LoRa Library LoraNode (V3.0)

Introduzione

La nuova classe **LoraNode** è stata introdotta per semplificare al massimo la creazione e l'utilizzo di un nodo software basato sullo shield per Arduino o sulla mini scheda Maduino o sul micro radio-comando. In ogni caso utilizza il modulo radio SX1278 SEMTECH o un equivalente come lo RFM98.

Questa classe utilizza e maschera la più complessa classe LORA, che a sua volta utilizza la gestione di base del modulo (classe SX1278). In ogni caso le funzioni sottostanti possono essere sempre utilizzate facendo riferimento alla notazione LR. o SX.

Prima di parlare della classe LoraNode occorre illustrare la struttura della rete in cui un nodo LoraNode si inserisce.

La rete:

- è configurabile in termini di numero di indirizzi utilizzabili per i nodi; maggiore è il numero di periferiche minore saranno i possibili valori utilizzabili per l'identificativo della rete; (numero definibile di nodi: da 7 a 8191)
- è criptata con metodo AES256 la cui chiave estesa è generata da un “seme” intero;
- un byte casuale (marker) è aggiunto al messaggio per ulteriori usi da parte dell'utente;

La classe LoraNode utilizza diversi valori predefiniti per ridurre al massimo il coding. Ovviamente questi valori possono essere sovrascritti. In particolare sono predefiniti:

- la tipologia: 15 periferiche al massimo
- l'identificativo di rete: 2345 (ma può essere uno qualunque tra 1 e 4095)
- una chiave per la crittografia (“seme”)
- la frequenza radio : 433.6 Mhz
- lo spreading factor : codice 10
- l'ampiezza di banda : codice 8
- la potenza : codice 2 (10dBm ovvero 10mW)
- il buffer di trasmissione/ricezione : 64 char (64 byte + null byte) (null terminated string)

L'identificativo del nodo stesso, ovvero il suo indirizzo all'interno della rete, è caricato dalla EEPROM se è presente in essa, ovvero se è stato salvato, oppure può essere fornito direttamente all'atto dell'istanza dell'oggetto nodo.

Protocollo di comunicazione

La libreria utilizza un protocollo proprietario definito dalla classe LORA. Questo protocollo è inserito all'interno del protocollo LoRa del modulo radio. Ovvero il segmento dati del protocollo LoRa è composto a sua volta dai seguenti campi (gli ultimi 3 criptati insieme come un unico campo):

net_id&node_id (to)	net_id&node_id (from)	marker	message

La trasmissione del modulo radio è comunque **half-duplex** (o trasmette o riceve)

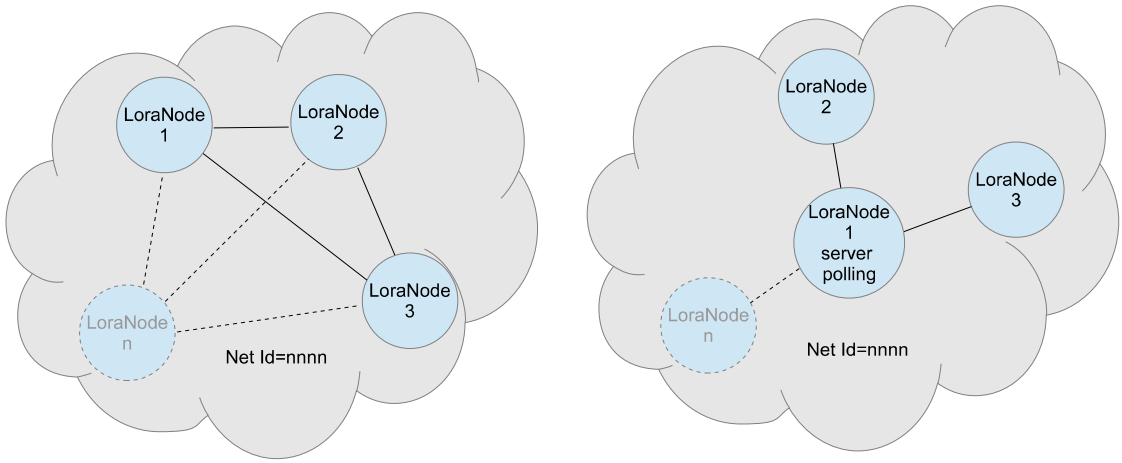

Fig. 1 - Mesh network or Star network

Funzioni principali

- Istanza : **LoraNode()** ; in questo caso l'indirizzo del nodo stesso viene caricato dalla EEPROM se è stato precedentemente salvato. Altrimenti rimane predefinito come 1; in questo caso si può utilizzare la funzione per definirlo successivamente (setLoraNode(id)).
  Oppure : **LoraNode(numero_nodo)**.
- Attivazione del modulo radio: **begin()**; se il modulo ha problemi ritorna “falso”
- Spedisce un messaggio: **writeMessage(a_chi,stringa,timeout)**; la funzione controlla se l'antenna rileva del traffico sullo stesso canale (stessa frequenza,stesso spreading factor e stessa banda) ed attende al massimo per “timeout” millisecondi finché non trova il canale libero. In caso contrario ritorna (“falso”) senza spedire.
- Prova a ricevere: **newMessAvailable(timeout)**; se è arrivato un messaggio entro i “timeout” millisecondi ritorna “vero” altrimenti ritorna “falso”. Esiste anche la versione che è specializzata a ricevere solo da un certo mittente. NB il buffer di ricezione predefinito è lungo 64 caratteri; se si vogliono ricevere messaggi più lunghi bisogna ampliare il buffer con la funzione changeMessageBufferLen(int maxMesslen)
- Legge il testo (decodificato) e le altre caratteristiche del messaggio appena arrivato:
  - **getMessage()**;
  - **getSender()**;
  - **getMarker()**;

Oltre alle funzioni base sono presenti le funzioni che permettono di sovrascrivere i valori di default e di salvare l'indirizzo del nodo ed altre caratteristiche della rete su EEPROM. Tutte le funzioni sono illustrate nelle pagine di help e sono commentate anche nel file di libreria “LoraNode.h”.

In particolare è presente una funzione (setAutomaticAck(vero/falso)) che attiva o disattiva una risposta automatica di conferma (i due caratteri”AK”). Se attivata, alla ricezione viene spedita la conferma, ed alla spedizione viene attesa la risposta di conferma. Per definizione è disabilitata.

In ogni caso va tenuto sempre presente che il modulo radio è half-duplex, per cui non può ricevere mentre trasmette e la gestione di questi flussi va gestita con attenzione per non perdere messaggi.

La classe LoraNode è orientata alla trasmissione di messaggi stringa (a caratteri), se si vogliono trasmettere dati binari si possono usare le versioni a byte delle funzioni writeMess() e newMessAvailable(); informazioni maggiori le trovate sulle pagine di help e sul file LoraNode.h

Esempio base di sketch

Semplice software per un nodo che esegue richieste dalla rete e spedisce la comunicazione di eventi locali a destinatari prestabiliti.

```
#include "LoraNode.h"    //Include libreria
LoraNode Node;           //Instance (se  node_id è presente in EEPROM lo carica)
                         //altrimenti usate: LoraNode Node(id); perché se no id=1
bool SHIELD=true;        //Verifica del collegamento con il modulo radio
void setup() {
 Serial.begin(9600);
 if (!Node.begin()) {Serial.println("No LoRa module!");SHIELD=false;return;}
 Node.printConfig();        //Print (su Serial) della configurazione radio (per informazione)
 Node.printAddresses();     //Print (su Serial) della configurazione della rete (per informazione)
}
//Loop: prova a ricevere per 1000 millisecondi
// e di seguito verifica se si è rilevato un evento locale da comunicare
void loop() {
  if (!SHIELD) return;
```

```
  if (Node.newMessAvailable(1000)){execRequest();} //ricezione
  if (event()) {sendEvent();}                       //spedizione
}
// routine che gestisce la richiesta proveniente dalla rete
void execRequest()
{
  char* text=Node.getMessage();
  int sender=Node.getSender();
// qui va inserito il codice che si occupa della richiesta:
}
// routine che verifica gli eventi locali
bool event()
{
// qui va inserito il codice che controlla gli eventi locali
// se l'evento dura troppo poco (<1000 millisecondi)
// conviene gestirlo con un interrupt che alza un “flag”
// (senza interferenze perché la libreria non usa interrupt)
}
// routine che gestisce gli eventi locali
void sendEvent()
{
// qui va inserito il codice che decide il destinatario
// ed  elabora l'evento per la spedizione
  Node.writeMessage(to_id,mess,300);
}
```

Modifiche al “setting” predefinito

I nodi di rete non possono avere id=0 perché lo 0 come indirizzo è stato previsto per un eventuale messaggio “broadcast” (a tutti).

Se 15 nodi di rete sono troppo pochi per la vostra applicazione potete portarli a:

- 31 utilizzando la funzione: setMaxDevices(5) . Ma allora l'Id di rete può avere un valore tra 1 e 2047
- 63 utilizzando la funzione: setMaxDevices(6) . Ma allora l'Id di rete può avere un valore tra 1 e 1023
- 127 utilizzando la funzione: setMaxDevices(7) . Ma allora l'Id di rete può avere un valore tra 1 e 511
- 255 utilizzando la funzione: setMaxDevices(8) . Ma allora l'Id di rete può avere un valore tra 1 e 255

Per valori maggiori si veda la pagina help relativa alla classe LORA. Si tenga però presente che con numeri elevati di nodi le trasmissioni potrebbero sovrapporsi ed inoltre il modulo radio è half-duplex. Il problema non si pone se si adotta un funzionamento di tipo “server polling”, dove un nodo si preoccupa di interrogare a turno gli altri nodi, ovvero una struttura di rete a stella.

L'Id della rete può essere cambiato con la funzione setNetId(nuovo_id), mentre la chiave crittografica può essere cambiata con la funzione setKey(numero_intero).

Lo “spread factor“, e la larghezza di banda utilizzata, determinano la sensibilità della ricezione e la velocità di trasmissione, ma in modo inverso. Ovvero, un alto spread (codice 12) accoppiato ad una bassa banda (codice 0) determinano la massima sensibilità (fino a -150dBm), ma anche la minima velocità (11 pbs), mentre un basso spread

(codice 6) accoppiato ad una larga banda (codice 9) determinano un'alta velocità (23438 bps), ma una sensibilità di soli -113dBm. I valori preimpostati (SPR=10 e BW=8) consentono una sensibilità di -130dBm e una velocità di 1221 bps. Nel caso si volesse aumentare la sensibilità, e quindi la portata, suggeriamo di diminuire la banda con la funzione setBandWidth(5), e aumentare lo spread con la funzione setSpreadingFactor(11). In questo modo la sensibilità si porta a -140dBm anche se la velocità si riduce a un decimo (112 bps).

Consumo

Per aumentare la portata si può anche agire sulla potenza di emissione, tenendo presente che con questa aumenta anche l'assorbimento di corrente. Per esempio con 10mW (codice 2), che è il valore predefinito, la trasmissione si porta via circa 65mA solo per il modulo radio (circa 80/90 mA per la scheda con Arduino). Modificando la potenza con la funzione setPower(1), si porta la potenza a 5mW (7dBm) e il consumo si riduce a circa 50mA per il modulo radio (circa 70/80 per la scheda con Arduino). Questo consumo dura il tempo di trasmissione che ovviamente è legato alla lunghezza del messaggio ed alla configurazione di trasmissione vista precedentemente.

Per applicazioni particolari si può ridurre ulteriormente la potenza, e quindi il consumo, utilizzando una funzione della classe SX1278 che ha il formato: SX.setLowPower(byte dBm); dove “dBm” è il valore in dBm e può andare da 2 a 6. Con un valore di potenza pari a 2dBm (1.5mW) l'assorbimento si riduce a circa 35mA per il modulo radio. La potenza può essere aumentata fino a 100mW (codice 5), ma bisogna tener presente le disposizioni a riguardo. Il modulo radio in condizioni di ricezione consuma circa una decina di mA; mentre in “standby”, ovvero quando non riceve né trasmette consuma circa 2mA. Ma volendo il modulo radio si può porre in modalità “sleep” con la funzione LR.setSleepState(true). In questo stato il consumo del modulo radio è pressoché nullo (pochi uA).

Mode Radio Module		Entire System (with Arduino)	Just radio module
Sleep		25mA / 12mA (V>3.7V / battery)	<< 1mA
Standby		27mA / 14mA	1.7 mA
Receive		37mA / 25mA	11 mA
Transmit 2dBm (min)	1.5mW	63 / 53 mA	38 mA
3dBm	2 mW	65 / 56 mA	40 mA
4dBm	2.5mW	68 / 59 mA	43 mA
5dBm	3.1mW	70 / 62 mA	45 mA
7dBm	5mW	75 / 66 mA	50 mA
10dBm	10mW	85 / 80 mA	60 mA
13dBm	20mW	103 / 99 mA	78 mA
17dBm	50mW	133 / 126 mA	108 mA
20dBm	100mW	148 / 136 mA	123 mA

Consumo

Il consumo in trasmissione dura solo per il tempo in cui viene inviato il pacchetto. Questa durata dipende dai parametri di trasmissione: spreading factor (SF) e larghezza di banda (BW), che però sono legati in maniera inversa alla sensibilità. In figura 2 sono mostrati i tempi misurati per alcuni valori di sensibilità e con diverse lunghezze del messaggio. Come si vede c'è uno zoccolo fisso, a prescindere dalla lunghezza del messaggio, dovuto al sovraccarico del protocollo LoRa e di quello della libreria.

SF	BW	dBm	bps	fly time (milliseconds)			Bps	1/Bps (mS/B)
				1 to 10 char	50 char	100 char		
7	8	-121	6836	45	121	200	650	1.5
8	7	-127	1953	141	401	664	200	5
10	8	-130	1221	248	641	1032	127	8
9	5	-135	367	740	2116	3488	38	26
11	5	-140	112	2564	6883	10809	12	82
12	4	-144	46	6840	16261	26731	5	200
12	1	-149	15	20514	48781	80200	1.5	650

Tempi di trasmissione in funzione della sensibilità

Utilizzo dell'EEPROM

Oltre all'indirizzo del nodo stesso, anche tutta la configurazione della rete (Id rete,numero massimo di nodi e la chiave di crittografia) possono essere salvati su EEPROM, occupando i primi 7 byte con la funzione saveNetConfig() a cui fa da inverso la funzione loadNetConfig(). Infine anche le caratteristiche fondamentali della trasmissione radio, ovvero i parametri che potrebbero essere adattati a diverse esigenze, possono essere salvati in EEPROM con la funzione saveRadioConfig() a cui corrisponde la funzione inversa loadRadioConfig(). I pasametri sono: frequenza, spreading factor, larghezza di banda e potenza. Queste grandezze occupano ulteriori 7 byte.

Per questo motivo si consiglia di utilizzare per le proprie applicazioni le posizioni a partire dal byte 20 (per lasciare l'espansione ad eventuali altre versioni). La funzione resetEEPROM() cancella la memoria fino alla posizione 14 (porta a 255 il contenuto dei byte). Quindi cancella anche l'Id del nodo salvato.