end of this long discussion on how to build a MiniReteLoRa, I hope I have been clear in the explanation and have stimulated the interest of the reader.

For some of you C++ experts, perhaps the programs presented will not be the top of perfection, however, being all open source, you can try your hand at modifying what you want, within the limits of the Creative Commons license mentioned at the beginning of the article.

Everything I have stated has been tried and works. The programs can be downloaded at this link.

https://github.com/paolometeo/MiniReteLoRa

I remain available to respond to your comments and reports of possible errors.

Good work!

Paolo