Come costruirsi una “MiniReteLoRa”

… e monitorare qualsiasi cosa (o quasi)

Paolo Bonelli

Versione 3: 29/11/2022

Sommario

Introduzione

Oggi si parla tanto di IOT (Internet Of Things), intendendo oggetti sparsi per la casa, la fabbrica o un campo aperto che mandano dati su Internet allo scopo di monitorare parametri critici ed eventualmente attivare segnali di allarme.

Sistemi del genere sono già in commercio, ma, oltre al prezzo, l'ostacolo principale per un chi cerca un'applicazione particolare è la loro scarsa duttilità.

Quali tecnologie può usare un privato che vuol fare tutto da sé?

Per un privato con poca dimestichezza con l'elettronica è necessaria una tecnologia di trasmissione dati che usi hardware economico e software open source possibilmente di facile uso; inoltre può essere utile che questi sistemi funzionino anche fuori dalle mura di casa, all'aperto, dove non è presente alimentazione elettrica e rete WiFi.

In questo articolo si impara a costruire una rete di sensori che trasmettono i dati su Internet a basso costo, muovendono i primi passi verso un'esperienza entusiasmante, forse al pari di quella provata da Guglielmo Marconi alla fine del XIX secolo, quando, nel parco della sua villa, vide arrivare, con grande giubilo sul suo ricevitore rudimentale, un segnale trasmesso via radio e partito a qualche chilometro di distanza.

Cos'è e a cosa serve una MiniReteLoRa

Partiamo da alcuni esempi di applicazioni pratiche:

- in un campo coltivato c'è la necessità di misurare la temperatura dell'aria, del suolo, il contenuto di acqua nel terreno e altro, ma siamo lontani da una casa con WiFi;
- sempre in campagna, dobbiamo comandare una pompa per sollevare l'acqua da un pozzo e riempire un serbatoio quando sta per vuotarsi, ma questo si trova parecchio distante; vogliamo inoltre che la pompa si accenda automaticamente e seguire il funzionamento di tutto il sistema su Internet;
- abbiamo una serra dove è importante tenere sotto controllo la temperatura dell'aria in vari punti per comandare gli impianti di riscaldamento o l'apertura delle prese d'aria, però non vogliamo tirare fili in mezzo alle piante;
- vogliamo sapere dove è andato a finire il nostro cane o la mucca che pascola o la capra che si arrampica su un dirupo.

Fin qui si tratta di applicazioni agricole, ma pensiamo anche alla misura delle polveri sottili in aria nei giardinetti dove giocano i nostri figli, a scuola o più in generale all'inquinamento di ecosistemi acquatici come fiumi e laghi.

Tutte queste applicazioni richiedono sensori, elettronica di acquisizione-trasmissione dei dati e un ricevitore concentratore. Quale sistema si può usare per trasmettere i dati? La rete GSM-4G? Una rete WiFi? Nel primo caso dovremmo pagare un canone e dotare ogni scheda-sensore di una SIM con un consumo di energia che non ci permette di far durare la carica della batteria per mesi; nel secondo caso avremmo bisogno di schede-sensore dotate di WiFi, sempre con un consumo di batteria non trascurabile e un router WiFi/GSM come concentratore con una copertura che non potrebbe superare i 100-200 m.

Da alcuni anni una nuova tecnologia di trasmissione si è resa disponibile con il relativo Hw e Sw: si tratta di LoRa (Long Range). La sigla indica una tecnica di modulazione del segnale radio che permette di avere una grande sensibilità nel punto di ricezione, ne consegue una copertura di chilometri, come dice il nome, con potenza di uscita del trasmettitore ridotta al minimo e tempi di trasmissione brevi per ogni scheda-sensore.

LoRa, in Europa, usa la banda libera di 868 MHz a patto di non occuparla per più del 1% del tempo. Immaginate un sensore che trasmette pochi dati ogni 5 minuti con un tempo di trasmissione di 70 ms (millisecondi), la percentuale di occupazione della banda è di circa 0,02 % che ci permetterebbe di avere fino a 50 schede sensori!! Il consumo tipico di una scheda sensore è di 140 mA in trasmissione e 0,30 mA a riposo, (con la CPU in sleep), quindi il consumo medio, sempre per lo stesso esempio, è di 0,32 mA. In queste valutazioni non ho tenuto conto del consumo dei sensori, che comunque può essere trascurabile se questi sono attivati solo poco prima della trasmissione dei loro dati.

L'uovo di Colombo per le nostre necessità?

Sì ma quali difficoltà si incontrano a costruirsi una rete simile? Esiste in commercio qualcosa "chiavi in mano"? Bisogna essere super esperti nel Hw/Sw?

Qui ci viene incontro il mondo dell'open source: quell'universo di conoscenze disponibili sulla Rete, gratuite e spesso rese facili da abili divulgatori. In più esiste un mercato dell'hardware a basso costo che si espande sempre di più offrendoci sensori, microcontrollori e soluzioni di facile cablaggio. Se siamo già capaci di collegare un sensore ad un microcontrollore come Arduino, possiamo con poco sforzo entrare nel mondo dell'Internet of Things (IOT) che usa LoRa e costruirci la nostra MiniReteLoRa.

Adesso cercherò di descrivere l'architettura di quella che io chiamo "MiniReteLoRa", una rete di sensori-trasmettitori e ricevitore concentratore, completamente autonoma, che non ha bisogno cioè di appoggiarsi a particolari server esterni e sottostare a complessi protocolli.

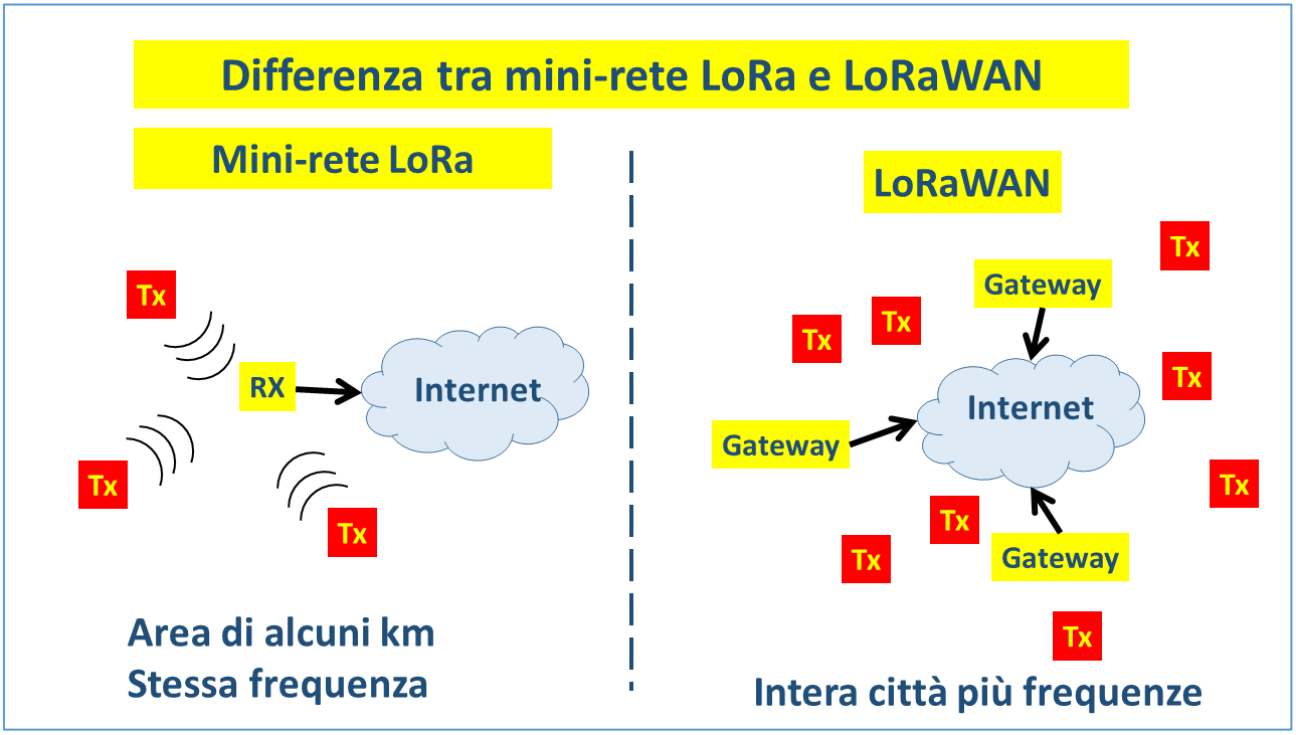


Figura 1 - Differenze tra una MiniReteLoRa e LoRaWAN

Una MiniReteLoRa è una tecnologia che consente l'acquisizione e trasmissione di grandezze rilevate da sensori, posti in località prevalentemente all'aperto, quindi prive di connettività WiFi, richiedendo poca energia e bassi costi. Nulla vieta comunque di usare questa tecnica anche in un ambiente chiuso, come la nostra casa o fabbrica.

Mentre il termine LoRa indica la tecnologia di modulazione di una portante radio, brevetto della ditta Semtech, i termini MiniReteLoRa e LoRaWAN sono usati per definire un tipo di architettura di dispositivi, con i loro software, che sfruttano la tecnologia LoRa per trasmettere e ricevere dati.

La MiniReteLoRa, che descrivo qui, non va confusa con il protocollo LoRaWAN, creato per un'infrastruttura di rete più complessa e più adatta ad un utilizzo commerciale, fatta per moltissimi nodi trasmettitori e alcuni ricevitori (gateway) collegati in Internet ad un network server gestito da chi offre il servizio. LoRaWAN, avendo più gateway in rete offre una copertura spaziale maggiore.

Nella Figura 1 cerco di illustrare in modo semplice la differenza tra questi due approcci, mentre la Figura 2 mostra lo schema dettagliato di una rete LoRaWAN.

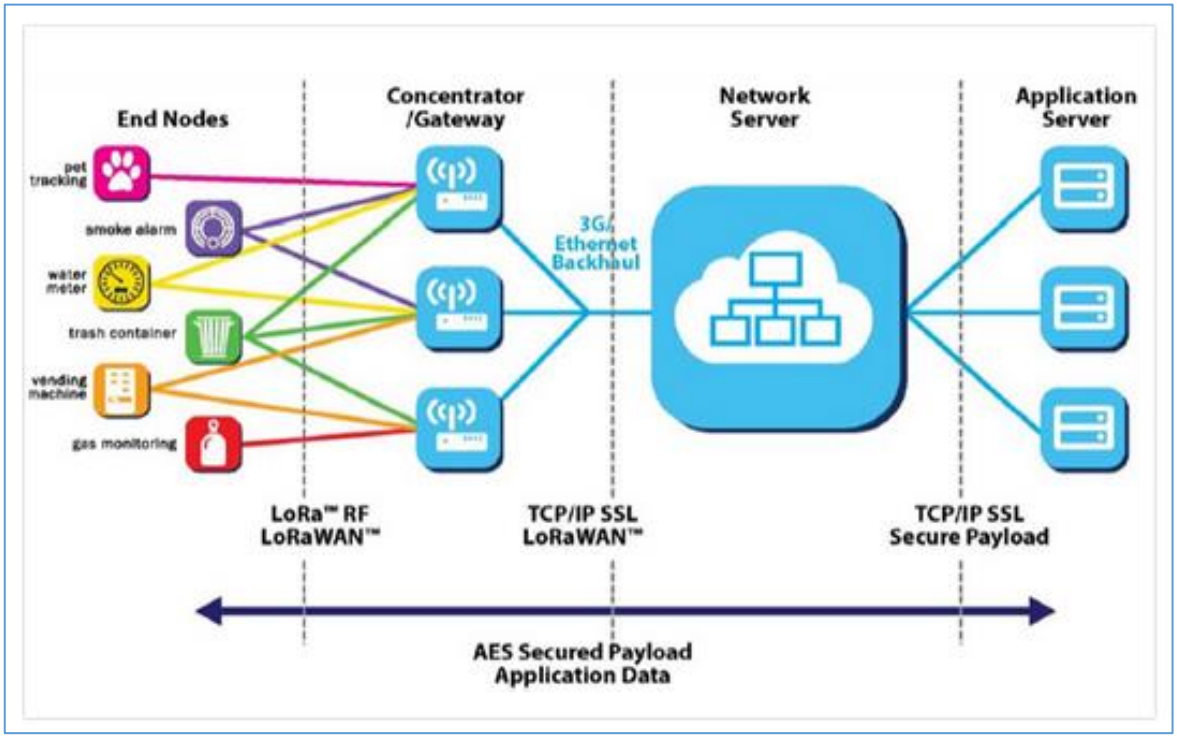


Figura 2 - Schema dettagliato di una rete LoRaWAN (tratto da www.internet4things.it)

La MiniReteLoRa l'ho concepita per funzionare senza infrastrutture di terze parti, quindi può essere impiegata ovunque e implementata da chiunque. La MiniReteLoRa ha un'architettura che prevede:

- più nodi rilevatori/trasmettitori (TX) che svolgono la funzione di acquisire i dati dai sensori, confezionarli in un messaggio che riporta anche alcune informazioni sul nodo e trasmettere in modo broadcast questi messaggi con la tecnologia LoRa;
- un nodo ricevitore (RX) in grado di ricevere e selezionare i messaggi inviati dai nodi TX, estrarne i dati e presentarli in vari modi, quali: display, LED, datalogger. Un'altra possibilità del nodo RX è quella di essere capace di collegarsi ad Internet tramite un hotspot WiFi ed accedere ad un server sul quale depositare i dati ricevuti.

I nodi TX e RX sono costituiti da schede hardware dotate di un microcontrollore programmabile con la IDE di Arduino e di un modulo radio LoRa, gestibile con apposite librerie. Queste schede possono svolgere sia la funzione di TX che di RX, cambiando solo il software.

La trasmissione dei dati può avvenire in modo periodico, controllata da ogni nodo TX in modo autonomo, o in seguito ad un evento esterno registrato dai sensori. In ogni caso si tratta di una trasmissione a senso unico, non è previsto che il TX invii dati su richiesta di un RX. Quest'ultima caratteristica è necessaria se si vuole che il nodo TX consumi meno energia possibile, infatti lo stato di "ascolto" permanente consuma più energia di una trasmissione periodica.

Tipicamente un TX che trasmette ogni 5 minuti un breve messaggio di 30 byte per 70 millisecondi e poi si mette in uno stato di "sleep" per il resto del tempo, può consumare anche solo 0,3 -0,4 mA, mentre un RX sempre in ascolto consuma anche 100 volte tanto. Per questo motivo i nodi RX devono essere alimentati con una fonte di energia illimitata o batterie di grande capacità, se si vuole che la MiniReteLoRa funzioni per mesi senza interventi di manutenzione. Questa necessità è ancora più evidente se il nodo RX è collegato ad altre schede che svolgono funzioni di datalogger o invio dati su Internet con WiFi.

In certe applicazioni particolari il nodo TX può mettersi in ascolto di un messaggio "di avvio" mandato da RX, per esempio per comandare la partenza di più TX in tempi diversi, rimane però sempre la necessità che, a regime, il TX svolga esclusivamente la trasmissione dei dati.

Nonostante la possibilità di impostare da software piccole variazioni di frequenza vicino a quella principale di 868,0 MHz, tutti i nodi TX e il nodo RX di una MiniReteLoRa devono funzionare sulla stessa frequenza. Le possibili interferenze sono ridotte al minimo se il tempo tra due trasmissioni dei nodi TX è lungo rispetto al tempo di occupazione della frequenza nel momento della trasmissione. Essendo quest'ultimo tempo dell'ordine di frazioni di secondo, una periodicità di trasmissione di alcune decine di secondi può essere sufficiente a garantire una bassissima probabilità di interferenze tra alcuni nodi TX. Ancora meglio sarebbe impostare una periodicità di trasmissione diversa per ogni nodo, magari pari a una sequenza di numeri primi di secondi.

A questo punto della mia spiegazione, qualcuno si porrà una domanda:

È affidabile una rete dove i dati viaggiano a senso unico? Infatti non è previsto alcun feedback tra il ricevitore e il trasmettitore, se un dato viene perso il trasmettitore non lo saprà mai.

Cerco di rispondere con due argomenti:

- Il sistema di trasmissione LoRa, gestito dalla libreria che useremo nelle nostre applicazioni, ha un suo controllo interno di integrità dei dati. Quindi se un messaggio in arrivo al RX è corrotto, non verrà processato.
- L'affidabilità va quindi valutata in base agli eventuali messaggi persi rispetto al totale (messaggi trasmessi ma non ricevuti o non processati per errori interni). Ma la ridondanza di messaggi farà aumentare l'affidabilità della mini rete. Mi spiego meglio: se per conoscere l'andamento nel tempo di una grandezza ambientale basta un dato per esempio ogni 10 minuti, il nodo TX lo trasmetterà tre o quattro volte ogni 10 minuti, in modo da essere sicuri che almeno uno arrivi a destinazione corretto.

Perché scegliere una MiniReteLoRa

- Hardware poco costoso;
- Bassi consumi;

- Software semplice e di facile comprensione;
- Librerie consolidate;
- Compatibilità con Arduino e il suo ambiente di programmazione;
- Ricezione e archiviazione dei dati in locale senza necessità di server esterni;
- Eventuale ri-trasmissione dei dati su Internet a diversi server.

Come si fa?

Parliamo adesso prima dell'hardware necessario alla costruzione di una MiniReteLoRa e poi del software. Per semplicità presento solo tre schede microcontrollore, da me testate con successo; ma è bene sapere che MiniReteLoRa può costruirsi anche con altre schede apportando solo piccolissime modifiche al software che presenterò più avanti.

Hardware

In commercio si trovano diverse schede che possono svolgere la funzione dei nodi TX ed RX o tutti e due contemporaneamente, in una MiniReteLoRa. Possiamo anche costruire noi una di queste schede, semplicemente mettendo insieme un microcontrollore e un modulo radio LoRa comprati separatamente.

Nel seguito, per ragioni di semplicità, parlerò solo di due schede, una adatta ai nodi TX e l'altra, più potente per il nodo RX, capace anche di collegarsi ad una rete WiFi.

La scheda-sensore TX: LILYGO®TTGO T-Deer Pro Mini

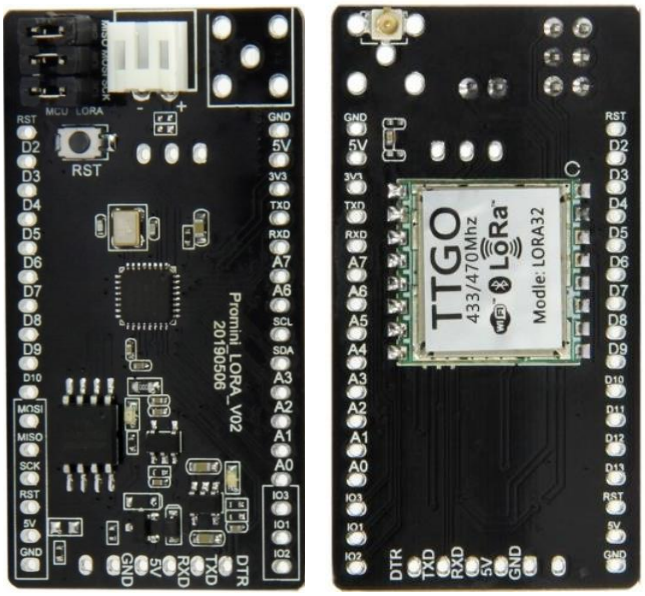


Figura 3- Scheda per nodo TX

La Figura 3 mostra la scheda della ditta LILYGO-TTGO formata da una CPU ATmega328P con clock a 16 MHz simile a quella di Arduino UNO e con un modulo radio LoRa. Questa scheda, come l'Arduino Pro Mini, va programmata con un cavo USB/ FTDI a 3.3 V per i segnali e 5 V per l'alimentazione, non avendo una presa USB a bordo. I pin FTDI sono quei 6 in basso nella figura.

La scheda è dotata di un connettore JST per una eventuale batteria LiPo da 3.7 V che si ricarica collegando la scheda con il cavo FTDI al PC.

Bisogna fare attenzione al collegamento del cavo USB/FTDI: il filo nero (GND), tra i 6 che escono dal connettore FTDI, deve andare sull'ultimo a sinistra guardando la prima immagine della scheda in figura. Attenzione anche a rispettare le polarità della pila LiPo: seguire quelle scritte sulla scheda PCB e non farsi ingannare dai colori rosso-nero dei fili.

Di tutti i pin accessibili sulla scheda, bisogna tener conto, che molti sono impegnati per le funzionalità radio, più precisamente:

D13, D12, D11, D10, D9, D2 destinati alla comunicazione con il modulo radio; D8 è libero, ma nelle mie applicazioni, lo uso per collegare un LED che segnala la trasmissione o ricezione di un messaggio; A0 deve essere collegato al D6 se vogliamo monitorare la tensione della batteria.

Quindi i pin liberi rimasti sono: D0 (RX), D1(TX), D3, D5, D7, A1, A2, A3, A4, A5, A6, A7

Tener presente che se si usano D0 e D1 si rischia di interferire con il monitor seriale quando lo usiamo durante i test.

A6, A7 sono solo input analogici, non possono essere usati come output digitali.

Il LED rosso del Power, presente sulla scheda, può essere interrotto, distruggendolo con le pinzette, in modo da ridurre il consumo di corrente. Per accorgersi se la scheda sta funzionando è possibile collegare un LED verde al pin D8 che faremo lampeggiare ad ogni messaggio inviato.

L'antenna LoRa si collega tramite un "pig tail", un cavetto che da una parte ha la presa femmina SMA sulla quale va avvitata un'antenna adatta alla frequenza 868 MHz e dall'altra la spina U.FL che va collegata alla piccola presa sul PCB.

Per maggiori dettagli su questa scheda, rimando al link:

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

La scheda Ricevitore RX: LILYGO®TTGO LoRa32 T3_V1.6.1 (LoRa32 V2.1_1.6)

Tra parentesi è il nuovo nome assegnato dalla Lilygo.

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

La scheda si programma con la IDE di Arduino, per mezzo di un cavo micro-USB. La CPU ESP32 è molto potente, ha anche una memoria flash, quella che ospita il programma, da 4 MB contro i 32KB di Arduino.

Il protocollo usato per la MiniReteLoRa, che vedremo più avanti, consente alla scheda RX di ricevere i messaggi trasmessi dai vari nodi TX, decodificarli e inviare i dati dei nostri sensori ad un server internet che possiamo scegliere a piacimento tra quelli disponibili gratis o a pagamento.

La Figura 5 mostra alcuni particolari hardware della scheda, quali i due interruttori, i LED. La scheda può essere alimentata tramite la presa micro USB connessa ad una fonte a 5V, usata anche per il collegamento al PC, oppure con una batteria LiPo da collegare al connettore JST. La batteria può essere ricaricata connettendo il cavo USB ad una fonte a 5V.

Figura 4 - Scheda per nodo RX

La scheda in

Figura *4* è una vera e propria "bomba"! Ha tante cose a bordo. Una potente CPU ESP32, un modulo radio LoRa, un display OLED, uno slot per una SD card e soprattutto un modulo WiFi. L'abbinamento tra i due moduli LoRa e WiFi consente di inviare su Internet i dati ricevuti dai nodi TX via LoRa.

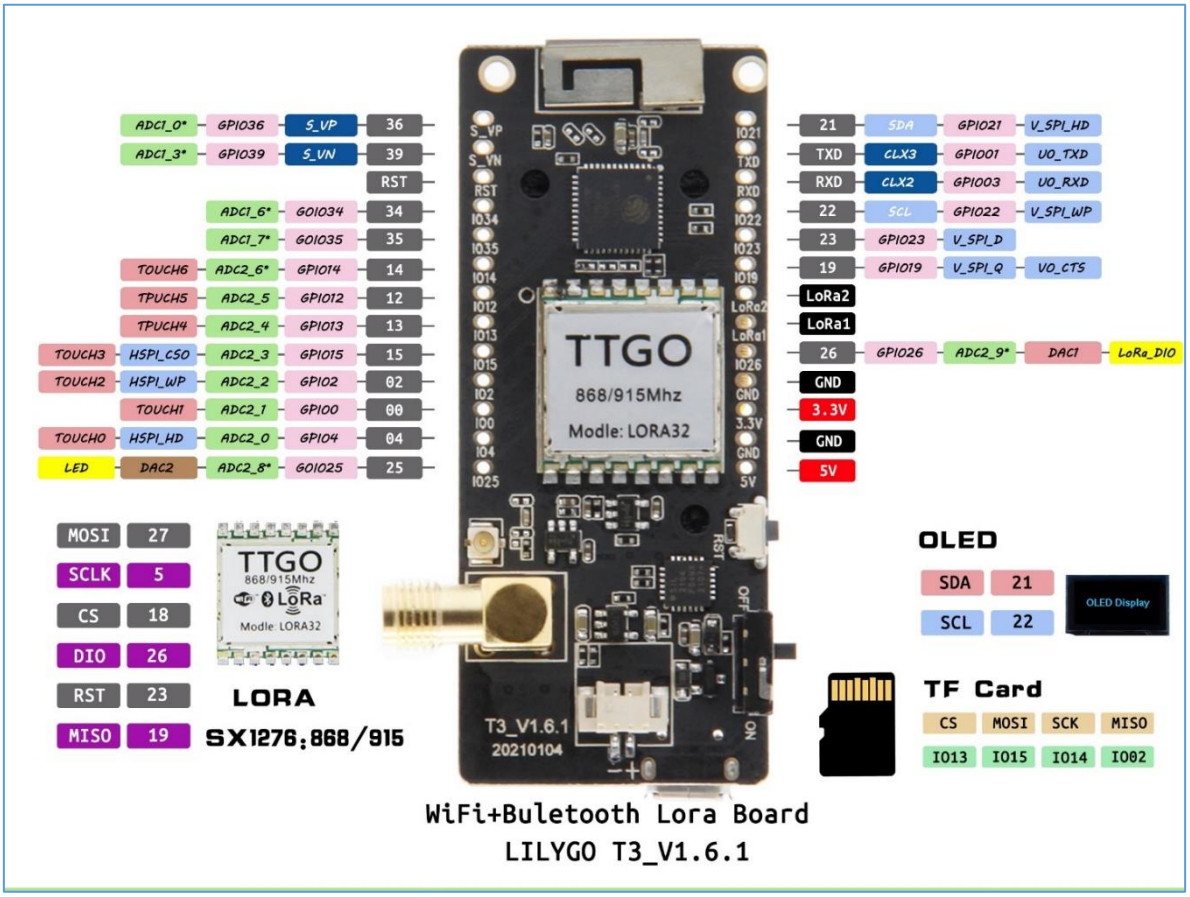
L'antenna deve essere collegata alla presa SMA sul PCB. Per maggiori dettagli su questa scheda rimando al link:

Pin usati per i vari dispositivi sulla scheda:

- Per LoRa: 5,14,18,19,23,26,27
- Per OLED (I^2C): 21,22
- Per SD card (attualmente non è stata testata): 2,4,12,13,14,15
- Per LED verde (usato da LoRa): 25
- Per VBAT: 35

Pin liberi

ADC a 12 bit (0 – 4095): 34, 36, 39, 00



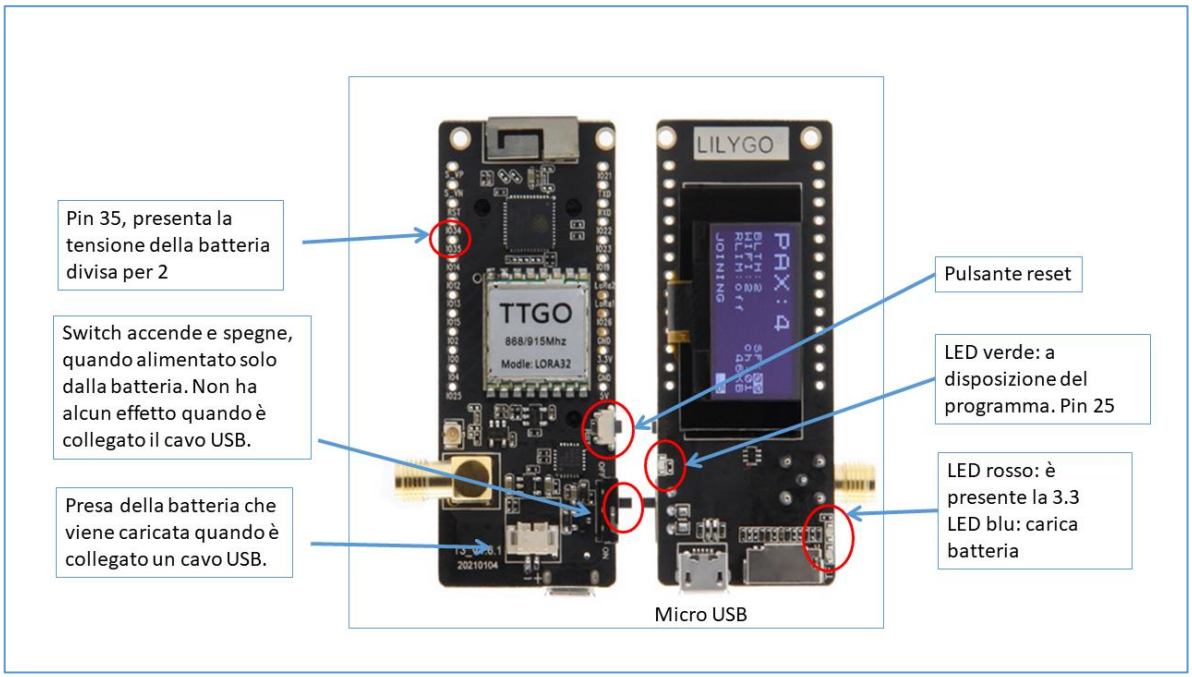


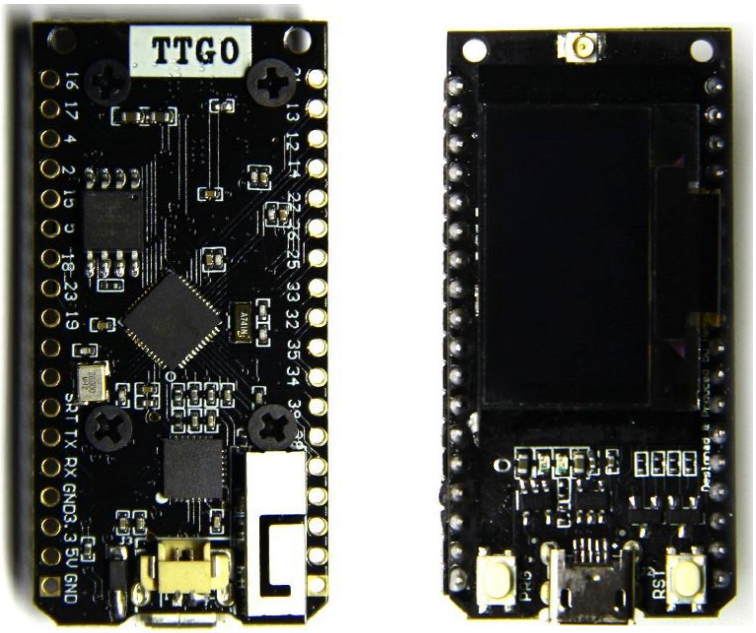
Figura 5 – Alcuni particolari hardware della scheda ESP32

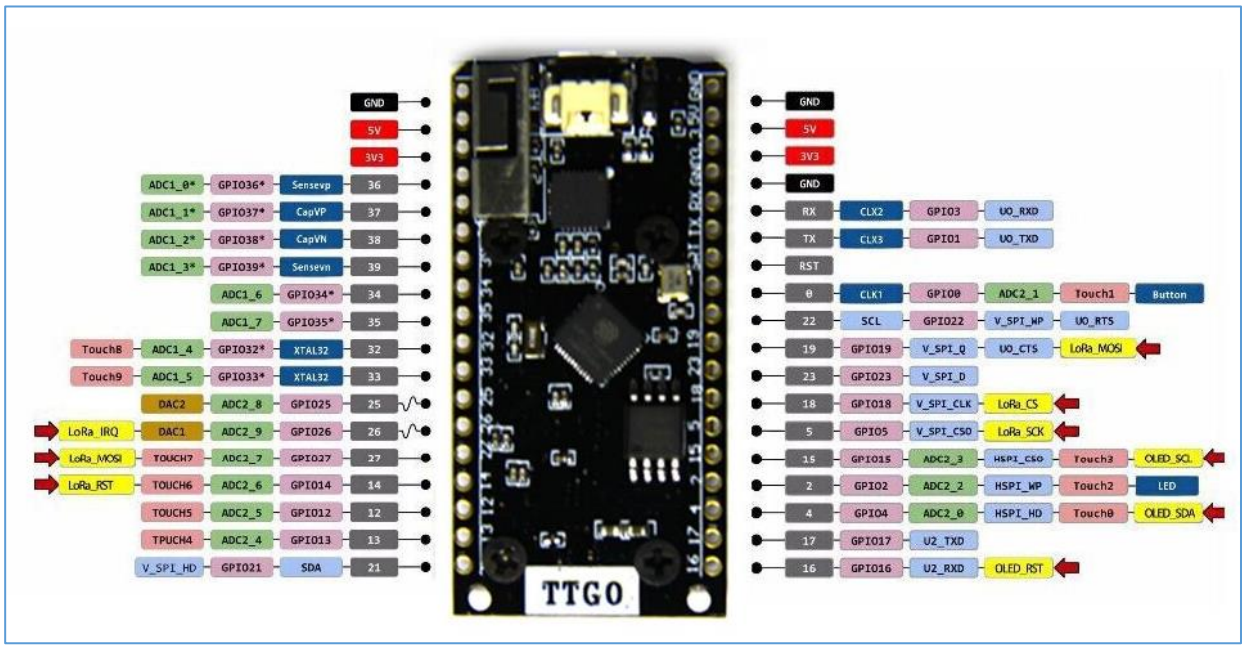
La scheda Ricevitore RX: LILYGO® TTGO LORA32 868/915Mhz (LoRa32 V1.0)

Tra parentesi è il nuovo nome assegnato dalla Lilygo.

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

Questa scheda, anche se ha meno cose a bordo della precedente, è ugualmente valida per costruire un ricevitore per una MiniReteLoRa. Infatti può essere caricata con lo stesso programma cambiando solo alcune righe che definiscono i pin dello schermo OLED.





Software

Il presente capitolo presuppone che il lettore abbia minime nozioni di programmazione C++ come quelle necessarie per programmare Arduino. Se già avete scritto qualche riga di codice con la IDE, non dovrebbe essere difficile capire quanto riportato di seguito; altrimenti più avanti troverete un paragrafo dove spiegherò l'uso di due programmi "Template" sia per TX che per RX, con i quali il lettore potrà costruire sue applicazioni, adattate alla sua MiniReteLoRa, cambiando solo alcune parti del codice senza la necessità di capire tutto il suo significato.

La struttura del messaggio inviato

Prima di passare ad esaminare il software da caricare sui nodi TX ed RX, spiegherò come è fatto il messaggio trasmesso dai nodi TX in una MiniReteLoRa.

Seguendo la Figura 7 vediamo che il messaggio trasmesso è composto da una serie di byte: alcuni costituiscono il "preambolo" e il terminatore del messaggio, altri, quelli che ci interessano, sono il cosiddetto "payload" che conterrà i nostri preziosi dati.

Per chi invece vuole sapere di più circa l'intero formato del messaggio usato dalla libreria RadioHead.h, che gestisce il modulo radio (Transceiver), la composizione del pacchetto trasmesso è riportata in Figura 6.

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

Figura 6 -Struttura completa di un messaggio LoRa

Nel payload possiamo mettere quello che vogliamo e la sua lunghezza può essere variabile (fino a 251 byte), ma è qui che è importante progettare una struttura dei messaggi utile alla loro gestione in fase di ricezione. Quello che ho sperimentato con successo è la struttura visibile in Figura 7. Qui il payload è composto da una parte di lunghezza fissa (8 byte), che chiamo intestazione, e una parte di lunghezza variabile dove andranno

i valori numerici delle grandezze misurate dai sensori dei vari nodi. Quest'ultima è di lunghezza variabile perché potrei avere sensori diversi per ogni nodo TX.

I valori numerici della grandezze misurate dai sensori, nel messaggio, sono codificati in formato binario interno e non in carattere. Per spiegarmi meglio, immaginiamo il numero 25.4 che rappresenta i gradi centigradi di un sensore di temperatura, la sua rappresentazione in byte può essere in formato carattere ASCII (carattere "2", carattere "5", carattere ".", carattere "4") per un totale di 4 byte, oppure in formato interno come una variabile `float` di 4 byte, oppure sempre in formato interno come una variabile `int` di 2 byte se moltiplico per 10 il valore portandolo a 254.

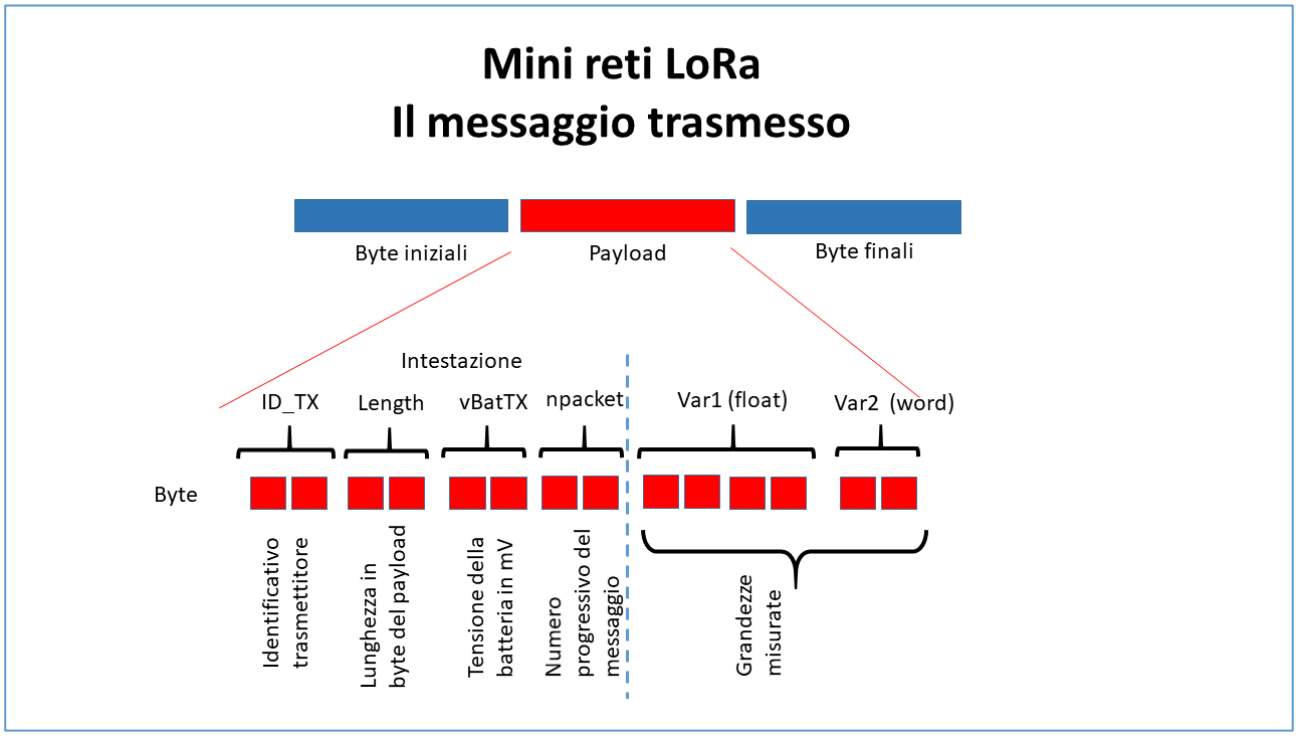


Figura 7 – Struttura del PayLoad di un messaggio per una MiniReteLoRa

Vediamo adesso i vantaggi e svantaggi delle due codifiche. Il formato carattere ASCII ha il vantaggio di essere accettato da qualunque sistema informatico, essendo un codice usato universalmente, lo svantaggio sta nella necessità di occupare un byte per ogni cifra o simbolo, per esempio se la grandezza da trasmettere è una pressione espressa in millibar, 1020.5, ci vogliono 6 byte. Inoltre le CPU dei microcontrollori eseguono i calcoli usando la codifica interna, ad esempio per ricavare una pressione a livello del mare devo inserire quello che il sensore misura in una formula; una volta trovato il risultato, questo dovrà essere convertito in caratteri ASCII e il numero di byte richiesto dipenderà dal risultato di questi conti.

Il formato interno è più compatto, pensate infatti che in una variabile `float` di 4 byte possono starci numeri grandissimi, con segno, con una precisione di 5 o 6 cifre significative. Se devo fare dei conti all'interno del nodo TX sulle grandezze misurate e non conosco a priori il range di variabilità del risultato, usando variabili `float` nel programma posso stare tranquillo. Ma la programmazione C, quella che si usa con questi microcontrollori, prevede l'uso anche di variabili intere di 2 `byte` o di un solo `byte`, utili per compattare ancora di più il nostro messaggio. Teniamo presente che la lunghezza massima consigliata per un messaggio LoRa è di 30-40 `byte` come payload.

Dunque le grandezze misurate dai nostri sensori occuperanno la parte di byte che sta dopo l'intestazione del messaggio, mentre questa è composta da numeri interi in formato interno come descritto in Tabella 1.

Tabella 1 - Formato dell'intestazione del messaggio

Byte	**contenuto**	**Dichiarative delle variabili in formato interno numerico**		**Nome della variabile usata nei programmi**
		C++	**Arduino**	
0 - 1	Indicativo numerico del nodo TX	uint16_t	word	ID_TX
2 - 3	Numero di byte di tutto il payload	uint16_t	word	Length
4 - 5	Tensione della batteria in mV	uint16_t	word	vBatTX
6 - 7	Numero progressivo del messaggio inviato	uint16_t	word	npacket

L'indicativo del nodo (ID_TX) può essere un numero da 1 a 65535, ma, per essere coerenti con i programmi presentati qui, conviene usare sempre numeri di 5 cifre: da 10000 a 65000.

Vediamo adesso le dichiarative per le variabili che contengono le grandezze misurate. Immaginiamo un nodo TX con il numero 12345, che deve trasmettere il contenuto di una variabile `float`, una `int` e una `byte`, per un totale di 7 byte + 8 di intestazione, grandezze prelevate da uno o più sensori, le dichiarative saranno:

```
float temp;   // temperatura dell'aria come numero di gradi centigradi con un decimale (°C)
int umidita;  // umidità dell'aria come numero intero (%)
byte piove;   //  pioggia = 1 o non pioggia = 0
```

La trasmissione dei valori di queste variabili, sia con LoRa o in qualche altra modalità seriale, necessita che venga trasmesso un byte alla volta. Quindi i valori vanno convertiti in un `array` di `byte`.

Questa conversione è illustrata in Figura 8.

Così facendo la CPU può accedere alle aree di memoria che contengono i valori misurati, sia come variabili `float`, `int`, `byte`, sia come singoli `byte` di un `array`.

Alle variabili definite nella struttura **`packet12345`** sarà assegnato un valore nel corso del programma, meglio se per mezzo di una nostra funzione apposita (p.e. ReadSensor), L'unica cosa di cui ricordarsi è che, essendo state definite in una struttura, messa poi in **`union`** con un `array`, verranno chiamate così:

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

Anche gli elementi di **`bufVar`** saranno chiamati in questo modo:

```
minni.bufVar[i]  = …
```

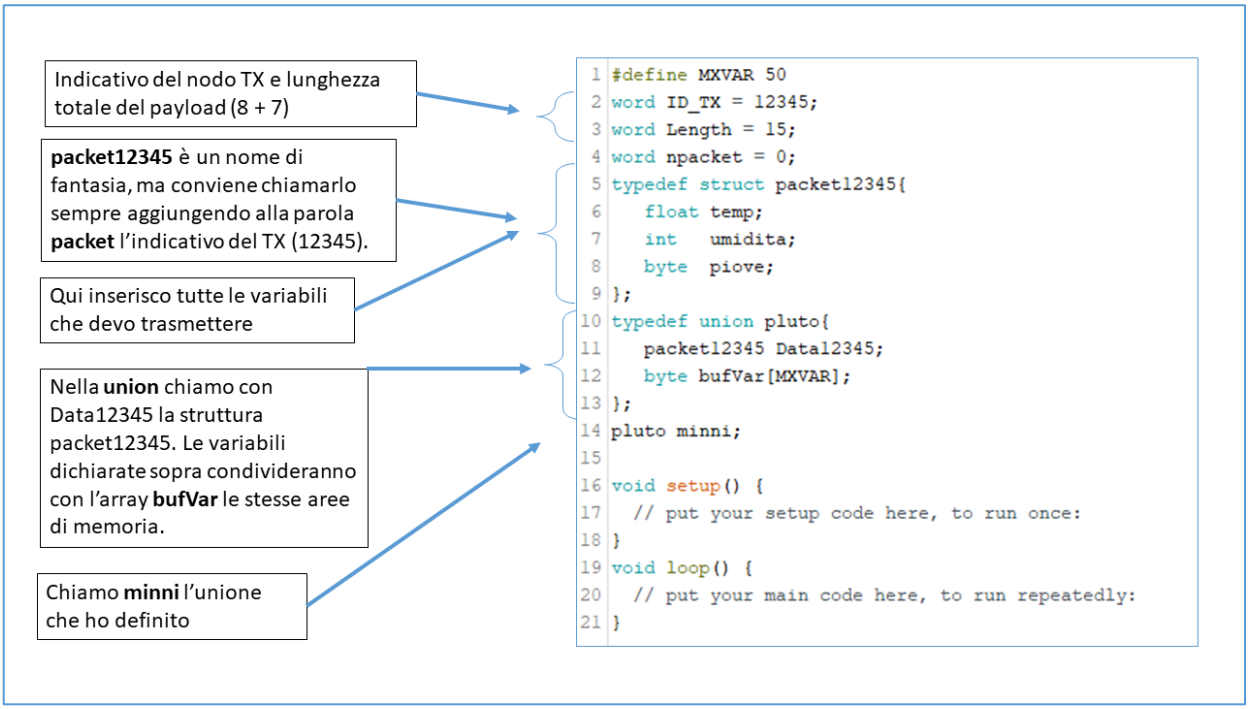


Figura 8 -Dichiarative delle variabili da trasmettere

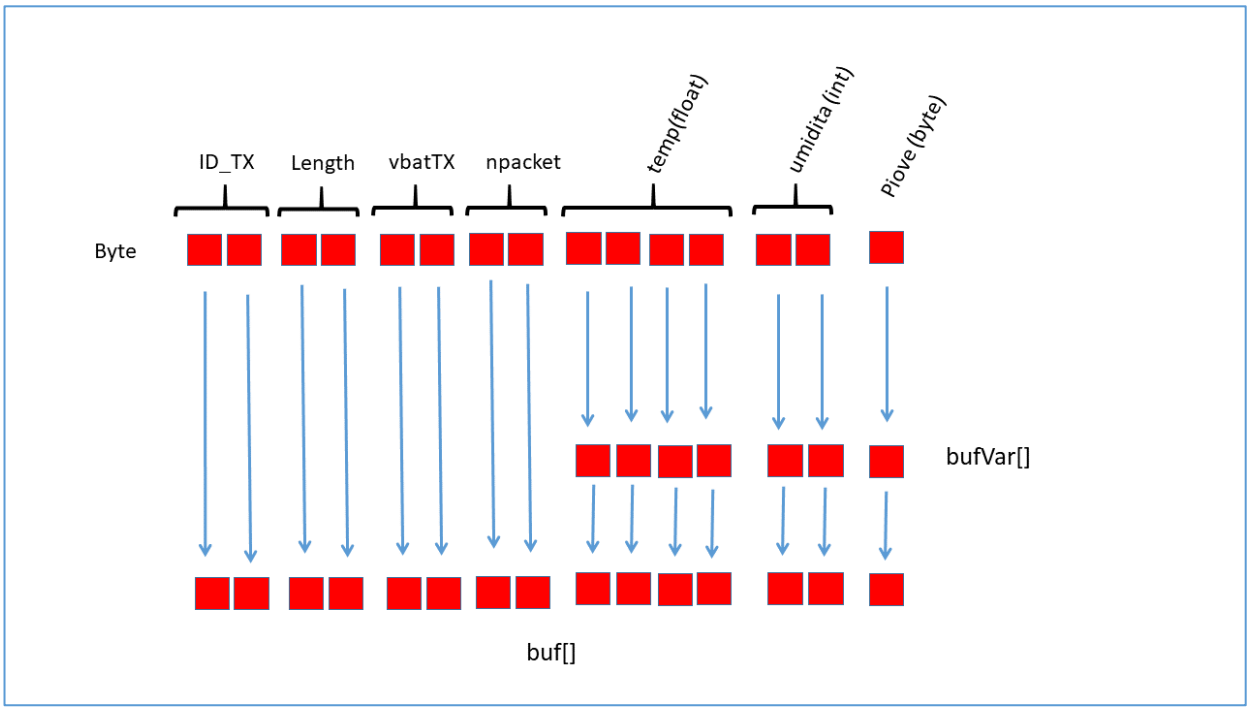


Figura 9 - Corrispondenza tra i byte delle variabili e quelli degli array

E' buona norma dichiarare le variabili in una struttura in ordine decrescente di numero di byte, i.e. prima quelle da 4 `byte`, poi quelle da due, poi le variabili di un solo `byte`. Questo per evitare errori di abbinamento con `array` nella `union`.

Il vantaggio di questo modo di procedere sta anche nella possibilità di effettuare il passaggio inverso. Ad esempio ricevere un array di byte tramite LoRa o un collegamento seriale e decodificarlo in variabili di diverso tipo, dichiarate in una struttura.

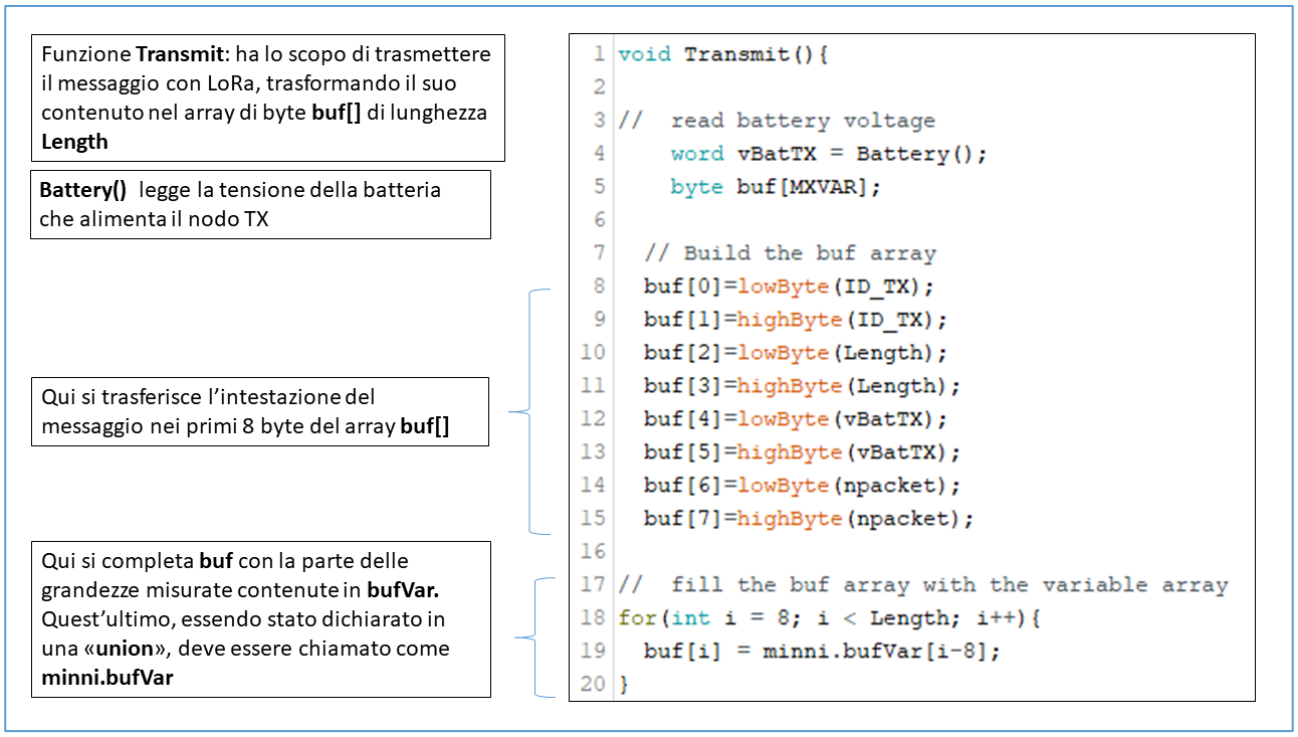


Figura 10 - Funzione Transmit, prima parte

Nella Figura 10 si vede la prima parte della funzione `Transmit`, che è sempre uguale per tutte le applicazioni, e che prepara l'`array buf` caricandogli sia gli 8 `byte` dell'intestazione che l'`array bufVar` contenente le grandezze misurate.

La seconda parte della funzione `Transmit` è visibile in Figura 11 e comprende: le chiamate alle funzioni della libreria `RadioHead` preposte alla trasmissione del messaggio, l'incremento di `npacket` che sarà inserito nel prossimo messaggio e la chiamata alla funzione che fa lampeggiare il LED come segnale diagnostico dell'avvenuta trasmissione.

Lo scopo del numero progressivo di messaggio `npacket` è quello di poter diagnosticare, in ricezione, eventuali messaggi persi.

Programma per un nodo TX: TX_Generic

Spiegherò in questo paragrafo un programma completo per un nodo TX reale, costituito da una scheda TTGO, presentata prima, e un sensore di temperatura e umidità DHT22. Il sorgente di questo programma, scritto tramite la IDE di Arduino, è diviso in "**folder**"; in un folder può esserci una lista di dichiarative (tipo di variabili, `#define`, `#include` di librerie) oppure una funzione. Tutto questo serve per una maggiore chiarezza, ma soprattutto per creare un "**Template**", cioè un programma generico per un TX di una MiniReteLoRa, dove ci saranno solo alcuni folder da modificare adattandoli alle nostre esigenze. Nella Tabella 2 presento i folder del programma `TX_Generic`, che misura temperatura e umidità tramite un sensore DHT22. Questo programma è il "Template" di cui parlavo prima.

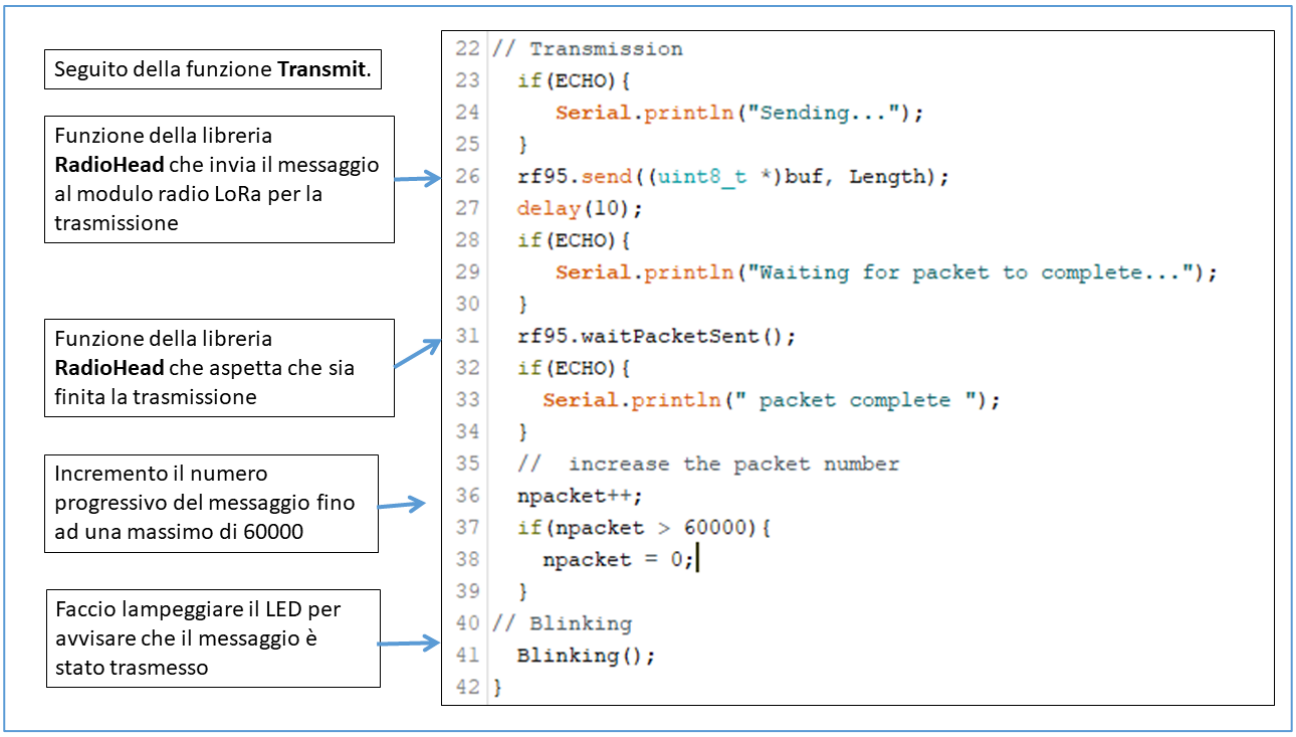


Figura 11 - Funzione Transmit, seconda parte

Tabella 2 - Elenco dei folder nel programma TX_Generic

Nome del folder	**Cosa contiene**	**Da modificare**
`TX_Generic`	Main program	Si Modificare se necessario le variabili `ncicliMax`, `RF95_FREQ`,`RF95_POW`, ECHO, e azzerare nel setup() le variabili contenute in Packet.h
`Battery`	Funzione per leggere la tensione della batteria	No
`Blinking`	Funzione lampeggio del LED	No
`Nword`	Funzione per approssimare all'intero più vicino	No
`Packet.h`	Dichiarative delle variabili misurate dal sensore e da trasmettere	Si Inserire le variabili lette dai vari sensori, che si vuole trasmettere, identificativo del TX e lunghezza del payload
`Radio.h`	Dichiarative riguardanti la radio	No Solo se si usa un hardware diverso, bisogna modificare alcuni #define, peraltro già presenti e commentati.
`RadioInit`	Funzione di inizializzazione Radio	No
`ReadSensor`	Funzione per la lettura sensore	Si Inserire le istruzioni di lettura dei sensori.
`Sens.h`	Dichiarative riguardanti il sensore	Si Inserire le dichiarative necessarie ai sensori comprese eventuali loro librerie
`SensInit`	Funzione di inizializzazione sensore	Si Inserire le inizializzazioni necessarie alla lettura dei sensori
`Transmit`	Funzione per trasmette il pacchetto di dati	No
`altDelay`	Funzione di ritardo	No

Come si può vedere dalla Tabella 2 alcuni folder hanno il nome con un "`.h`" finale. Questi sono usati per le dichiarative di variabili globali e inclusioni di librerie, che normalmente sono inserite sopra la "`void setup()`". Nel programma principale questi folder sono inclusi con l'istruzione `#include`:

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

I folder che contengono funzioni, possono essere chiamati come si vuole, ma per chiarezza è meglio chiamarli con lo stesso nome della funzione, così la funzione

```
void RadioInit(){    …    }
```

sarà dentro al folder RadioInit.

Il folder `TX_Generic` contiene il programma principale, visibile in Figura 12, che è abbastanza corto. Andiamo ad esaminarlo da vicino.

All'inizio del programma, righe 18 – 20, vengono inclusi i folder che contengono le dichiarative valide per i sensori, come `Sens.h`, che contiene anche le librerie necessarie per la loro lettura e come `Packet.h` che contiene le variabili che dovranno essere trasmesse, come spiegato precedentemente.

Alle righe 21 e 22 sono definiti i valori di frequenza in MHz e della potenza di trasmissione, il cui massimo è 23 dBm.

Alle righe 23 e 24 sono definite una variabile contatore del numero di giri del `loop` e il numero massimo di questa, raggiunto il quale verrà chiamata la funzione che trasmette il messaggio. La variabile `ECHO` serve, se posta a 1, ad attivare le scritture di controllo sul monitor seriale; per alcune schede, ma non la nostra, è necessario porre ECHO a 0 quando non si usa più il monitor seriale.

Nel `setup` sono inserite tutte le istruzioni che servono ad inizializzare:

- la radio (riga 28);
- frequenza e potenza (righe 30 e 35);
- sensori (riga 39).

Le successive righe 42 e 43, inizializzano le due variabili da trasmettere, dichiarate in `Packet.h`, che poi verranno trasmesse alla riga 44.

La funzione `altDelay` alla riga 45 ha lo stesso scopo della `delay`**,** ma è creata con istruzioni di ritardo costruite ad hoc.

La libreria `RadioHead`, usata per la trasmissione radio, consente di impostare altri parametri oltre la frequenza e la potenza; sono i parametri tipici della trasmissione LoRa, quali: Bandwidth, Code rate, Spreading factor. Le mie prove sono state fatte sempre con le impostazioni di default per questi parametri e mi sono sempre trovato bene, per chi volesse "divertirsi" a cambiare questi parametri, consiglio di leggersi la documentazione della libreria, di cui ho riportato il link più avanti.

La frequenza di trasmissione può essere impostata da 868.0 a 868.6 MHz a passi di 0.1.

```
18 #include "Radio.h"
19 #include "Sens.h"
20 #include "Packet.h"
21 #define RF95_FREQ 868.0  // LoRa Frequency
22 #define RF95_POW   23    // Transmitting power dBm
23 word ncicli =    0;
24 word ncicliMax = 10;     // Number of loop cycles after which it transmits
25 byte ECHO  = 0 ;
26 void setup() {
27   //  Initializing the radio
28     RadioInit();
29     Serial.println(F("TX_Generic.ino"));
30     rf95.setFrequency(RF95_FREQ);
31     if (ECHO) {
32       Serial.print(F("Set Freq to: ")); Serial.println(RF95_FREQ);
33     }
34 //  Set transmission Power
35     rf95.setTxPower(RF95_POW, false);
36     if (ECHO) {
37     Serial.print(F("Set Power to: ")); Serial.println(RF95_POW);
38   }
39   SensInit();
40
41 //   First transmission with all the variables at 0
42     minni.Data30003.Humi = 0;
43     minni.Data30003.Temp = 0;
44     Transmit();
45     altDelay(5000);
46 }
48 void loop() {
49   ncicli++;
50   if(ncicli >= ncicliMax){
51    ReadSensor();
52 // Transmit data
53     Transmit();
54     ncicli = 0;
55   }
56   if(ECHO){
57      altDelay(5000);
58   }else{
59 //  The radio goes to sleep
60      rf95.sleep();
61 // It goes to sleep for 8 seconds.
62      Watchdog.sleep(8000);
63   }
64 }
```

La parte di loop del programma è fatta in modo che ad ogni ciclo la CPU e la radio vengano messe in sleep (righe 58 – 63), consumando meno energia possibile, per otto secondi (questo è il massimo accettabile dalla funzione `Watchdog.sleep`).
Per allungare il tempo di sleep si ricorre al contatore `ncicli`, che, raggiunto il suo valore massimo `ncicliMax`, passa il controllo alle funzioni ReadSensor e Transmit (righe 51 – 53).

Durante la fase di test del programma, ponendo `ECHO = 1`, si può avere il ritardo che si vuole senza ricorrere allo sleep, agendo sui valori di `altDelay` e `ncicliMax`.

Figura 12 - Main program di TX_Generic

Il folder `Packet.h` si presenta così:

```
1  //  30003;    DHT22        byte 8 + 8;
2  uint16_t ID_TX = 30003;
3  uint16_t Length = 16;
4  uint16_t   npacket = 0;
5  // put here your list of variables to be transmitted
6  typedef struct packet30003{
7  float      Temp;
8  float      Humi;
9  };
10 //  these declarations must not be changed
11  typedef union pluto {
12   packet30003 Data30003;          //  Dataxxxxx is the n
13   byte bufVar[MXVAR];             //  array of bytes to
14 };
15 pluto minni;                     //  minni is the name o
```

Figura 13 . Folder Packet.h del programma TX

Gli altri folder sono ampiamente commentati all'interno del sorgente e quindi ritengo che non sia utile riportarli qui nel testo. Invito il lettore a leggere e capire per prima cosa i folder che devono essere modificati (vedi Tabella 2) che sono abbastanza semplici. Gli altri folder, che non richiedono modifiche, possono essere tralasciati.

Struttura del programma per un nodo RX

RX_Generic_autoconnect

ATTENZIONE! Il programma `RX_Generic_autoconnect` è ancora in fase di test per problemi di stabilità. Al suo posto può essere usato `RX_Generic_NoAutoconnect`, che necessità la scrittura all'interno del programma delle credenziali di accesso a WiFi e a Server

In questo capitolo spiegherò il programma che permette alle due schede ESP32 presentate prima, di ricevere via LoRa i dati di molti nodi TX, anche diversi tra loro, e trasmetterli ad un server su Internet.

Il server che ho scelto è quello di AdafruitIO (https://io.adafruit.com/) che offre spazio gratuito e la possibilità di creare grafici interattivi con il proprio browser.

La scheda RX può fare anche altre cose, come ad esempio attivare dei LED se si superano alcune soglie delle grandezze ricevute, allarmi e attuatori vari, tutto questo cambiando aggiungendo al programma poche righe.

Anche qui, come per il nodo TX, il programma è suddiviso in folder, alcuni da non modificare ed altri da adattare alle proprie esigenze.

Autoconnessione della scheda RX al server

Il programma `RX_Generic_autoconnect` è stato scritto per ricevere tre nodi TX diversi (30002, 30003, 50000) dei quali il 30002 è quello a cui si riferisce il nodo TX presentato precedentemente (`TX_Generic`).

Il programma usa delle librerie particolari che permettono l'auto-connessione ad una rete WiFi e ad un account del server AdafruitIO. Questa funzione offre il vantaggio di non dover scrivere le credenziali di accesso nel programma e doverlo così ricaricare ogni volta che si fa un cambiamento di rete o di account al server.

L'utilità di tutto ciò sta nel fatto che è possibile costruire una mini rete LoRa composta da nodi sensori e un ricevitore gateway e poi trasportarla ovunque senza bisogno di modificare e ricaricare il programma.

Il programma dovrà essere modificato solo se cambiano i nodi TX da ricevere e/o i loro messaggi. Nel seguito si spiega come fare questi cambiamenti.

Tabella 3 - Elenco dei folder del programma RX_Generic_autoconnect

folder	Cosa contiene	Da Modificare
`RX_Generic_autoconnect`	Main program Funzioni della libreria AdafruitIO	Si
`Battery`	Legge la tensione di un eventuale batteria di alimentazione collegata alla presa JST	No
`Blinking`	Funzione che fa lampeggiare un led ad ogni messaggio ricevuto.	No
`Packet.h`	Contiene: Dichiarative delle variabili misurate dai vari sensori TX che vogliamo ricevere. Le strutture sono uguali a quelle dei singoli sketch dei TX.	Si
`Print30002 ; Print30003; Print50000`	Funzioni che portano il nome del TX e servono a stampare su monitor seriale e su eventuali Display. Ce n'è una per ogni TX.	Si, da creare per ogni nuovo TX
`Radio.h`	Dichiarative per la radio LoRa	No
`Receive`	Funzione per la ricezione dati via radio LoRa	No
`SaveConfig.h`	Gestisce la configurazione della rete	No
`board_defs.h`	Dichiarative dei pin della scheda	Si, solo se si cambia scheda
`ReadParameter`	Legge i parametri della configurazione	No
`AutoConnection`	Gestisce le connessioni alla rete e al server	No

`RX_Generic_autoconnect` è il main program, ha bisogno di essere modificato solamente nella parte `loop()`.

Come si può vedere dal pezzo di codice in Figura 14, nel "if" principale si eseguono gruppi di istruzioni a seconda dell'identificativo del nodo TX (ID0, ID1…) definiti nella sezione `Packet.h` descritta più avanti. Nei gruppi vengono dichiarati i "feed" di **AdafruitIO** dove andranno i valori dei dati ricevuti e le istruzioni per inviarli. Non è necessario che i nomi dei Feed siano gli stessi delle variabili che contengono i dati, ma risulta più chiaro chiamarli allo stesso modo.

`Packet.h` contiene l'elenco degli indicativi dei nodi TX da ricevere assieme alle rispettive strutture dei dati, così come sono scritte nello stesso folder dei nodi TX (vedi Figura 15 e Figura 16).

```
void loop() {
  io->run();
  byte Roger = 0;
  Roger = Receive();
  if(Roger){
//  Identification and re-transmission
        if (ID_TX == ID0){
              AdafruitIO_Feed *Temp = io->feed("Temp");
              AdafruitIO_Feed *Humi = io->feed("Humi");
              Temp ->    save(minni.Data30003.Temp);
              Humi ->    save(minni.Data30003.Humi);
              Print30003();
        }else if(ID_TX == ID1){
              AdafruitIO_Feed *Tair = io->feed("Tair");
              AdafruitIO_Feed *Umid = io->feed("Umid");
              AdafruitIO_Feed *Light = io->feed("Light");
              Tair->    save(minni.Data50000.Tair);
              Umid->    save(minni.Data50000.Umid);
              Light->    save(minni.Data50000.Light);
              Print50000();
        }else if(ID_TX == ID2){
              AdafruitIO_Feed *npulse = io->feed("npulse");
              AdafruitIO_Feed *sPeriod = io->feed("sPeriod");
              npulse->    save(minni.Data30002.npulse);
              sPeriod->    save(minni.Data30002.sPeriod);
              Print30002();
        }else{
              Serial.println(" ID_TX not permitted ");
        }
    Roger = 0;
  }
}
```

Figura 14

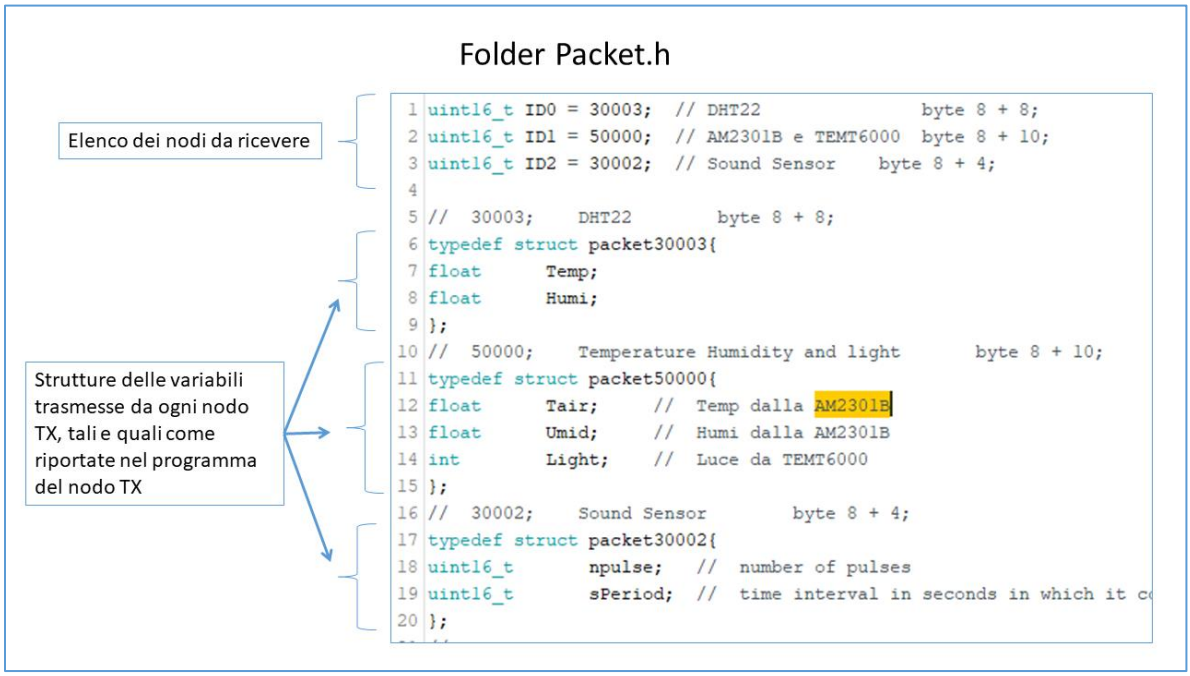


Figura 15 - Folder Packet.h del programma per RX

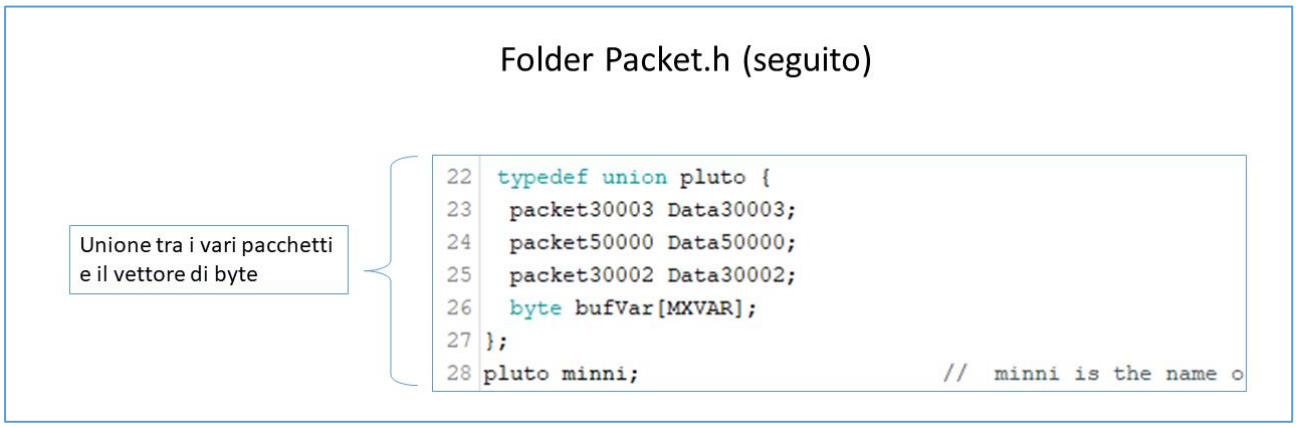


Figura 16 - Folder Packet.h (seguito)

RX_Generic_NoAutoconnect

Questo programma è più semplice di quello descritto prima. In pratica non richiede le librerie per l'autoconnessione contenute nella cartella principale e non ha le seguenti cartelle:

`SaveConfig.h`	Gestisce la configurazione della rete
`ReadParameter`	Legge i parametri della configurazione
`AutoConnection`	Gestisce le connessioni alla rete e al server

In aggiunta invece è inserita la cartella `ServerConfig.h` che contiene le credenziali di accesso alla rete WiFi e al server AdafruitIO

Il livello del segnale RSSI e le antenne

Il parametro RSSI (Receive Signal Strenght Indicator), restituito da un'apposita funzione della libreria LoRa, indica la forza del segnale radio LoRa ricevuto. E' espresso in dBm negativi, per cui, se lo visualizziamo in valore assoluto, questo numero può andare da qualche decina, con TX e RX vicini, a circa 110 con TX e RX alla distanza di 1 Km o più. La relazione tra distanza e RSSI non è univoca, dipende da molti fattori esterni che attenuano il segnale, come orografia, vegetazione e tempo atmosferico. Nel programma RX_Generic questo parametro viene visualizzato sul monitor seriale e sul display.

Anche il tipo di antenna usata per TX e RX influenza il RSSI. In base alla mia esperienza consiglio di usare o antenne a stilo o quelle "Ground Plane" più costose ma più performanti, come in Figura 17.

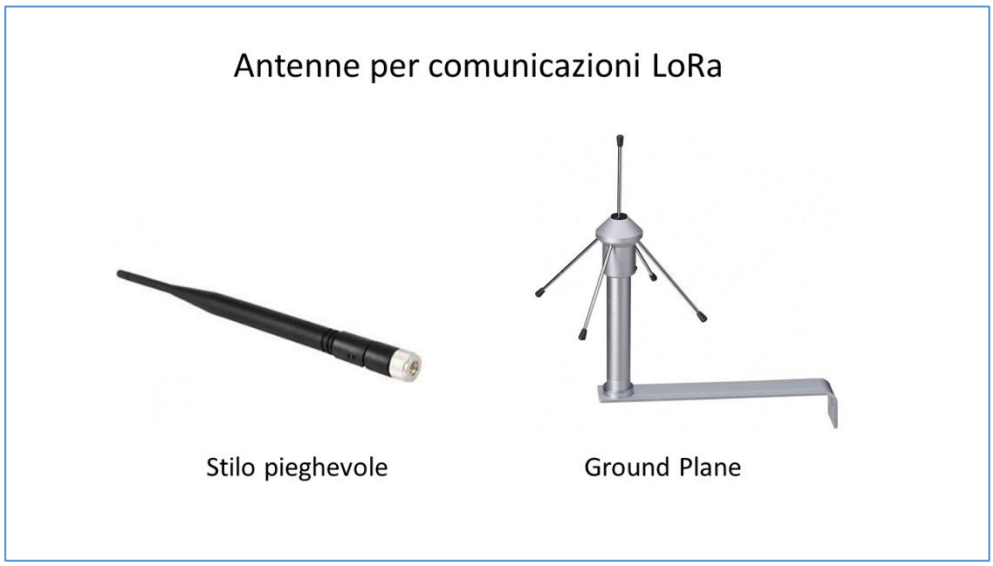


Figura 17 - Antenne per TX e RX

Librerie

Per il programma **`TX_Generic`** si usano le seguenti librerie:

```
#include <RH_RF95.h>
```
Questa libreria fa parte del pacchetto RadioHead, la cui documentazione è reperibile qui:

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

```
#include "Adafruit_SleepyDog.h"
```
Contiene le funzioni per mandare in sleep la CPU.

```
#include xxxx
```
Eventuali librerie necessarie per la lettura dei sensori.

Per il programma **`RX_Generic_autoconnect`** si usano le seguenti librerie:

```
//  Gestisce le trasmissioni LoRa by Sandeep Mistry
 #include <LoRa.h>

// Gestiscono le operazioni di collegamento alla rete WiFi e al server AdafruitIO
#include <WiFiManager.h>
#include <ArduinoJson.h>
#include <LittleFS.h>
#include "AdafruitIO_WiFi.h"
```

```
// interfaccia I2C
#include <Wire.h>

// Libreria per il funzionamento del display  OLED incorporato nella Scheda
#include <Adafruit_SSD1306.h>
```

Tutte le librerie possono essere incluse nella IDE tramite la funzione “Gestione Librerie” della stessa IDE di Arduino. Occorre anche includere i seguenti link nella finestra “URL aggiuntive per il gestore schede” nella voce “Impostazioni” del menù “File”.

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

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

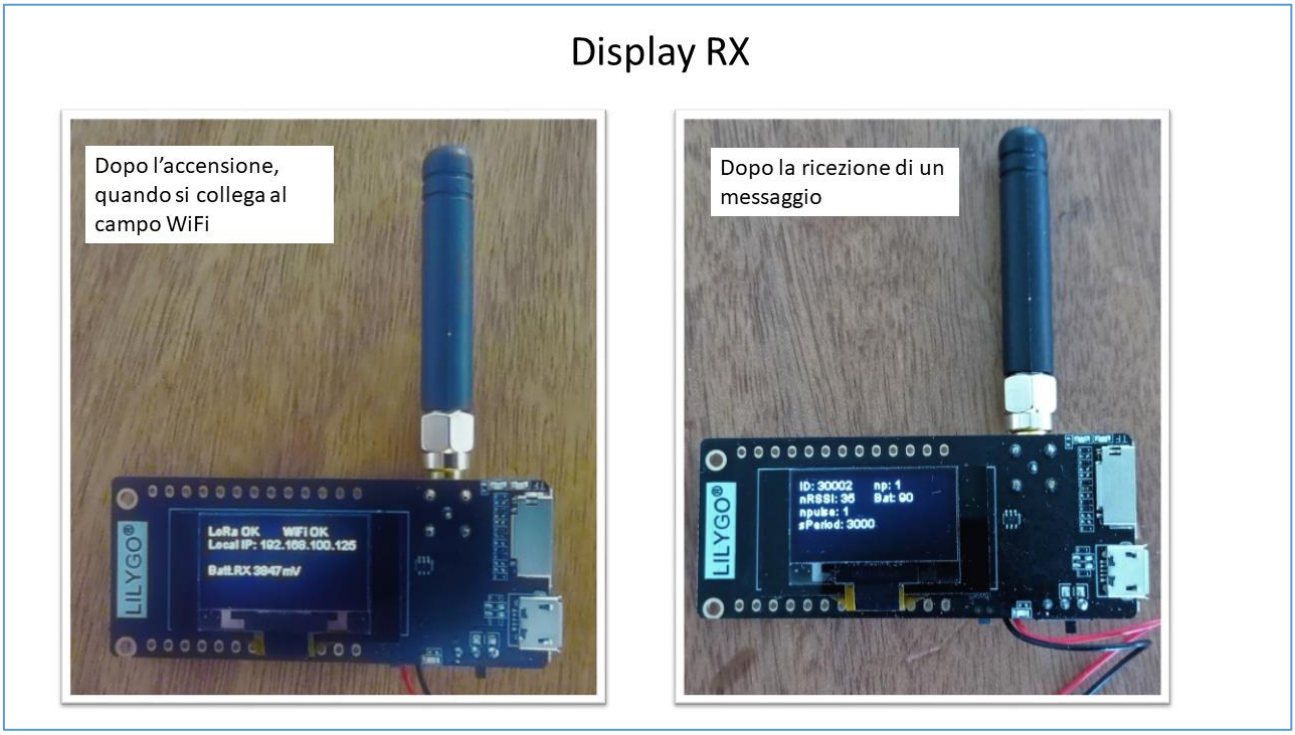


Figura 18 - Due momenti del funzionamento del RX

Server esterni

Come detto precedentemente, si è usato il server della ditta Adafruit:

https://io.adafruit.com/

dove è possibile creare un account gratuito solo con un proprio indirizzo di posta elettronica. Questo servizio consente di inviare dati ad un database con il programma `RX_Generic_autoconnect`, descritto sopra, e realizzare rappresentazioni grafiche di vario tipo.

Una volta avuto l'account, non c'è bisogno di predisporre niente sul server Adafruit IO affinchè questo acquisisca e memorizzi i dati che gli mandiamo. Bisogna invece prendere nota delle credenziali di accesso che si trovano alla voce "Key" del menù in alto che compare una volta collegati. Cliccando sulla voce "Key", compare la finestra in Figura 19, dove basta copiare le due righe in basso, sotto la scritta Arduino .

Una volta inviati i primi dati al server Adafruit, si può creare una "Dashboard" che conterrà i grafici che imposteremo. La creazione di grafici è molto semplice e si fa per mezzo del nostro browser e dei menù presenti sul sito. In Figura 200 è visibile un grafico temporale della variabile `npulse` trasmessa dal nodo TX 30002.

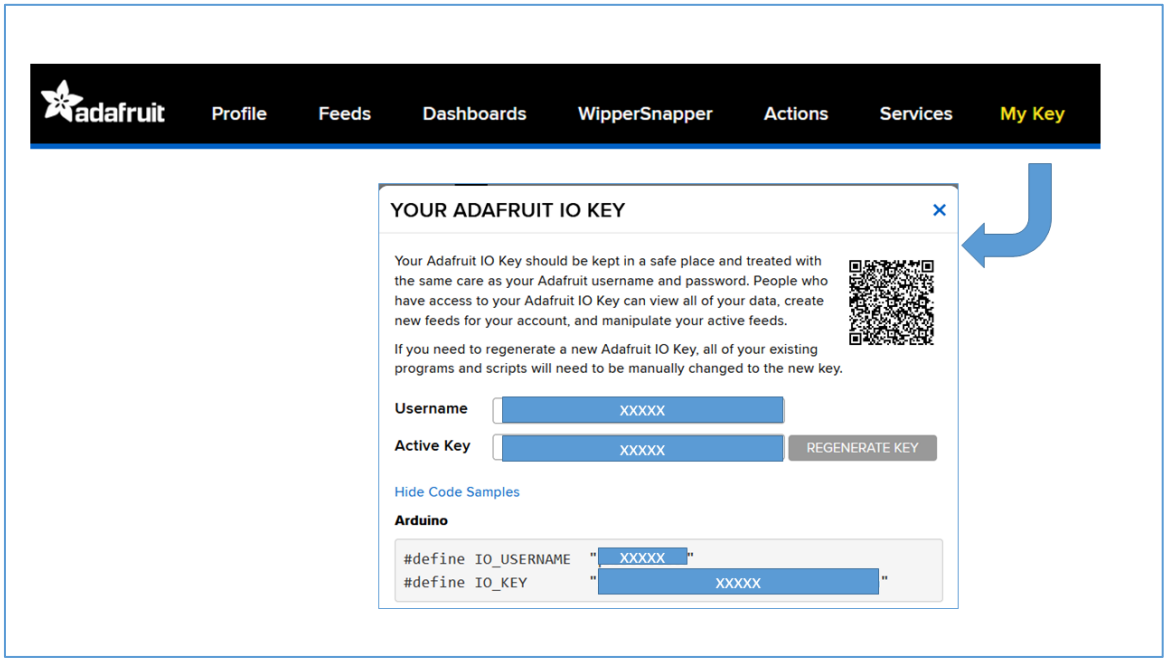


Figura 19 - Credenziali di accesso al server Adafruit IO

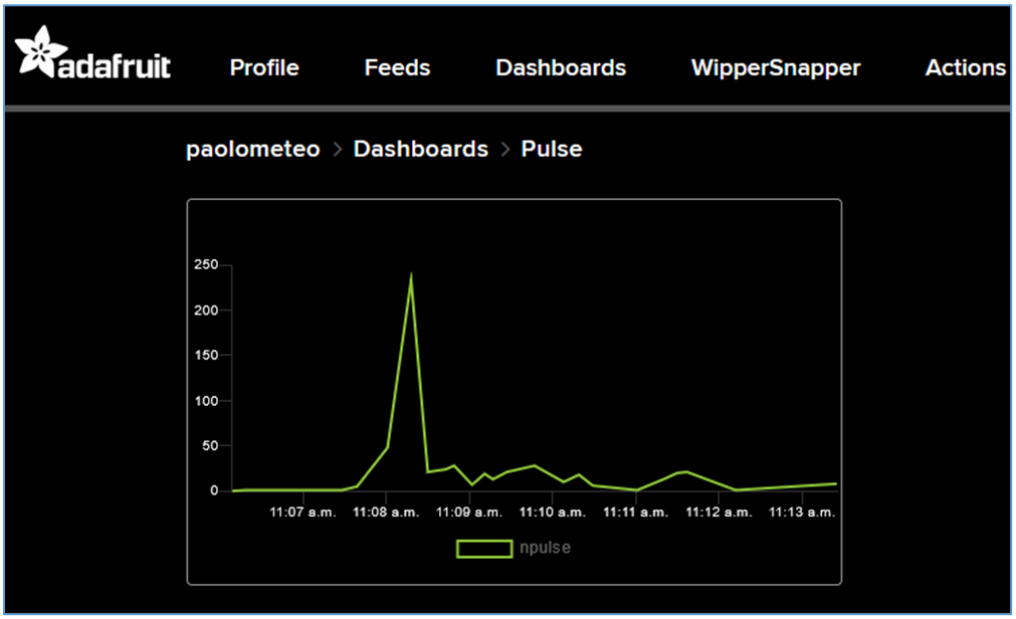


Figura 200 - Esempio di grafico temporale creato nella dashboard di AdafruitIO

Funzionamento del nodo RX (autoconnect)

Quando si dà alimentazione al nodo RX, questo genera una sua rete WiFi con il nome (SSID):

`WiFi Setup.`

- Questo SSID compare nella lista delle reti attive sul nostro PC o cellulare.
- Selezioniamo questa rete e aspettiamo la notifica di avvenuto collegamento; non è richiesta la password.
- Subito siamo avvisati con una notifica che dobbiamo visualizzare il menù sul browser; se ciò non avviene, possiamo collegarci all'URL: `http://192.168.4.1.` Il menù appare come qui sotto.

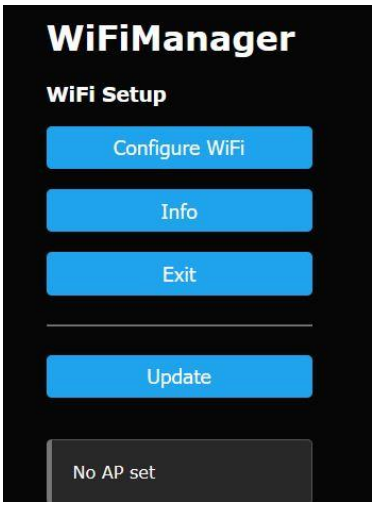


Figura 21 - Menù di collegamento

Cliccando il primo tasto azzurro "Configure WiFi" si passa ad un'altra pagina dove si possono scrivere SSID e password della rete WiFi a cui si deve connettere la scheda RX (il programma presenterà una lista delle reti attive) e le credenziali di accesso al server Adafruit IO.

Infine si preme il tasto "Save" per chiudere questo breve dialogo con la scheda. Il ricevitore si connette alla rete WiFi scelta e al server Adafruit IO, l'avvenuta connessione è visualizzata sul display OLED. A questo punto ogni messaggio LoRa ricevuto, se tra quelli dichiarati, è girato al server.

Funzionamento del nodo RX (NoAutoconnect)

E' necessario inserire nel programma le credenziali di accesso alla rete WiFi e al server AdafruitIO nella cartella ServerConfig.h. Poi il programma va compilato e caricato sulla scheda RX.

Prove di affidabilità di una MiniReteLoRa

Qualcuno potrebbe avere dei dubbi sull'affidabilità di una MiniReteLora specialmente quando più nodi TX trasmettono ad un ricevitore tutti sulla stessa frequenza. Ebbene una valutazione di questa affidabilità viene da un'applicazione pratica realizzata presso la Cascina S. Ambrogio di Milano con una stazione meteo-ambientale dotata di tre nodi TX che trasmettono le misure di molte variabili, 27 per l'esattezza, verso un nodo RX che le memorizza su una scheda SD mediante un datalogger. I dati che sono stati raccolti ed analizzati vanno dal 12/03/2022 al 13/04/2022, poco più di un mese.

I tre nodi TX inviano dati con tre periodicità diverse:

nodo	periodo (secondi)
12120	87
12121	278
12122	179

I nodi TX erano distanti circa 500 metri dalla stazione di ricezione RX.

Avendo a disposizione la data e l'ora di arrivo dei messaggi di ogni nodo e il numero progressivo del messaggio è stato calcolato il numero di messaggi arrivati con l'intervallo di un periodo, di due periodi, i.e. un messaggio saltato, e più di due. In tabella sono rappresentati i risultati in percentuale.

Nodo TX	Totale messaggi	Messaggi arrivati dopo 1 periodo	Messaggi arrivati dopo 2 periodi	Messaggi arrivati dopo 3 periodi
12120	31669	99,37 %	0,62 %	0,01 %
12121	9900	99,31 %	0,67 %	0,01 %
12122	15436	99,40 %	0,60 %	0

Come si può notare la percentuale di messaggi persi dai tre nodi è molto bassa, nonostante non sia stato predisposto alcun sistema di sincronizzazione sull' invio dei messaggi e la frequenza LoRa utilizzata sia stata la stessa per tutti.

Questa prova dimostra che una MiniReteLoRa dotata di 3 nodi TX che inviano messaggi di lunghezza diversa e con periodicità diversa ad un solo nodo RX, ha un'affidabilità di tutto rispetto e promette risultati interessanti anche con un numero di nodi maggiore.

Conclusioni

Siamo arrivati alla fine di questa lunga trattazione su come costruirsi una MiniReteLoRa, spero di essere stato chiaro nella spiegazione e aver stimolato l'interesse nel lettore. Per alcuni di voi, esperti di C++, forse i programmi presentati non saranno il massimo della perfezione, comunque, essendo tutto open source, potete cimentarvi a modificare quello che volete, nei limiti della licenza Creative Commons citata all'inizio dell'articolo.

Tutto quello che ho esposto è stato provato e funziona. I programmi possono essere scaricati a questo link.

https://github.com/paolometeo/MiniReteLoRa

Rimango disponibile a rispondere ai vostri commenti e segnalazioni di possibili errori.

Buon Lavoro!

Paolo