RAK LoRa Starter Kit

User Manual

Miha Smaka

Table of Contents

1. Introduction

This guide assumes that whoever is using it has a working knowledge of what LoRa is and at least somewhat of an idea of what they would like to accomplish with it. With that in mind, I still feel it is somewhat beneficial to provide a quick summary of the above points, for people who might have accidentally stumbled onto this guide or are not as acquainted with the LoRa/LoRaWAN suite. Since I do not think I am in any sense an authority on the matter, I will provide a summary as per my current understanding. **DISCLAIMER:** The reader is urged not to take anything for granted and encouraged to use common sense, their own knowledge and further reading provided below.

LoRa (INSERT CITATION) is one of the various services that have emerged under the IoT moniker in the past decade. Its specialty is providing low power but long-range communication capabilities, making it almost ideal for vast connections of devices with limited access to power (e.g. various sensors, networks sensors). LoRa (i.e. the chip) specifically takes care of the *physical layer* from a networking standpoint – the actual "long-range communication link" [1], while LoRaWAN defines the network architecture, the "MAC" layer, the application interfaces along with protocols and modes of operation.

LoRaWAN defines a "star-of-stars" [1] network topology in which multiple end nodes (usually sensors of some kind) are connected to a gateway, multiple gateways in turn being connected to a network server which is connected to several application servers. This set-up was chosen to conserve energy and simplicity as opposed to more standard mesh configurations in which "nodes receive and forward information from other nodes that is likely irrelevant for them" [2].

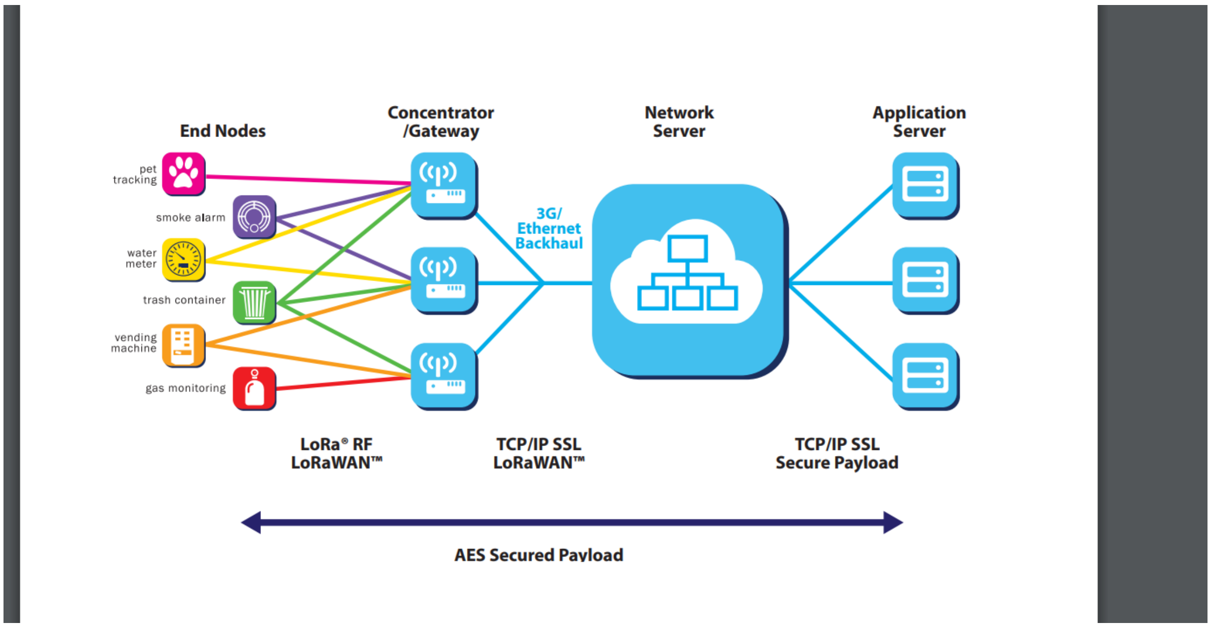


Figure 1 LoRa Network Topology [2]

An end node would thus be any device "too small" or low power that would benefit from a more low-level protocol like LoRa – for an example you can think of an Arduino or anything an Arduino would be used to prototype. A 'gateway' in terms of networking is a device that can communicate (i.e. 'speak') two completely different protocols or protocol suites at once. In the above example it connects to the end nodes on one side using LoRa and on the other side to the network server using classic TCP/IP connections. The gateway takes the payload from the LoRa node and transforms it into a TCP/IP payload transferring it on to the Network Server. It is comprised from several different components, but essentially takes care of the communication/routing between any application servers working through the standard Internet and the end nodes. It thus makes sure to correctly connect an application (e.g. a web server displaying the locations of water sensors) to the end nodes, through the relevant gateways.

One could possibly disregard LoRaWAN and still use LoRa with devices embedded with a LoRa chip with a different network configuration. LoRAWAN is only defined as a standard [3] by the "LoRa Alliance", an organization similar to IEEE, and no official implementation exists. LoRa devices normally do not come shipped with it and it is left up to the user to install it if they wish to do so. However, because of security and efficiency reasons it is very advisable to do so, or at least use another heavily tested alternative as IoT (like printers in the past) devices are very often sought by potential attackers (i.e. "hackers") because of lax security. Additionally, I find it hard to think of many use cases on a basic level where one would not want to use the above architecture, even if resulting in a single application server.

Moreover, since the standard is the only official documents describing the protocols suite, multiple implementations exist (as with TCP/IP) with varying levels of difficulty as to their set-up and how much they incline towards the open-source paradigm. This gives userbase the option of forming private or public networks, with open source or classic network provider (e.g. ISPs) implementations. The most accessible and user-friendly of this is likely "The Things Network" (abbr. "TTN" or "ttn"), which is open source as well. It provides a completely public and open network infrastructure which allows anyone to connect new gateways and end nodes (with specific applications), while running the whole Network Server infrastructure and allowing applications to communicate with it. Their plan is to eventually allow anyone to setup another part of the Network Server thus creating a completely open and distributed network.

In the guide I will be using their services simply because of ease of configuration – it is definitely possible for someone to configure their own network/application server, but I fail to see the benefit of doing that in a guide designed for an introduction. Before I dive into actually coding applications, I will provide an overview of what one needs to do first to set-up the devices.

(I also highly suggest that you at least read the TTN introduction before continuing, it is short, informative and well-written)

2. Necessary materials & device set-up

You will always need two basic things: an end node and a gateway that can form all the necessary connections (e.g. power, monitors, Internet). In some cases, you will not need to connect an additional gateway since your area might already be covered by another public one from TTN – LoRa transmissions from your end node will travel in all directions and could potentially be picked up by some already existing receiver. In a lot of cases you will however want your own gateway either to increase coverage or to make sure traffic for your nodes is not throttled.

a. RAK831 Raspberry Pi LoRA Gateway

This is the device we will be using in our case. You may already receive the device assembled or you may not. Since there are many great guides on this topic, including how to install the correct OS and the correct packet forwarder and exactly the steps one needs to take to ready the device, I will simply provide guides below [4] [5]. I will also give a couple of helpful tips that could've saved me a lot of time:

- **MAKE SURE THE ANTENNA IS ATTACHED/SCREWED ON BEFORE YOU POWER IT ON.**
- Try to make your Raspberry Pi headless first as the guide in [4] suggests – that means configuring it so that you can use it without a screen. You will need a **monitor with a HDMI input** and **HDMI cable** to connect it to the Raspberry Pi so that you can enable SSH on the Pi.
- It might be easier in some cases to just use the Ethernet port and an **Ethernet cable** to connect your Pi directly to your router instead. Your router is likely running DHCP, meaning it can configure the connection completely automatically.
- If there is a lightning bolt on the screen when the RPi is booting up, it means that it does not have enough power. It is recommended to use a power supply of **5V/2A**, as otherwise it will be stuck in a loop of rebooting.
- It will greatly help if you have ever used a Linux terminal before.
- A rule of thumb is that any problem you might encounter before setting up the packet forwarder is probably a **Raspberry Pi/OS** problem and should be Google searched as such. Any problem after that is likely a **TTN** configuration/installation/registration problem and again should be Google searched as such.
- Make sure the *global_conf.json* and *local_conf.json* are configured as instructed in [5].
- If you get your LoRa Gateway and TTN reports and error saying it is already registered, fear not. You will just not be able to access it in the console – make sure the configuration is correct and the gateway will work, nevertheless. You can check that it is connected to TTN on this webpage by using the simple CTRL + F command and searching for the Gateway EUI, or by adding it at the end of the URL. The EUI ("extended unique identifier") is usually obtained through the installation process, if not you can find it in the *global_conf.json* or *local_conf.json*. If it is not there, you obtained a device with an unfinished installation process – the EUI is obtained by running the "ifconfig" command through SSH and taking the MAC address found under the "eth0" denomination. Insert the two bytes "FF FE" after the 3rd byte of the MAC address and you will obtain the device EUI. This is a deprecated algorithm from the IEEE, but so far no alternatives seem to have been proposed.

In summary, I suggest the following; use guide [4] to assemble and setup the Raspberry Pi, preparing it for use. Use this Github page for reference instructions to download all the software, and cross-reference it with [4]. In the end use the above tips if in doubt and [5] to verify everything is running correctly. System logs are available by running the "(sudo) systemctl status ttn-gateway.service" command. If you see an uplink and a downlink acknowledgment, then the gateway is connected and running, ready to receive.

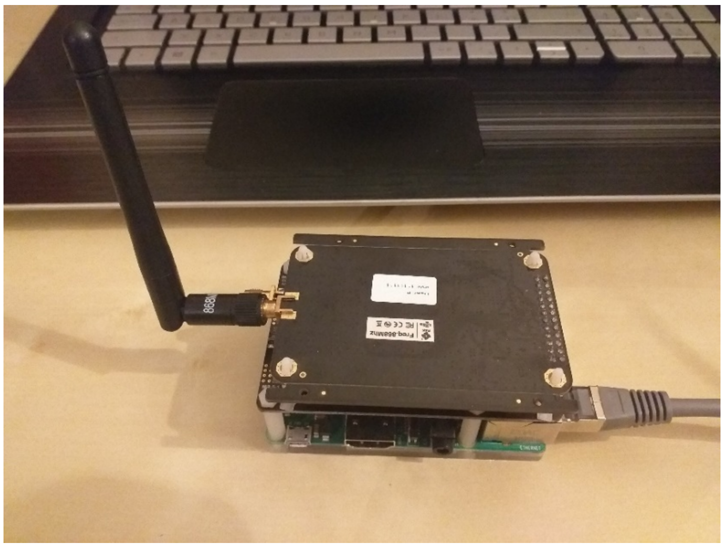


Figure 2 RAK 831 Gateway with attached antenna and connected Ethernet cable

b. Arduino Uno with Dragino LoRa shield

This will be the end node which we will be using. You will need a USB A to B cable to connect it to your computer which should come with all the equipment. If you received the shield and Arduino separately, I assume you received instructions as to how to assemble it – normally just considering the "form factor" of the Arduino and the shield should make it clear. In any case, again **MAKE SURE THE ANTENNA IS ATTACHED/SCREWED ON BEFORE YOU POWER IT ON/CONNECT IT TO THE COMPUTER.**

You will also require the latest version of the Arduino IDE [6] as well as the MCCI LoraWAN-MAC-in-C library [7] (v2.3), with installation instructions being to go on the toolbar under "Sketch" -> "Include Library" -> "Manage Libraries" and use the search bar. Finally, you will definitely need at least a working knowledge of coding in C and low-level computing in order to be able to code for the Arduino.

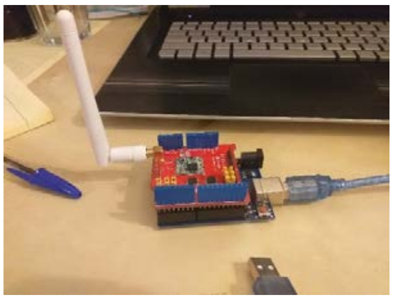


Figure 3 Arduino with Dragino shield and USB cable connected

3. First steps – sending data from end node

All the code/config file are readily available at my Github repository.

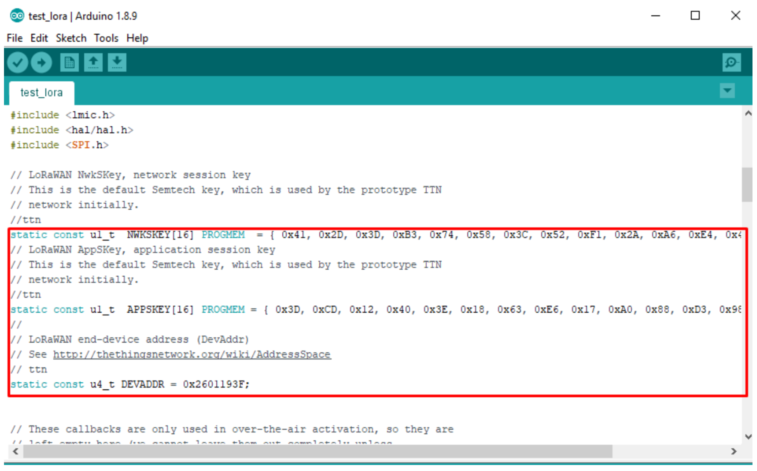
a. Basic test

Now both devices should be set-up. In case that you have a sensor, you can connect it to the Arduino and test it independently from the LoRa shield. If that is not the case, we will simply test it with pre-made data. I highly suggest that you use the Sketch provided by Dragino itself, found here [8], which you will need to slightly modify.

Before we get it into the actual code, it is important to understand that there are 2 different ways of "activating" LoRAWAN devices, meaning them joining/being recognized by the network server [9]. The first one is OTAA ("over-the-air-activation") in which the device is assigned a NwkSKey (used for message verification) and an AppSKey (used for message encryption) with every activation procedure. It is generally considered to be the more secure of the two, and the one used in production. The second one is ABP ("activation-by-personalization"), in which the keys are assigned once and function for as long as the device remains registered. For our prototyping purposes, ABP is more than sufficient and what we will use.

To that purpose if you have not yet created an account for TTN, you should do it now and sign-in. The process should be very self-explanatory. Then, create an application, a suitable ID would probably be "Name_test" or something similar. Make sure you choose the right handler for your region, which would likely be "ttn-handler-eu" (depending on the regi. After that create a device, you may choose the Device ID to be whatever as long as it is unique to be to the application – "Ard1" is probably a good idea. For the Device EUI the guidelines say that the user can choose their own – it is very suggested for that choice to be random, for example on this website. Alternatively, the generation could be done in the same manner as for the Gateway – take the MAC address and insert the "FF FE" bytes after the 3rd byte. Go under the "Settings" tab and choose the ABP method of activation, then remember to "Save" at the bottom of the page.

Connect the Arduino to your computer via the USB and open the Arduino IDE. Copy the sketch from the link found on [8] into the IDE. You will need to go to your application page in the TTN console and copy the Network ("NWSKEY") and Application ("APPSKEY") key, as well as the Device Address in the correct fields (and correct form).

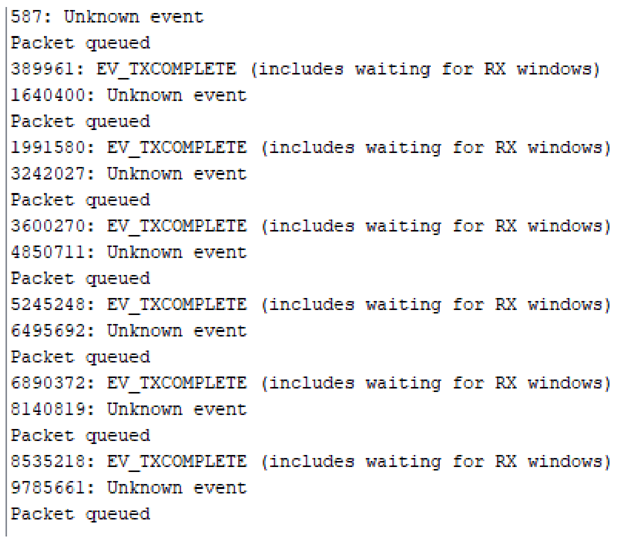


```
#include <lmic.h>
#include <hal/hal.h>
#include <SPI.h>

// LoRaWAN NwkSKey, network session key
// This is the default Semtech key, which is used by the prototype TTN
// network initially.
//ttn
static const u1_t  NWKSKEY[16] PROGMEM  = { 0x41, 0x2D, 0x3D, 0xB3, 0x74, 0x58, 0x3C, 0x52, 0xF1, 0x2A, 0xA6, 0xE4, 0x4
// LoRaWAN AppSKey, application session key
// This is the default Semtech key, which is used by the prototype TTN
// network initially.
//ttn
static const u1_t  APPSKEY[16] PROGMEM = { 0x3D, 0xCD, 0x12, 0x40, 0x3E, 0x18, 0x63, 0xE6, 0x17, 0xA0, 0x88, 0xD3, 0x9
//
// LoRaWAN end-device address (DevAddr)
// See http://thethingsnetwork.org/wiki/AddressSpace
// ttn
static const u4_t DEVADDR = 0x2601193F;


// These callbacks are only used in over-the-air activation, so they are
```

Figure 4 Marking where the keys go

Be sure to copy both the keys and the address in the form you see on the picture – the keys go byte-by-byte prefixed by "0x" and separated by a comma, the address copied directly and prefixed by "0x". Verify that the Gateway is functioning again, through either the TTN console or the API (adding your Gateway EUI at the end of the URL) if you do not have access to the Gateway. Moreover, you will need to go into the directory where your Arduino IDE stores it's add-on libraries (on Windows/OSX this would be the "Documents" folder, on Linux it is under "home\<username>\"). In there you should find the "MCCI_LoRaWAN_LMIC_library\project_config" folder and in it the "lmic_project_config.h". Go to the relevant Github repo and look at the file named the same there.

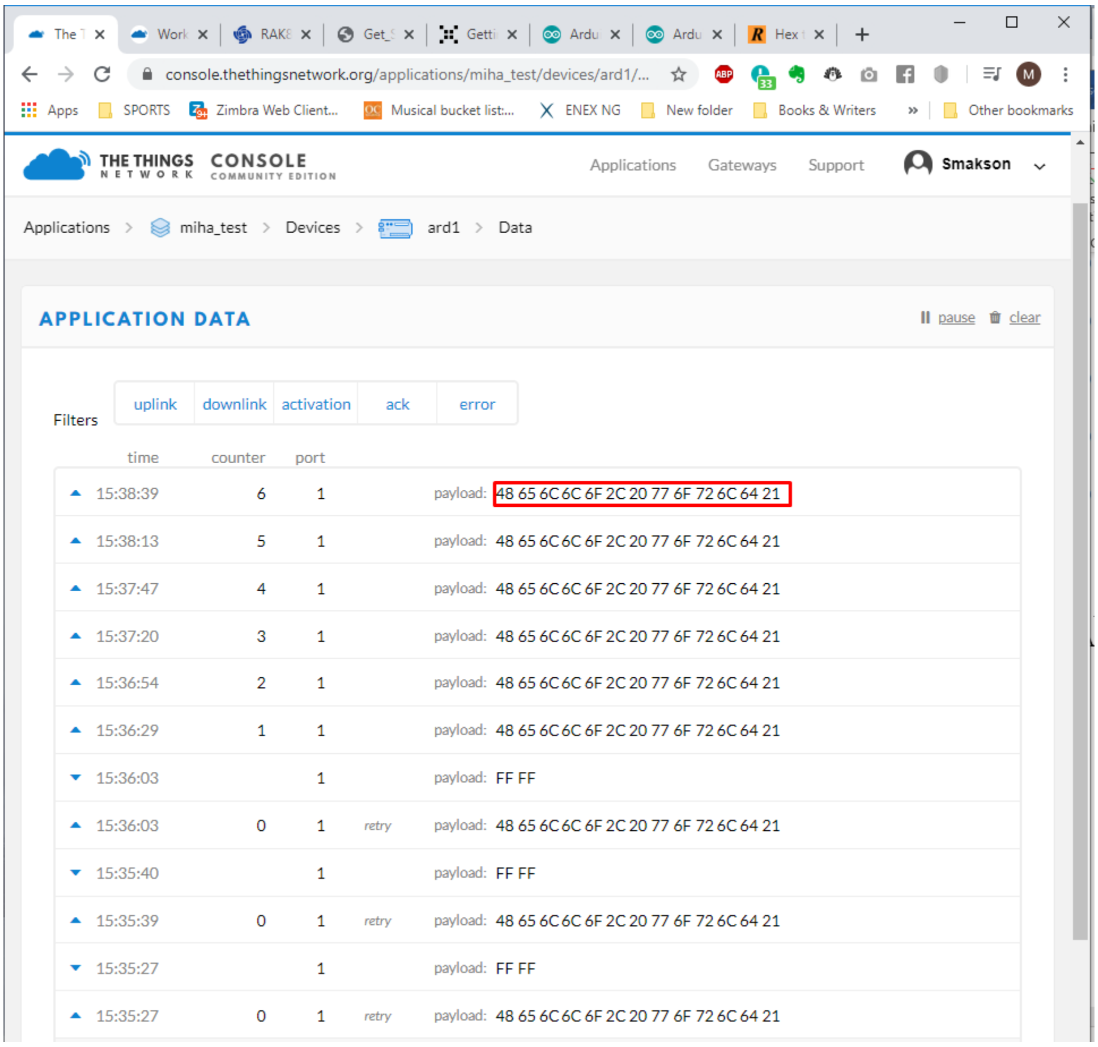
Use your favourite text editor (preferably something that parses C code as well to make it easier, Visual Studio Code is a good open-source one) to make sure that the files match. In the Arduino IDE, first go under "Tools" and under "Port" choose whatever port appears – if there are none, check your USB connection with the Arduino. In the top right corner, there should be an icon which after highlighting says "Serial Monitor", open it so you see what your Arduino is sending back to the computer. Then go under "Sketch" and "Verify/Compile", you should not get any errors, and then "Upload". This is what the Serial Port output should be:



```
587: Unknown event
Packet queued
389961: EV_TXCOMPLETE (includes waiting for RX windows)
1640400: Unknown event
Packet queued
1991580: EV_TXCOMPLETE (includes waiting for RX windows)
3242027: Unknown event
Packet queued
3600270: EV_TXCOMPLETE (includes waiting for RX windows)
4850711: Unknown event
Packet queued
5245248: EV_TXCOMPLETE (includes waiting for RX windows)
6495692: Unknown event
Packet queued
6890372: EV_TXCOMPLETE (includes waiting for RX windows)
8140819: Unknown event
Packet queued
8535218: EV_TXCOMPLETE (includes waiting for RX windows)
9785661: Unknown event
Packet queued
```

Figure 5 Screenshot of Serial Port output

You should not worry because of the “Unknown event” – this only appears because the latest version of this LMIC library includes a new event “EV_TXSTART” which the Dragino code hasn’t been updated to handle yet. If you go under the TTN console you should see the following:


console.thethingsnetwork.org/applications/miha_test/devices/ard1/...

Applications > miha_test > Devices > ard1 > Data

APPLICATION DATA

Filters: uplink | downlink | activation | ack | error

	time	counter	port		
▲	15:38:39	6	1		payload: 48 65 6C 6C 6F 2C 20 77 6F 72 6C 64 21
▲	15:38:13	5	1		payload: 48 65 6C 6C 6F 2C 20 77 6F 72 6C 64 21
▲	15:37:47	4	1		payload: 48 65 6C 6C 6F 2C 20 77 6F 72 6C 64 21
▲	15:37:20	3	1		payload: 48 65 6C 6C 6F 2C 20 77 6F 72 6C 64 21
▲	15:36:54	2	1		payload: 48 65 6C 6C 6F 2C 20 77 6F 72 6C 64 21
▲	15:36:29	1	1		payload: 48 65 6C 6C 6F 2C 20 77 6F 72 6C 64 21
▼	15:36:03		1		payload: FF FF
▲	15:36:03	0	1	retry	payload: 48 65 6C 6C 6F 2C 20 77 6F 72 6C 64 21
▼	15:35:40		1		payload: FF FF
▲	15:35:39	0	1	retry	payload: 48 65 6C 6C 6F 2C 20 77 6F 72 6C 64 21
▼	15:35:27		1		payload: FF FF
▲	15:35:27	0	1	retry	payload: 48 65 6C 6C 6F 2C 20 77 6F 72 6C 64 21

Figure 6 Screenshot of TTN console

The part marked in red is the payload encoded in ASCII, the way C encodes text by default. If you decode it to a string using your favourite web-app (CyberChef is extremely useful for all such operations), you will see your payload. Congrats, you just sent your first piece of data over LoRa!

(If you are having problems and the above didn’t work, I suggest that you go step by step and eliminate each part of the connection: node to gateway, and gateway to server. To that point I found it very useful to have a SDR USB dongle that is basically somewhat of a radio frequency spectrometer – if you are not sure whether it is your LoRa gateway or LoRa node which is not working you can use it to check that the node is actually transmitting something at the right frequency. They are relatively cheap and should be available in any computer hobbyist/electrician shop. On the other hand it might also be useful to check whether the TTN servers are even online here)

Of course, this would not be used in production for several reasons: strings are extremely inefficient and costly for transmitting data, usually there would be a sensor involved and included in the code, to conserve power one would likely power down the processor in between transmissions/scans, finally one would probably want to have an application server somewhere to display the data, or possibly respond to the node. I will dedicate a section to each of these below.

(P.S.: If you power off the Arduino and then restart it, you will need to reset the frame counter on the TTN console. A frame counter simply keeps track of the number of frames sent by the device as to avoid something called replay attacks. The device will otherwise start sending frames with the count of 0, because of the “LMIC_reset()” command.)

DISCLAIMER: I am not sure that the above library is good for actual deployment/production low-power implementations of sensors, but it is likely good enough for any prototype you might need.

b. Principles of sending data over LoRa

LoRa is by design a communication protocol that trades date-rate for range and low-power requirements. If you are running under TTN, there is a fair-usage policy as well as government duty cycle limitations which encourage the user (you) to limit the amount of airtime which you spend transmitting [10]. Even if you are not, you would likely want to conserve the airtime since that implies you are conserving the battery life as well since transmitting takes up a lot of electricity. To that respect it is important to encode data as sparingly as possible, using the least amount of bits/bytes needed. While I could expound more on this, I find the TTN document [11] to be very well written, as well as provide a good introduction to the ideas of data encoding and working with bits/bytes.

c. LMIC library (v2.3)

Before getting into more complex code of sensing data and sending it back, I feel it is important to understand the library we are using more. One can find the latest documentation under the “doc” folder on the official Github repository [7], in which the programming model of the library is well-explained and illustrated. Since that version is as of now in pre-release, you can find the documentation relevant to the one we are using in your Arduino library folder for Windows(and OS X)/Linux respectively in “(Documents/<username>)\Arduino\MCCI_LoRaWAN_LMIC_library\doc”.

At its core the library provides an OS which takes care of timing and scheduling/queueing TX/RX windows as well as other tasks which are called “jobs” – these can do anything as long as it does not run too long. This means that jobs should generally only update states and schedule further actions, that will trigger events. Similarly, to Javascript, the code of the application is driven with an event handler that includes a variety of events that might occur running a LoRa node (transmission starts, transmission completes and RX windows were listened to, Joining/Resetting OTAA connection, etc.) [12]. The control of the application is thus being passed between the event handler and the jobs, which means that we thus need to make sure that a job is being scheduled at all points, otherwise the code will restart (since code in Arduino runs in loops). The simplified diagram below should make it clearer what the control flow of the app looks like.

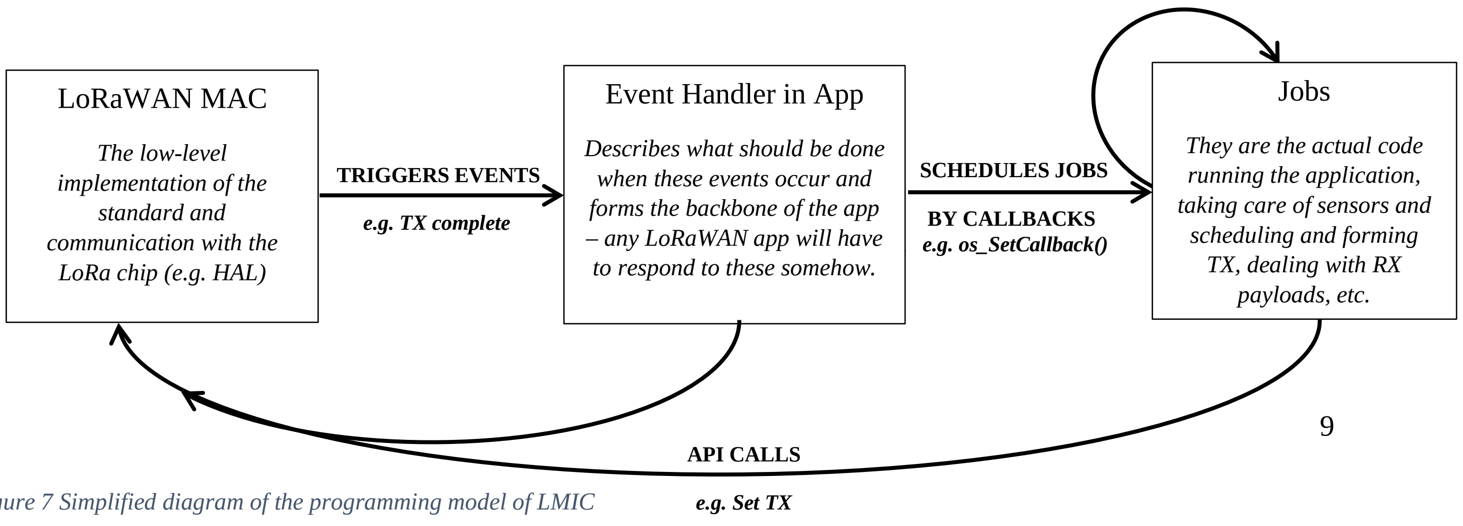
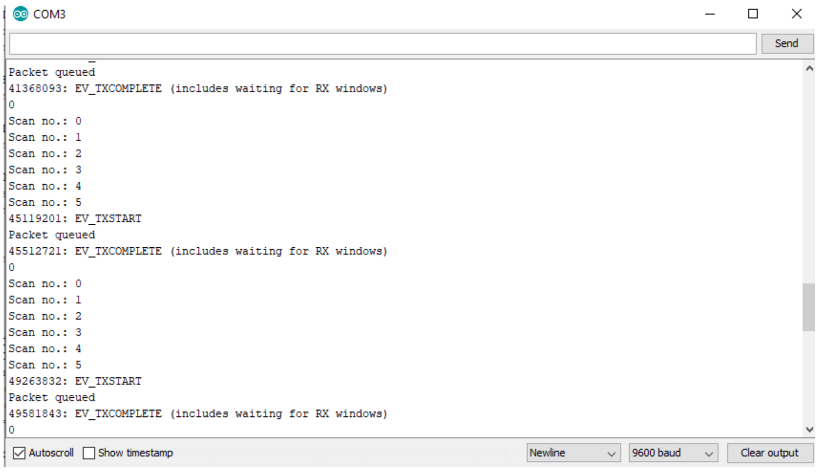

LoRaWAN MAC
The low-level implementation of the standard and communication with the LoRa chip (e.g. HAL)
TRIGGERS EVENTS
e.g. TX complete
Event Handler in App
Describes what should be done when these events occur and forms the backbone of the app – any LoRaWAN app will have to respond to these somehow.
SCHEDULES JOBS
BY CALLBACKS
e.g. os_SetCallback()
Jobs
They are the actual code running the application, taking care of sensors and scheduling and forming TX, dealing with RX payloads, etc.
API CALLS
e.g. Set TX

Figure 7 Simplified diagram of the programming model of LMIC

d. Sensors

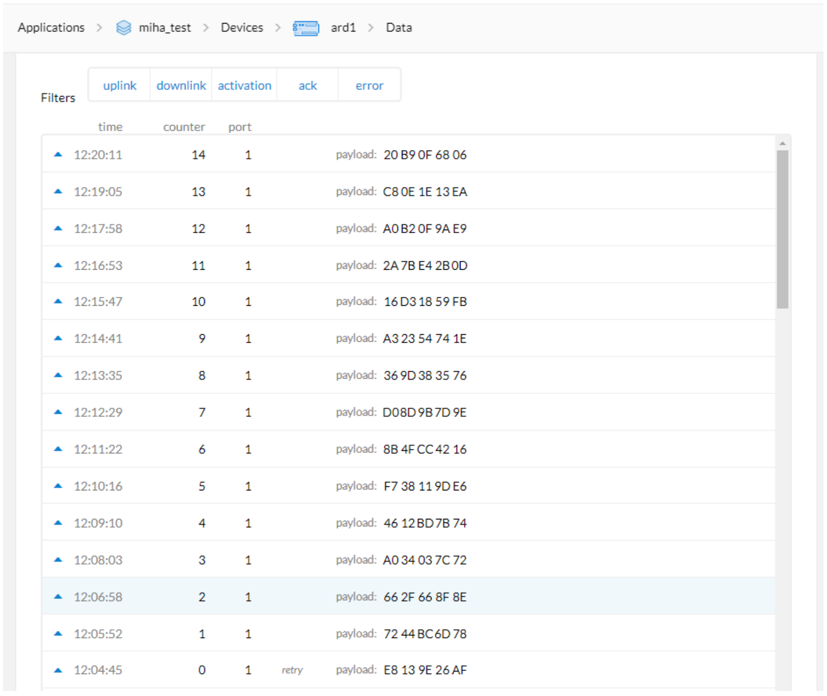
Understanding how the library is intended to work is important if we wish to do anything more than just send a “Hello, world!” message. With any sensor on the other it would probably be a good idea to have more than one job and pass the tasks in a slightly more complex manner. I coded a short extension of the basic example and put it on the Github repo with comments that explain what I was trying to do. It is tested and should work as long as the Arduino has power and the time intervals are not set too long since there are issues related to the way LMIC time works. It can be found under the name of “loraABPsensor.ino” and should be explained in the comments of the file itself. The output on the Serial Monitor should look like:



```
Packet queued
41368093: EV_TXCOMPLETE (includes waiting for RX windows)
0
Scan no.: 0
Scan no.: 1
Scan no.: 2
Scan no.: 3
Scan no.: 4
Scan no.: 5
45119201: EV_TXSTART
Packet queued
45512721: EV_TXCOMPLETE (includes waiting for RX windows)
0
Scan no.: 0
Scan no.: 1
Scan no.: 2
Scan no.: 3
Scan no.: 4
Scan no.: 5
49263832: EV_TXSTART
Packet queued
49581843: EV_TXCOMPLETE (includes waiting for RX windows)
0
```

Figure 8 Output of Serial Monitor for 'sensor' example

Since I did not have a sensor, I simply used the “random(255)” command to generate random bytes, simulating the data one would get from sensor. Instead of transmitting every time that I made a scan, I found it more useful to simply keep an array of several scans and transmit only when I reach a certain amount of “MAX_SCANS”. The output on the TTN console can be seen here:



time	counter	port		
▲ 12:20:11	14	1		payload: 20 B9 0F 68 06
▲ 12:19:05	13	1		payload: C8 0E 1E 13 EA
▲ 12:17:58	12	1		payload: A0 B2 0F 9A E9
▲ 12:16:53	11	1		payload: 2A 7B E4 2B 0D
▲ 12:15:47	10	1		payload: 16 D3 18 59 FB
▲ 12:14:41	9	1		payload: A3 23 54 74 1E
▲ 12:13:35	8	1		payload: 36 9D 38 35 76
▲ 12:12:29	7	1		payload: D0 8D 9B 7D 9E
▲ 12:11:22	6	1		payload: 8B 4F CC 42 16
▲ 12:10:16	5	1		payload: F7 38 11 9D E6
▲ 12:09:10	4	1		payload: 46 12 BD 7B 74
▲ 12:08:03	3	1		payload: A0 34 03 7C 72
▲ 12:06:58	2	1		payload: 66 2F 66 8F 8E
▲ 12:05:52	1	1		payload: 72 44 BC 6D 78
▲ 12:04:45	0	1	*retry*	payload: E8 13 9E 26 AF

Figure 9 TTN console output of the same Sketch

This was done to avoid several overhead power costs with powering up the antenna at the beginning of the transmission and the receive windows after it, which would not be necessary in real sensor application either. Of course, this is not yet truly low power. For that one would need to put the processor to sleep in between scans and keep the data in persistent, non-volatile memory.

e. Low power

From [13] you can see that there are extensive options for configuring the Arduino so that it consumes less power. I have fairly limited knowledge about microprocessors though, so I am unsure how much all of these can be applied to the Arduino Uno which is not on a breadboard and moreover has to communicate with the LoRA shield. It can be seen in [13] that even if I knew what I was doing it on a circuit level, the biggest power drops come from using a bare processor coupled with avoiding the power regulators – something I did not wish to try on a borrowed board.

I think it is a better idea to look at a more high-level approach, such as one implemented in [14] and simply provide more conceptual code rather than something that would function the best in production. It turns out that you do not need to save anything in persistent memory as when you put the processor to sleep, volatile memory (SRAM on Arduino Uno persists). This means that we need to simply modify the previous code a bit and take into account that the device will be sleeping in between.

The file in the Github repository is called 'lowpower_lora.ino', but I have not had the opportunity to test it yet so there is a possibility it does not work. It should still have the right idea as to what needs to be done. Make sure that you install the Low Power library before you run the code, otherwise it will not work. The code itself is again explained in the comments. Again, keep in mind however that to make a really low power solution to whatever you wish to solve with LoRa it will be much more power efficient if you make your own board (as on a breadboard) and likely that the circuit implements parts of LoRaWAN already. The above is only good for prototyping and learning purposes.

f. End notes

I would like to reiterate that none of the above code is ready for production and true deployment. It likely does not adhere to the fair usage policy and/or duty cycle limitations, as well as not being completely secure because of the use of ABP as opposed to OTAA. It does however form a legitimate basis out of which proper applications can be created.

4. Application server

On the TTN website it says that they provide several SDKs for different language, one of them being in Python, my preferred language to code in. It turns out that this SDK has been discontinued however and we will need to take an alternative approach. TTN provides two different APIs; one which provides the whole device, application and gateway management platform – like the console – and a simpler API, used simply for managing, receiving and sending messages. I believe that for most use cases, the TTN console is already more than practical for managing the application and devices so I will only focus on the Data API here.

Since not everybody can have a client running at all times, it makes MQTT subscriptions slightly impractical – they proceed in real-time and if there is no client to receive and store them will be gone forever. For that reason, TTN provides an automatic "Integration", the option to store your message for up to a week and access them in bulk with a REST API. You can access it in your Console, in your application overview under "Integrations". From then on choose Storage Integration and finish subscribing. You will get a link to a webpage with examples of all the possible queries and ways to produce them, as well as example output.

TO KEEP IN MIND: Data will only start getting stored AFTER you activate the option in integrations. All your data before that will be lost, with no way to get it back. There might also be already existing solutions to whatever you might need, it is not necessary to code your own – what is below is done primarily for learning purposes.

a. MQTT API

Before we begin developing even a barebones form of a web app, we need to make sure that we can even keep track of messages outside of one. There exists a library for MQTT Clients in more or less any language you might want to choose, while I will provide one coded in Python. For that to work you need the "paho-mqtt" library installed. Using the API reference and the library instructions, there is a simple example in the Github repo that simply prints messages as they are received, under the name of "basicMQTT.py". The sample output is:

b. Django

While it would likely be useful enough just to have a CLI which would allow us to run a real-time client, it would likely be more practical to have at least a simple GUI. Since we would want it to be readily accessible by the internet, to use it even if we cannot locally access the computer, the simplest idea is to make a web application. Python provides several different frameworks that allow this, but the simplest one which comes with everything included is Django. It allows you to use a common interface for both the server and application, comes with an integrated database and implements best security practices automatically.

A fairly simple, but working, implementation can be found under the name "basicDjango" in the same Github repository. It makes use of both the MQTT API as well as the Storage API – it checks at the beginning in bulk for all the messages it might have missed based on the timestamps, after that running the MQTT client to keep up with any new messages that might arrive in real-time. It also keeps all the past ones in a database and allows the user to modify it after a credential check, using the default Django admin functionality. It is currently only made to accommodate one device, but can be adapted for multiple if necessary. Potentially it could also be used independently of TTN, if one was to take care of the routing of the packets (Network Server) on their own. More explanations are found in the comments. If you are unfamiliar with Django, I highly suggest that you follow the Getting Started guide – that should also help you learn how to run the project.

c. Downlink issues

Ideally there are a lot of situations in which we would like to send messages back to the device, what is referred to as downlink in LoRaWAN terminology. I've had extensive issues doing this with LMIC, which I had not managed to solve before returning the device. From scouring various forums I've grown to believe that it is an issue regarding timing, for which there exists a hack "setClockError()", which is not truly consistent either. Moreover it is power intensive, making it non-ideal even if it would work.

I am however sure that this bug will be corrected in newer versions of the library, so I extended the "lowpower_lora.ino" to include the option that downlink reception defines the sample rate. I tried to keep the message format as short as possible, meaning it would be only 2 bytes – the 1st one being how many measurements should be taken per day, the 2nd one defining how large the buffer should be, how many measurements can be kept before transmitting. I did not get a chance to test it out because of the previously mentioned problems with downlink messages. The code for the node can be found in "downlink.ino". The Django app already supports it, with there being a relevant downlink form.

5. RAK811 WisNode

The WisNode presented me with a quite a few issues and I haven't managed to get it working yet. The documentation that exists on the topic is a bit conflicting as different versions of the module seem to have different guides, but not all of them accessible on the RAK website. The two versions which I found are the older one and the one found on the newer Github repository, neither of which I could follow. I first tried simply connecting it to my computer with a Micro USB cable and control it using the Serial Tool mentioned in the 1st guide [15] to no success. I could not find the one mentioned in the 2nd guide [16], so I assumed that meant that I needed to update the firmware for which I would need to move the bridge connecting two pins [15] [16]. The only way I could think of doing that was by soldering, which I did not wish to attempt since I have no experience of soldering on circuit boards and did not wish to damage it.

I thought of using it with the Arduino, but this forum post seemed to suggest that I needed to make external connections [17], which again I did not have the hardware for or the experience to do so surely and safely. I am leaving the references here in hopes that they might help someone else.

References/Further Reading

[1] LoRa Alliance, “About LoRaWAN,” 2019. [Online]. Available: https://lora-alliance.org/about-lorawan. [Accessed 30 December 2019].

[2] LoRa Alliance, “What is LoRaWAN,” [Online]. Available: https://lora-alliance.org/sites/default/files/2018-04/what-is-lorawan.pdf. [Accessed 30 December 2019].

[3] LoRa Alliance, “LoRaWAN Specification,” 2018. [Online]. Available: https://lora-alliance.org/sites/default/files/2018-07/lorawan1.0.3.pdf. [Accessed 2 January 2020].

[4] K. S, “RAK831 + RASPI 3 LORA GATEWAY : STEPS TO AN ONLINE FUNCTIONAL GW,” The Things Network, 8 October 2017. [Online]. Available: https://www.thethingsnetwork.org/labs/story/rak831-lora-gateway-from-package-to-online. [Accessed 7 January 2020].

[5] J. Stoking, “Semtech UDP Packet Forwarder,” The Things Network, 26 July 2018. [Online]. Available: https://www.thethingsnetwork.org/docs/gateways/packet-forwarder/semtech-udp.html. [Accessed 7 January 2020].

[6] Arduino, “Arduino - Software,” Arduino, 2020. [Online]. Available: https://www.arduino.cc/en/main/software. [Accessed 7 January 2020].

[7] G. contributors, “MCCI LoraWAN-in-C Arduino Library,” Github, [Online]. Available: https://github.com/mcci-catena/arduino-lmic. [Accessed 7 January 2020].

[8] Dragino, “Connect to TTN,” MediaWiki, 10 October 2018. [Online]. Available: https://wiki.dragino.com/index.php?title=Connect_to_TTN#Use_LoRa_Shield_and_Arduino_as_LoRa_End_Device. [Accessed 8 January 2020].

[9] M. Monneret, “Addressing & Activation,” The Things Network, 19 November 2019. [Online]. Available: https://www.thethingsnetwork.org/docs/lorawan/addressing.html. [Accessed 01 January 202].

[10] M. Monneret and H. Visser, “Duty Cycle,” The Things Network, 25 October 2019. [Online]. Available: https://www.thethingsnetwork.org/docs/lorawan/duty-cycle.html. [Accessed 14 January 2020].

[11] M. Arits, F. Zandbergen and A. Mellbratt, “Working with Bytes,” The Things Network, 10 April 2018. [Online]. Available: https://www.thethingsnetwork.org/docs/devices/bytes.html. [Accessed 15 January 2020].

[12] T. Moore, “Arduino LoRaWAN MAC in C (LMIC),” 10 November 2018. [Online]. Available: https://github.com/mcci-catena/arduino-lmic/tree/master/doc. [Accessed 16 January 2020].

[13] N. Gammon, “Gammon Forum : Electronics : Microprocessors : Power saving techniques for microprocessors,” Nick Gammon, 2013 March 2013. [Online]. Available: http://www.gammon.com.au/forum/?id=11497&reply=5#reply5. [Accessed 1 January 2020].

[14] D. Rhee, “ProMini_Lora_Otaa_temperture,” 29 June 2016. [Online]. Available: https://github.com/DiederikRhee/ProMini_Lora_Otaa_temperture/blob/master/Lora_temp_hum/Lora_temp_hum.ino. [Accessed 16 January 2020].

[15] RAK Wireless, “Get_Start_with_RAK811_WisNode-LoRa.pdf,” 23 August 2019. [Online]. Available: Get_Start_with_RAK811_WisNode-LoRa.pdf. [Accessed 18 January 2020].

[16] RAK Wireless, “RAK811_LoRaNode,” Github, 29 March 2019. [Online]. Available: https://github.com/RAKWireless/RAK811_LoRaNode. [Accessed January 18 2020].

[17] F. contributors, “RAK811 Wisnode Lora Module v1.2 with Arduino UNO NOT WORKING,” RAK Wireless, November 2019. [Online]. Available: https://forum.rakwireless.com/t/rak811-wisnode-lora-module-v1-2-with-arduino-uno-not-working/1101. [Accessed 18 January 2020].

[18] R. Sanchez-Iborra, J. Sanchez-Gomez, J. Ballesta-Viñas, M.-D. Cano and A. Skarmeta, “Performance Evaluation of LoRa Considering Scenario Conditions,” *Sensors,* vol. 18, p. 772, 2018.

[19] W. contributors, “Gateway (telecommunications),” Wikipedia, The Free Encyclopedia, 30 December 2019. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Gateway_(telecommunications)&oldid=933132218. [Accessed 7 January 2020].

[20] F. l. h. MaartenArits, “Device Registration,” The Things Network, 8 November 2018. [Online]. Available: https://www.thethingsnetwork.org/docs/devices/registration.html. [Accessed 9 January 2020].