LoRaWAN Applications Workshop

Part 2 – Customized LoRa end-device and end-user applications

1 Workshop Goal

To understand the way in a full LoRa network is configured using The Things Network (TTN) services, from LoRa end-devices to the LoRa application server, how LoRa end-devices receive data, the technical limitations of this technology and how own LoRa end-devices firmware and Cloud endpoints communicate through the TTN.

For the second part of the workshop, a single LoRa gateway will be used for all groups. This gateway will be configured by the tutor.

2 Tools

Each group will make use of the following hardware:

- One configurable LoRa end-device supported by RIOT-OS (Board B-L072Z-LRWAN1)
- One PC with
  - RIOT-OS configured with the toolchain for the LoRa end-device,
  - GCC compiler with Paho and JSON-C libraries properly configured

During this workshop a single LoRa Gateway will be provided by the tutor for all groups.

3 Introduction

This document is composed of several sections, each of which provides instructions for different tasks. The tutor will guide you through each one of the tasks during the workshop class.

Each task has an objective, a short theoretical background, a description of the steps to follow, a procedure for verifying that the objective of a section was achieved, and a set of questions to be answered by the students. At the end of each task, a questions and discussion session will be open during the workshop class for cross-verification of the achieved results of each task.

You should not continue to the next section if the verification procedure was not successful.

4 LoRaWAN Uplink Application in RIOT-OS

In part A of this workshop, you were using an example application from RIOT-OS running on the ST board to test the transmission of LoRa frames. The objective of this task is to get familiar with the RIOT-OS LoRa API for end-device transmission, allowing you developing your own LoRa application in the future. Each group of students will develop their own firmware for the same LoRa end-device (ST board).

4.1 Task steps

4.1.1 Project configuration

1. Copy the whole content of the folder "RIOT/examples/lorawan" as a subfolder named "workshop3b" within the folder "RIOT/iot_workshops" (i.e., RIOT/examples/lorawan -> RIOT/iot_workshops/workshop3b)
2. Edit the "Makefile" file (using e.g., nano) and change the values of DEVICEEUI, APPEUI and APPKEY (lines 10-12) as to match in TTN.
3. Execute "`make`" without parameters from the command line, to verify that the Makefile was properly edited.
4. If properly modified, execute "`make flash`" after plugging in your ST board. Verify that the end-device joins your TTN application and sends a frame with the content "This is RIOT!".

You may read more information about configuring your project for LoRa here.

4.1.2 RIOT-OS Application

Take your time to read the whole example and application. If needed, take some time to read the description of the LoRa functions used – enumerated below - of this example here:

- semtech_loramac_send (line 102)
- semtech_loramac_init (line 156)
- semtech_loramac_set_deveui (line 157)
- semtech_loramac_set_appeui (line 158)
- semtech_loramac_set_appkey (line 159)
- semtech_loramac_set_dr (line 162)
- semtech_loramac_join (line 169)

Please modify the example by making the application send a LoRa frame every time the blue button B1 of the board is released (you may reuse the code from previous Workshops). The frame content should include the time duration since the button was pressed encoded as an uint32_t - in 4 bytes - as result of xtimer_usec_from_ticks.

4.2 Verification procedure

Check in the TTN that your LoRa end-device joins and that the messages are received. You may opt for printing out on the ST board console the sent time from your end-device, so you can verify its reception on the TTN live-view console.

4.3 Questions and Analysis

Is the semtech_loramac_send function blocking or non-blocking? Is the semtech_loramac_join function blocking or non-blocking? Attempt to measure the execution time of each function using xtimer. For semtech_loramac_send, does the execution time correspond to the air-time of the device? You may use the LoRa airtime calculator here.

5 MQTT Uplink Client

In part A of this workshop, you were using the Mosquitto command line subscribe utility to read-out uplink LoRa frames from the LoRa Application Server. The objective of this task is to develop your own application to read-out such frames from the LoRa Application Server. We will be using the Paho C Client Library for its implementation. Source files of this library can be located in the "paho.mqtt.c" folder within home.

5.1 Task steps

1. Create your own folder within the home directory. You may name it "workshop3b_mqqt_sub_client", for instance.
2. Copy the file located at "paho.mqtt.c/src/samples/MQTTClient_subscribe.c" within your home directory to the directory created in the step before.
3. Compile the example application using GCC as shown below:

```
gcc -o my_mqtt_client_sub MQTTClient_subscribe.c -lpaho-mqtt3c
```

4. Execute the generated file (my_mqtt_client_sub) from the previous compilation step and verify that it runs properly
5. Modify lines 22-24 with the MQTT connection and subscription parameters used with the Mosquitto subscribe utility in Part A of this workshop. You are free to provide your own CLIENTID string.
6. Insert a new line before line 74 to include your username and password of to the `conn_opts` structure. Check the fields of the structure here.
7. Compile and execute your client as shown in step 3. Verify that your MQTT client receives data from your LoRa end-device.
8. In your C file include the header <json-c/json.h>
9. Follow this example for parsing the incoming JSON message from TTN using JSON-C within the `msgarrvd` function (near line 38). You shall look for the value of the key "uplink_message.frm_payload" in the incoming JSON string.
10. Print the value of this key using an integer 32-bits format.

5.2 Verification procedure

Make sure your application is correctly transmitting your values by checking in the ST board console and on the MQTT client console the printed values.

5.3 Questions and Analysis

1. What is the purpose of the functions MQTTClient_create, MQTTClient_connect and MQTTClient_subscribe? What are their parameters? Are they blocking?
2. Which callbacks can be set in the PAHO API? In which cases are they needed? Provide examples of real-world situations when such callbacks are needed.

6 MQTT Downlink Client

The purpose of this task is to test the PAHO API for downlink message transmission. For this, your PAHO client will publish to the TTN topic for downlink LoRa message transmission.

6.1 Task steps

1. Create your own folder within the home directory. You may name it "workshop3b_mqqt_pub_client", for instance.
2. Copy the file located at "paho.mqtt.c/src/samples/MQTTClient_publish.c" within your home directory to the directory created in the step before.
3. Compile the example application using GCC as shown below:

```
gcc -o my_mqtt_client_pub MQTTClient_publish.c -lpaho-mqtt3c
```

4. Execute the generated file (my_mqtt_client_pub) from the previous compilation step and verify that it runs properly
5. Modify lines 22-25 with the MQTT connection and message to be published that were used with the Mosquitto publish utility in Part A of this workshop. You are free to provide your own CLIENTID string.
6. Insert a new line before line 44 to include your username and password of to the conn_opts structure. Check the fields of the structure here.
7. Compile and execute your client as shown in step 3. Verify that your MQTT client publishes data to the TTN correctly.

6.2 Verification procedure

Same as in Part A Section 9 "MQTT Downlink Client".

6.3 Questions and Analysis

1. What is the purpose of the function MQTTClient_publishMessage? What are its parameters? Is it blocking?

7 MQTT Client Integration and Logic

In this task we will develop a single MQTT client that both subscribes and publishes in a single thread.

7.1 Task steps

The MQTT client shall receive (via MQTT subscribe) upstream measurements from LoRa end-device. Once a new measurement is received, it should store it in internal volatile memory, compute the mean and standard deviation of the last N = 100 values, and send (via MQTT publish) downstream to the reporting LoRa end-device both computed values as truncated uint32, for a total of 8 bytes of payload.

For this, since there is only one MQTT broker used for both services, you shall make use of a single MQTT client, object and connection.

7.2 Verification procedure

Same as in Part A Section 9 “MQTT Downlink Client”.

7.3 Questions and Analysis

N.A.

8 LoRaWAN Downlink Application in RIOT-OS

8.1 Task steps

1. Open the file “RIOT/tests/pkg_semtech-loramac/main.c” in your virtual machine
2. The RIOT application creates a thread for receiving messages
3. Copy and paste the necessary code to your application (RIOT/iot_workshops/workshop3b), adding a new thread.
4. Make sure the new (receiving) thread correctly decodes the payload (two uint32 numbers) and prints out through the console both received values.

8.2 Verification procedure

You may print out in the console of the MQTT client the two transmitted values for each client. This will facilitate the verification of their correct transmission.

8.3 Questions and Analysis

N.A.