A guide to LoRa Nodes and Servers

Gareth Waymark

August 2021

1 Introduction

With the Internet of Things (IOT) becoming more and more popular a wireless method of data transmission was created that had very long range and used very little power. This long range (LoRa) system has a range of up to 10 miles with the correct conditions and devices can run on batteries lasting years. This protocol is not designed for high data rates or constant transmission. It is perfect for remote sensors wishing to transmit data periodically, maybe a single data packet every few minutes or as little as once a day.

This guide is *not* a step by step tutorial. It aims to consolidate lots of useful information that will assist you in getting a LoRa node up and running. The guide will also cover setting up your own LoRa server to give you complete control over your own network.

2 What is a LoRaWAN?

LoRa®, a trademarked protocol by Semtech, is used for data transmission via radio. It describes the way that data is transmitted by the radio transmitter and detected by a receiver. LoRaWAN® is a network layer protocol that is based on LoRa. It is managed by the LoRa Alliance® and describes how the network must work. Chirpstack is an open source collection of software programs that implement a LoRaWAN. This can be hosted on your own hardware or on a large, public server such as The Things Network.

First, nodes broadcast data packets using the LoRa protocol and these are received by a LoRa Gateway. The LoRa Gateway then sends these packets of data to a LoRa Network server. A LoRa Network Server is a piece of software that runs in the background either on the gateway or on a remote server via the internet. A LoRa Application Server runs in a similar way to the network server, either locally or remotely, and provides a web interface to manage the nodes and gateways. Together, these services make up a LoRa network. There are other services required in order to get the whole system to work and these will be covered later in the guide. The diagram below shows a basic setup. The green node could be an Arduino/LoRa shield, the blue elements represent a Raspberry Pi with a Gateway hat and the red represents a separate server. This could be another Raspberry pi, a PC or it could be run on the same machine as the gateway. Figure 1 shows a basic LoRaWAN network.

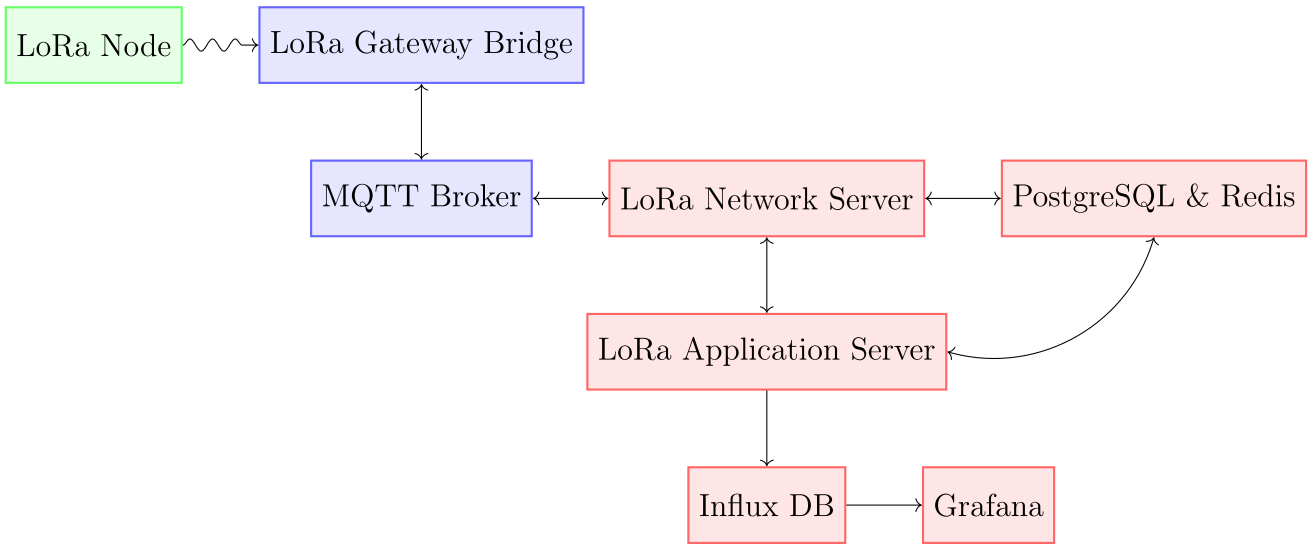


Figure 1: A simple LoRaWAN overview

3 Fair Usage Policy

This is *really* important. Not only do LoRa servers such as TTN impose their own limits on the amount of airtime you can use, there are government set limits as well. It takes an amount of time for a node to transmit a packet. This is called the airtime. There are LoRaWAN airtime calculators available online and these can be very useful. For example, it takes at least 46.3 ms to send a 2 byte payload packet. The Things Network impose a 30 seconds per 24 hour period limit. This means that a node sending 2 bytes can only do so every 134 seconds (2 minutes, 14 seconds) on average in 24 hours. The European limit is given as 1% duty cycle. 1% of 24 hours is 864 seconds; considerably more than 30 seconds allowed by The Things Network. This still means a minimum time of 4.6 seconds between packets therefore it becomes clear that a LoRaWAN is not suitable for large amounts of data transfer. It is also worth noting that if the data rate is reduced to increase range the airtime increases and therefore the time between packets. Using the lowest data rate (DR0) would result in 1.1 hours between packets when using The Things Network.

4 Setting Up An Arduino Node

At its heart, a node is based on a transceiver chip that is capable of using the LoRa modulation protocol. The same chip is often capable of other modulation schemes such as FSK and MSK among others. One common family of chip you will likely encounter is the Semtech SX127x family. These ICs require a micro-controller to configure and control them. There are many prototype boards available on the market and some are easier than others to setup and use. Some boards simply bring the pins of the SX127x IC to pin headers to allow bread boarding and prototyping. This board would require a micro-controller to be programmed to interface with it. This can be a complex task and is not recommended for beginners. Alternatively there are boards that include a pre-programmed micro-controller that manages the interface for you and exposes a high level serial interface. This allows, relatively simple, AT commands to be used to control the LoRa chip. One such board is the RAK811 by RAKWireless. They can come as plain boards or as Arduino Shields/Pi Hats. This guide will be based around the

RAK811 Arduino shield, however it should be relevant to any RAK811 device.

The first step is to read the manufacturers documentation and install any required libraries. There are two different connection methods available; Activation By Personalisation (ABP) and Over The Air Activation (OTAA). ABP & OTAA are two methods that nodes can connect to a LoRaWAN. ABP is easier but less secure and is not recommended so OTAA is the preferred method. This guide will discuss the use of OTAA.

Every node that will connect to a LoRaWAN will have a unique device identification number called a 'Dev Eui'. If you already know the Dev Eui then you can skip this part. To get the Dev Eui from the RAK811 we need to run a command using the serial pass-through sketch or similar (this can be found under the example sketches that came with the RAK811 library you installed). Sending `at+get_config=dev_eui` should return the Dev Eui.

Included in the appendix is an example Arduino sketch that initiates an OTAA connection and then sends temperature readings every 5 minutes. The App Eui & App Key will be provided by The Things Network or Chirpstack. These will be discussed later in the guide. The demo node uses a DHT11 sensor to measure temperature and humidity. This data is formatted to take up as little space as possible in the LoRa packet.

Tip, ensure the Arduino Serial Monitor line endings are set to the correct type, often CR and NL. The RAK811 requires both carriage return (CR) and newline (NL) characters to be sent to signal the processing of a command.

Tip, the RAK811 requires its default serial speed setting to 9600 baud An important sketch is the 'SetBaudRate' sketch (Not all nodes will require this step). This will change the baud rate of the RAK811 LoRa Node to 9600. This is required to fix compatibility issues with Arduino. Simply load the sketch and that should be enough to re-configure the node. If there is not a sketch to change the baud rate you will need to change it using the Serial Pass-through sketch.

Tip, If you have trouble with commands, ensure you are using the correct version of documentation. In the case of the particular RAK811 used for this example we needed to use version 1.4 of the AT commands as the latest V2 commands were not recognised.

Tip, If you choose to use ABP, ensure that when you are configuring a new end device that you enable 'frame counter restarting'. ABP increments a counter with each packet that it receives. If the Arduino does not send the packet with the same frame number the LoRa network will reject it. If you find that the node data is received but then, when you try again later, nothing works you may have forgotten to enable this mode. The correct method is for the node to keep track of the frame counter, even after a reset. This would require writing the frame counter to an EEPROM for example. The OTAA connection method automatically resets the frame counter as soon as a connection is established which avoids this problem.

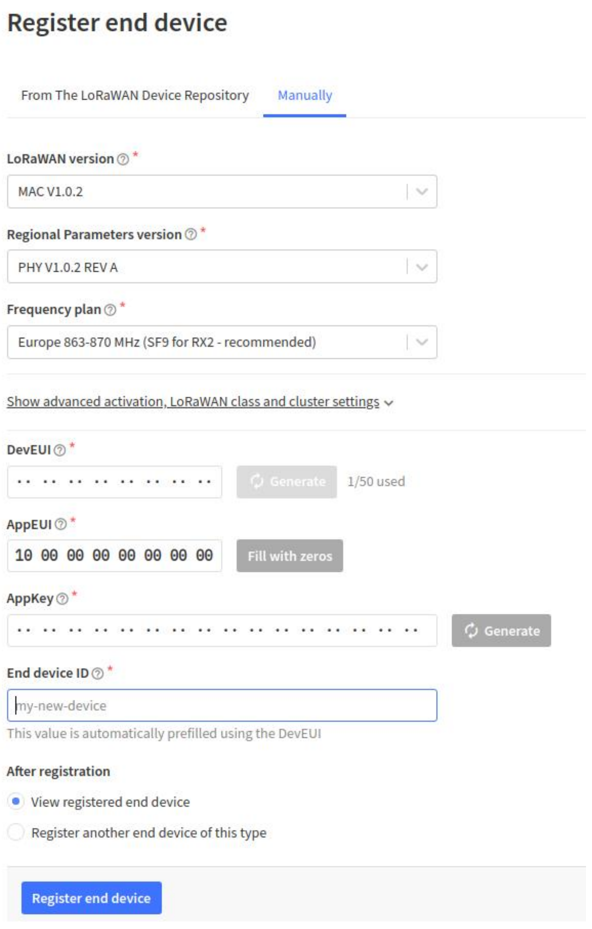
Tip, ensure you send full bytes. The RAK811 only allows you to send full bytes. When you type the AT command the data needs to be typed in hex format, including leading zeros. For example, the integer value of 285_{10} is $11D_{16}$ in hex. To send this on the RAK811 we need to type the data as $011D_{16}$. This is two full bytes. The RAK811 will reject the packet it you try to send $11D_{16}$. An example of this would be to use sprintf with a format specifier such as %04X. This will force a hex value at least 4 numbers long, padded with zeros. This would be suitable for 16 bit ADC values for instance.

5 Configuring The Things Network

5.1 Setting up a new node

This section assumes that you have either a gateway of your own or access to a shared gateway that is connected to TTN. Create an account if you have not already and login. The default version of TTN as of writing is V3. There are a lot of V2 tutorials online but these are not always relevant.

Within both TTN and Chirpstack an 'Application' is a collection of devices/nodes. These devices share some common settings configurable under the application. Therefore the first step is to create a new application. This is as simple as choosing a name and a description. Next you can add end devices. TTN will offer you a choice of pre-configured devices or to manually add a device; Choose manually add. Here you will need to add some information about the node. When using a RAK811 you should select the LoRaWAN version to be MAC v1.0.2 and Rev A. For the frequency plan you should choose 'Europe 863870 MHz (SF9 for RX2 recommended)' Next you can enter the 'Dev Eui'. For the 'App Eui' do not 'Fill with zeros', instead fill with zeros and changed the first zero to a one. The RAK811 will not accept an App Eui of all zeros. Generate an App Key and record it. This needs to be added to the Arduino code to allow the node to connect. Figure 2 shows some basic settings for a new node.



Register end device

From The LoRaWAN Device Repository | Manually

LoRaWAN version *
MAC V1.0.2

Regional Parameters version *
PHY V1.0.2 REV A

Frequency plan *
Europe 863-870 MHz (SF9 for RX2 - recommended)

Show advanced activation, LoRaWAN class and cluster settings

DevEUI *
.. .. Generate 1/50 used

AppEUI *
10 00 00 00 00 00 00 00 Fill with zeros

AppKey *
.. .. Generate

End device ID *
my-new-device
This value is automatically prefilled using the DevEUI

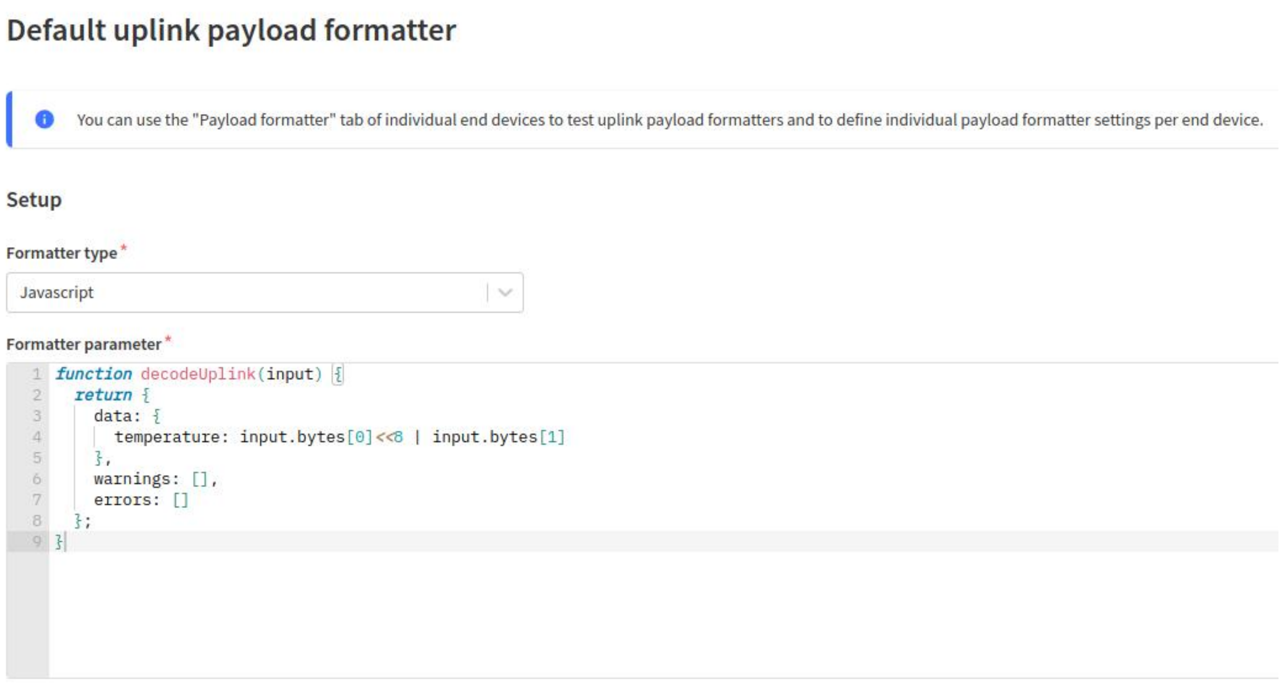
After registration
● View registered end device
○ Register another end device of this type

Register end device

Figure 2: A basic new device setup on TTN v3

5.2 Payload formatting

When you take a measurement using the node, temperature for example, you would have had to convert it to some kind of byte format. That may have been the raw ADC result or some other arbitrary method. TTN & Chirpstack allow you to create a payload formatter that will decode the data received from the node and forward it on in a much more usable format. For example, in the example code in appendix A, the temperature is read as a floating point value, say 23.5. This is multiplied by 10 to give 235. This value can be sent as a single byte value. Unfortunatly, this method is limited to temperatures up to 25.5 degrees as a full byte is 255. We can use an additional byte to allow temperatures well beyond those expected to be encountered in a typical application. These bytes will need a small amount of processing by the payload formatter. Figure 3 shows a simple Javascript payload formatter example with some simple byte shifing to reconstruct the 16 bit temperature value.


Default uplink payload formatter

You can use the "Payload formatter" tab of individual end devices to test uplink payload formatters and to define individual payload formatter settings per end device.

Setup

Formatter type*

Javascript

Formatter parameter*

```
function decodeUplink(input) {
  return {
    data: {
      temperature: input.bytes[0]<<8 | input.bytes[1]
    },
    warnings: [],
    errors: []
  };
}
```

Figure 3: A simple Javascript payload formatter example

6 Getting Your Data From TTN

Data that has been received and processed by TTN is not stored by default. Instead, it is passed on to other services for storage or processing. This section will look at using Node.JS to access packets from TTN using MQTT and store them in a MYSQL database, or similar, on your own computer. The data stored in the database can then be accessed by applications of your choice. A good example is an application called Grafana which can retieve your data and visualise it.

When TTN receives a packet from a LoRa node it is processed and then published via the MQTT broker to all devices that have subscribed to it. The data will be published regardless of there being any subscribers, in other words unless a device is subscribed the data will be lost. Figure 4 shows a basic example of a device, such as a Raspberry Pi, subscribing to an MQTT broker provided by TTN.

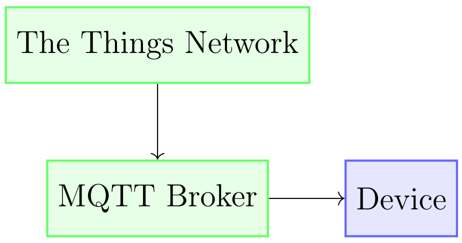


Figure 4: Overview of a simple MQTT publish/subscribe model

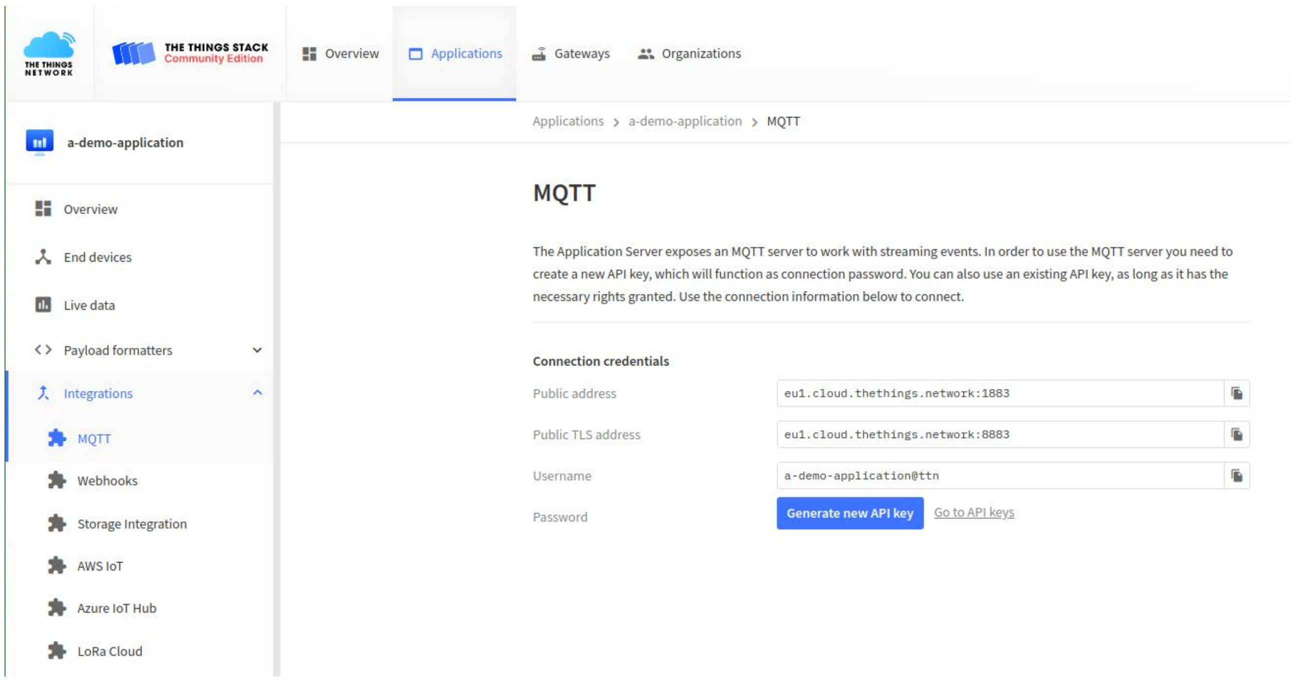
Figure 5 shows an example configuration when setting up MQTT for your TTN application. Generate a new API key and record it somewhere safe. Once you have created a key, you will only have access to it once. If you lose the key you can simply generate a new one. This key can then be used, along with the addresses shown, to subscribe to the TTN MQTT broker. A good way to test this is to use a program called mosquitto. This is a linux command line program that allows you to subscribe to and publish to a broker. You can install mosquitto on a Raspberry Pi or a Linux computer using a command such as

```
sudo apt install mosquitto
```

Once installed the following example command can be run to subscribe to the TTN broker.

```
mosquitto_sub -h eu1.cloud.thethings.network \
              -p 1883 -u a-demo-application@ttn \
              -P {This is the API key generated by TTN} \
              -d -t "v3/a-demo-application@ttn/devices/a-demo-node/up"
```

As packet data is received it is processed by TTN and forwarded to the mosquitto client that is running and these appear on the command line. Mosquitto_sub can be exited once it has been established that the packets are being received correctly.



MQTT

The Application Server exposes an MQTT server to work with streaming events. In order to use the MQTT server you need to create a new API key, which will function as connection password. You can also use an existing API key, as long as it has the necessary rights granted. Use the connection information below to connect.

Connection credentials

Public address	eu1.cloud.thethings.network:1883
Public TLS address	eu1.cloud.thethings.network:8883
Username	a-demo-application@ttn
Password	Generate new API key Go to API keys

Figure 5: The MQTT settings page on TTN

Appendix B includes an example Node.JS script that subscribes to a TTN MQTT broker and then saves the data into a MYSQL database. You should familierise yourself with MYSQL and be able to create an empty database, create a user and create a table. The example uses a database called 'ttn_demo_db'. Within this database is a table called 'sensor_data' with columns for 'time' and 'temperature'.

Tip, MYSQL likes times to be in UTC format. The example script performs some basic manipulation on the incoming time and converts it to UTC. This will be useful later on when we use Grafana.

The Node.JS script can be executed either on the command line or set to run in the background. Once data is being stored correctly we can use Grafana to access it. Install Grafana and, once finished, you may need to start the service (see the Grafana documentation). Open a web browser and type the address of the computer where Grafana is installed followed by the port number such as `http://localhost:3000`. You should be presented with a login page. The default username and password are 'admin', 'admin'.

A Example Arduino Code

An example Arduino script that takes a temperature measurement and sends it to a LoRaWAN.

```
/* Includes for DHT11 sensor */
/* Requires the Arduino DHT Library */
#include <Adafruit_Sensor.h>
#include <DHT.h>
#include <DHT_U.h>

/* Includes for LoRa Node */
#define DEBUG_MODE
#include "RAK811.h"
#include "SoftwareSerial.h"

/* DHT defines */
#define DHTPIN 2
#define DHTTYPE DHT11
DHT_Unified dht(DHTPIN, DHTTYPE);

/* LoRa Node defines */
#define TXpin 11
#define RXpin 10
#define PACKET_DELAY 300 // Seconds between packets
SoftwareSerial RAKSerial(RXpin, TXpin);
RAK811 RAKLoRa(RAKSerial, Serial);

/* Configure OTAA connection details */
// PLEASE PUT YOUR OWN VALUES HERE
String DevEui = "393935347738XXXX";
String AppEui = "1000000000000000";
String AppKey = "C9838344C8E9C13038D8F8219EFBXXXX";
```

```
//==================================================//


void setup() {
  Serial.begin(9600);
  /* Configure DHT11 */
  dht.begin();

  /* Configure LoRa Node */
  pinMode(12, OUTPUT); //Reset Pin
  digitalWrite(12, LOW); //Pull down Reset
  digitalWrite(12, HIGH); //Release reset

  RAKSerial.begin(9600);
  delay(100);
  while (!InitLoRaWAN());

}

void loop() {
  sensors_event_t event; // Create a variable to capture the sensor data
  dht.temperature().getEvent(&event); // Get temperature

  char temp_char[5]; // A temporary variable to store data packet.
  // We need 4 bytes to send the 16bit hex value plus 1 for the string null
  // character.
  int temp;

  /* This is an important step. Here we are taking the floating point value,
   * e.g. 25.32. Next we multiply by 10 to give 253.2 and then convert to an
   * integer 253.  *
   */
  temp = (int)(event.temperature * 10);
  sprintf(temp_char, "%04X", temp); // Convert to a 4 digit hex with leading zeros
  Serial.println(temp_char); // Send the value to the Arduino serial monitor

  int packetType = 0; // 0: Unconfirmed packets, 1: confirmed packets
  RAKLoRa.rk_sendData(packetType, 1, temp_char); // Broadcast the data packet
  int i;
  for (i=0; i<PACKET_DELAY; i++)
  {
    delay(1000); // This 1 second delay is repeated PACKET_DELAY times.
  }
}

char InitLoRaWAN(void)
{
  RAKLoRa.rk_setWorkingMode(LoRaWAN);
  if (RAKLoRa.rk_initOTAA(DevEui, AppEui, AppKey))
  {
    Serial.println("OTAA init OK");
  }
  else
  {
    Serial.println("OTAA init FAILED");
  }

  delay(5000);
  while (RAKLoRa.rk_joinLoRaNetwork(OTAA))
  {
    String ret = RAKLoRa.rk_recvData();
    Serial.println(ret);
    return 1;
  }
}
```

B Example Node.JS Code

An example Node.JS script to subscribe to TTN packets and store in a MYSQL database.

```
// A Node.js script to subscribe to a TTN MQTT server.

const mqtt = require('mqtt');
const mysql = require('mysql');
const moment = require('moment-timezone');

// Configure our MYSQL database user
// ! UPDATE THESE !
const MYSQLoptions={
    host: 'localhost',
    user: 'username',
    password: 'pa$$word',
    database: 'ttn_demo_db'
};

// Configure our TTN connection details
// ! UPDATE THESE !
const TTNoptions={
  username:"a-demo-application@ttn",
  password:"YOUR.API.KEY.J6FWODELULF44GGFUFVKVJIH2X4AWJXWWDUAKOIQJAZWG6GCPK4Q",
  clean:true
};

//Connect to the MQTT Server
var client  = mqtt.connect('mqtt://eu1.cloud.thethings.network:1883', TTNoptions);

//Listen for a CONNACK signal. This tells us that the connection was
//successfull
```

```
client.on("connect", function() {
  console.log("MQTT Connected: "+client.connected);
});

//Listen for any errors that occur during connection
client.on("error", function() {
  console.log("Can't connect "+error);
  process.exit(1)
});

//Listen for a new message and save in database
client.on('message',function(topic, message, packet){
  var data = JSON.parse(message);
  var timestamp = moment.tz(data.time, "Europe/London");
  var timestamputc = timestamp.utc();
  const query = "INSERT INTO sensor_data SET ?";
  const values = {
    time: timestamputc.format("YYYY-MM-DD HH:mm:ss"),
    temperature: data.uplink_message.decoded_payload.temperature / 10
  };

  const con = mysql.createConnection(MYSQLoptions);

  con.connect(function(err) {
    if (err) throw err;
  });

  con.query(query, values, function (err, results) {
    if(err) throw err;
    console.log("Record inserted: ",results.insertId);
  });

  con.end(function(err) {
    if(err) throw err;
  });
});

//Subscribe to the TTN application topic. This is for TTN V3
var topic="v3/a-demo-application@ttn/devices/demo-node-1/up"; // ! UPDATE THESE !
client.subscribe(topic,{qos:1});
```