Capire e usare LoRa®

5

di PIER CALDERAN

Impariamo a utillizzare l'interfaccia di Node-RED per inviare i dati dal web al nostro dispositivo finale, sfruttando The Things Network e un gateway multicanale per implementare il downlink su LoRaWAN.

Nella scorsa puntata abbiamo spiegato come ricevere i dati provenienti dai dispositivi finali LoRaWAN e in che modo visualizzarli con Node-RED.
In queste pagine ci cimenteremo con l'esatto contrario, cioè mandare i controlli da Node-RED al dispositivo finale, attraverso The Things Network. Nell'esempio di oggi spiegheremo come creare dei pulsanti per accendere e spegnere un LED collegato al dispositivo finale.

MODULO NODE-RED UI LED

Oltre alla dashboard che abbiamo già installato nella scorsa puntata per aggiungere i misuratori di temperatura, umidità e luminosità all'interfaccia di Node-RED, abbiamo pensato che fosse utile installare un modulo che simuli le funzioni di un LED. Fra i tanti presenti in rete, abbiamo scelto il modulo **Node-RED UI LED**, il cui sito di riferimento è il seguente: *https://flows.nodered.org/node/node-red-contrib-ui-led*.
Qui potete trovare le istruzioni per usarlo così com'è ma anche i comandi per personalizzare il colore del LED. Per installare il modulo, procedete come al solito con il comando *npm* dal terminale:

```
npm install node-red-contrib-ui-led
```

Per impostazione predefinita il LED ha tre stati, evidenziati da tre colori di base: uno rosso, uno verde e uno grigio, come illustrato in **Fig. 1**. Per approfondire l'uso di questo modulo si consiglia di caricare nel flow gli esempi preimpostati usando la voce *Import* dal menu di Node-RED.

ARCHITETTURA DI RETE LORAWAN

Prima di affrontare l'argomento di oggi, ci sembra doveroso spiegare la logica che sta alla base dell'architettura di rete LoRaWAN. Come già detto fin dalla prima puntata, il protocollo LoRaWAN non è stato introdotto per controllare "velocemente" i dispositivi finali.

Fig. 1 - L'interfaccia grafica del modulo Node-RED UI LED.

È un sistema efficace, ma abbastanza lento. In altre parole, se pensate di usare un gateway come un radiocomando per l'apricancello, il protocollo LoRaWAN non ve lo permette. Abbiamo spiegato in questo corso come usare la tecnologia LoRa per la ricetrasmissione punto-punto a lunga distanza, ma senza utilizzare gateway e il protocollo LoRaWAN.
Deve essere chiara una cosa: il protocollo LoRaWAN è stato pensato per l'utilizzo dei canali LoRa in modo tale da evitare collisioni, attacchi o connessioni fallite. Quindi, se volete sfruttare il cloud per avere il controllo totale dei vostri dispositivi finali, continuate a leggere. I sistemi di backend di The Things Network sono responsabili dell'instradamento dei dati tra i dispositivi e le applicazioni. Se per esempio guardiamo una normale rete IoT, questa richiede un gateway come ponte tra specifici protocolli radio (per esempio Wi-Fi) e Internet. Nei casi in cui i dispositivi stessi supportino lo stack IP, questi gateway devono solo inoltrare i pacchetti a Internet. Invece, i protocolli non IP, come LoRaWAN, richiedono una forma di instradamento ed elaborazione prima che i messaggi possano essere consegnati a un'applicazione. La rete TTN è posizionata tra i gateway e le applicazioni e si occupa di queste fasi di instradamento ed elaborazione (vedere la **Fig. 2**).
Il compito di The Things Network è quello di eseguire tutte queste funzioni di routing in modo decentralizzato e distribuito. L'obiettivo è quello di essere molto flessibile in termini di opzioni di distribuzione pubblica tramite le applicazioni create nel backend. Ovviamente è possibile anche distribuire reti private, eseguendo tutti questi componenti in un ambiente privato. In questo modo, tutti i dati rimarranno all'interno dell'ambiente privato, ma sarà comunque sempre possibile utilizzare l'account ospitato da TTN per l'autenticazione e l'autorizzazione.

FUNZIONALITÀ DI BASE DI TTN

The Things Network fornisce un server di rete LoRaWAN ovvero un protocollo cosiddetto *network-intensive*. "Intensivo"

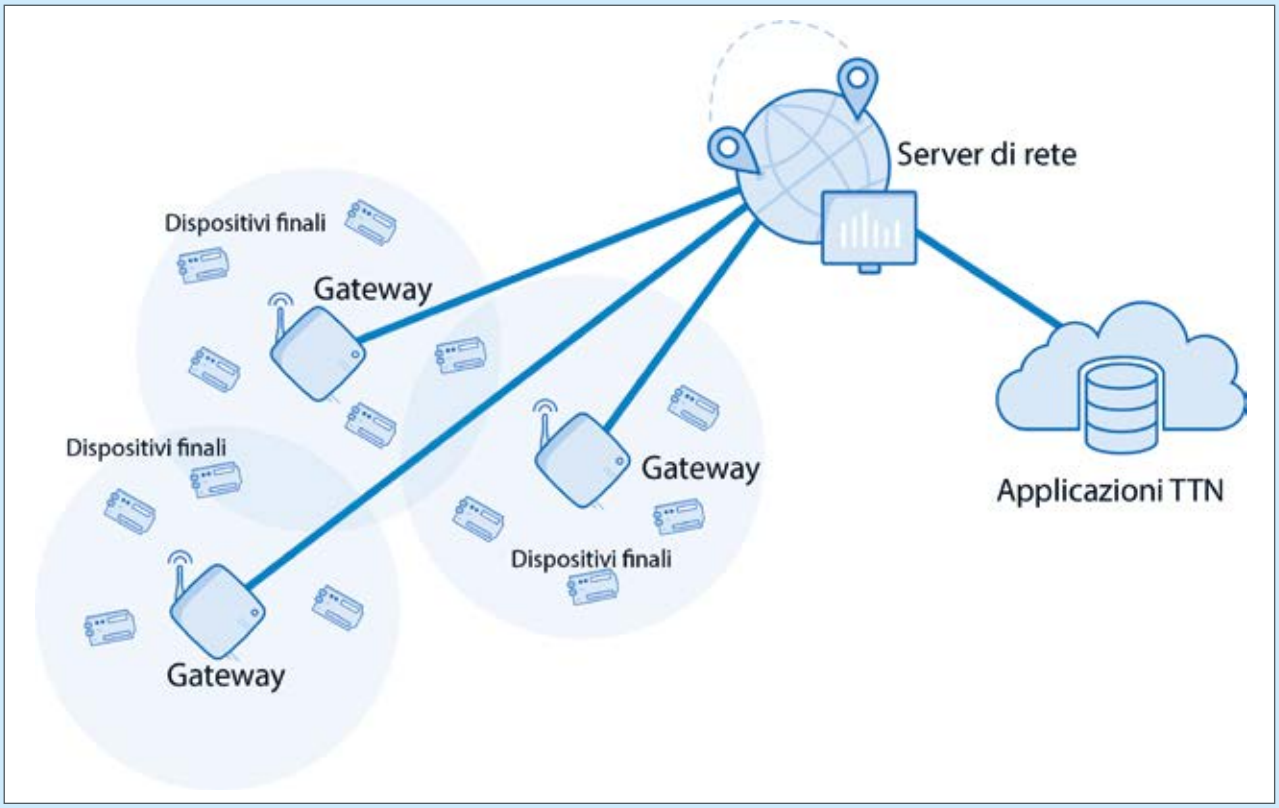

Fig. 2 - Architettura di rete LoRaWAN.

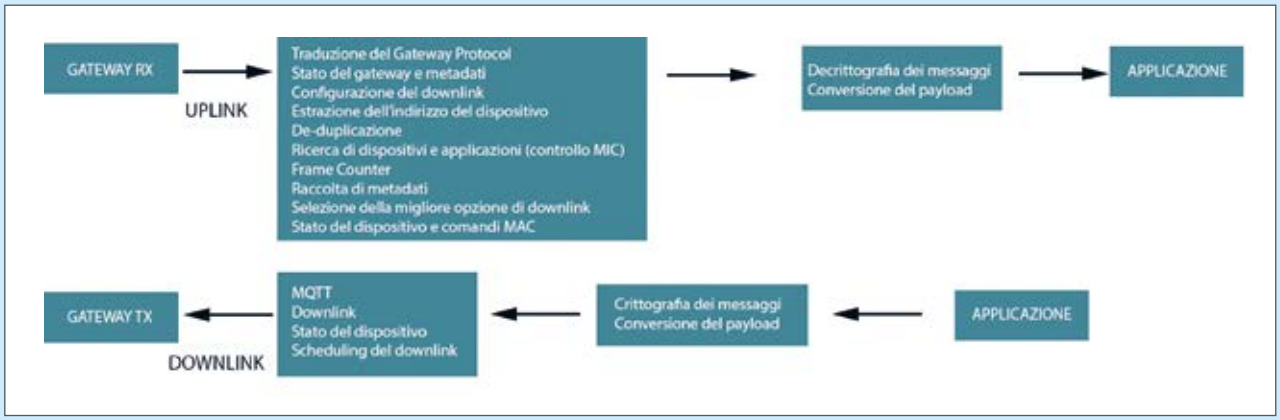

Fig. 3 Flusso di elaborazione dei messaggi di uplink e downlink.

nel senso che a causa dell'approccio dei milioni di dispositivi connessi, i sistemi di backend sono responsabili della maggior parte della logica LoRaWAN. Quindi per funzionare su un'infrastruttura distribuita come The Things Network, è stato necessario seguire alcune funzioni.
Nel backend di The Things Network si possono distinguere una serie di diverse funzioni principali. Vediamo quali.

- **Scheduling** (pianificazione): le funzioni di *scheduling* servono all'utilizzo dei gateway. Lo *scheduling* è necessario perché un gateway possa eseguire solo una trasmissione o ricezione alla volta. Le informazioni sull'utilizzo vengono usate per distribuire uniformemente il carico su diversi gateway e per essere conformi ai cicli di lavoro (*duty cycle*) europei.
- **Stato dei dispositivi finali**: è necessario gestire lo stato dei dispositivi nella rete, poiché l'indirizzo del dispositivo non è univoco. Il server deve tenere traccia di quali indirizzi vengono utilizzati da quali dispositivi per mappare un messaggio al dispositivo corretto e all'applicazione. Altre cose di cui la rete deve tenere traccia sono le chiavi di sicurezza e i contatori di frame.
- **Funzionalità delle applicazioni**: per esempio, i broker MQTT e i gestori devono sapere a quale server deve essere inoltrato il traffico per un'applicazione specifica.
  I gestori devono sapere come interpretare i dati binari e creare collegamenti a protocolli di livello superiore, come AMQP e MQTT.
- **Funzionalità della distribuzione**: la funzionalità di rilevamento dei servizi aiuta i componenti a determinare dove deve essere instradato il traffico. Attualmente, questo è implementato come un server di rilevamento centralizzato, dando a The Things Network il controllo su quali componenti sono autorizzati ad annunciare servizi specifici.

Per rendere tutto questo lavoro possibile, si è reso necessario separare i compiti. L'idea di base è che il router sia responsabile di tutte le funzionalità relative al gateway. Un broker gestisce un intervallo di indirizzi dei dispositivi ed è responsabile di trovare il gestore giusto a cui inoltrare ogni messaggio. Il server di rete è responsabile del mantenimento dello stato di tutti i singoli dispositivi.
Il gestore TTN è responsabile della crittografia, decrittografia e della conversione dei messaggi e dell'inoltro dei messaggi alle applicazioni.

ELABORAZIONE DEI MESSAGGI DI UPLINK E DOWNLINK

Sulla base di questa separazione dei compiti, il backend di The Things Network deve eseguire una serie di attività durante l'elaborazione dei messaggi di *uplink* e *downlink*.
Una panoramica di questo flusso è illustrata nella **Fig. 3** e spiegata qui di seguito:

- traduzione del Gateway Protocol;
- stato del gateway e metadati;
- configurazione del downlink;
- estrazione dell'indirizzo del dispositivo;
- deduplicazione;
- ricerca di dispositivi e applicazioni (controllo MIC);
- Frame Counter;
- raccolta di metadati;
- selezione della migliore opzione di downlink;
- stato del dispositivo e comandi MAC;
- decrittografia dei messaggi;
- conversione del payload;
- MQTT;
- downlink.
- Stato del dispositivo
- Scheduling del downlink

TRADUZIONE DEL GATEWAY PROTOCOL

Quando un gateway riceve un messaggio che è stato trasmesso su LoRa, viene incapsulato e inoltrato a TTN.
La maggior parte dei protocolli gateway (*Gateway Protocol*) ha la stessa struttura ovvero quando vengono ricevuti uno o più messaggi. Il loro payload binario viene inoltrato al backend, insieme ai metadati.

Periodicamente il gateway invia anche alcune informazioni di stato *alive* (vivo) sul gateway stesso: per esempio, coordinate GPS, potenza del segnale RSSI, rapporto segnale/rumore SNR, il numero di pacchetti ricevuti/trasmessi e altre metriche.

STATO DEL GATEWAY E METADATI

Le coordinate GPS di un gateway possono essere particolarmente rilevanti per l'applicazione, pertanto il backend memorizza l'ultimo messaggio di stato inviato dal gateway e inietta le informazioni GPS nei metadati di ogni messaggio di *uplink*.

CONFIGURAZIONE DEL DOWNLINK

In LoRaWAN la risposta di *downlink* a un messaggio di *uplink* dipende fortemente dalla regione geografica del gateway e sono descritte nella specifica "LoRaWAN Regional Parameters". Poiché il router è responsabile di tutti i dettagli relativi al gateway e specifici della regione, il router deve determinare come inviare una risposta in *downlink* a un dispositivo.
Dopo ogni messaggio di *uplink*, ci sono due finestre di ricezione, una esattamente 1 secondo dopo l'*uplink*, l'altra dopo 2 secondi. Pertanto, per ogni gateway che ha ricevuto il messaggio di *uplink*, un router crea due configurazioni di *downlink*.
Per selezionare l'opzione migliore in un secondo momento, il router deve inoltre calcolare un tempo per ciascuna opzione. Questo tempo è influenzato da una serie di fattori.
Al momento consideriamo il tempo di trasmissione, la potenza del segnale, l'utilizzo del gateway e le trasmissioni già programmate (*scheduled*), in quanto un gateway non può fare due trasmissioni contemporaneamente. La pianificazione (*scheduling*) di un messaggio di *downlink* su un gateway che aveva una migliore potenza del segnale rende anche più probabile che un nodo riceva correttamente il *downlink*.
La combinazione del tempo di trasmissione di un messaggio e l'utilizzo di un gateway viene utilizzata per ottimizzare la rete nel suo complesso. Poiché ogni trasmissione blocca per qualche tempo i ricevitori di un gateway, è meglio inviare messaggi in un tempo più breve. Pertanto, i messaggi di *downlink* a una velocità dati maggiore sono preferiti rispetto ai messaggi con una velocità dati inferiore.

ESTRAZIONE DELL'INDIRIZZO DEL DISPOSITIVO

Il primo passaggio nell'instradamento di un pacchetto è basato sull'indirizzo del dispositivo. Come abbiamo avuto modo di vedere, questo *DevAddr* è un indirizzo a 32 bit impostato dall'utente (modalità ABP) o dal gestore (modalità OTAA). TTN ha scelto di distribuire il traffico ai broker in base al prefisso dell'indirizzo del dispositivo.

DEDUPLICAZIONE

LoRaWAN è un protocollo che rende probabile che il messaggio venga ricevuto da più di un gateway. Ciò significa che il backend deve eseguire una sorta di deduplicazione per recapitare un messaggio solo una volta all'applicazione.
Ciò non significa che i duplicati non siano importanti. Anche i metadati di questi messaggi potrebbero essere preziosi. Per esempio, quando si combinano le posizioni dei gateway che hanno ricevuto il messaggio con l'ora di ricezione e la potenza del segnale, potrebbe essere possibile determinare la posizione del dispositivo che ha inviato il messaggio.
L'attuale implementazione del backend deduplica localmente i messaggi di *uplink*. Il payload del messaggio sarà lo stesso per tutti i duplicati e la possibilità che durante il periodo di deduplicazione (solitamente un paio di secondi) arrivi un messaggio diverso è estremamente bassa.
Per raccogliere i metadati aggiunti al messaggio da ciascun gateway, il broker deve memorizzare nel buffer i duplicati per un certo periodo di tempo, abbastanza lungo da raccogliere il maggior numero possibile di duplicati, ma abbastanza breve da dare all'applicazione il tempo sufficiente per rispondere a un messaggio nella finestra di ricezione che verrà aperta 1 secondo dopo la trasmissione.
Le nostre misurazioni nell'attuale distribuzione di The Things Network hanno dimostrato che il ritardo medio tra il primo e l'ultimo duplicato è di poco inferiore a 100 ms, con il ritardo massimo di circa 300 ms.

RICERCA DI DISPOSITIVI E APPLICAZIONI

Poiché gli indirizzi dei dispositivi non sono univoci, è necessario determinare il dispositivo esatto che ha inviato il messaggio e l'applicazione a cui appartiene. Per fare ciò, il backend deve eseguire una serie di controlli MIC (*Message Integrity Code*), uno per ogni dispositivo che utilizza lo stesso indirizzo del dispositivo. Il broker richiede pertanto un elenco di dispositivi con l'indirizzo del dispositivo specificato dal server di rete e controlla se il MIC può essere convalidato utilizzando la chiave di sessione di rete. Se non viene trovata alcuna corrispondenza, il messaggio viene ignorato.

FRAME COUNTER

Il contatore di frame nei messaggi LoRaWAN è una misura di sicurezza utilizzata per rilevare gli attacchi di replay.
Dopo aver convalidato il MIC, il broker controlla se il contatore di frame è valido. Poiché i contatori di frame possono solo aumentare, un messaggio con un contatore di frame inferiore all'ultimo contatore di frame noto dovrebbe essere eliminato. Inoltre, il broker deve verificare che lo spazio tra l'ultimo contatore di frame noto e il contatore nel messaggio non sia troppo grande.

RACCOLTA DI METADATI

Quando tutti i controlli hanno avuto esito positivo, il broker può continuare a elaborare il messaggio. Innanzitutto, unisce i duplicati ricevuti da tutti i diversi router e gateway.
In questo passaggio è importante distinguere tra metadati uguali per ogni gateway che ha ricevuto il messaggio e metadati specifici per ogni ricezione. Per esempio, la frequenza, la modulazione e la velocità dei dati saranno le stesse per tutti

i gateway, quindi deve essere inoltrato solo una volta. D'altra parte, la potenza del segnale, l'ora di ricezione e le coordinate GPS di ciascun gateway dovrebbero essere tutte incluse quando si inoltra il messaggio.
In questa fase vengono raccolte anche le diverse configurazioni di *downlink* per selezionare l'opzione migliore nella fase successiva.

SELEZIONE DELLA MIGLIORE OPZIONE DI DOWNLINK

Il broker deve selezionare l'opzione migliore per una risposta in *downlink* a un messaggio. Poiché il broker non dispone di alcuna informazione sul gateway che ha ricevuto un messaggio, è molto difficile farlo. Pertanto il router ha già calcolato un tempo per ciascuna configurazione di *downlink*.
Se questo calcolo del tempo viene eseguito in modo standard, il broker deve solo ordinare tutte le possibili opzioni di *downlink* e utilizzare l'opzione migliore.

STATO DEL DISPOSITIVO E COMANDI MAC

Prima di inoltrare il messaggio di *uplink* al gestore, viene prima inviato al server di rete in modo che lo stato del dispositivo possa essere aggiornato. Il server di rete aggiunge anche un template di *downlink* al messaggio. Questo template può essere utilizzato dal gestore per inviare un messaggio di *downlink* al dispositivo. Contiene tutti i valori necessari (come il contatore di frame, il tipo di messaggio e gli indicatori di opzione) in modo che il gestore debba solo aggiungere al messaggio il payload dell'applicazione. Inoltre, ciò offre al server di rete la possibilità di aggiungere comandi MAC al messaggio. Per esempio, in base al numero di gateway che hanno ricevuto un messaggio e alla loro potenza del segnale, il server di rete può aggiungere comandi MAC che istruiscono il dispositivo a trasmettere a una velocità dati maggiore.

DECRITTOGRAFIA DEI MESSAGGI

Poiché i messaggi sono crittografati end-to-end, il backend è anche responsabile della decrittografia dei messaggi.
Dopo aver decrittografato il payload del messaggio, il gestore può passare il messaggio all'applicazione. Tuttavia, per molte applicazioni, sono necessarie alcune semplici operazioni di decodifica e conversione del payload.

CONVERSIONE DEL PAYLOAD

Dopo la decrittografia, il gestore è in grado di decodificare e convertire il payload in un formato facilmente accessibile dall'applicazione. L'implementazione include quindi le cosiddette funzioni di formato del payload.
Queste funzioni sono semplici funzioni JavaScript che possono essere utilizzate per decodificare, convertire e convalidare i dati (vedere la scorsa puntata). Il decodificatore (*decoder*) viene utilizzato per decodificare il payload binario in un formato più appropriato.
Per esempio, la seguente funzione decodifica un valore di temperatura inviato come due byte a un oggetto JSON:

```
function (bytes) {
    var data = (bytes[0] << 8) | bytes[1];
    return { temperature: data / 100.0 };
}
```

Il convertitore (*converter*) facoltativo può convertire i valori nell'oggetto JSON decodificato. Per esempio, potrebbe trattarsi di una conversione da una temperatura in Celsius a una temperatura in Fahrenheit.
Il validatore (*validator*) facoltativo può essere utilizzato per verificare la validità dei dati ed eliminare i valori anomali.

MQTT

L'implementazione del gestore predefinito pubblica su un broker MQTT semplicemente una rappresentazione JSON dei messaggi di *uplink* a un topic di questo tipo:

```
<app_id>/devices/<dev_id>/uplink
```

Ciò consente alle applicazioni di iscriversi semplicemente allo stesso topic MQTT ed elaborare i dati in qualsiasi modo.

DOWNLINK

Dopo aver pubblicato il messaggio di *uplink* su MQTT, il gestore determinerà se è necessario rispondere al dispositivo con un messaggio di *downlink*. Esistono tre situazioni in cui è necessario inviare un messaggio di *downlink*:

- il primo caso è quando l'applicazione ha un payload disponibile da inviare al dispositivo. In questo caso, il payload viene aggiunto al template di risposta generato dal server di rete;
- il secondo caso è quando il messaggio di *uplink* richiede una conferma. Indipendentemente dal fatto che un payload di *downlink* sia disponibile o meno, deve essere inviato un *acknowledgement* (riconoscimento);
- la terza situazione è quando il server di rete deve inviare comandi MAC al dispositivo.

Analogamente a quanto avviene con i messaggi di *uplink*, il gestore è responsabile della crittografia del payload del messaggio. Se non è disponibile alcun payload di *downlink*, il gestore può scegliere di attendere un breve periodo per consentire all'applicazione di preparare un messaggio di *downlink* in base al messaggio di *uplink* appena ricevuto. Trascorso tale termine, il gestore dovrà inviare nuovamente il messaggio di *downlink* al broker.

STATO DEL DISPOSITIVO

Dopo che il broker riceve un messaggio di *downlink* da un gestore, invia il messaggio che aggiornerà lo stato del dispositivo (in particolare, i contatori di frame) nel database e genererà il MIC del messaggio.

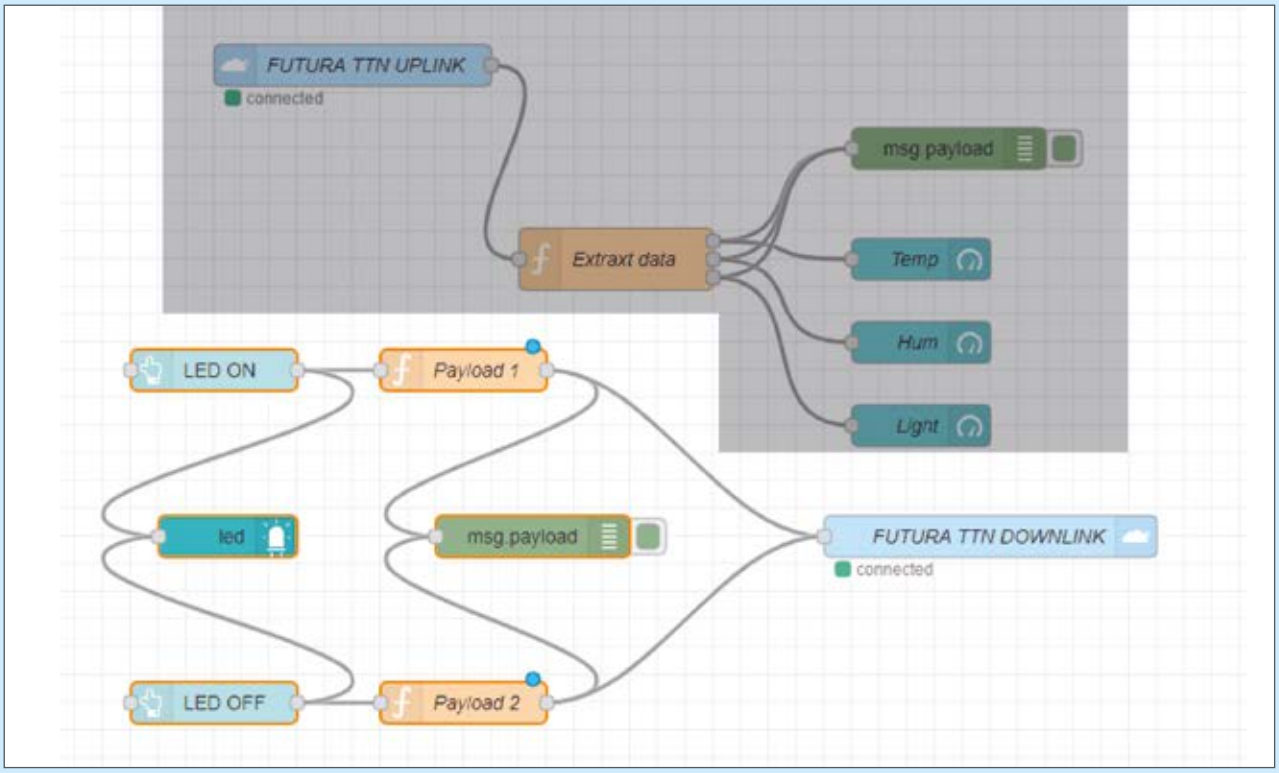

Fig. 4 - Il flusso utilizzato per leggere temperatura, luce e umidità, che qui riutilizziamo.

Successivamente, il broker inoltra il messaggio di *downlink* al router responsabile del gateway che deve trasmettere il messaggio di *downlink*.

SCHEDULING DEL DOWNLINK

Come accennato all'inizio di questo capitolo, il router è responsabile della gestione della pianificazione (*scheduling*) del gateway. Poiché la maggior parte dei gateway ha solo un buffer di un messaggio di *downlink*, il router deve memorizzare nel buffer i messaggi pianificati (*scheduled*) fino all'ultimo momento, quindi inviare ogni messaggio appena in tempo al gateway.

AGGIUNGERE DUE PULSANTI E UN LED

Dopo la spiegazione dell'architettura di rete LoRaWAN, passiamo alle cose più semplici e vediamo come aggiungere due pulsanti alla nostra interfaccia di Node-RED.
Questi pulsanti manderanno dei messaggi di *downlink* a TTN, che a sua volta li manderà al nostro gateway tramite l'applicazione *futura_devices*. L'applicazione convertirà questi messaggi per il *device_1*, al quale avremo collegato un LED.
Se guardate la **Fig. 4** potete notare nella parte grigia lo stesso flow della scorsa puntata, usato per l'*uplink* dei dati di temperatura, umidità e luminosità.
A questo flow abbiamo aggiunto i seguenti nodi:

- nodo downlink **FUTURA TTN DOWNLINK**
- nodo button **LED ON**
- nodo function **Payload 1**
- nodo button **LED OFF**
- nodo function **Payload 2**
- nodo **led**
- nodo **debug** (opzionale)

Una volta inseriti tutti i nodi e collegati fra loro, è necessario aprire il rispettivo editor e inserire tutti i dati di ciascun nodo nel modo seguente. Il risultato sarà simile alla **Fig.5**., nella quale è proposto il **Nodo downlink**, i cui parametri sono:
Name: FUTURA TTN DOWNLINK
App: futura_devices
Device ID: 2601164B
Port: 1
Schedule: replace

Abbiamo poi i **Nodi button (Fig. 6)** così impostati.
Group: DOWNLINK
Size: 4x2
Label: LED ON (LED OFF per il secondo nodo button)
Port: 1
Payload: true (false per il secondo nodo button)
Topic: led

Invece il **Nodo led (Fig. 7)** ha queste impostazioni.
Group: DOWNLINK
Size: auto

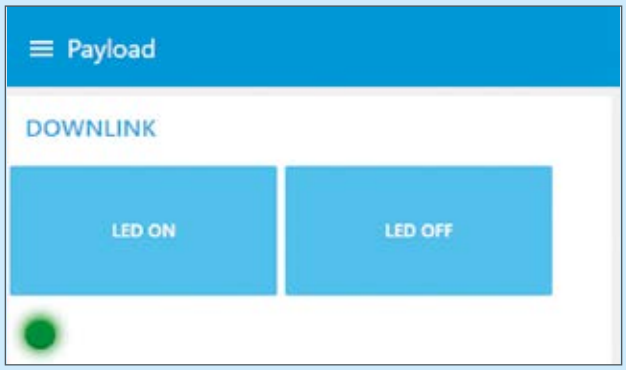

Fig. 5 - Nodo downlink.

Msg.payload: false (color red)
Msg.payload: true (color green)

Andiamo quindi ai **Nodi function (Fig. 8)**.
Name: Payload 1 (Payload 2 per il secondo nodo function)
Function:

```
return {
  dev_id: "device_1",
  port: 1,
  payload:"31"
}
```

Function (per il secondo nodo function):

```
return {
  dev_id: "device_1",
  port: 1,
  payload:"32"
}
```

È importante notare che impostando il payload dei due nodi button rispettivamente a *true* e *false* si può controllare lo

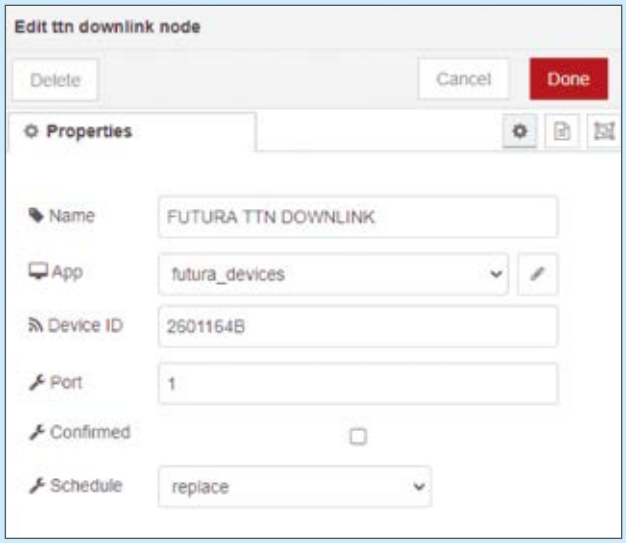

Fig. 6 - Editor del nodo downlink

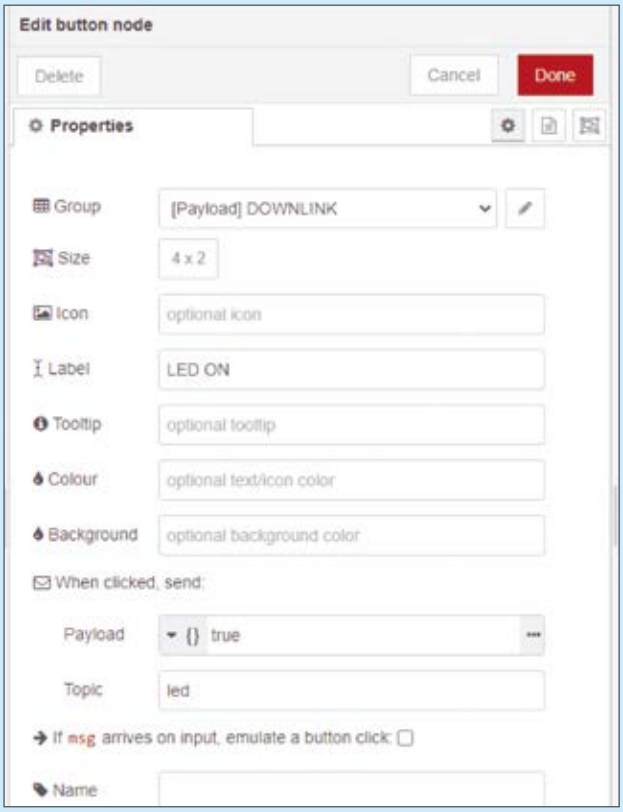

Fig. 7 - Editor del nodo button.

stato del nodo *led*: questo diventa verde se il *msg.payload* del primo button è *true* e diventa rosso se il *msg.payload* del secondo button è *false*. Ovviamente questo procedimento serve solo di esempio e non verifica lo stato effettivo del LED fisico. Per fare questo bisognerebbe creare una serie di operazioni

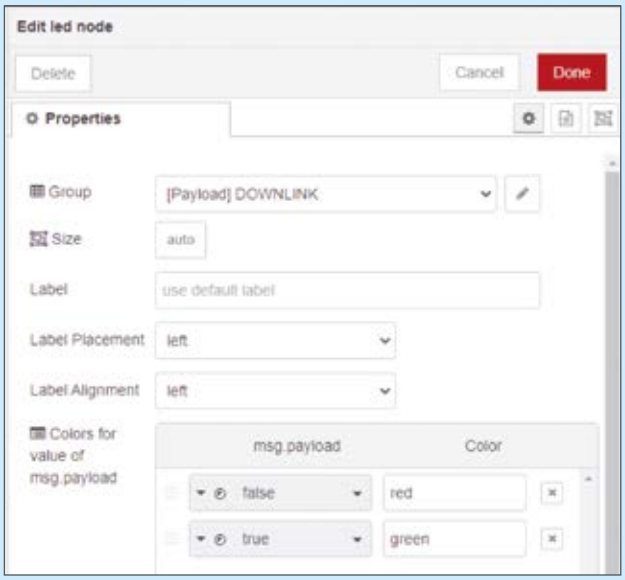

Fig. 8 - Editor del nodo led.

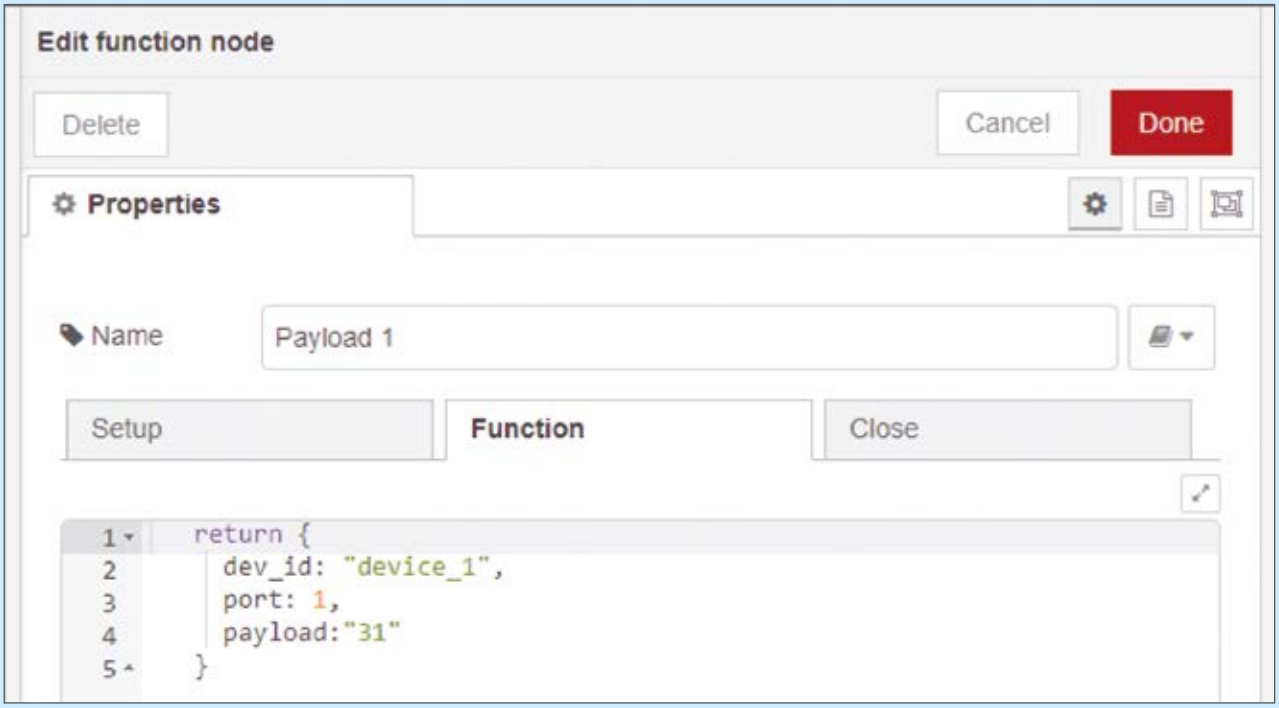

Fig. 9 - Editor del nodo function.

di *uplink* e *downlink* molto complesse che non trattiamo qui. Nel primo nodo function abbiamo inserito nel messaggio JSON i seguenti parametri che TTN leggerà e "schedulerà" sotto forma di messaggi di *downlink* al *device_ 1*:

- **dev_id**: "device_1", ovvero il nome del nostro dispositivo finale;
- **port**: 1, la porta disponibile al momento;
- **payload**: "32", ovvero il valore in esadecimale equivalente 49 decimale, che in ASCII corrisponde al numero 1.

Nella seconda funzione abbiamo inserito nel messaggio JSON gli stessi parametri eccetto quello per il payload: **payload**: "32", ovvero il valore in esadecimale corrispondente al decimale 50 che nella tabella ASCII corrisponde al numero 2. Riportiamo le due funzioni complete nel **Listato 1**.
In questo modo, cliccando su LED ON il nodo *led* diventerà verde e verrà spedito un messaggio di *downlink* con il payload 0x31 (decimale 49, simbolo ASCII 1). Cliccando su LED OFF, il nodo *led* diventerà rosso e verrà spedito un messaggio di *downlink* con il payload 0x32 (decimale 50, simbolo ASCII 2). Quello che si vedrà nella sezione **Application Data** del *device_ 1* sarà simile alla **Fig. 10**.
Come si può vedere, insieme ai messaggi di *uplink* di temperatura, umidità e luminosità, ci sono anche i messaggi di *downlink* "31" e "32" marcati come *scheduled*. Questi messaggi verranno eseguiti dopo il successivo messaggio di *uplink*. La scelta di questi valori per il payload è arbitraria e ovviamente si possono mandare messaggi di qualsiasi tipo, sempre in esadecimale.
Per esempio, se volete provare a mandare un payload con il testo "Hello world", dovrete scrivere il payload in questo modo:

"48656C6C6F20776F726C64"

Che corrisponde ai caratteri ASCII e relative lettere:
72 (H)
101 (e)
108 (l)
108 (l)
111 (o)
32 (spazio)
119 (w)
111 (o)
114 (r)
108 (l)
100 (d)

Listato 1

```
Function payload 1
  return {
     dev_id: "device_1",
     port: 1,
     payload:"31"
   }

Function payload 2
  return {
     dev_id: "device_1",
     port: 1,
     payload:"32"
   }
```

LO SKETCH PER IL DOWNLINK

A questo punto, tutto è pronto per modificare lo sketch che avete usato nella scorsa puntata per l'*uplink*. In pratica, basta aggiungere all'evento **EV_TXCOMPLETE** il codice per l'analisi della finestra di ricezione **RX**: se questa finestra conterrà il payload 0x31 o 0x32 di *downlink* da TTN, che sono gli stessi valori 0x31 o 0x32 mandati da Node-RED, potremo leggerli e abbinare a questi il codice per l'accensione e lo spegnimento del LED. Nel **Listato 2** potete vedere le modifiche apportate che riguardano sostanzialmente l'accensione e lo spegnimento di un LED collegato al pin digitale 5 del nostro dispositivo finale. Il circuito elettronico andrà modificato di conseguenza, come illustrato in **Fig. 11**. Il codice aggiunto si riferisce alla definizione della variabile **PIN_LED** come pin 5 e la sua modalità come **OUTPUT**. La parte di codice per il *downlink* è stato inserita nella funzione **onEvent**. All'interno dello switch della funzione, in caso l'evento sia **EV_TXCOMPLETE** verrà analizzata la finestra di **RX** che viene ricevuta dopo l'*uplink* dei dati avvenuto con successo. All'interno della finestra di ricezione del *downlink* si potranno analizzare i dati con le seguenti funzioni della libreria LMIC:

- **LMIC.dataLen**: può essere 0 (nessun dato) oppure maggiore di 0 (byte di dati);
- **LMIC.dataBeg**: può essere 0 (nessun dato) o inizio dei dati

Se **LMIC.dataLen** è maggiore di zero allora si entra nel ciclo *for* per memorizzare i dati nella variabile *mydata*. Sappiamo che i dati ricevuti possono essere 0x31 o 0x32, per cui li memorizzeremo come *char* nel primo elemento dell'array **mydata[0]**. A questo punto il gioco diventa facile:

```
if (mydata[0] == '1') digitalWrite(PIN_LED,1);
if (mydata[0] == '2') digitalWrite(PIN_LED,0);
```

Significa che se il primo elemento dell'array *mydata* ha il valore ASCII 1 allora il LED si accende. Se il valore ASCII è 2, lo spegnerà. Si fa notare che nello sketch abbiamo impostato l'intervallo di **TX_INTERVAL** a 10 secondi per ottenere un *downlink* in questo lasso di tempo. Questo significa che il LED potrà venire spento o acceso entro l'intervallo di 10 secondi, per cui non aspettatevi la contemporaneità degli eventi.

Listato 2

```
#define PIN_LED 5

void onEvent (ev_t ev) {
    switch(ev) {
…
…
…
        case EV_TXCOMPLETE:
            Serial.println(F("EV_TXCOMPLETE (includes waiting for RX windows)"));

char mydata[2]={0};
        if (LMIC.dataLen)
        {
        Serial.println(F("Received "));
        Serial.print(LMIC.dataLen);
        Serial.println(F(" bytes of payload"));
        Serial.print("txCnt :");
        Serial.println(LMIC.txCnt);
        Serial.print("txrxFlags :");
        Serial.println(LMIC.txrxFlags);
        Serial.print("dataBeg :");
        Serial.println(LMIC.dataBeg);
        for (int i = 0; i < LMIC.dataLen; i++)
        {
          if (LMIC.frame[LMIC.dataBeg + i] < 0x10) {
            Serial.print(F("0"));
          }
          Serial.print(LMIC.frame[LMIC.dataBeg + i], HEX);
          mydata[i]=char(LMIC.frame[LMIC.dataBeg + i]);
          }
          }
            Serial.println();
            Serial.println(mydata);

            if (mydata[0]=='1') digitalWrite(PIN_LED,1);
            if (mydata[0]=='2') digitalWrite(PIN_LED,0);

            // Schedule next transmission
            os_setTimedCallback(&sendjob, os_getTime()+sec2osticks(TX_INTERVAL), do_send);
            break;
}

void setup() {
    pinMode(PIN_LED,OUTPUT);
…
…
…
}
```

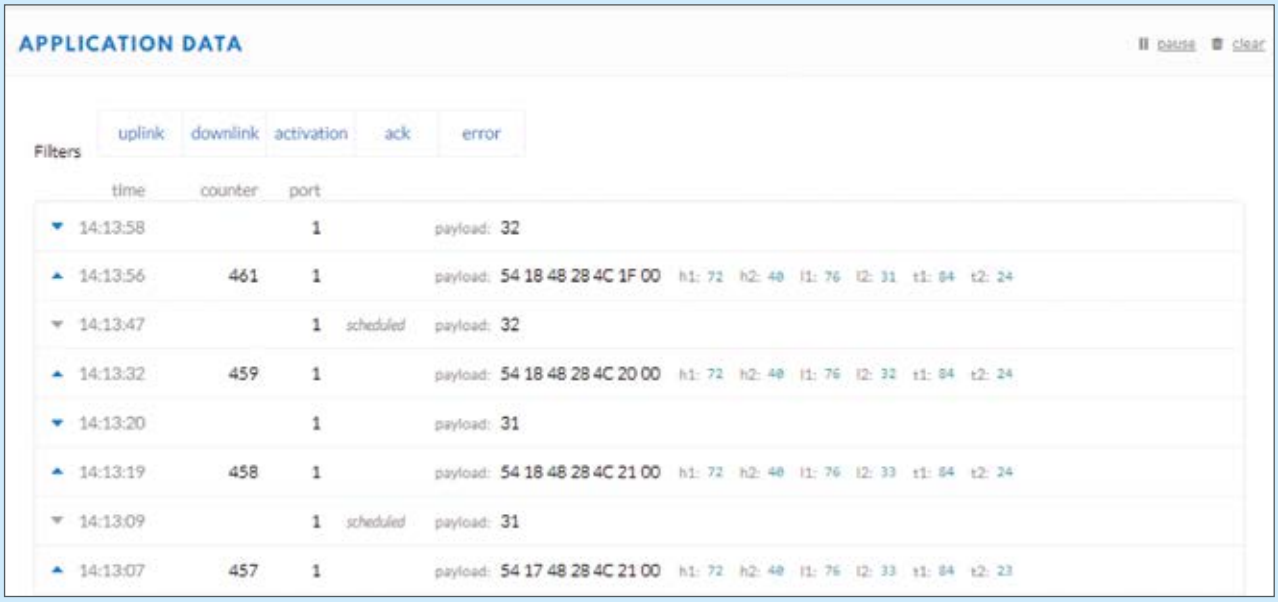

Fig. 10 - I dati di uplink e downlink dell'applicazione.

In altre parole, quando premete LED ON o LED OFF possono passare da 2 a 10 secondi prima di vedere accendersi o spegnersi il LED. Questo dipende dal tempo di *scheduling* dalla ricezione del messaggio del successivo *uplink*.
Quello che abbiamo proposto qui è solo un semplice esempio per iniziare a sperimentare. Per chi vuole dilettarsi con le API della libreria LMIC, consigliamo di visitare la pagina web *https://lora-developers.semtech.com/resources/tools/basic-mac/programming-model-and-api/*.

UPLINK E DOWNLINK CON IC880A

Il gateway che abbiamo usato fino ad oggi per le operazioni di *uplink* è a canale singolo e per usarlo abbiamo fruito del *single channel packet forwarder* di Thomas Telkamp (https://github.com/tftelkamp/single_chan_pkt_fwd). Purtroppo, come dice lo stesso autore, con questo software non è possibile effettuare il *downlink* da TTN. Come abbiamo avuto modo di sottolineare, l'uso di un gateway a canale singolo può essere considerato come un modo veloce ed econo-

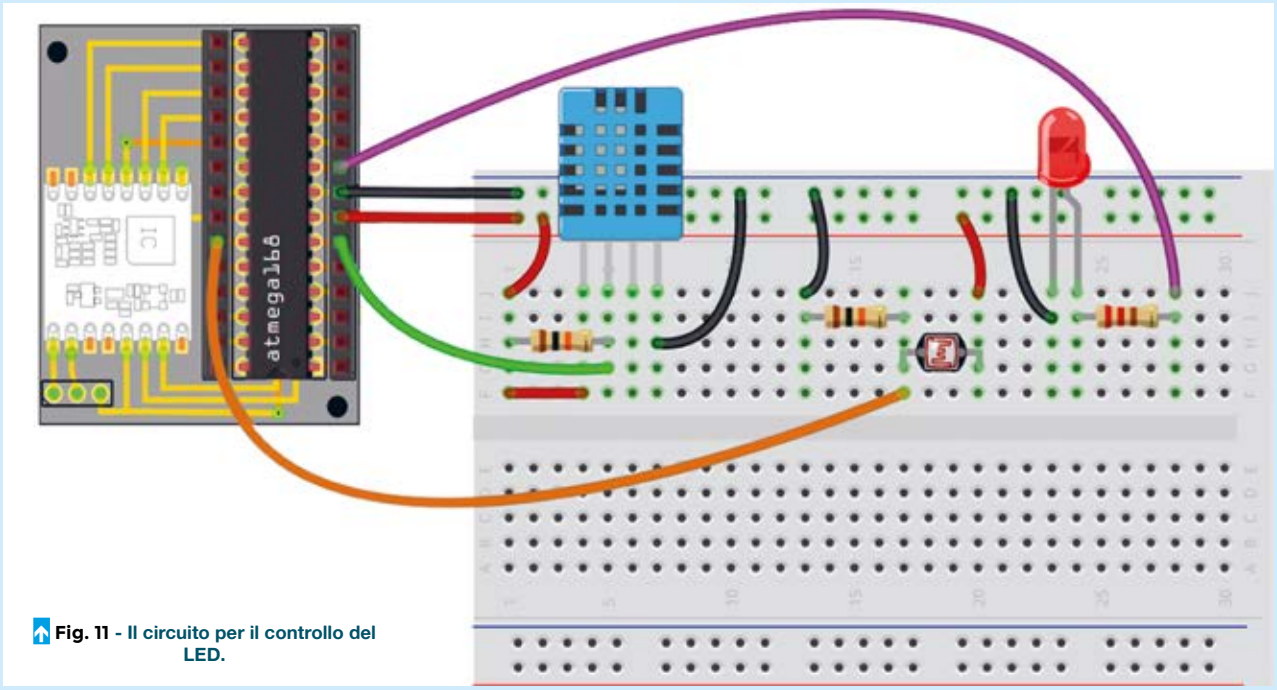

Fig. 11 - Il circuito per il controllo del LED.

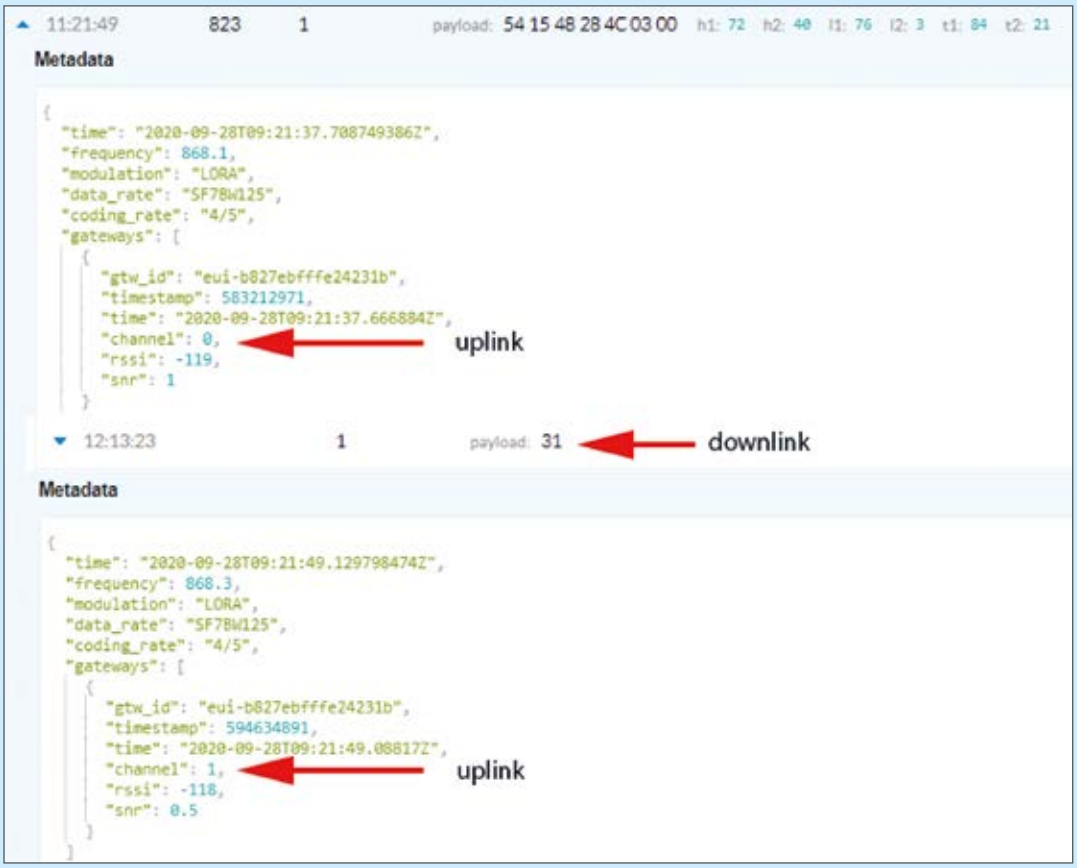

Fig. 12
I due uplink prima e dopo un downlink.

mico per sperimentare con LoRaWAN, ma per applicazioni professionali diventa obbligatorio usare un gateway a 8 canali. Per esempio il gateway iC880A della Wireless Solutions (*https://wireless-solutions.de/products/long-range-radio/ic880a.html*). Ricordiamo che questo gateway è basato sul chip multicanale SX1301 e pertanto è adatto per il *packet forwarder* bidirezionale di Semtech (*https://github.com/Lora-net/packet_forwarder*). Per il collegamento del gateway a Raspberry Pi rimandiamo alla seconda puntata di questo corso.
Facciamo notare che non è possibile utilizzare il *single channel packet forwarder* con il gateway iC880A. Perché non sia possibile usare un gateway single channel per l'*uplink* e il *downlink* è spiegabile osservando la **Fig. 12**, dove vedete due *uplink* avvenuti prima e dopo un *downlink*: notare che vengono usati due canali di *uplink* diversi ogni volta.

THE THINGS NETWORK ZURICH

Oltre al già citato *packet forwarder* di Semtech (che comunque vi invitiamo a provare) è disponibile un *packet forwarder* di The Things Network Zurich. Questa è una comunità di hacker/maker svizzeri che ha modificato il *packet forwarder* orignale di Semtech, apportando molte migliorie. La comunità ha al suo attivo circa 200 gateway sparsi nel territorio svizzero ed è in continua espansione. Oltre al *packet forwarder*, il GitHub della comunità mette a disposizione una serie di utility molto interessanti che vi invitiamo a provare a visitare *https://github.com/ttn-zh*.
Al link *https://github.com/ttn-zh/ic880a-gateway/wiki* trovate un'ottima guida "*From zero to LoRaWAN in a weekend*" per costruire il vostro gateway con Raspberry Pi e iC880A in un weekend, come viene dichiarato nel titolo. Seguendo le istruzioni passo passo, capirete come installare facilmente il *packet forwarder* ed essere operativi in pochi minuti.
La **Fig. 13** illustra il nostro gateway su TTN che abbiamo chiamato ttn zh. Per completezza di informazione, vi diamo alcune indicazioni per la corretta installazione e l'uso di questo packet forwarder della comunità svizzera.
Dal terminale di Raspberry Pi digitate questi comandi per clonare il repository e installare il packet forwarder:

```
git clone https://github.com/ttn-zh/ic880a-gateway.git ~/ic880a-gateway
cd ~/ic880a-gateway
sudo ./install.sh spi
```

L'istallazione chiederà di inserire alcune informazioni; cliccate enter ad ogni opzione in modo da accettare le impostazioni predefinite. Per impostazione predefinita, il programma di installazione cambia il nome host del Raspeberry Pi in ttn-gateway (per evitare collisioni con altre Raspberry Pi eventualmen-

Fig. 13 - Il nostro gateway ttn zh.

te connessi a Internet). Dopo l'installazione viene riavviato il sistema; dopo il riavvio, dal terminale digitate questi comandi per entrare nella directory dove si trova l'eseguibile ed avviarlo con uno script:

cd /opt/ttn-gateway/bin
sudo ./start.sh

Se aprite lo script **start.sh** con un editor di testo, noterete che contiene già le istruzioni per il reset del gateway iC880A il cui pin di reset va collegato al pin GPIO25 di Raspberry Pi. Una volta avviato il *packet forwarder* si vedrà qualcosa di simile alla **Fig. 14**.

CONCLUSIONI

Bene, termina qui questa puntata dedicata alla tecnologia LoRa e alla rete LoRaWAN, nella quale vi abbiamo descritto come usare l'interfaccia di Node-RED per inviare i dati dal web al vostro dispositivo finale, sfruttando The Things Network e un impiegando gateway multicanale per implementare il downlink sulla rete LoRaWAN. Non ci resta che darvi appuntamento alla prossima puntata!

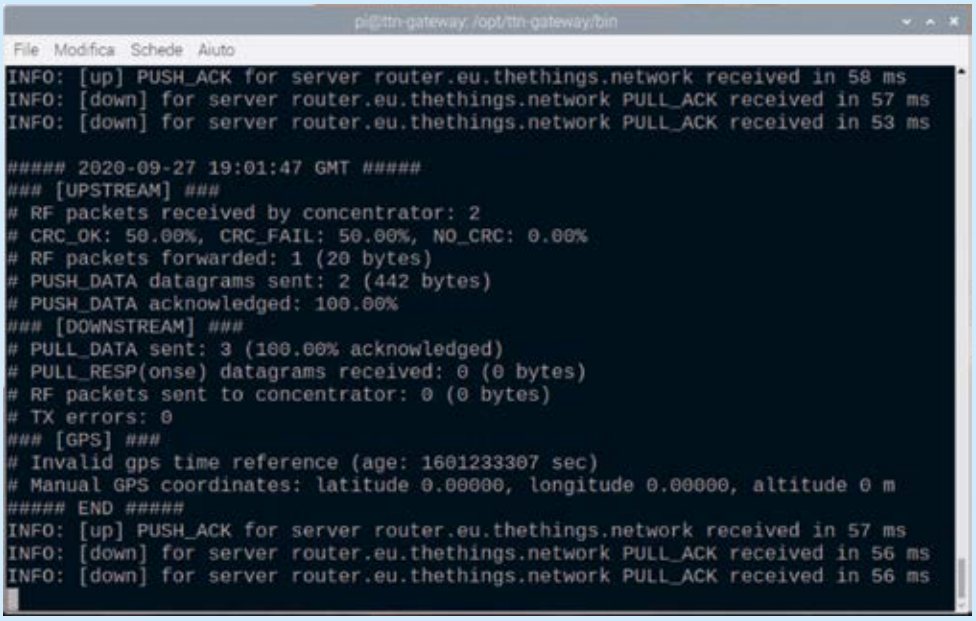

Fig. 14
La finestra del packet forwarder di The Things Network Zurich.