2 Introduction to TTN and LoRaWAN Simulator

Table of Contents

2.0 Basics of TTN (The Things Network)

This lab will introduce The Things Network (TTN) which acts as a LoRaWAN (Long Range Wide Area Network) server. Parts of TTN is illustrated below in figure 1, but it got much much more functionality in reality (see figure 4 for a TTN overview). Ubidots can be thought of as a special database where all sensor data is stored since TTN normally not save any data, it just passes it further. We will use a virtual sensor node/device as publisher (IoT device) and LoRaWan gateway simulator in the lab.
We are going to write some Python, C# and C/C++ code with MQTT so we can subscribe to the TTN broker. It is basically the same principles as in lab 1 but a bit more complex and closer to reality. We will also write some Unit Tests for our C# code.

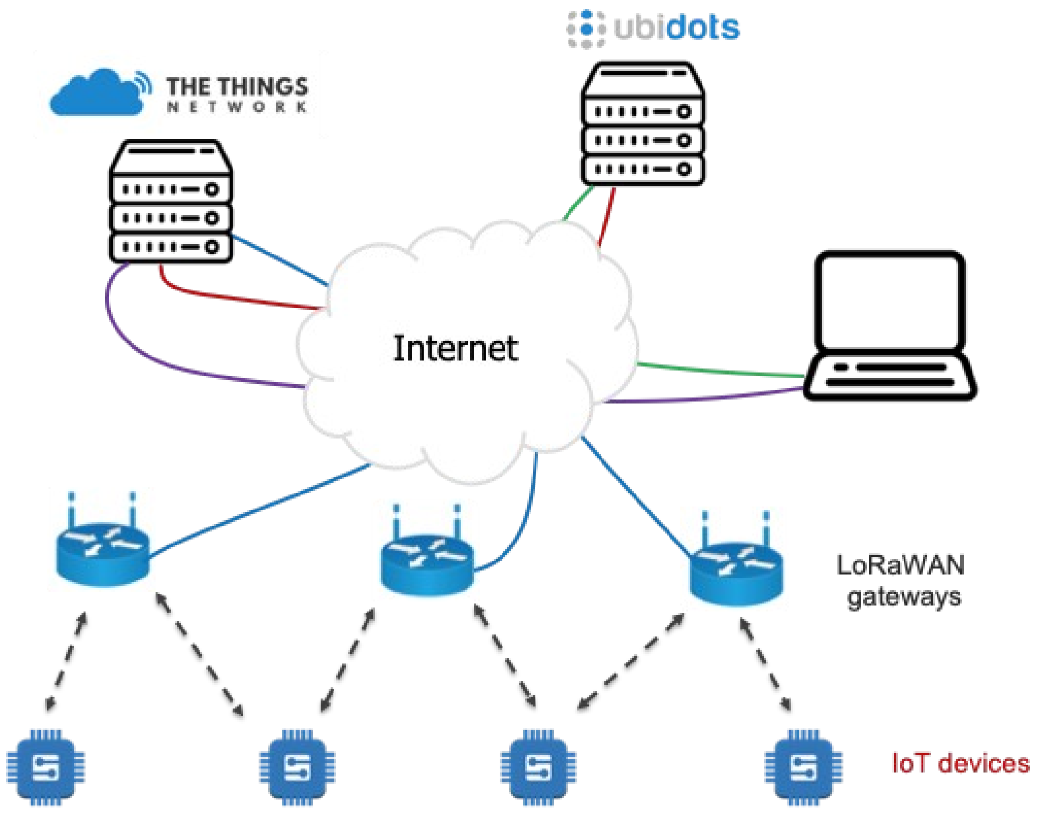


Figure 1. IoT overview, see figure 4 for a TTN overview.

For linked devices with MQTT in TTN the following applies:
Application ID = The ID of the application and also MQTT username
Application Key = The Access Key of the application and also the MQTT password

Requirements
C# compiler, Python, Curl and a web-browser.

To get started look thru this weeks learning topics and links in Learn.

Then follow the two first lessons in: "Getting Started with LoRaWAN" at:
https://www.youtube.com/playlist?list=PLM8eOeiKY7JVwrBYRHxsf9p0VM_dVapXl to get a good understanding how TTN works and a walk-thru for how to solve the following steps.

The instructions in section 2.1 builds on above lessons so you have to watch them when performing the steps. Otherwise you will be lost!

You must register a personal account on TTN which is free.

Arm Mbed also have a good guide at:
https://os.mbed.com/docs/mbed-os/v5.14/tutorials/LoRa-tutorial.html how to
1. add an application
2. register a device (yes it is possible without hardware)

So you get the needed keys:
- Device EUI (Extended Unique Identifiers)
- Application EUI (Extended Unique Identifiers)
- App Key

to perform the 2.1 task.

2.1 LoRaWAN Device and Gateway Simulator

2.1.1 Task

1. With the keys from TTN navigate to: **https://labs.mbed.com/simulator** which acts as both a sensor device and a LoRaWAN gateway.
2. Select LoRaWAN, press the Load demo button and look for the comment //Device credentials, register device as OTAA in The Things Network and copy your credentials here (see step 3)
   Note! If you want to run the simulator (https://github.com/armmbed/mbed-simulator) native on one of your own computers or via Docker, see section 2.1.2.
3. Put in your TTN keys (DEV_EUI, APP_EUI, APP_KEY) into the program code.
4. Press Run and wait for the Arm Mbed OS Simulator (your device) to connect to your application at TTN. This may take several tries before it works. But when successfully have connected it will stick for a while if you do not close the browser.
5. When connected (Connection – Successful will be shown in the Serial output) press the button (center on the electronic green card).

6. If all goes well and you have performed the steps correct you will have sent your first message to TTN and via HTTP Integration to: https://requestbin.com/, as in the "Getting Started with LoRaWAN" videos or: https://www.thethingsnetwork.org/docs/applications/http/ out to an external application.
7. Add the humidity slider to the C/C++ code which is sent in the message. Also add another component, a LED which can change brightness for every sent message to the C/C++ code. See figure 2.

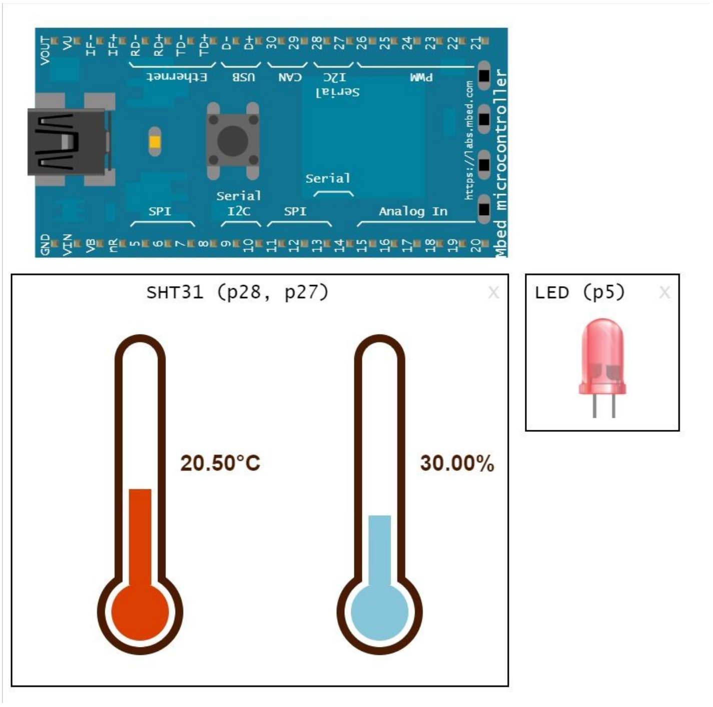
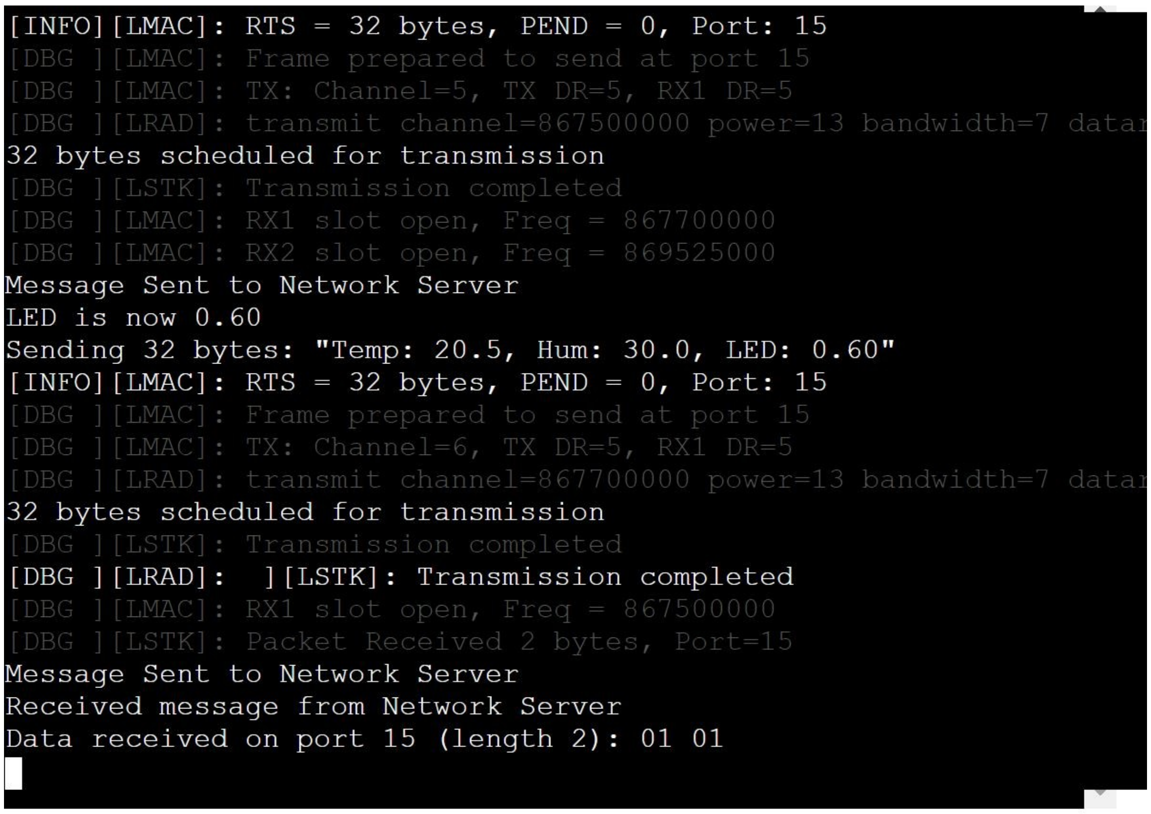


Figure 2. Temperature and humidity slider and a LED light.

When your C/C++ code works with the added components the Serial output should look similar to figure 3 when pressing the button (center on the electronic green card).



```
[INFO][LMAC]: RTS = 32 bytes, PEND = 0, Port: 15
[DBG ][LMAC]: Frame prepared to send at port 15
[DBG ][LMAC]: TX: Channel=5, TX DR=5, RX1 DR=5
[DBG ][LRAD]: transmit channel=867500000 power=13 bandwidth=7 datar
32 bytes scheduled for transmission
[DBG ][LSTK]: Transmission completed
[DBG ][LMAC]: RX1 slot open, Freq = 867700000
[DBG ][LMAC]: RX2 slot open, Freq = 869525000
Message Sent to Network Server
LED is now 0.60
Sending 32 bytes: "Temp: 20.5, Hum: 30.0, LED: 0.60"
[INFO][LMAC]: RTS = 32 bytes, PEND = 0, Port: 15
[DBG ][LMAC]: Frame prepared to send at port 15
[DBG ][LMAC]: TX: Channel=6, TX DR=5, RX1 DR=5
[DBG ][LRAD]: transmit channel=867700000 power=13 bandwidth=7 datar
32 bytes scheduled for transmission
[DBG ][LSTK]: Transmission completed
[DBG ][LRAD]:  ][LSTK]: Transmission completed
[DBG ][LMAC]: RX1 slot open, Freq = 867500000
[DBG ][LSTK]: Packet Received 2 bytes, Port=15
Message Sent to Network Server
Received message from Network Server
Data received on port 15 (length 2): 01 01
```

Figure 3. Serial output.

As illustrated in figure 3 (Data Received on port 15 ...) I have also sent a message to TTN with the attached curl2_downlink.bat. Note that TTN will in turn send my message back downlink to the device **ONLY** next time the device connects with an uplink message.

2.1.2 Other simulator run options

Native

I have corrected and updated the installation instructions somewhat to make it easier here: http://users.du.se/~hjo/cs/gmi28v/resources/Arm%20Mbed%20OS%20simulator/

Running the simulator native on your own computer will make it much faster and easier to run the programs since you can edit the source code in a real editor and compile it yourself. You will also be able to run and monitor your own LoRa Gateway with TTN.

If you select this route. When you start the simulator look for the rows in the console window:
LoRaWAN information:

Gateway ID:	AA:BB:CC
Packet forwarder host:	router.eu.thethings.network
Packet forwarder port:	1700

Make sure the gateway registered in the network server running the *legacy packet forwarder*

When putting in the Gateway ID in the TTN console the resulting ID is:
AABBCC0000000000

Docker
A third option to run the LoRaWAN gateway simulator is with a Docker container as described at the Github page.
I have however not been able to run the container successfully yet. I get a similar problem as stated here: https://medium.com/@stavrosk/mbed-os-simulator-3a59fdae7459 when the demos are about to be compiled: libmbed.bc does not exist. Building…
If someone get the Docker option to work please notify me.

Report:
Show that you have a working implementation of the task above and you can send uplink messages from the device and downlink messages via curl. This can be done at the lab or via screen recording as Windows built-in game recording (Xbox Game Bar). Use "Windows key + Alt + R" to toggle start and stop of the recording.

Alternative recording software: https://atomisystems.com/screencasting/record-screen-windows-10/

2.2 Python MQTT TTN Receiver

Task
Write a Python script that can subscribe to your TTN application and receive messages from your virtual device via the TTN broker and MQTT.

Review the TTN Python documentation at:
https://www.thethingsnetwork.org/docs/applications/python/ to solve the task.

You may reuse some the principles in your Python programs from lab1.

My output from the application when pressing the simulator button from task 2.1

```
Connected to: hajo66-test-one => your_access_key
Waiting for messages...

Received uplink from dev_id: node
Message object: MSG(app_id='hajo66-test-one', dev_id='node',
hardware_serial='008FA7FD523199F8', port=15, counter=25, payload_raw='D2cBEA9ogQ8DACg=',
payload_fields=MSG(analog_out_15=0.4, relative_humidity_15=64.5, temperature_15=27.2),
metadata=MSG(time='2019-10-06T14:18:36.190224345Z', frequency=867.9, modulation='LORA',
data_rate='SF7BW125', airtime=69888000, coding_rate='4/6', gateways=[MSG(gtw_id='eui-
0242ee000084ee70', timestamp=2335051000, time='2019-10-06T14:18:36.182Z', channel=2, rssi=-
35, snr=5, rf_chain=0)], latitude=60.487545, longitude=15.409067, altitude=160,
location_source='registry'))

As JSON: ["hajo66-test-one", "node", "008FA7FD523199F8", 15, 25, "D2cBEA9ogQ8DACg=", [0.4,
64.5, 27.2], ["2019-10-06T14:18:36.190224345Z", 867.9, "LORA", "SF7BW125", 69888000, "4/6",
[["eui-0242ee000084ee70", 2335051000, "2019-10-06T14:18:36.182Z", 2, -35, 5, 0]],
60.487545, 15.409067, 160, "registry"]]
```

Report:
Show that you have a working and correct implementation of the task above. Hand in your source code and a screen recording over mission accomplished.

2.3 C# MQTT TTN Receiver

Task

Write a C# application that can subscribe to your TTN application and receive messages from your virtual device via TTN and MQTT like in task 2.2.

I used a C# help library ALoRa (which is attached): https://github.com/ekwus/ALoRa that sits on-top of the M2Mqtt package: https://m2mqtt.wordpress.com/, but I guess you can use whatever TTN C# code library you want to make it work.

I added AloRA to my project and updated the library with a .NET Core 3.1 compliant version of M2Mqtt as M2MqttDotnetCore: https://www.nuget.org/packages/M2MqttDotnetCore/1.1.0 Adding libraries as source code to your own project may give you some hints how to solve task in general since they usually contains example code or Unit Tests that shows how the library is used.

My output from the application when pressing the button 3 times in figure 2.

```
DotNet-Core 3.1 With The Things Network (TTN) And The C# Library ALoRa

Connected to TTN and listening for MQTT LoRa/LoRaWAN TTN Messages with Virtual Sensor

Press return to exit!

M_client_MqttMsgSubscribed: uPLibrary.Networking.M2Mqtt.MqttClient

GMT Message Timestamp: 2019-10-02T09:16:58, Device: node,
Topic: hajo66-test-one/devices/node/up, Payload: 54-65-6D-70-3A-20-32-30-2E-35-2C-20-48-75-
6D-3A-20-33-30-2E-30-2C-20-4C-45-44-3A-20-30-2E-30-30
Decoded into Temp: 21.5, Hum: 65.0, LED: 0.00
Device location is Latitude: 60.487545, Longitude: 15.409067, Altitude: 160m

GMT Message Timestamp: 2019-10-02T09:20:10, Device: node,
Topic: hajo66-test-one/devices/node/up, Payload: 54-65-6D-70-3A-20-32-30-2E-35-2C-20-48-75-
6D-3A-20-33-30-2E-30-2C-20-4C-45-44-3A-20-30-2E-32-30
Decoded into Temp: 32.8, Hum: 68.2, LED: 0.20
Device location is Latitude: 60.487545, Longitude: 15.409067, Altitude: 160m

GMT Message Timestamp: 2019-10-02T09:20:23, Device: node,
Topic: hajo66-test-one/devices/node/up, Payload: 54-65-6D-70-3A-20-32-30-2E-35-2C-20-48-75-
6D-3A-20-33-30-2E-30-2C-20-4C-45-44-3A-20-30-2E-34-30
Decoded into Temp: 32.8, Hum: 68.2, LED: 0.40
Device location is Latitude: 60.487545, Longitude: 15.409067, Altitude: 160m
...
```

When writing the code for decoding the hex message create a help class for this with static methods. You need a HexToAscii converter for this. Create Unit Tests for your methods and try to use a naming convention for the tests: https://stackoverflow.com/questions/155436/unit-test-naming-best-practices, for example:

- `<method>_Should<expected>_When<condition>`
- `HexToAscii_ShouldConvertValidValues_WhenDecodingInput`
- `HexToAscii_ShouldThrowException_WhenUsingBadInput`
- `...`

So it throws exceptions and can validate a correct hex to ASCII conversion. It is very convenient to create Unit Tests and validate the parsing logic since otherwise you have to wait for messages to arrive. Something that may take a long time if you cannot control the sending interval yourself.

Report:
Show that you have a working and correct implementation of the task above. Also show the Unit Tests for code you have created. Hand in your source code and a screen recording over mission accomplished.

2.4 Lab feedback

a) Were the labs relevant and appropriate and what about length etc?
b) What corrections and/or improvements do you suggest for these labs?

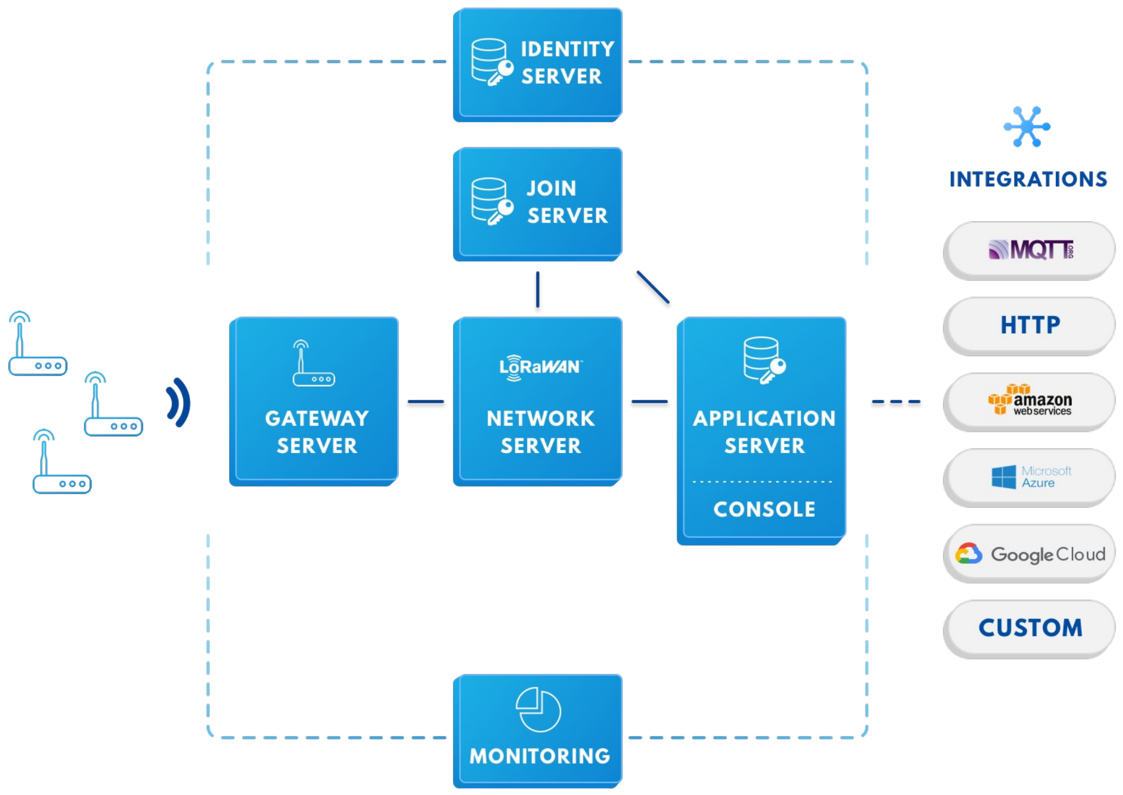


Figure 4. TTN building blocks overview.