EE4500-CNS-2025-BL2 LAB 2 LoRaWAN Node

Course	25-EE4500_EE5510-CNS_TSV-CAM-BL2
Unit	Lab 2
Title	LoRaWAN Node
Version	1.2
Author(s)	Bruce Belson
Date created	1-Jun-2024
Date updated	8-Mar-2025

Contents

Objectives

The purpose of this lab exercise is to set up a LoRaWAN Node using a supplied kit of components.

i. The node kit should be constructed.
ii. Firmware for the Pico should be written to connect the node to The Things Network, as a LoRaWAN end device.
iii. A TTN Application should be created, and the node should be connected to it.
iv. Encrypted data arriving at the TTN application should be decrypted and parsed into an object and packaged as JSON, ready for transmission to the next stage of the pipeline.

Deliverables

There are two target deliverables for the lab: a functioning LoRaWAN node and a TTN application receiving and parsing its data stream.

Deliverable 1: Node

The node is specified below.

1) A fully functioning LoRaWAN node device, based around the following configuration:
    a. Raspberry Pi Pico.
    b. Waveshare SX1303 915M LoRaWAN Gateway HAT.
    c. COM-11120 RGB LED.
    d. NSL-19M51 Light Dependent Resistor.
2) The device should connect to a TTN (The Things Network) application and run the following software:
    a. LoRaWAN Pico firmware stack.
    b. Application based on example `otaa_temperature_led`.
    c. Application also uploads light level detected by the NSL-19M51.
    d. Application also sets the RGB to a colour value and brightness according to bytes 2-4 of downlink messages.

The device should be demonstrated working by monitoring the TTN Application page and by controlling the RGB LED.

Deliverable 2: TTN Application

1) A TTN application should be created as follows:
    a. Application ID & name: cc4950-jc123456 where *123456* is your JC number.
    b. Description: CC4950 Lab Application for [your name]
    c. Add user bbfnq2 to the Application as a collaborator: grant all current and future rights.
2) The device should be connected as follows:
    a. Connected to The Things Network.
    b. Frequency plan: Aus FSB 2
    c. LoRaWAN version: 1.0.3
    d. Join EUI: All zeroes
    e. DevEUI & AppKey: Generated
    f. Device ID: eui-[DevEUI] (Note that DevEUI must be converted to lower case)

Websites

Raspberry Pi Pico	LearnJCU > Reading Resources > Raspberry Pi Pico
Waveshare product site	https://www.waveshare.com/wiki/Pico-LoRa-SX1262
LoRaWAN Pico firmware stack	https://files.waveshare.com/upload/d/d9/Pico-LoRa-SX1262-868M_Code_%281%29.zip
The Things Network	https://www.thethingsnetwork.org/

Procedure

Node construction

Assemble the node, referring to the instructions at the Waveshare website.

Make sure that you put the Pico on to the hat the right way round.

Raspberry Pi Pico configuration

If you do not already have a Pi Pico development environment, install the Pi Pico development tools following the instructions on LearnJCU (See ***Websites*** above).

Check that your build environment is working with a Blinky program (or similar) that also writes to the console.

Download the LoRaWAN Pico firmware stack by copying the zip file. Open the root folder of the expanded zip in VS Code. Set the kit and build the repo, checking whether there are any errors.

Hint: Be sure to start the Pico version of VS Code, or to start it from the Pico - Developer Command Prompt.

You may find an error as follows:

```
lorawan.c:324:36: error: invalid initializer
```

This indicates that the function `make_timeout_time_ms()` has not been defined in `lorawan.c`.

Add the appropriate include file to src/lorawan.c at line 31, immediately following the line containing `#include "pico/lorawan.h"`.

Create Application on The Things Network

- Log in to The Things Network.
- Find your way to the Console and make sure that you're in the Australia 1 cluster.
- Create an Application as specified in the Objectives. Remember to add the lab tutor as a collaborator.
- Add an end device as specified in the Objectives. This corresponds to the Pico node you have just bult. (Refer to the Waveshare documentation – noting that it is not up to date.)

Update Pico Firmware

Using the two values generated on the TTN console, modify the IDs in the Pico `otaa_temperature_led` project's source code. Additionally, correct the `LORAWAN_REGION` value.

Build the project and deliver the .uf2 to the Pico. Use the Serial Monitor inside VS Code or Putty to monitor the console output, as you prefer.

In TTN, observe the ***Live Data*** panel, and watch the encrypted data arriving. You should be seeing new uplink data messages arrive approximately every 30 seconds. Why 30 seconds?

Expand a data message by selecting it. In the ***Event details*** panel, there is a field named `data.uplink_message.frm_payload`. This contains an encrypted representation of the payload, which was constructed in the main loop of the node application, and sent to TTN using `lorawan_send_unconfirmed()`.

Decode data packet

In the TTN application, add a Payload Formatter for the uplink data messages, selecting a `Custom Javascript formatter`. Leave the default formatter function's code unchanged at this point.

Watch the next data message and look for the `data.uplink_message.decoded_payload` field. Compare its value to the value reported in the Pico's console output.

Event details


```
],
"received_at": "2024-08-17T04:36:35.794465749Z",
"uplink_message": {
  "session_key_id": "AZFejYuGtDHN4W7cZKUYbg==",
  "f_port": 2,
  "f_cnt": 41,
  "frm_payload": "Fw==",
  "decoded_payload": {
    "bytes": [
      23
    ]
  },
```


Now make the payload meaningful. Edit the formatter function to return an object containing the raw decoded payload and a meaningful parsed value, e.g.:

```
{
  "bytes": [<contains all input bytes>],
  "internal_temperature": <value of bytes[0] as Number>
}
```

Observe the new contents of the decoded payload field in the TTN ***Live Data Event Details*** panel.

Extend upload data

1. Place the Pico on the breadboard and wire up the light dependent resistor to feed an input voltage into an ADC pin which is not used by the Waveshare hat.
2. Extend the data buffer used to send data from the Pico to the TTN to size 3, and populate it with the temperature as a single byte and the LDR as a pair of bytes following.
3. Update the firmware.
4. Test that the data is being received by the TTN application.
5. Extend the payload formatter to decode the LDR value; compare the values in the TTN console and the VS Code console.

Downlink data

Set up the node to respond to download link data messages as follows.

1) Wire up the RGB LED to be driven by GPIO pins 16, 17 & 18, via an appropriate resistance (e.g. 220 ohm).
2) Update the initialisation code for the Pico to control the pins with PWM.
3) Update the code in the downlink message handler to set the LED values as follows:
    a. On-board LED is on if data[0] is non-zero, off if data[0] is 0. (This is the behaviour of the existing code.)
    b. Red value = data[1] * data[1]
    c. Green value = data[2] * data[2]
    d. Red value = data[3] * data[3]

Hint: Check the length of the received data to avoid bounds errors.

In a real application a separate cloud application would create downlink messages – to be sent to a specific node immediately after the next uplink message is received.

We don't have an upstream application, so we can simulate this, using the TTN Console.

1. Select the ***End Devices*** tab of your TTN Application.
2. Select your device in the list.
3. Select the ***Messaging*** Tab.
4. Select the ***Downlink*** panel.
5. Enter a payload containing 4 bytes (or more) with each byte written as a pair of hex values.
6. Click on ***Schedule Downlink***.

When the next uplink message is received, observe the Pico console output stream, and the behaviour of the RGB and on-board LEDs.

Appendix – Hardware list

For each student:

- Raspberry Pi Pico
- Micro USB cable
- Waveshare Pico-LoRa-SX1262-915M LoRaWAN Pico HAT
- COM-11120 RGB LED
- NSL-19M51 Light Dependent Resistor
- Bread board
- Various resistors

For the lab:

- One or more TTN Gateways