RuuviTag on The Things Network Tutorial Documentation

Release 1.0.0

Rafael Römhild

Feb 17, 2021

Contents:

1 Hardware **3**

2 TL;DR **5**

3 LoRaWAN limitations **7**

4 Device setup **9**

5 TTN Device Registration **11**

6 Disable WiFi **13**

7 Hands on code **15**

7.1 Configuration (settings.py) 15

7.2 LoRaWAN node (node.py) 16

7.3 Main (main.py) 16

8 TTN Payload Format Decoder **19**

In this tutorial we use a LoPy or FiPy to track temperature and humidity from a location with no WiFi and (in my case) no power supply socket and publish the RuuviTag sensor data on The Things Network with the MicroPython RuuviTag Scanner.

This tutorial uses settings specifically for connecting to The Things Network within the European 868 MHz region. For another usage, please see the settings.py and node.py files for relevant sections that need changing.

Note: The code in this tutorial is not bounded to The Things Network and can also be used with other LoRaWAN networks.

CHAPTER 1

Hardware

- 1 x LoPy or FiPy from pycom
- 1 x LoRa Antena
- 1 x Expansion Board
- 1 x Case
- 1 x Lithium Ion Polymer Battery or battery pack
- 1 or more RuuviTags

CHAPTER 2

TL;DR

If you are familiar with MicroPython, LoPy, RuuviTag, TTN or just want to get started now, you can get the up to date snippets from the tutorial repository on GitHub. Modify the settings.py file and copy all *.py files to your device.

CHAPTER 3

LoRaWAN limitations

LoRaWAN is limited on how much data can be send over the network. You can read more about the limitations on:

- Best practices to limit application payloads
- Limitations: data rate, packet size, 30 seconds uplink and 10 messages downlink per day Fair Access Policy

In this tutorial we only want temperature and humidity from our tags. To save space we use the RuuviTag Data Format 5. But you can pack the data in any format you like or add more sensor data.

Note: A good goal is to keep the payload under 12 bytes. For temperature and humidity we need 2 bytes for each and one extra byte as a identifier, if we plan to scan more than one tag. In sum we have 5 bytes for each tag. As we don't need to send updates every minute we can add two or three more tags to the payload and send the measurements every 5 or 10 minutes.

CHAPTER 4

Device setup

If you are new to LoPy/FiPy, I recommend you start with updating your device firmware and go on with REPL & uploading code. To upload code you can use FTP, the Pymakr Plugins or use something like the mpfshell. I prefer mpfshell. With mpfshell you can easily manage the data on your device similar to an ftp client program and also access the REPL prompt.

After your are familiar with your device and updated to the latest firmware, it's time to install the MicroPython RuuviTag Scanner. Copy the `ruuvitag` directory from the repository to your device `/flash/lib/` directory.

In example with mpfshell on Linux with the device connected to `ttyUSB0`:

1. Clone the repository:

```
git clone https://github.com/rroemhild/micropython-ruuvitag.git
```

2. Go to the module directory:

```
cd micropython-ruuvitag
cd ruuvitag
```

3. Copy the files to your device

```
mpfshell ttyUSB0
cd lib
md ruuvitag
cd ruuvitag
mput .
```

CHAPTER 5

TTN Device Registration

Before we can start hacking, we need to add our new device to The Things Network.

1. Start a MicroPython REPL prompt.
2. Run the following commands in the REPL:

```
>>> import ubinascii
>>> from network import LoRa
>>> lora = LoRa(mode=LoRa.LORAWAN)
>>> print(ubinascii.hexlify(lora.mac()).upper().decode('utf-8'))
```

This should print your Device EUI like:

```
70B3D5499C89D4CE
```

3. Follow the steps to register your device.

Note: We'll use the Over The Air Activation (OTAA) to negotiate session keys for further communication.

CHAPTER 6

Disable WiFi

To disable wifi on boot, connect to the REPL prompt and run the following commands:

```
>>> import pycom
>>> pycom.wifi_on_boot(False)
```

This will disable WiFi on your pycom board and it will persist between reboots.

CHAPTER 7

Hands on code

The device is ready, our TTN is setup, finally we can start add our code. We have 3 files: settings.py, node.py and main.py. Create these files on your workstation and copy them later to the device root (`/flash`) directory. The main.py is the file that run on each boot.

7.1 Configuration (settings.py)

This file contains settings for the device. As we use the OTAA activation copy the `Application EUI` and `Application Key` from the TTN console "device overview" to the appropriate variables.

The *RUUVITAGS* variable is used as a device whitelist for the RuuviTags you want to publish data. In our application we use the tuple index from the mac addresses as a device id. In example the first mac address is the device on the stable box and the second from the greenfield sites.

Note: Keep in mind that we want a small payload, only allow some tags to publish sensor data.

```
DEBUG = True

# OTAA activation
NODE_APP_EUI = ''
NODE_APP_KEY = ''

# Device deepsleep time in seconds
NODE_DEEPSLEEP = 30

# RuuviTags whitelist, other tags will be ignored, all lowercase
# RUUVITAGS = (b'aa:bb:02:03:04:05', b'aa:ab:12:13:14:15')
RUUVITAGS = (b'c2:0d:d5:a3:b3:54')

# Bluetooth scan timeout
TIMEOUT = 10
```

7.2 LoRaWAN node (node.py)

This file contains the LoRaWAN network setup and don't need to be modified for the European 868 MHz region. The LoRaWAN class abstract the network access. I'll not get in details with this code to keep the focus on the RuuviTag code. In short this class handle the LoRaWAN setup for the European 868 MHz region, joins the network, prepare a socket and take care of the OTAA session keys.

7.3 Main (main.py)

The main.py file scans the RuuviTags in range, prepare sensor data and send the payload to The Things Network. To extend the battery life the device goes into deepsleep mode, wake up after 5 minutes and repeats.

We start to import required modules, node, settings and the RuuviTagScanner.

```
import machine

import settings

from ustruct import pack
from lorawan import LoRaWAN
from ruuvitag.scanner import RuuviTagScanner
```

Now we need to pack the sensor data. We use the data format from RuuviTag for temperature and humidity and add a tag id:

Offset	Description
0	Tag ID (8bit)
1-2	Temperature in 0.005 degrees (16bit signed)
3-4	Humidity in 0.0025% (16bit unsigned)

In example for 2 RuuviTags the following payload will be send:

```
b'\x00\x11\x8aZ\xd0\x01\x06&y\xe0'
```

If we split this into parts, we get:

Stable			Greenfield sites		
ID	Temp	Hum	ID	Temp	Hum
00	118A	5AD0	01	0626	79E0

To achieve this we add two functions to our code:

```
LoRaWAN.DEBUG = settings.DEBUG


def pack_temp(temp):
    """Temperature in 0.005 degrees as signed short"""
    temp_conv = round(round(temp, 2) / 0.005)
    return pack("!h", temp_conv)


def pack_humid(hum):
```

(continues on next page)

(continued from previous page)

```
    """Humidity in 0.0025 percent as unsigned short"""
    hum_conv = round(round(hum, 2) / 0.0025)
```

To bring all the sensor data together we add a payload variable with an empty bytes objects, later we'll add the packed sensor data to this object:

```

```

Now we initialize the *RuuviTagScanner*. Remember to not scan all the tags around you and add just the ones you need.

```
    payload = b""
```

We are setup and can start scanning for the tags and pack the data together. You can set a higher timeout in the settings.py file if your tag is on a longer range.

The tag id is his index from the whitelist tuple we set in *settings.RUUVITAGS*. When you unpack the payload on the target platform you have to remember the tag position from the tuple.

```
    # get all data and prepare payload and add them to the payload
    print("Scan for ruuvitags")
    for ruuvitag in rts.find_ruuvitags(timeout=settings.TIMEOUT):
        id_payload = settings.RUUVITAGS.index(ruuvitag.mac.encode())
        payload = (
```

When all tags where processed and our payload is ready, we setup up LoRaWAN and send out the payload.

```
            + pack_temp(ruuvitag.temperature)
            + bytes([ruuvitag.rssi * -1])  # rssi
```

At the end we send the device into deepsleep mode for 5 minutes:

```
    print("Setup lorawan")
```

The complete main.py file:

```
import machine

import settings

from ustruct import pack
from lorawan import LoRaWAN
from ruuvitag.scanner import RuuviTagScanner


LoRaWAN.DEBUG = settings.DEBUG


def pack_temp(temp):
    """Temperature in 0.005 degrees as signed short"""
    temp_conv = round(round(temp, 2) / 0.005)
    return pack("!h", temp_conv)


def pack_humid(hum):
    """Humidity in 0.0025 percent as unsigned short"""
    hum_conv = round(round(hum, 2) / 0.0025)
```

(continues on next page)

(continued from previous page)

```
    return pack("!H", hum_conv)


def main():
    payload = b""

    rts = RuuviTagScanner(settings.RUUVITAGS)

    # get all data and prepare payload and add them to the payload
    print("Scan for ruuvitags")
    for ruuvitag in rts.find_ruuvitags(timeout=settings.TIMEOUT):
        id_payload = settings.RUUVITAGS.index(ruuvitag.mac.encode())
        payload = (
            payload
            + bytes([id_payload])
            + pack_temp(ruuvitag.temperature)
            + pack_humid(ruuvitag.humidity)
            + pack("!H", ruuvitag.battery_voltage - 1600)  # battery
            + bytes([ruuvitag.rssi * -1])  # rssi
        )

    print("Setup lorawan")
    node = LoRaWAN(settings.NODE_APP_EUI, settings.NODE_APP_KEY)

    print("Send payload")
    node.send(payload)
    node.shutdown()

    if settings.DEBUG is False:
        print("Enter deepsleep for {} seconds".format(settings.NODE_DEEPSLEEP))
        machine.deepsleep(settings.NODE_DEEPSLEEP * 1000)


if __name__ == "__main__":
    main()
```

Now reset your device and watch the incoming application data on the TTN console.

CHAPTER 8

TTN Payload Format Decoder

On The Things Network Console we can now decode our payload with the following javascript example. Remember the position from each tag from the settings.RUUVITAGS variable, we will give them names in the decoded output.

```
function Decoder(bytes, port) {
  var ruuvitags = {};
  var tagname = "";
  var tags = bytes.length / 5;

  for (i=0;i<tags;i+=1) {
    var temperature = (bytes[1] << 8) | bytes[2];
    var humidity = (bytes[3] << 8) | bytes[4];

    if (bytes[0] === 0) {
      tagname = "stable";
    }
    else if (bytes[0] === 1) {
      tagname = "greenfield";
    }

    ruuvitags[tagname] = {
        "humidity": parseFloat((humidity * 0.0025).toFixed(2)),
        "temperature": parseFloat((temperature * 0.005).toFixed(2))
    };

    bytes.splice(0, 5);
  }

  return ruuvitags;
}
```

1. Find out how many tags are included in the payload. Since we know that one tag use 5 bytes for payload, we divide the payload length with 5.
2. Iterate over the payload and unpack the data. In position zero we find the tag number from our tuple. We do a simple if-else match and give each RuuviTag a name.

3. Add the RuuviTag name with the measurements to the `ruuvitags` object, remove the first 5 bytes from `bytes` and iterate over the next payload data.
4. Return the decoded data.

When you take a look on the TTN Console Device Overview in the data section, you can see the incoming unpacked payload.