

ELEC423

The Internet of Things: Architecture and Applications

Assignment 2 - Design and Implementation of LoRaWAN-Based IoT Application with ABP Activation for Transmitting and Receiving Messages to/from TTN

Khanthapak Thaipakdee

Department of Electrical Engineering and Electronics,
University of Liverpool,
Brownlow Hill, Liverpool L69 3GJ, UK

This assignment was done on IoT kit.

Table of Contents

	Page
Table of Figures.....	ii
Table of Code Snippets.....	iii
1. Introduction.....	1
2. Description of Each Task.....	2
Task 1.....	2
Task 2.....	4
Task 3.....	4
Task 3. (i).....	5
Task 3. (ii).....	6
Task 3. (iii).....	6
Task 3. (iv).....	6
Task 3. (v).....	6
Task 3. (vi).....	7
Task 3. (vii).....	9
Task 4.....	9
3. Discussion and Conclusion	10
Appendix A: Code	11
Appendix B: Evidence of Work.....	17

Table of Figures

	Page
Figure 1 Screenshot of the IoT application on The Things Network.....	2
Figure 2 Program design.....	3
Figure 3 Screenshot of the end device: Overview tab.....	4
Figure 4 Screenshot of the end device: Payload formatters tab.	4
Figure 5 Error scenario - screenshot of incorrect input.....	5
Figure 6 Capture of the display represents the contents of the messages.	7
Figure 7 TTN console: Live data.	8
Figure 8 TTN console — Live data: event details.	8
Figure 9 Screenshot of receiving messages – (a) downlink and (b) no downlink scenarios.....	9

Table of Code Snippets

	Page
Code Snippet 1.....	4
Code Snippet 2.....	5
Code Snippet 3.....	5
Code Snippet 4.....	6
Code Snippet 5.....	6
Code Snippet 6.....	6
Code Snippet 7.....	7
Code Snippet 8.....	7
Code Snippet 9.....	8
Code Snippet 10.....	9

1. Introduction

Python 3 is an interpreted, open-source, and high-level programming language with versatile applications, supporting both object-oriented and procedural programming. Users can interact with Python 3 through the terminal using the "python3" command, and it offers an interactive shell known as the Python shell for code interpretation. Additionally, Python 3 can be utilized with Integrated Development Environments (IDEs), such as Thonny. Thonny facilitates editing, executing, and debugging Python 3 code, finding applications in the Internet of Things (IoT), web development, GUIs, data analysis, and scripting.

Python supports various data formatting mechanisms, including eXtensible Markup Language (XML), JavaScript Object Notation (JSON), and Cayenne Low Power Payload (LPP) for generating messages at the application layer of the TCP/IP suite. Cayenne LPP stands out for its simplicity in creation, allowing multiple sensor values within a single frame. This format is particularly advantageous for constrained devices and networks due to its reduced payload size. The Cayenne LPP adheres to payload size restrictions, which can be minimized to as low as 11 bytes. The Cayenne LPP frame format comprises data channel, data type, and the actual data, facilitating effective communication.

LoRa (Long Range) is a proprietary physical layer protocol designed by Semtech Corporation, featuring an open MAC layer through the LoRa Alliance. It operates on spread spectrum technology, utilizing three crucial parameters: spreading factor, bandwidth, and coding rate. The LoRa Alliance oversees the Long-Range Wide Area Network (LoRaWAN) protocol, offering connectivity for IoT devices with a focus on security. In LoRaWAN, security is based on two keys—network session key and application session key—effectively separating the network server from the data message.

LoRaWAN accommodates three device classes—Class A, Class B, and Class C—offering flexibility between power consumption and transmissions. Class C is used for mains-powered devices with no latency, enabling the server to send downlink messages anytime.

The Things Network (TTN) is an open, decentralized, and community-driven IoT data network adhering to the LoRaWAN standard and European Telecommunications Standards Institute (ETSI) regulations. TTN enables individuals and organizations to establish and contribute to LoRaWAN networks collectively. The Seed Studio LoRa/GPS HAT serves as a hardware component for TTN, featuring a Semtech SX1276/SX1278 transceiver, an L80 GPS module, and SPI interface, supporting low-power, long-range wireless communication using LoRa modulation.

To configure the Semtech LoRa chipset and create LoRaWAN frames, the PyLoRa module is employed. This module facilitates communication with TTN using two activation mechanisms: Over-the-Air Activation (OTAA) and Activation by Personalization (ABP). OTAA involves a handshake process for secure activation, while ABP doesn't require a handshake but necessitates tracking the frame counter. The Seed Studio LoRa/GPS HAT, integrated with PyLoRa, thus provides a solution for building IoT applications on TTN.

In this assignment, an IoT solution will be crafted using the IoT kit with Raspberry Pi OS. The emphasis lies in employing the LoRa physical layer protocol for long-distance communication, alongside the LoRaWAN protocol executed by TTN. The chosen activation method is ABP, ensuring a secure activation process. Messages will be formatted using Cayenne LPP for efficient sensor data encapsulation. This IoT solution will seamlessly transmit temperature and humidity values from the DHT11 sensor module, along with a specific numerical output, over the LoRaWAN network.

2. Description of Each Task

The program's objective is to establish an IoT application on The Things Network, register an end device for data reception using the ABP activation method, and utilize Cayenne LPP as the payload formatter for uplink messages. Additionally, it requires the creation of a Python 3 script in the working directory to facilitate data exchange with TTN via LoRa/LoRaWAN. The script should

- Prompt the user for information, such as the GPIO pin number in the BCM numbering system for managing the DHT11 module and the frame counter for sending messages
- Include a "DIGIT_SUM" variable containing the digit sum of the student ID
- Gather the current temperature and humidity values from the DHT11 module and ensure their validation
- Crafting a Cayenne LPP-formatted message encompassing the sensor values and integrating the "DIGIT_SUM" as a digital output
- Configure the LoRa modulation for a bit rate of approximately 1760 bits/s (DR3)
- Sent five Cayenne LPP-formatted messages to TTN application, with random frequency selection from the G1 group and a bandwidth of 125 kHz using LoRaWAN ABP activation method
- Handle received messages from TTN. If no downlink messages are received within two times the specified 'desired Rx1 delay' seconds, the script should cease waiting.

Additionally, the IoT solution must align with ETSI EU regulations and TTN policies to maximize the number of messages sent per day.

The program design is illustrated in Figure 2. In the task of sending five Cayenne LPP-formatted messages to TTN application, the "number_sender" variable is employed to monitor the count of uplink messages. In the absence of downlink messages, the terminal will show "No downlink message." Conversely, if a downlink message is received, it will be displayed on the terminal.

Task 1

To develop an IoT application on The Things Network, begin by logging into TTN Console based in the EU via <https://eu1.cloud.thethings.network/console/>. Navigate to "Go to applications," where you'll enter the application ID as "rp-dht11-iot" and proceed to click the "Create application" button. The resulting IoT application interface will resemble the layout depicted in Figure 1.

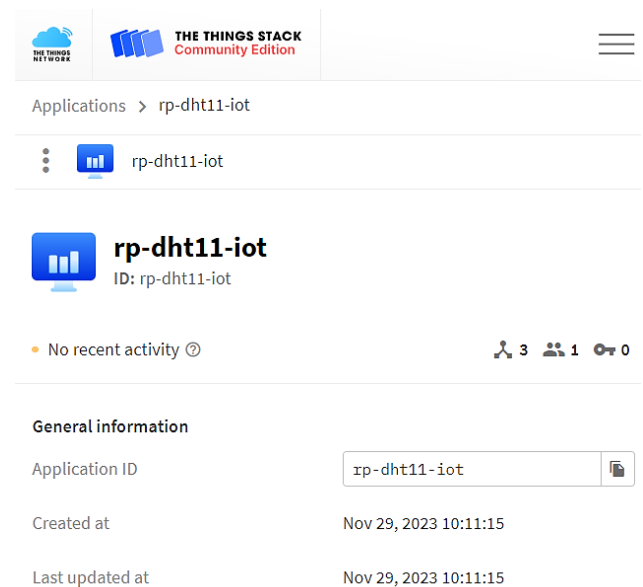


Figure 1 Screenshot of the IoT application on The Things Network.

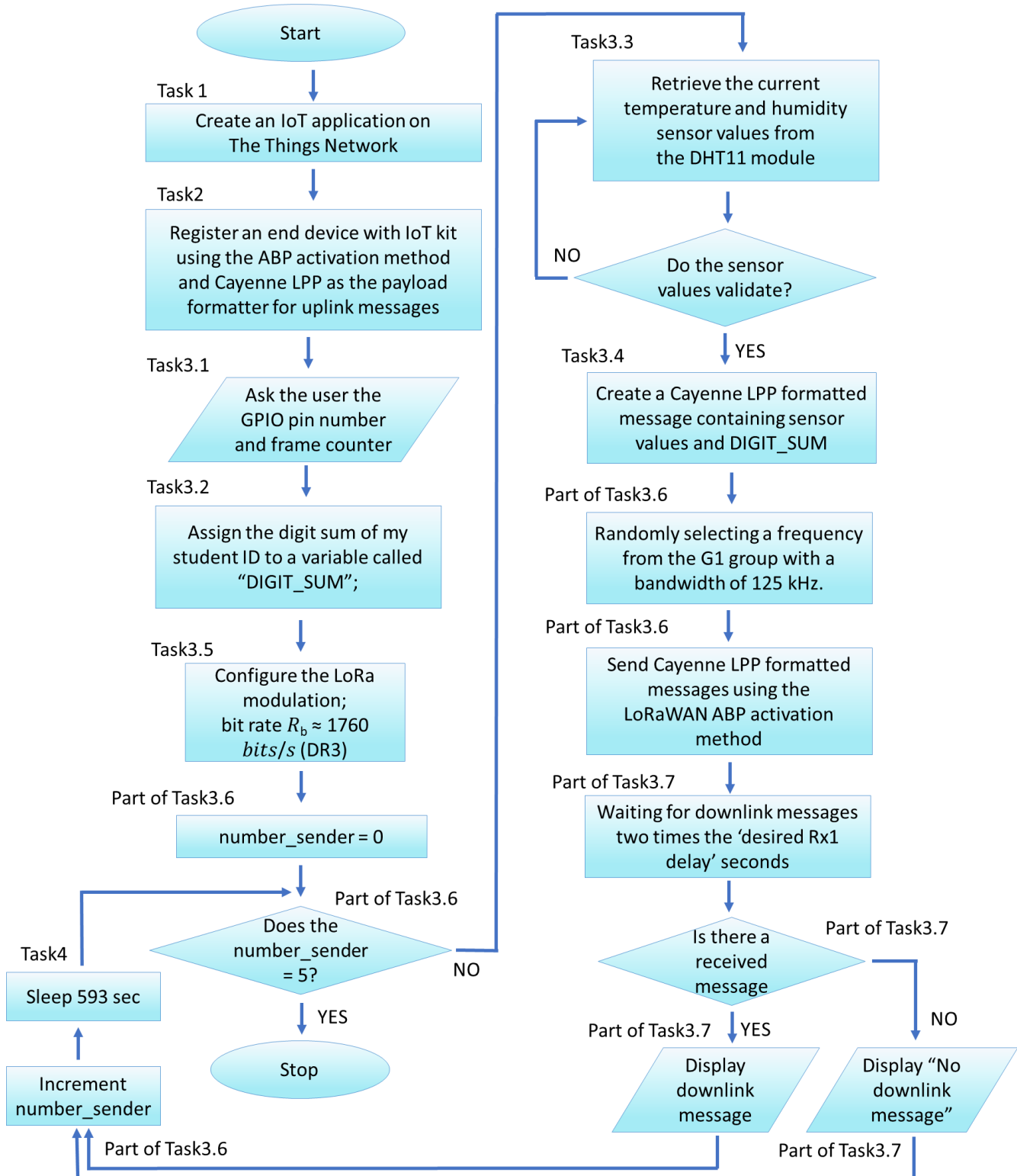


Figure 2 Program design.

Task 2

To enrol an end device for receiving data from the IoT kit via LoRa/LoRaWAN using the ABP activation method, start by accessing the IoT application established in Task 1. Navigate to the "End device" menu and proceed to click on the "Register end device" button. Choose manual registration by selecting "Enter end device specifics manually." For the frequency plan, choose "Europe 863-870 MHz (SF9 for RX2 - recommended)," and opt for "LoRaWAN Specification 1.0.0." Click on "Show advanced activation, LoRaWAN class, and cluster settings."

In the Activation mode, pick "Activation by personalization (ABP)" and set LoRaWAN class to "Class C (Continuous)" since the device is continuously listening for messages, allowing the server to send downlink messages at any time. Click "Generate" for DevEUI, Device address, AppSKey, and NwksKey. Finally, click "Register end device" to complete the process. The resulting layout for the end device should resemble Figure 3.

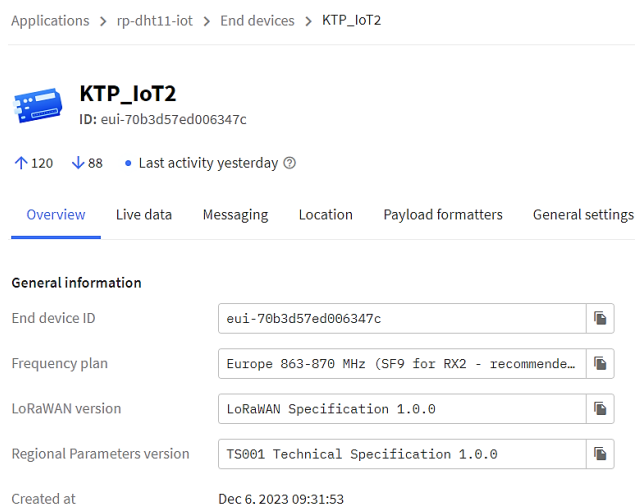


Figure 3 Screenshot of the end device: Overview tab.

To configure Cayenne LPP as the payload formatter for uplink messages, navigate to the "Payload formatters" tab, click on "uplink," and then select "CayenneLPP" as the formatter type, as illustrated in Figure 4.

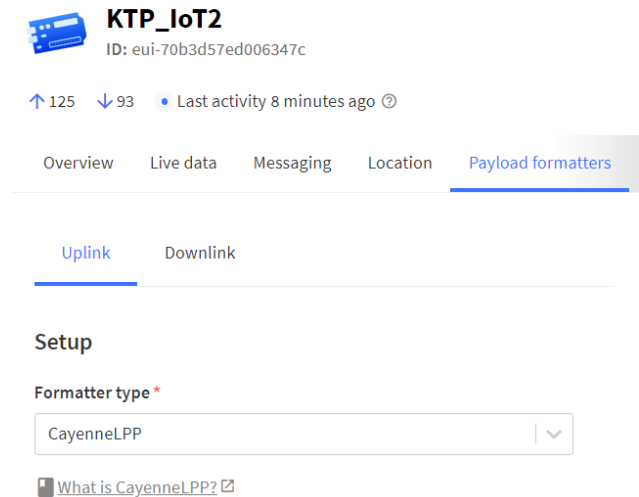


Figure 4 Screenshot of the end device: Payload formatters tab.

Task 3

This task requires a Python 3 script with specifications for sending and receiving data to/from TTN via LoRa/LoRaWAN. To develop the Python 3 script, navigate to the working directory, "LoRaWan_TTN", using the "cd" command. Subsequently, employ the "thorny" command to open Thonny for editing, executing, and debugging, as depicted in Code snippet 1.

```
~ $ cd LoRaWan_TTN
~/LoRaWan_TTN $ thorny tx_rx_assignment2.py
```

Code Snippet 1

The imports include the "LoRaWANtxrx" class for handling LoRaWAN communication, the "sleep" function for introducing delays, the "LoRa" class and "board" configuration from the "SX127x" module, "RPi.GPIO" for Raspberry Pi

GPIO handling, “sys” for system-related functionalities, and additional modules for Cayenne LPP and DHT11 sensor interaction (refers to Code Snippet 2).

```
from LoRaWAN_TTN import LoRaWANTxrx
from time import sleep
from SX127x.LoRa import *
from SX127x.board_config import BOARD

import RPi.GPIO as GPIO
import sys

from cayennelpp import LppFrame, LppUtil
import dht11

import random
```

Code Snippet 2

Task 3. (i)

This task involves gathering terminal input, particularly the GPIO pin number (BCM numbering system) for managing the DHT11 module. Additionally, the solution mandates the frame counter for sending messages. The "sys" module is utilized to extract input values from attributes separated by a space, with the "argv" list object holding this input, where argv[0] designates the script name, argv[1] represents the pin number, and argv[2] denotes the frame counter. The received input type is "str," so the "int" function is applied to convert the string into an integer number. A try-except block handle potential errors, particularly in cases like converting "23.5" to an integer. It triggers an alert stating "There was an exception of type" followed by type of error and “Error: expected an integer” then terminating the script with "sys.exit(2)" (refer to Code Snippet 3).

For DHT11 management, pin 26 is the correct input in BCM numbering. An if-else statement is used to verify that the user inputs the correct pin number; otherwise, it triggers an alert stating "Invalid pin number" and terminates the script using "sys.exit(1)."

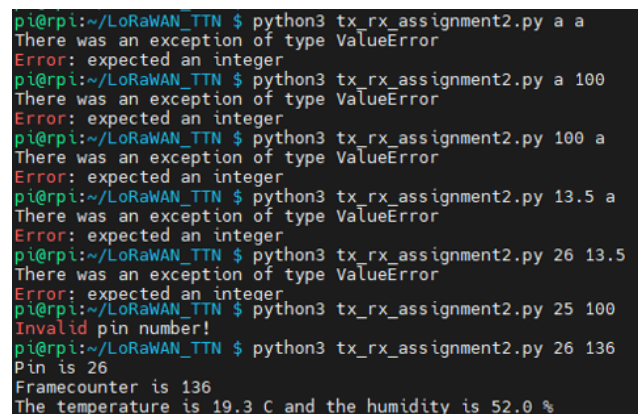
Figure 5 illustrates error scenarios, detecting "ValueError" if 'a' or '13.5' is used as the pin number or frame counter. Additionally, if the user inputs any integer other than 26 as the pin number, it prompts "Invalid pin number!" and terminates to prevent DHT11 sensor module errors.

```
# Retrieve input from the terminal (Task 3(i))
try:
    #Ask user the pin number
    pin = int(sys.argv[1])
    #Ask user the frame counter
    framecounter = int(sys.argv[2])

except Exception as e:
    print("There was an exception of type",
          e.__class__.__name__, "Error: expected an
          integer")
    sys.exit(2) #sys.exit(2) refers to incorrect
               type of input

if pin==26:
    # Access the GPIOs using the BCM
    numbering system
    GPIO.setmode(GPIO.BCM)
    print("Pin is",pin)
    print("Framecounter is",framecounter)
else:
    print("Invalid pin number!")
    sys.exit(1) #sys.exit(1) refers to invalid pin
               number for DHT11 module
```

Code Snippet 3



```
pi@rpi:~/LoRaWAN_TTN $ python3 tx_rx_assignment2.py a a
There was an exception of type ValueError
Error: expected an integer
pi@rpi:~/LoRaWAN_TTN $ python3 tx_rx_assignment2.py a 100
There was an exception of type ValueError
Error: expected an integer
pi@rpi:~/LoRaWAN_TTN $ python3 tx_rx_assignment2.py 100 a
There was an exception of type ValueError
Error: expected an integer
pi@rpi:~/LoRaWAN_TTN $ python3 tx_rx_assignment2.py 13.5 a
There was an exception of type ValueError
Error: expected an integer
pi@rpi:~/LoRaWAN_TTN $ python3 tx_rx_assignment2.py 26 13.5
There was an exception of type ValueError
Error: expected an integer
pi@rpi:~/LoRaWAN_TTN $ python3 tx_rx_assignment2.py 25 100
Invalid pin number!
pi@rpi:~/LoRaWAN_TTN $ python3 tx_rx_assignment2.py 26 136
Framecounter is 136
The temperature is 19.3 C and the humidity is 52.0 %
```

Figure 5 Error scenario - screenshot of incorrect input.

Task 3. (ii)

This task required calculating the sum of my student ID manually, and subsequently, the variable "DIGIT_SUM" was assigned the computed value, as depicted in Code Snippet 4.

```
DIGIT_SUM=25
```

Code Snippet 4

Task 3. (iii)

This task (refer to Code Snippet 5) initializes the DHT11 sensor module with a specified GPIO pin number and enters an infinite loop to continuously read temperature and humidity data. Within the loop, the code checks the validity of the retrieved data using the "is_valid()" method. If the data is valid, it extracts the temperature and humidity values with the "get_temperature()" and "get_humidity()" methods, respectively. The obtained values are then printed to the console. The loop continues until valid data is retrieved, and the break statement is used to exit the loop.

```
# Task 3. (iii)
# Initial the DHT11 module
dht11_sensor_module = dht11.DHT11(pin)

# Retrieve the data from the sensors (check that data is valid)
while True:
    data = dht11_sensor_module.read()
    # Validate the data retrieved
    if data.is_valid():
        # Extract the temperature and humidity values
        temperature = data.get_temperature()
        # Celsius
        humidity = data.get_humidity() # %
        print("The temperature is",temperature, "C and the humidity is", humidity,"%")
        break
```

Code Snippet 5

Task 3. (iv)

In Code Snippet 6, a Cayenne LPP message is constructed. The code initializes an empty frame using the "LppFrame" class and adds "temperature" variable, "humidity" variable, and "DIGIT_SUM" variable to designated channels. "temperature" is in Celsius (channel 1), "humidity" is a percentage (channel 2), and "DIGIT_SUM" serves as a digital output (channel 3). The frame is then converted into a Cayenne LPP formatted byte sequence, resulting in the variable "msg_cayenne_lpp". This concise and structured message encapsulates essential sensor data and digital output, suitable for transmission.

```
# Create a Cayenne LPP formatted message (Task 3.(iv))

# Create an empty frame
frame =LppFrame()

# Add the temperature, humidity values and DIGIT_SUM to the frame

frame.add_temperature(1,temperature)
frame.add_humidity(2, humidity)
frame.add_digital_output(3,DIGIT_SUM)

# Create the message in CayenneLPP format
msg_cayenne_lpp = bytes(frame)
```

Code Snippet 6

Task 3. (v)

In this task (refers to Code Snippet 7), the LoRa modulation parameters are configured for RX1. The configuration is based on Lecture 15 specifications, where the bit rate (R_b) is set to 1760 bits/s, Spreading Factor (SF) is set to 9, and the channel bandwidth is 125 kHz. The coding rate (CR) is determined from the formula $R_b = SF * \frac{BW}{2^{SF}} * \frac{4}{CR+4}$, and in this case, CR is set to 1.

The code utilizes the “set_coding_rate()”, “set_bw()”, and “set_spreading_factor()” functions to set the coding rate to 1, bandwidth to 125 kHz, and spreading factor to 9, respectively.

```
# Configure LoRa for RX1 (Task 3. (v))

# Rb (bits/s) = 1760, Configuration =SF9/125kHz
Data Rate Mode = DR3

# Set the coding rate
lora.set_coding_rate(1)

# Set the bandwidth : 7 for 125 kHz
lora.set_bw(7)

# Set the spreading factor = 9
lora.set_spreading_factor(9)
```

Code Snippet 7

Task 3. (vi)

In this task (refers to Code Snippet 8), the LoRaWAN parameters are pre-configured for ABP. The device address (devaddr), network session key (nwkskey), and application session key (appskey) are set to specific values. The LoRaWANTxrx object is created, and the module is configured for both transmission (TX) and reception (RX). The code sets up the LoRaWAN module for TX with ABP, specifying the device address and keys.

```
# (Part of Task 3. (vi))
# Pre-configured details for ABP

# Create the LoRaWANTxrx object
lora = LoRaWANTxrx(False)

# Configure LoRaWAN for TX with ABP
lora.set_devaddr(devaddr)
lora.set_nwkskey(nwkskey)
lora.set_appskey(appskey)
```

Code Snippet 8

The script (refers to Code Snippet 9) then enters a loop to send five messages with a randomly chosen frequency from the G1 group (868.1, 868.3, 868.5 MHz) and a bandwidth of 125 kHz. The frame counter is updated for each message. The code initiates the transmission, waits for completion, and updates the counter accordingly. The process is repeated, updating new values of the temperature and humidity, until five messages are sent. The sleep function introduces a delay, which will be discussed later in this report. Overall, this code snippet manages the configuration and transmission of LoRaWAN messages with ABP.

Figure 6 represents the terminal output from tasks 3. (i), and 3. (iii-vi). It displays the corrected pin number and current frame counter. Additionally, it exhibits the retrieved temperature and humidity values, along with the random frequency from the G1 group and the Cayenne LPP-formatted byte sequence.



```
pi@rpi:~/LoRaWAN_TTN $ python3 tx_rx_assignment2.py 26 111
Pin is 26
Framecounter is 111
The temperature is 21.9 C and the humidity is 38.0 %
The frequency is 868.1

b'\x01g\x00\xdb\x02hI\x03\x01\x19'
Sending LoRaWAN message

TxDone
Receiving LoRaWAN message in RX1

Downlink message is 9677434aecace466bd0fb517c41f96141db8b
ede283d7111bdb08c96470eeee22040868f0742

The temperature is 22.5 C and the humidity is 43.0 %
The frequency is 868.1

b'\x01g\x00\xe1\x02hV\x03\x01\x19'
Sending LoRaWAN message

TxDone
Receiving LoRaWAN message in RX1

Downlink message is 9677434aecace466bd0fb517c41f96141db8b
ede283d7111bdb08c96470eeee22040868f0742

The temperature is 22.6 C and the humidity is 40.0 %
The frequency is 868.5

b'\x01g\x00\xe2\x02hP\x03\x01\x19'
Sending LoRaWAN message

TxDone
Receiving LoRaWAN message in RX1

Downlink message is 9677434aecace466bd0fb517c41f96141db8b
ede283d7111bdb08c96470eeee22040868f0742
```

Figure 6 Capture of the display represents the contents of the messages.

```

# variable use to count number of sending the
messages
number_sender = 0;
while (number_sender < 5):

    # Task 3. (ii, iii, iv)
    msg_cayenne_lpp =
    retrieve_sensor_Cayenne(pin,DIGIT_SUM)

    # (Part of Task 3.(vi))
    # Create array of G1 frequency from the G1
    group with a bandwidth of 125 kHz
    freqG1=[868.1,868.3,868.5]
    freqrandm=random.choice(freqG1)
    print("The frequency is",freqrandm)
    # Set the channel frequency
    lora.set_freq(freqrandm)

    # Set the frame counter
    lora.set_tx_fcnt(number_sender +
    framecounter)
    # Create a frame by setting the payload
    lora.set_tx_data(list((msg_cayenne_lpp)))

    try:
        print("Sending LoRaWAN message\n")
        lora.start()
        while True:
            sleep(.1)
            if lora.has_tx_done() == True:
                print("TxDone")
                break
            # receiving message part
        finally:
            sys.stdout.flush()
            lora.set_mode(MODE.SLEEP)
            sleep(593)

    # Update number of sending messages
    # Use this to update frame counter
    number_sender = number_sender + 1;

```

Code Snippet 9

From Figure 7, the uplink message is displayed in the TTN live data. Additionally, in the TTN payload section, the decoded payload will be shown, where channel 1 represents temperature, channel 2 represents humidity, and channel 3 represents the digital output. If the TTN console does not display the decoded payload, revisit the settings made during Task 2. Also, if the sensor values or the "DIGIT_SUM" appear incorrect, review the "retrieve_sensor_Cayenne" function. Also, in Figure 8, you can verify the correctness of Spreading Factor (SF), bandwidth, Coding Rate (CR), and random frequency.

↓ 13:54:28	MAC payload:	27 70 FE 6C A1 18 5C 84 C7 D6 76 F3 B1 3F 11 47 7D D9 76 9F 3A 75 BD AC
↑ 13:54:28	Payload:	{ digital_out_3: 25, relative_humidity_2: 42, temperature_1: 22.7 }
↑ 13:54:28		
↓ 13:44:25	MAC payload:	26 E2 0B AA FD 7D FF 32 D2 46 2D 8B C2 B4 54 8B C5 A2 C6 E7 27 99 BE 41
↑ 13:44:25	Payload:	{ digital_out_3: 25, relative_humidity_2: 44, temperature_1: 21.6 }
↑ 13:44:25		
↓ 13:34:21	MAC payload:	BE 36 31 58 96 32 15 4C E8 C5 58 E6 29 36 FA 99 55 A9 76 24 4E 6E C6 79
↑ 13:34:21	Payload:	{ digital_out_3: 25, relative_humidity_2: 43, temperature_1: 22.6 }
↑ 13:34:21		
↓ 13:24:18	MAC payload:	38 F0 BE 0E F3 E7 DC 92 6C 97 22 8B C9 0C D4 7A 6C D0 57 7D 6C CC 19 34
↑ 13:24:18	Payload:	{ digital_out_3: 25, relative_humidity_2: 43, temperature_1: 21.7 }
↑ 13:24:18		
↓ 13:14:14	MAC payload:	FB 2A D8 B0 0D 85 5A 58 A2 6D E4 2D 21 A4 78 08 D2 13 48 DC 37 D2 FD 03
↑ 13:14:14	Payload:	{ digital_out_3: 25, relative_humidity_2: 41, temperature_1: 22.3 }

Figure 7 TTN console: Live data.

```

54  "settings": {
55      "data_rate": {
56          "lora": {
57              "bandwidth": 125000,
58              "spreading_factor": 9,
59              "coding_rate": "4/5"
60          }
61      },
62      "frequency": "868300000",
63      "timestamp": 639491932,
64      "time": "2023-12-15T13:54:27.702721118Z"
65  },
66  "received_at": "2023-12-15T13:54:28.244305285Z",
67  "consumed_airtime": "0.205824s",

```

Figure 8 TTN console — Live data: event details.

Task 3. (vii)

The task (refers to Code Snippet 10) is designed to handle the reception of LoRaWAN messages in RX1. It initiates the reception process, introducing a delay equivalent to two times the ‘desired Rx1 delay’ seconds using the “sleep(10)” function. The code then checks for received messages, and if a downlink message is available, it prints the message. In case no downlink message is received within the specified time, it outputs “No downlink message”. Figure 9 shows a screenshot of receiving messages – (a) downlink and (b) no downlink scenarios.

```
try:
    # Sending message part

    print("Receiving LoRaWAN message in
    RX1\n")

    # Handle received messages from the
    TTN and (Task 3. (vii))
    # Wait until two times the 'desired Rx1
    delay' seconds

    while True:
        sleep(10)
        if lora.has_rx_done() == True:
            print("Downlink message is
            ",lora.get_rx_data_bytes(),"\n\n")
            break
        else:
            print("No downlink message\n\n")
            break
    finally:
        sys.stdout.flush()
        lora.set_mode(MODE.SLEEP)
```

Code Snippet 10

(a)

```
TxDone
Receiving LoRaWAN message in RX1

Downlink message is 9677434aecace466bd0fb517c41f96141db8b
ede283d7111bdb08c96470eeee22040868f0742
```

(b)

```
TxDone
Receiving LoRaWAN message in RX1

No downlink message
```

Figure 9 Screenshot of receiving messages – (a) downlink and (b) no downlink scenarios.

Task 4

The solution demonstrates compliance with both ETSI EU regulations and TTN policies regarding LoRaWAN uplink transmissions. According to ETSI EU regulations, the duty cycle (DC) for each of the eight frequencies in the EU is set at 1%. The specific frequencies allowed for uplink messages are 867.1, 867.3, 867.5, 867.7, 867.9, 868.1, 868.3, and 868.5 MHz. To align with these regulations, the solution randomly selects frequencies from the list [868.1, 868.3, 868.5].

Figure 7 provides evidence of this task, showcasing the compliance with TTN policies specifying a maximum uplink Time on Air (ToA) of 30 seconds per day per device. The solution accurately calculates the transmit time interval (T_{interval}) using the formula $T_{\text{interval}} = (\text{ToA} / \text{DC}) - \text{ToA}$. The specific ToA value of 0.205824 seconds is derived from the TTN console's Live Data: event details (Figure 8). With the TTN's DC approximately at 0.035% (precisely 0.034722222%), the calculated T_{interval} is approximately 592.56 seconds, translating to around 10 minutes. To align with these specifications, the script incorporates “sleep(593)” to effectively manage LoRaWAN uplink transmissions within the defined constraints.

3. Discussion and Conclusion

The design and implementation of the LoRaWAN-based IoT application presented in this project involve several key components and considerations. The application uses the Activation by Personalization method, streamlining communication between end devices and The Things Network without the need for a handshake. This simplifies the device setup process but requires keeping track of the frame counter. Additionally, in this solution, there is a check for an integer frame counter from the user; although it cannot confirm that the user's input matches the current frame counter.

Cayenne Low Power Payload (LPP) serves as the chosen payload formatter for uplink messages, offering a standardized format for sensor data. The application retrieves temperature and humidity values from the DHT11 module, validating the data for accuracy. Therefore, in the working directory, “dht11.py” has to be present. Additionally, the script incorporates a digital output, namely the “DIGIT_SUM”, to be included in the Cayenne LPP-formatted message. The payload is then transmitted via LoRa/LoRaWAN with a configured bit rate of approximately 1760 bits/s and adheres to ETSI EU regulations and TTN policies.

To assess the compliance with TTN policies, the application adheres to the specified duty cycle, ensuring that the transmission time interval stays within the allowable limits. The transmit time interval calculation considers the maximum uplink Time on Air (ToA) of 30 seconds per day per device, as specified by TTN policies.

The Seed Studio LoRa/GPS HAT is a crucial hardware component for TTN, featuring a Semtech SX1276/SX1278 transceiver, an L80 GPS module, and an SPI interface. This HAT enables low-power, long-range wireless communication using LoRa modulation. The

Semtech transceiver ensures reliable data transmission, while the L80 GPS module provides location information. The SPI interface facilitates seamless integration with compatible devices, allowing for straightforward communication and control.

In conclusion, this project successfully creates a functional LoRaWAN-based IoT application with ABP activation, Cayenne LPP payload formatting, and adherence to regulatory and policy requirements. The implemented Python script effectively handles data exchange with TTN, showcasing the capabilities of LoRaWAN technology in IoT applications. The learning experience includes using the terminal and Thonny IDE for Python3 scripting, working with the DHT11 module, understanding the Seed Studio LoRa/GPS HAT, and gaining insights into LoRa, LoRaWAN, and TTN workings.

Appendix A: Code

```
#!/usr/bin/env python3

from LoRaWAN_TTN import LoRaWANtxrx
from time import sleep
from SX127x.LoRa import *
from SX127x.board_config import BOARD

# Import the RPi module to work with the GPIOs
import RPi.GPIO as GPIO
# Import the DHT11 module to manage the sensor module
import dht11
# Import the system module
import sys

# Import the Cayenne LPP as the payload formatter for uplink messages
from cayennelpp import LppFrame, LppUtil

# Import random class
import random

# The BOARD.setup() function is a method used to initialize the board configuration
# for the SX127x LoRa module.
GPIO.setwarnings(False)
BOARD.setup()
```



```

def retrieve_sensor_Cayenne(pin,DIGIT_SUM):

    # Initialise the DHT11 sensor module (Task 3. (iii))
    dht11_sensor_module = dht11.DHT11(pin)

    # Retrieve the data from the sensors (check that data is valid)
    while True:
        data = dht11_sensor_module.read()
        # Validate the data retrieved
        if data.is_valid():
            # Extract the temperature and humidity values
            temperature = data.get_temperature() # Celsius
            humidity = data.get_humidity() # %
            print("The temperature is",temperature, "C and the humidity is", humidity,"%")
            break

    # Create a Cayenne LPP formatted message (Task 3. (iv))

    # Create an empty frame
    frame =LppFrame()

    # Add the temperature, humidity values and DIGIT_SUM to the frame
    frame.add_temperature(1, temperature) # Celsius
    frame.add_humidity(2, humidity)# %
    frame.add_digital_output(3, DIGIT_SUM)# DIGIT_SUM

    # Create the message in CayenneLPP format
    msg_cayenne_lpp = bytes(frame)
    return msg_cayenne_lpp

```

```

def main ():

    # Retrieve input from the terminal (Task 3. (i))
    try:
        #Ask user the pin
        pin = int(sys.argv[1])
        #Ask user the frame counter
        framecounter = int(sys.argv[2])

    except Exception as e:
        print("There was an exception of type", e.__class__.__name__,"\nError: expected an integer")
        sys.exit(2) #sys.exit(2) refers to incorrect type of input

    if pin==26:
        # Access the GPIOs using the BCM numbering system
        GPIO.setmode(GPIO.BCM)
        print("Pin is",pin)
        print("Framecounter is", framecounter)
    else:
        print("Invalid pin number!")
        sys.exit(1) #sys.exit(1) refers to invalid pin number for DHT11 module

    # Manually calculate the digit sum of my student ID (Task 3. (ii))
    DIGIT_SUM = 25

    # Pre-configured details for ABP (Part of Task 3. (vi))
    devaddr = [0x26, 0x0B, 0xCA, 0x05]
    nwkskey = [0x16, 0xA7, 0x91, 0x5B, 0x09, 0x67, 0x8C, 0x2B, 0xF9, 0xC9, 0x0F, 0xFD, 0xE9, 0xFB, 0xD4, 0x7F]
    appskey = [0xD4, 0x80, 0xC9, 0xA9, 0xAC, 0x94, 0xD8, 0x20, 0x60, 0xF1, 0x57, 0xFE, 0x98, 0x93, 0x3D, 0xDC]

    # Create the LoRaWANtxrx object
    lora = LoRaWANtxrx(False)

    # Configure the module for TX
    lora.set_mode(MODE.SLEEP)
    lora.set_dio_mapping([1,0,0,0,0])
    lora.set_pa_config(pa_select=1, max_power=0x0F, output_power=0x0E)

    # Configure the module for RX
    lora.set_sync_word(0x34)

```

```
# From Lecture 15,  $R_b$  (bits/s) = 1760, Configuration = SF9/125kHz,  
# Data Rate Mode = DR3  
# SF9 refers to Spreading factor = 9,  
# The channel bandwidth is 125 kHz  
#  $R_b = SF * BW / \text{chips} * 4 / (CR+4)$   
# We can find CR from above equation and  $CR = 1$ 
```

```
# Configure LoRa for RX1 (Part of Task 3. (v))
```

```
# Enable CRC  
lora.set_rx_crc(True)
```

```
# Set the coding rate  
lora.set_coding_rate(1)
```

```
# Set the bandwidth : 7 for 125 kHz  
lora.set_bw(7)
```

```
# Set the spreading factor = 9  
lora.set_spreading_factor(9)
```

```
# Configure LoRaWAN for TX with ABP (Part of Task 3. (vi))  
lora.set_devaddr(devaddr)  
lora.set_nwkskey(nwkskey)  
lora.set_appskey(appskey)
```

```

# variable use to count number of sending the messages
number_sender = 0;

while (number_sender < 5):

    # Task 3.(ii,iii,iv)
    msg_cayenne_lpp = retrieve_sensor_Cayenne(pin,DIGIT_SUM)

    # (Task 3.(vi)) Create array of frequency from the G1 group with a bandwidth of 125 kHz
    freqG1=[868.1,868.3,868.5]
    freqrandm=random.choice(freqG1)
    print("The frequency is",freqrandm)

    # Set the channel frequency
    lora.set_freq(freqrandm)
    # Set the frame counter
    lora.set_tx_fcnt(number_sender + framecounter)
    # Print Cayenne LPP formatted byte sequence
    print("\n\n",msg_cayenne_lpp)
    # Create a frame by setting the payload
    lora.set_tx_data(list((msg_cayenne_lpp)))

    assert(lora.get_agc_auto_on() == 1)

    try:
        print("Sending LoRaWAN message\n")
        lora.start()
        while True:
            sleep(.1)
            if lora.has_tx_done() == True:
                print("TxDone")
                break

        print("Receiving LoRaWAN message in RX1\n")

        # (Task 3. (vii)) Handle received messages from the TTN and
        # Wait until two times the 'desired Rx1 delay' seconds
        while True:
            sleep (10)
            if lora.has_rx_done() == True:
                print("Downlink message is ",lora.get_rx_data_bytes(),"\n\n")
                break
            else:
                print("No downlink message\n\n")
                break

```

```

except KeyboardInterrupt:
    sys.stdout.flush()
    print("\nKeyboardInterrupt")
    sys.exit(3) #exit3 refers to keyboard interrupt

finally:
    sys.stdout.flush()
    lora.set_mode(MODE.SLEEP)

    if (number_sender==4):
        # Clean up the GPIOs used and the SPI interface
        BOARD.teardown()

        # Clean up the channel
        GPIO.cleanup()

        # No error causes during program launching
        sys.exit(0)

# Design the solution with the ETSI EU regulations and TTN policies
# that enforces a daily limit on the number of messages transmitted. (Part 4)
# See more detail about this number in the report
sleep(593)

# Update number of sending messages
# Use this to update frame counter
number_sender = number_sender + 1;



if __name__ == '__main__':
    main()

```

Appendix B: Evidence of Work

Applications - Console - The Th

eu1.cloud.thethings.network/console/applications



Applications (1)

Search

Create application

ID	Name	End devices	Created at
rp-dht11-iot	rp-dht11-iot	3	16 days ago

© 2023 The Things Stack


EN

v3.28.2 (77f83ca93)

Documentation

Status page



Get support



11:39
15/12/2023


Overview - KTP_IoT2 - The Thin

eu1.cloud.thethings.network/console/applications/rp-dht11-iot/d...



Applications > rp-dht11-iot > End devices > KTP_IoT2

rp-dht11-iot

**KTP_IoT2**
ID: eui-70b3d57ed006347c

↑ 77 ↓ 57 • Last activity 2 days ago

OverviewLive dataMessagingLocationPayload formattersGeneral settings

General information

End device ID

eui-70b3d57ed006347c

Frequency plan

Europe 863-870 MHz (SF9 for RX2 - recommen...

LoRaWAN version

LoRaWAN Specification 1.0.0

Regional Parameters version

TS001 Technical Specification 1.0.0

Created at

Dec 6, 2023 09:31:53

Activation information

AppEUI

n/a

DevEUI

70 B3 D5 7E D0 06 34 7C

Session information

Session start

Dec 7, 2023 14:40:08

Device address

26 0B CA 05

NwkSKey

.....

SNwkSIntKey



.....

11:45

15/12/2023


Gateway data - UoL xLive data - KTP_IoT xUplink payload form x

eu1.cloud.thethings.network/console/applications/rp-dht11-iot/dev... ☆



☰

Applications > rp-dht11-iot > End devices > KTP_IoT2 > Payload formatters > Uplink

**KTP_IoT2**
ID: eui-70b3d57ed006347c

↑ 100 ↓ 70 • Last activity 29 seconds ago ?

OverviewLive dataMessagingLocationPayload formattersGeneral settings

UplinkDownlink

Setup
Formatter type*
CayenneLPP | v
[What is CayenneLPP?](#)

Test
Byte payloadFPort

1

Test decoder

Decoded test payload

13:3415/12/2023

```
ols Games Settings Macros Help
2. 192.168.7.1 (pi)
pi@pi:~/LoRaWAN_TTN $
pi@pi:~/LoRaWAN_TTN $ python3 tx_rx_assignment2.py a a
There was an exception of type ValueError
Error: expected an integer
pi@pi:~/LoRaWAN_TTN $ python3 tx_rx_assignment2.py a 100
There was an exception of type ValueError
Error: expected an integer
pi@pi:~/LoRaWAN_TTN $ python3 tx_rx_assignment2.py 100 a
There was an exception of type ValueError
Error: expected an integer
pi@pi:~/LoRaWAN_TTN $ python3 tx_rx_assignment2.py 13.5 a
There was an exception of type ValueError
Error: expected an integer
pi@pi:~/LoRaWAN_TTN $ python3 tx_rx_assignment2.py 26 13.5
There was an exception of type ValueError
Error: expected an integer
pi@pi:~/LoRaWAN_TTN $ python3 tx_rx_assignment2.py 25 13.5
There was an exception of type ValueError
Error: expected an integer
pi@pi:~/LoRaWAN_TTN $ python3 tx_rx_assignment2.py 25 100
Invalid pin number!
pi@pi:~/LoRaWAN_TTN $ python3 tx_rx_assignment2.py 26 136
Pin is 26
Framecounter is 136
The temperature is 19.3 C and the humidity is 52.0 %
MobaXterm by subscribing to the professional edition here: https://mobaxterm.mobatek.net
14:18
17/12/2023
```

```
ols Games Settings Macros Help
2. 192.168.7.1 (pi)
The temperature is 21.8 C and the humidity is 43.0 %
The frequency is 868.3
b'\x01g\x00\xda\x02hV\x03\x01\x19'
Sending LoRaWAN message
TxDone
Receiving LoRaWAN message in RX1
No downlink message
The temperature is 21.8 C and the humidity is 41.0 %
The frequency is 868.5
b'\x01g\x00\xda\x02hR\x03\x01\x19'
Sending LoRaWAN message
TxDone
Receiving LoRaWAN message in RX1
No downlink message
MobaXterm by subscribing to the professional edition here: https://mobaxterm.mobatek.net
17:11
15/12/2023
```

```
The temperature is 22.6 C and the humidity is 40.0 %
The frequency is 868.5

b'\x01g\x00\xe2\x02hP\x03\x01\x19'
Sending LoRaWAN message

TxDone
Receiving LoRaWAN message in RX1

Downlink message is 9677434aecace466bd0fb517c41f96141db8b
ede283d7111bdb08c96470eeee22040868f0742

The temperature is 22.6 C and the humidity is 40.0 %
The frequency is 868.1

b'\x01g\x00\xe2\x02hP\x03\x01\x19'
Sending LoRaWAN message

TxDone
Receiving LoRaWAN message in RX1

Downlink message is 9677434aecace466bd0fb517c41f96141db8b
ede283d7111bdb08c96470eeee22040868f0742

The temperature is 22.6 C and the humidity is 40.0 %
The frequency is 868.1

b'\x01g\x00\xe2\x02hP\x03\x01\x19'
Sending LoRaWAN message

TxDone
Receiving LoRaWAN message in RX1

Downlink message is 9677434aecace466bd0fb517c41f96141db8b
ede283d7111bdb08c96470eeee22040868f0742

pi@rpi:~/LoRaWAN_TTN $
```

2. 192.168.7.1 (pi)

ere: <https://mobaxterm.mobatek.net>

14:28
15/12/2023

Event details



```
2  name: "as.up.data.lo1waid",
3  "time": "2023-12-15T13:14:14.685397154Z",
4  "identifiers": [
5    {
6      "device_ids": {
7        "device_id": "eui-70b3d57ed006347c",
8        "application_ids": {
9          "application_id": "rp-dht11-iot"
10        },
11        "dev_eui": "70B3D57ED006347C",
12        "dev_addr": "260BCA05"
13      }
14    }
15  ],
16  "data": {
17    "@type": "type.googleapis.com/ttn.lorawan.v3.ApplicationUp",
18    "end_device_ids": {
19      "device_id": "eui-70b3d57ed006347c",
20      "application_ids": {
21        "application_id": "rp-dht11-iot"
22      },
23      "dev_eui": "70B3D57ED006347C",
24      "dev_addr": "260BCA05"
25    },
26    "correlation_ids": [
27      "gs:uplink:01HHPSSZJ9TMFV76V4Y0NFEC9X"
28    ],
29    "received_at": "2023-12-15T13:14:14.681667082Z",
30    "uplink_message": {
31      "f_port": 1,
32      "f_cnt": 96,
33      "frm_payload": "AWcA3wJoUgMBGQ==",
34      "decoded_payload": {
35        "digital_out_3": 25,
36        "relative_humidity_2": 41,
37        "temperature_1": 22.3
38      }
39    }
40  }
```



13:27

15/12/2023

Event details



```
83     "latitude": 53.4086364424694,  
84     "longitude": -2.99046188592911,  
85     "altitude": 18,  
86     "source": "SOURCE_REGISTRY"  
87   },  
88   "uplink_token": "CikKJwobdGVtcGVzdC1pb3QtYmFsdGljYnJvYWRI",  
89   "received_at": "2023-12-15T13:14:14.461810370Z"  
90 }  
91 ],  
92 "settings": {  
93   "data_rate": {  
94     "loras": {  
95       "bandwidth": 125000,  
96       "spreading_factor": 9,  
97       "coding_rate": "4/5"  
98     }  
99   },  
100   "frequency": "868100000",  
101   "timestamp": 1175404876,  
102   "time": "2023-12-15T13:14:00.331075Z"  
103 },  
104 "received_at": "2023-12-15T13:14:14.473757560Z",  
105 "consumed_airtime": "0.205824s",  
106 "network_ids": {  
107   "net_id": "000013",  
108   "ns_id": "EC656E0000000181",  
109   "tenant_id": "ttn",  
110   "cluster_id": "eu1",  
111   "cluster_address": "eu1.cloud.thethings.network"  
112 }  
113 }  
114 },  
115 "correlation_ids": [  
116   "gs:uplink:01HHPSSZJ9TMFV76V4Y0NFEC9X"  
117 ],  
118 "origin": "ip-10-100-13-189.eu-west-1.compute.internal",
```



13:27

15/12/2023

Event details



```
2  name: "as.up.data.lorawan",
3  "time": "2023-12-15T13:24:18.275316532Z",
4  "identifiers": [
5    {
6      "device_ids": {
7        "device_id": "eui-70b3d57ed006347c",
8        "application_ids": {
9          "application_id": "rp-dht11-iot"
10        },
11        "dev_eui": "70B3D57ED006347C",
12        "dev_addr": "260BCA05"
13      }
14    }
15  ],
16  "data": {
17    "@type": "type.googleapis.com/ttn.lorawan.v3.ApplicationUp",
18    "end_device_ids": {
19      "device_id": "eui-70b3d57ed006347c",
20      "application_ids": {
21        "application_id": "rp-dht11-iot"
22      },
23      "dev_eui": "70B3D57ED006347C",
24      "dev_addr": "260BCA05"
25    },
26    "correlation_ids": [
27      "gs:uplink:01HHPTCD0D0ZDXDR9KY6YQF5GZ"
28    ],
29    "received_at": "2023-12-15T13:24:18.271836417Z",
30    "uplink_message": {
31      "f_port": 1,
32      "f_cnt": 98,
33      "frm_payload": "AWcA2QJoVgMBGQ==",
34      "decoded_payload": {
35        "digital_out_3": 25,
36        "relative_humidity_2": 43,
37        "temperature_1": 21.7
38      }
39    }
40  }
```



13:30

15/12/2023

Event details



```
60     "gateway_ids": {
61       "gateway_id": "b827ebfffe6a06ee",
62       "eui": "B827EBFFFE6A06EE"
63     },
64     "time": "2023-12-15T13:24:17.544540882Z",
65     "timestamp": 3124293956,
66     "rssi": -78,
67     "channel_rssi": -78,
68     "snr": 11.5,
69     "uplink_token": "Ch4KHAoQYjgyN2ViZmZmZTZhMDZlZRIIuCfr//5q",
70     "received_at": "2023-12-15T13:24:18.059118106Z"
71   }
72 ],
73   "settings": {
74     "data_rate": {
75       "lorawan": {
76         "bandwidth": 125000,
77         "spreading_factor": 9,
78         "coding_rate": "4/5"
79       }
80     },
81     "frequency": "868100000",
82     "timestamp": 1778997780,
83     "time": "2023-12-15T13:24:03.919040Z"
84   },
85   "received_at": "2023-12-15T13:24:18.062453295Z",
86   "consumed_airtime": "0.205824s",
87   "network_ids": {
88     "net_id": "000013",
89     "ns_id": "EC656E00000000181",
90     "tenant_id": "ttn",
91     "cluster_id": "eu1",
92     "cluster_address": "eu1.cloud.thethings.network"
93   }
94 }
95 },
```



13:32

15/12/2023

Event details



```
3  "time": "2023-12-15T13:34:21.681801296Z",
4  "identifiers": [
5    {
6      "device_ids": {
7        "device_id": "eui-70b3d57ed006347c",
8        "application_ids": {
9          "application_id": "rp-dht11-iot"
10        },
11        "dev_eui": "70B3D57ED006347C",
12        "dev_addr": "260BCA05"
13      }
14    }
15  ],
16  "data": {
17    "@type": "type.googleapis.com/ttn.lorawan.v3.ApplicationUp",
18    "end_device_ids": {
19      "device_id": "eui-70b3d57ed006347c",
20      "application_ids": {
21        "application_id": "rp-dht11-iot"
22      },
23      "dev_eui": "70B3D57ED006347C",
24      "dev_addr": "260BCA05"
25    },
26    "correlation_ids": [
27      "gs:uplink:01HHPTYT8Z10XSBWRTXH8XNK9C"
28    ],
29    "received_at": "2023-12-15T13:34:21.678001652Z",
30    "uplink_message": {
31      "f_port": 1,
32      "f_cnt": 100,
33      "frm_payload": "AWcA4gJoVgMBGQ==",
34      "decoded_payload": {
35        "digital_out_3": 25,
36        "relative_humidity_2": 43,
37        "temperature_1": 22.6
```



13:38

15/12/2023

Event details



```
50     "uplink_token": "Ch4KHAoQYjgyN2ViZmZmZTZhMDZlZRIIuCfr//5qB
51     "received_at": "2023-12-15T13:34:21.439208288Z"
52   }
53 ],
54   "settings": {
55     "data_rate": {
56       "lorawan": {
57         "bandwidth": 125000,
58         "spreading_factor": 9,
59         "coding_rate": "4/5"
60       }
61     },
62     "frequency": "868100000",
63     "timestamp": 3727683244,
64     "time": "2023-12-15T13:34:20.925709009Z"
65   },
66   "received_at": "2023-12-15T13:34:21.471955862Z",
67   "consumed_airtime": "0.205824s",
68   "network_ids": {
69     "net_id": "000013",
70     "ns_id": "EC656E0000000181",
71     "tenant_id": "ttn",
72     "cluster_id": "eu1",
73     "cluster_address": "eu1.cloud.thethings.network"
74   }
75 },
76 },
77 "correlation_ids": [
78   "gs:uplink:01HHPTYT8Z10XSBWRTXH8XNK9C"
79 ],
80 "origin": "ip-10-100-13-189.eu-west-1.compute.internal",
81 "context": {
82   "tenant-id": "CgN0dG4="
83 },
84 "visibility": {
```



13:38

15/12/2023

Event details



```
3  "time": "2023-12-15T13:44:25.054489717Z",
4  "identifiers": [
5    {
6      "device_ids": {
7        "device_id": "eui-70b3d57ed006347c",
8        "application_ids": {
9          "application_id": "rp-dht11-iot"
10       },
11      "dev_eui": "70B3D57ED006347C",
12      "dev_addr": "260BCA05"
13    }
14  ],
15  "data": {
16    "@type": "type.googleapis.com/ttn/lorawan.v3.ApplicationUp",
17    "end_device_ids": {
18      "device_id": "eui-70b3d57ed006347c",
19      "application_ids": {
20        "application_id": "rp-dht11-iot"
21      },
22      "dev_eui": "70B3D57ED006347C",
23      "dev_addr": "260BCA05"
24    },
25    "correlation_ids": [
26      "gs:uplink:01HHPVH7GDCRYT47EVDFPW37DH"
27    ],
28    "received_at": "2023-12-15T13:44:25.051843375Z",
29    "uplink_message": {
30      "f_port": 1,
31      "f_cnt": 102,
32      "frm_payload": "AWcA2AJowAMBGQ==",
33      "decoded_payload": {
34        "digital_out_3": 25,
35        "relative_humidity_2": 44,
36        "temperature_1": 21.6
37      }
```



13:45

15/12/2023

Event details



```
115     },
116     "uplink_token": "CucCZXlKaGJHY2lPaUpCTVRJNFIwTk5TMWNpTENK",
117     "received_at": "2023-12-15T13:44:24.847711749Z"
118   },
119 ],
120   "settings": {
121     "data_rate": {
122       "loras": {
123         "bandwidth": 125000,
124         "spreading_factor": 9,
125         "coding_rate": "4/5"
126       }
127     },
128     "frequency": "868500000",
129     "timestamp": 2335745332,
130     "time": "2023-12-15T13:41:15.610583Z"
131   },
132   "received_at": "2023-12-15T13:44:24.846343676Z",
133   "consumed_airtime": "0.205824s",
134   "network_ids": {
135     "net_id": "000013",
136     "ns_id": "EC656E0000000181",
137     "tenant_id": "ttn",
138     "cluster_id": "eu1",
139     "cluster_address": "eu1.cloud.thethings.network"
140   }
141 },
142 },
143 "correlation_ids": [
144   "gs:uplink:01HHPVH7GDCRYT47EVDFPW37DH"
145 ],
146 "origin": "ip-10-100-15-173.eu-west-1.compute.internal",
147 "context": {
148   "tenant-id": "CgN0dG4="
149 },
```



13:46

15/12/2023

Event details



```
3  "time": "2023-12-15T13:54:28.451515948Z",
4  "identifiers": [
5    {
6      "device_ids": {
7        "device_id": "eui-70b3d57ed006347c",
8        "application_ids": {
9          "application_id": "rp-dht11-iot"
10        },
11        "dev_eui": "70B3D57ED006347C",
12        "dev_addr": "260BCA05"
13      }
14    }
15  ],
16  "data": {
17    "@type": "type.googleapis.com/ttn.lorawan.v3.ApplicationUp",
18    "end_device_ids": {
19      "device_id": "eui-70b3d57ed006347c",
20      "application_ids": {
21        "application_id": "rp-dht11-iot"
22      },
23      "dev_eui": "70B3D57ED006347C",
24      "dev_addr": "260BCA05"
25    },
26    "correlation_ids": [
27      "gs:uplink:01HHPW3MRK2RD0JFAX5042MKP6"
28    ],
29    "received_at": "2023-12-15T13:54:28.448633724Z",
30    "uplink_message": {
31      "f_port": 1,
32      "f_cnt": 104,
33      "frm_payload": "AwcA4wJoVAMBGQ==",
34      "decoded_payload": {
35        "digital_out_3": 25,
36        "relative_humidity_2": 42,
37        "temperature_1": 22.7
```



13:57
15/12/2023

Event details



```
47     "rssi": -91,  
48     "channel_rssi": -91,  
49     "snr": 11,  
50     "uplink_token": "Ch4KHAoQYjgyN2ViZmZmZTZhMDZlZRIIuCfr//5qB  
51     "received_at": "2023-12-15T13:54:28.189726135Z"  
52   },  
53 ],  
54   "settings": {  
55     "data_rate": {  
56       "lorawan": {  
57         "bandwidth": 125000,  
58         "spreading_factor": 9,  
59         "coding_rate": "4/5"  
60       }  
61     },  
62     "frequency": "868300000",  
63     "timestamp": 639491932,  
64     "time": "2023-12-15T13:54:27.702721118Z"  
65   },  
66   "received_at": "2023-12-15T13:54:28.244305285Z",  
67   "consumed_airtime": "0.205824s",  
68   "network_ids": {  
69     "net_id": "000013",  
70     "ns_id": "EC656E0000000181",  
71     "tenant_id": "ttn",  
72     "cluster_id": "eu1",  
73     "cluster_address": "eu1.cloud.thethings.network"  
74   }  
75 },  
76 },  
77 "correlation_ids": [  
78   "gs:uplink:01HHPW3MRK2RD0JFAX5042MKP6"  
79 ],  
80 "origin": "ip-10-100-13-189.eu-west-1.compute.internal",  
81 "context": {  
82   "tenant-id": "CgN0dG4=",  
83 },  
84 "visibility": {  
85   "rights": [  
86     "RIGHT_APPLICATION_TRAFFIC_READ"
```



13:58

15/12/2023