Internet of Things
Summer Semester 2025

Lab Exercise

LoRa Communication Lab

Prof. Dr. Anna Förster
Dr. Jens Dede
Dr. Asanga Udugama
Dr. Andreas Könsgen
Dr. Shadi Attarha

Prof. Dr. Andreas Willig
Dr. Gibson Kimutai
Thenuka Karunatilake
Saurabh Band
Jennifer Horstmann

Submission deadline: 21 May 2025, 12:00 midnight

You are required to work in groups of 3. The report to be written at the end should be developed by all members of the group together, but each member must upload a copy of the report, separately. The report should be in .pdf format and submitted using DoIT! at StudIP!.

In the last lab, you used WiFi with Arduino to transmit measured data to a server. Today, we will use LoRa for the data transmission and MicroPython as the programming language. The data is transmitted directly to the receiving node without any central Internet server. Especially for remote areas and isolated systems, this technology can be an alternative to infrastructure-based network architectures.

You are required to submit a report about the work, maximum 2 pages.
Details of the required content of the report can be found at the end of this document.

Caution!!!

Make sure the Antenna is connected to the board before powering up the device

Setup and Basics

Before you start with the actual task, it is important that you are familiar with the Thonny IDE and can navigate around easily. We use the MoleNet development board in this lab as well but with Micropython. To get the board up and running, follow the **Tutorial** given in *Thonny_intro.pdf* and return back to this point after completing the Tutorial. You will find this document on StudIP.

Now that you know how to upload the code to the development board, you are ready to start with the actual task.

In case you are not familiar with micropython, refer to the following link to get an idea about the basics: Micropython Basics [1]. Understanding the following concepts would be helpful: GPIOs, Pins, Timers, SPI, I2C

Task Description

The task consists of two parts: first, programming a Transmitter to send data over LoRa; second, optimizing the transmitter to improve efficiency and reliability. The transmitter node should read the data (temperature and Pressure) from the BME280 sensor. The data should be transmitted to the receiver node over the LoRa communication channel. You will find the details required to program the nodes in the '**Implementation Details**' Section. Following is the overview of the task:

[1] https://docs.micropython.org/en/latest/esp32/quickref.html#pins-and-gpio

You are required to demonstrate the working of the nodes to the tutor for Part 1 and Part 2 before end of the lab. Completing Part 1 is compulsory, while Part 2 will fetch you better grades.

Part 1: Basic

- For the transmitter, read out the sensor from the development board. Here, focus on the temperature (°C) and the air pressure (Pa). Transmit the sensor data along with group_id and packet_id in the given payload format below.
- Use the given LoRa parameters so that you use the appropriate channel to transmit
- Demonstrate the working by sending 30 packets, with an interval of 10 seconds

NOTE: The receiver is already set up. Ensure the selection of suitable parameters for successful Reception.

Part 2: Advanced

- Before optimizing the system, ensure that communication is working and note the results (payload size, packet reception ratio)
- Optimize the system (eg. reliable tx. or smaller payload). You are free to optimize single or multiple aspects of the system.
- Demonstrate the working by sending 30 packets, with an interval of 10 seconds
- Note the results and compare them with previous results. Discuss it in the report.

Report

- Attach the screenshot of the transmitted and received packet in the report. **The screenshot should display the payload data in the format given below, along with its size (length)**. [Refer to the report template below]
- (If applicable) Attach a separate screenshot showing the difference in payload size, or improvement in PRR, before and after optimization
- Later, we will provide you with the log file containing all the packets received by the receiver.

Implementation Details

In this section, you will find all the required details to program the transmitter. Download the library files and template code for this exercise from StudIP. The libraries contain drivers for BME280 ($bme280\_float.py$) and LoRa ($sx127x.py$).

Payload Format and Pin Mapping

The LoRa payload should contain the temperature and pressure reading from the BME280, the GROUP-ID given to you, and the PACKET-ID to keep track of the number of packets transmitted. Thus the payload should look as follows,

$$(group\_id,\ pkt\_id,\ temperature,\ pressure) \tag{1}$$

The temperature value should be measured in degrees Celsius (°C), with a resolution of 0.1 °C. While, the pressure should be measured in Pascal (Pa) with a resolution of 1. For eg: $temperature = 23.1°C$, $pressure = 101325\ Pa$. Thus the sample payload looks as follows:

$$(12,\ 25,\ 23.1,\ 101325)$$

Table 1 shows the pin mapping for the given development board. Use these pin mapping to use appropriate pins for I2C, SPI, and LoRa.

Table 1: Pin Mapping for MoleNet Board

Label	Pin
I2C_sda	9
I2C_scl	8
dio_0	46
ss	48
reset	45
sck	14
miso	21
mosi	47

Sensor Data

The development board has an onboard BME280 sensor, which can measure temperature, humidity, and pressure. The library provided in the template is taken from this GitHub repository [2]. You can refer to the repository to learn how to use the library. Here is an example of the usage:

Listing 1: Read Sensor Value

```
from machine import Pin
import bme280_float as bme280

i2c = machine.I2C(0, sda=machine.Pin(8), scl=machine.Pin(9))
bme = bme280.BME280(i2c=i2c)
```

In listing 1, the sensor values are read over the I2C bus. It is important to use appropriate SDA and SCL pins with respect to the development board to read the data successfully. Find the pin mapping for our development board in Table 1.

LoRa Setup and Parameters

The library used for LoRa is available on this GitHub repository [3]. Feel free to browse through the examples to learn the usage of the library. Here is sample code to setup LoRa using the provided library

Listing 2: LoRa Setup

```
from machine import Pin, SoftSPI, SPI
from sx127x import SX127x

### adapt the lora_default and lora_pins for your setup
lora_default = {
    'frequency': 869525000,
    'frequency_offset':0,
    'tx_power_level': 14,
    'signal_bandwidth': 125e3,
    'spreading_factor': 9,
    'coding_rate': 5,
    'preamble_length': 8,
    'implicitHeader': False,
    'sync_word': 0x12,
    'enable_CRC': True,
    'invert_IQ': False,
    'debug': False,
}

lora_pins = {
    'dio_0':26,
    'ss':18,
    'reset':16,
    'sck':5,
    'miso':19,
    'mosi':27,
}
```

[2] https://github.com/robert-hh/BME280/tree/master
[3] https://github.com/sergio303/micropython-sx127x/blob/master/examples/LoRaSender.py

Table 2: LoRa Parameters (*lora_default*)

Parameter	**Value**
Frequency	868000000
Spreading factor (sf)	9
Signal Bandwidth	125KHz
Coding Rate	4/5

```
lora_spi = SPI(
    baudrate=10000000, polarity=0, phase=0,
    bits=8, firstbit=SPI.MSB,
    sck=Pin(lora_pins['sck'], Pin.OUT, Pin.PULL_DOWN),
    mosi=Pin(lora_pins['mosi'], Pin.OUT, Pin.PULL_UP),
    miso=Pin(lora_pins['miso'], Pin.IN, Pin.PULL_UP),
)

lora = SX127x(lora_spi, pins=lora_pins, parameters=lora_default)
```

In the listing 2, we see the code to initialize the LoRa library. The communication between lora transceiver chip and microcontroller happens over SPI. It is important to use appropriate pins for SPI according to the development board in use. Get the correct values for *lora_pins* from Table 1 and for *lora_default* from Table 2. Use the default values from the library if the parameter in not mentioned in Table 2

Here, we use the spreading factor of 7 and coding rate of 4/5. Once the LoRa drivers are initialized, you can use the commands in listing 3 to transmit and receive the data over the LoRa channel.

Listing 3: Tx. and Rx. Lora

```
### Transmit over LoRa
lora.println(data)

### Receive over LoRa
payload = lora.readPayload().decode()
```

Following are the sample codes for transmission and reception over lora using the provided libraries shown in listing 4 and 5 respectively.

Listing 4: Transmission Example

```
counter = 0
while True:
    payload = 'Hello␣({0})'.format(counter)
    print('TX:␣{}'.format(payload))
    lora.println(payload)
    counter += 1
    time.sleep(5)
```

Listing 5: Reception Example

```
while True:
    ### read the buffer only if lora pkt received
    if lora.receivedPacket():
        try:
            payload = lora.readPayload().decode()
            rssi = lora.packetRssi()
            print("RX:␣{}␣|␣RSSI:␣{}".format(payload, rssi))
        except Exception as e:
            print(e)
```

Optimization

Now that you are able to transfer data over LoRa, do you think sending the payload as string is the most optimal way to do it? What is the size of the payload (in bytes) that you are transmitting? Can you optimize to use less number of bytes? Is the transmission reliable enough, or is there packet loss?

Try to optimize your system such that the transmission is reliable and/or efficient i.e reduce the packet loss and payload size as much as possible.

NOTE: Optimize one thing at a time so that you can get it working and store results for the report. Points will only be given if the results (screenshots) for the optimization are attached in the report and output demonstrated to the tutor

1. Reliability

In case you observe any packet loss, what do you think is the right way to deal with it? Can you design a system that ensures all packets are received at the receiver (Re-Tx., ACK, etc)?

Optimize to ensure that all packets are received successfully

The receiver is designed to send an acknowledgment for each successfully received packet.

The acknowledgement sent for each packet will consist of the PACKET-ID and GROUP-ID. Thus, the **acknowledgment payload** is as follows:

$$(group\_id,\ pkt\_id) \tag{2}$$

Make appropriate changes to the code to handle these acknowledgments. You can use the received acknowledgments to calculate PRR or retransmit the packets for which no ACK is received.

2. Efficiency

What is the size of the payload that you are transmitting? Can it be reduced to improve efficiency? You can measure the size of the payload using the command shown in listing 6

Listing 6: Calculate payload size

```
payload_len = len(payload)
```

One of the method to reduce payload size is to encode the data. The ustruct library [4] is available in micropython that allows you to encode and decode the data.

The receiver is capable of receiving encoded packets, provided that the data is encoded in the correct required format. Refer to Table 3 for the correct encoding format for each element of the payload

Table 3: Encoding Format for ustruct

Payload Field	C-Type	Python Type
group_id	unsigned char	integer
pkt_id	unsigned short	integer
temperature	float	float
pressure	unsigned long	integer

For *byte order*, use *network (=big-endian).*

Report

You need to submit a report, with maximum of 2 pages. The contents of this report should include the following information, as shown below.

Answer the following questions in the report

- Were all the packets received at the end? If not how many were lost?
- Did you use any techniques to ensure reliable transmission of packets? If yes, then explain the techniques and its effect on the transmission reliability.

[4] https://docs.micropython.org/en/v1.8/pyboard/library/ustruct.html

- Did you optimize the payload size? If yes, then explain how and its effects?

Following is the template for the report:

- Report title: LoRa Communications in the Internet of Things
- Administrative information
  - Your name(s) and your matriculation number(s)
- Section: Results and Discussion:
  - A short description of the procedure of the program
  - Attach results for Part 1 and discuss
  - Attach results for Part 2 and discuss (if applicable)
  - Take help of the above mentioned questions for discussion
- Screenshots as mentioned above