Laboratory: IoT Platforms

In this lab, you will learn how to deploy a simple solution capable of transmitting the data collected from sensors to a standardized IoT platform. You will be able to process data and create simple rules as well as friendly dashboards to present this information.

To achieve this, several tasks will presented for you to perform. In a first stage you will be exploring a commercial platform providing wide scale connectivity through LoRaWAN. Then, you are going to approach two of the most used IoT protocols to exchange data with the IoT platform.

Objectives

- Exchange data with The Things Network (TTN) platform thought LoRaWAN;
- Create rules and a simple dashboard to interact with the device using Node-RED;
- Understand Constrained Application Protocol (CoAP) protocol;
- Understand and use MQTT protocol to exchange data between the device and the Node-RED dashboard.

Tasks

I. LoRaWAN

At this part of the lab you will learn how to exchange data with the cloud, namely with The Things Network (TTN) platform, using LoRaWAN. LoRaWAN is a media access control (MAC) protocol for wide area networks. It is designed to allow low-powered devices to communicate with Internet-connected applications over long range wireless connections.

First you will need to get an account at TTN platform, then you will develop software to both send and get data from TTN platform using LoRaWAN. Finally, you will be challenged to develop a dashboard to interact with your device using Node-RED.

A. The Things Network (TTN)

TTN consists on a community that aims to build a global Internet of Things (IoT) using LoRaWAN. After getting an account you will be able to access a console that allows you to manage gateways and applications. Gateways refer to LoRa gateways you may want to register in order to contribute for a larger community and wider coverage. Applications refer to different utilization scopes in which you can register multiple devices.

1. Get a TTN Account

Access TTN console at https://console.thethingsnetwork.org/. You can either create your own account or use the following credentials to access a shared console:

Username: best-iot
Password: bestiot2017

2. Create a TTN Application

At TTN console, add an application. Make sure you use an unique identifier for it to avoid collisions with your colleagues application IDs. Choose the european handler (ttn-handler-eu) at handler registration.

Next, go to “Payload Formats” tab and change the payload format from “Custom” to “CayenneLPP”. This will be later explained in this lab.

3. Register a Device

After creating the application, you will need to register a device in order to get the required credentials for your device to connect to TTN.

In the application created before, go to “Devices” tab and register a device. After registering a device you should have three important codes which you may want to annotate: Device EUI, App EUI and App Key.

At this stage your application should be ready to receive/send data to/from your device. In the next section you will be writing some code to test this. You will be able to check the data being transmitted at the “Data” tab of your application console.

B. Exchanging data through LoRaWAN

You should write an application that will be able to both send and receive a few bytes of information using LoRaWAN. At this phase, some code samples are provided in order to help you understand the basics about both *micropython* syntax and LoRa communication.

1. Structure and configurations

In order to better organize your code, you are advised to have a separated configuration file for holding information such as EUIs, API Keys and WiFi credentials. Create a new directory for your project, and add three empty files to it: *config.py*, *main_lora.py* and *main.py*. The *main.py* file is the script that is executed after *LoPy* boots, the *config.py* is for holding any credentials or configurations, and the *main_lora.py* will have any code regarding LoRaWAN communications.

Write your TNN application and device EUIs and codes at *config.py* as follows:

config.py

```
DEV_EUI = 'XXXXXXXXXXXXXXXX'
APP_KEY = 'XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'
APP_EUI = 'XXXXXXXXXXXXXXXX'
```

In *main.py*, just keep an import of *main_lora.py*:

main.py

```
import main_lora
```

2. Connect to LoRa and send data to TTN

(You can also find the following file in the provided contents)

main_lora.py

```
from network import LoRa
import binascii
import pycom
import socket
import time
import config

# create OTA authentication params
dev_eui = binascii.unhexlify(config.DEV_EUI)
app_eui = binascii.unhexlify(config.APP_EUI)
app_key = binascii.unhexlify(config.APP_KEY)

# disable heartbeat so we can use the LED for feedback
pycom.heartbeat(False)

# turn LED on with red color while not joined
pycom.rgbled(0xff0000)

# Initialize LoRa in LORAWAN mode.
lora = LoRa(mode=LoRa.LORAWAN)

# join a network using OTAA
lora.join(activation=LoRa.OTAA, auth=(dev_eui, app_eui, app_key),
timeout=0)

# wait until the module has joined the network
while not lora.has_joined():
    time.sleep(2.5)
print('Successfuly joined!')

# change LED to green
pycom.rgbled(0x00ff00)

# create a LoRa socket
sock = socket.socket(socket.AF_LORA, socket.SOCK_RAW)

# set the LoRaWAN data rate
sock.setsockopt(socket.SOL_LORA, socket.SO_DR, 5)
```

```
# make the socket blocking
sock.setblocking(False)

time.sleep(1.0)

# change LED to blue
pycom.rgbled(0x0000ff)

for i in range (200):
    # send bytes
    sock.send(b'PKT #' + bytes([i]))

    # Wait some time before trying to receive data
    time.sleep(4)

    # Print data if it is available
    rx = sock.recv(256)
    if rx:
        print(rx)
    time.sleep(6)
```

3. Receive data from TTN

   The code from the previous exercise is also ready to receive information and print it. Send some random bytes from TTN console to check this.

C. Sending sensor data

In this section you should extend the code provided before for sending temperature and humidity values. You must have in consideration that LoRa is designed to transmit small payloads. Thus, values must be encoded for having the lowest possible size, and a decoder must be provided in TTN console for correctly decoding the information. In task A2 of this part of the lab, you were encouraged to select the Cayenne Low Power Payload (LPP) as the payload format in the TTN console. This is a format that considers the payload size restriction and for which TTN already have both encoders and decoders implemented. It is also an accepted format at https://cayenne.mydevices.com, which you can integrate with TTN to build an IoT dashboard[1].

1. Add Cayenne Low Power Payload (LPP) library

   In order for TTN to correctly decode your data using a CayenneLPP decoder, you must encode it in the same format. For this, a library is provided with a CayenneLPP encoder which you can use to send your data.

2. Send both temperature and humidity reads

   With the help of an additional provided module named *utils.py*, add the necessary code to periodically send temperature and humidity reads (use random values for now).

[1] You can read more about this at https://www.thethingsnetwork.org/docs/applications/Cayenne.

D. Create a Dashboard

At this section you are challenged to build a dashboard with Node-RED[2]. You can access Node-RED web interface at the address http://localhost:1880. After creating and deploying your flows for designing your dashboard, you will be able to access it at the address http://localhost:1880/ui.

1. Create *Node-RED* flow and deploy it

Using “ttn message” nodes (that connect directly to TTN), create a flow for building a simple dashboard that shows a gauge with the current value and a chart with the values of the last hour, for both temperature and humidity reads. At the end of this task, you are expected to have a flow like the one depicted on figure 1.

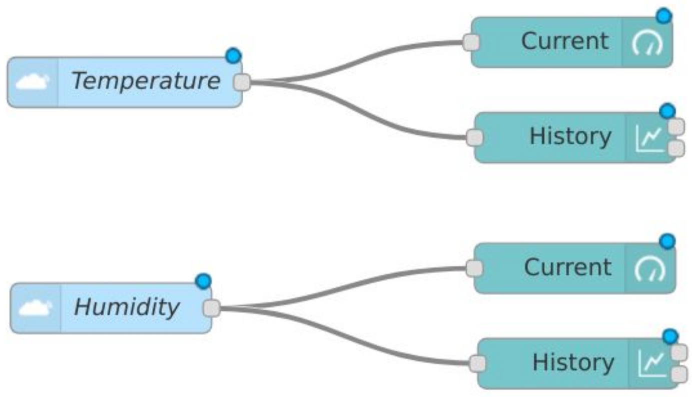

Figure 1: Node-RED flow for task D1.

Suggestion: When creating dashboard nodes, add a new tab named “LoRa”, and put temperature and humidity nodes in different groups.

E. **[Extra]** Interacting with the device

This is an extra task for this part of the lab. Here you are challenged to add a new group of nodes to your dashboard, where you will add inputs for interacting with your device. The following tasks imply that you both build flows at Node-RED and develop software at the device side.

[2] “a programming tool for wiring together hardware devices, APIs and online services in new and interesting ways” [https://nodered.org/]

1. Add a color picker to your dashboard in order to change LED color in your device
2. Add a switch to enable/disable the heartbeat of your device
3. Add a button to make the device pick a random color for its LED
4. Add a switch for enabling/disabling a mode in your device for periodically changing its LED to a random color
5. Add a switch for enabling/disabling a blink mode in your device. The blinking colors must also be chosen in the dashboard
6. Show a notification in dashboard when temperature is higher than a threshold

II. CoAP and MQTT

This part of the lab has the main objective of helping you understand how two of the most used protocols in IoT work. Currently, there is no CoAP implementation for LoPy (or micropython). Thus, that tasks for this protocol will be performed against a dummy device in your computer. Regarding MQTT, the tasks will be similar to what was done in the first part of this lab, where you will build a similar dashboard, but this time using WiFi and MQTT.

A. Exchange data using CoAP protocol

CoAP is a protocol that tries to do for constrained devices, what HTTP does for computers. It follows a request-reply model, and instead of working on top of TCP it makes use of UDP for transporting data. Using CoAP, devices can make their resources available under a URL, which can be accessed using methods such as GET, PUT, POST, and DELETE. Additionally, CoAP introduces an interesting feature named “observe”. When used, this feature allows clients to request updates of a resource when it changes.

1. Run a dummy device

Use the provided python script named *dummy_device.py* for running a simulated device in your computer. This device provides four resources as follows:

Resource	URL	Observe
Temperature	coap://localhost/temperature	yes
Humidity	coap://localhost/humidity	yes
Color	coap://localhost/color	yes
random_color	coap://localhost/settings/random_color	no

2. Interact using *aiocoap* library

Using *aiocoap-client* (already installed in the provided VM), perform the following subtasks:

a) Get the current values for temperature, humidity and color resources;
b) Same as above, but with the observe flag;
c) Change the color (use a string in the payload, e.g. “#00ff00”);
d) Activate the random_color mode (use string with “true” or “false”);

3. Interact using Node-RED

   Using Node-RED, create a simple dashboard (similar to the one created in the first part of this lab) for the available resources in this device. Consider only the temperature and humidity values.

   Suggestion: Add a new tab named “CoAP” to the dashboard

B. Exchange data using MQTT protocol

MQTT is one of the most adopted protocols in the IoT. It is a lightweight messaging protocol that follows a publish-subscribe model for communication between nodes. It requires the existence of a central message broker which is responsible for routing the messages to the corrected nodes.

Publish-subscribe refers to a model where nodes subscribe to a given resource in the broker, and only receive data when another node publishes data in that resource. One of the main advantages of this model is that one node can send data to multiple nodes by just publishing once in the respective resource.

In MQTT, resources are addressed using topics. A topic is a string that is treated as a hierarchy, using a slash (/) as a separator. This hierarchy along with the two wildcards available (“+” and “#”) in subscriptions, give this protocol an increased value. The “+” wildcard allows matching all items at a single level of the hierarchy, while the “#” allows matching all the remaining levels of the hierarchy. As an example, consider the following topics relative to a smart home environment:

1	/sensors/kitchen/temperature/value
2	/sensors/kitchen/humidity/value
3	/sensors/livingroom/temperature/value

When subscribing to these topics individually, you will only receive the messages regarding that specific resource. If you subscribe to the topic “/sensors/kitchen/+/value” (using “+” wildcard), you will receive messages regarding any sensor in the kitchen. Finally, if you subscribe to the topic “/sensors/#” you will receive messages regarding any sensor in the house.

In the following tasks consider using one of the following public MQTT brokers:

Host	**Port**
iot.eclipse.org	1883
test.mosquitto.org	1883

1. Using *Mosquitto* clients

Using the clients provided with *Mosquitto* (mosquitto_sub and mosquitto_pub), test the functionalities of MQTT. Try subscribing a topic and publishing data to it. Then use some wildcards in the subscribed topics in order to test this feature.

2. Publishing temperature and humidity reads from your LoPy device

Create a new file in your project folder named *main_mqtt.py*, and change the code in *main.py* accordingly. Develop an application for periodically publishing random temperature and humidity values to a broker using MQTT. Use the following template for topics:

```
/best-iot/DEVICE_EUI/RESOURCE
```

Replace DEVICE_EUI with the device EUI provided by TTN, and RESOURCE with the corresponding resource (temperature, humidity, etc.).

Use the MQTT library provided (*mqtt.py*). Remember that you need to connect your device to a WiFi network in order to be able to connect to a broker. Thus, consider adding the following new variables to your *config.py* file:

config.py

```
DEV_EUI = 'XXXXXXXXXXXXXXXX'
APP_KEY = 'XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'
APP_EUI = 'XXXXXXXXXXXXXXXX'

WIFI_SSID = 'BEST'
WIFI_PASS = 'bestiot2017'

MQTT_HOST = 'test.mosquitto.org'
MQTT_PORT = 1883
MQTT_USER = None
MQTT_PASS = None
```

3. Create *Node-RED* dashboard to visualize data

Similarly to what was done in the first part of this lab, create a dashboard for displaying the temperature and humidity using Node-RED.

Suggestion: Create a new tab named “MQTT” for holding these new nodes.

C. **[Extra]** Interacting with the device

This extra task consists in the same subtasks of the extra task presented in the first part of this lab, plus one more. However, here you are supposed to use MQTT instead of LoRa. Complete the following subtasks:

1. Add a color picker to your dashboard in order to change LED color in your device
2. Add a switch to enable/disable the heartbeat of your device
3. Add a button to make the device pick a random color for its LED
4. Add a switch for enabling/disabling a mode in your device for periodically changing its LED to a random color
5. Add a switch for enabling/disabling a blink mode in your device. The blinking colors must also be chosen in the dashboard
6. Show a notification in dashboard when temperature is higher than a threshold
7. Add a new graph to your dashboard with the values for both temperature and humidity of your colleagues. Suggestion: use MQTT wildcards.