POLITECNICO MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE E DELL'INFORMAZIONE

IoT Challenge #3, Exercises LoRaWAN

Internet of Things

Authors: **Kevin Ziroldi - 10764177**

Matteo Volpari - 10773593

Professors: Alessandro Redondi, Fabio Palmese, Antonio Boiano
Academic Year: 2024-2025
Version: 1.0
Release date: 27-4-2025

Contents

1 | EQ1 - LoRa SF calculation

1.1. Data

In this chapter we answer to question EQ1, asking to find the biggest LoRa SF for having a success rate of at least 70% in a LoRaWAN Network with the following parameters:

- Carrier frequency: $CF = 868MHz$
- Bandwidth: $BW = 125kHz$
- Number of gateways: $N_G = 1$
- Number of sensor nodes: $N_S = 50$
- Intensity of Poisson process: $\lambda = 1$ packet/minute
- Success rate: $SR \geq 0.7$

We compute the payload size based on the last two digits of the leader's person code (XY), according to the formula:

$$L = 3 + XY \text{ Bytes} \tag{1.1}$$

Our leader's person code is 10773593, so the payload size is:

$$L = 3 + 93 = 96 \text{ Bytes} \tag{1.2}$$

1.2. Maximum Spreading Factor calculation

Since LoRaWAN uses an ALOHA-like procedure to handle channel access and retransmissions, we compute the success rate, SR, as the ALOHA success rate:

$$SR = S/G = e^{-2G} = e^{-2N\lambda t} \tag{1.3}$$

Thanks to this formula, we can compute the maximum airtime to have a success rate greater than 70%.

$$SR \geq 0.7 \tag{1.4}$$

$$e^{-2N\lambda t} \geq 0.7 \tag{1.5}$$

By applying the natural logarithm, we get:

$$-2N\lambda t \geq ln(0.7) \tag{1.6}$$

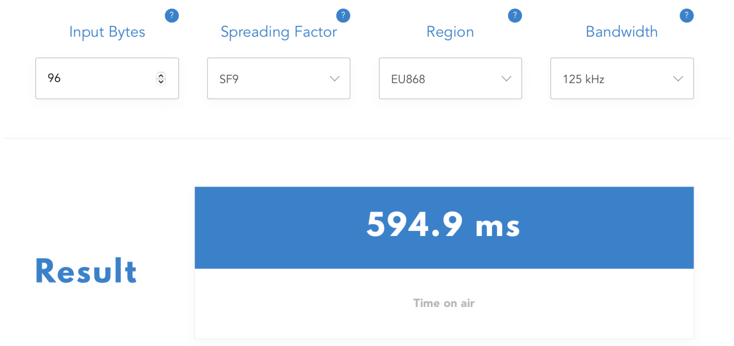
$$t \leq \frac{-ln(0.7)}{2N\lambda} = \frac{-ln(0.7)}{2 \cdot 50 \cdot \frac{1}{60 \cdot 10^3 \text{ ms}}} = 214.005 \text{ ms} \tag{1.7}$$

We now use the API `https://www.thethingsnetwork.org/airtime-calculator` to find the highest SF that guarantees an airtime smaller than the value we found. We use payload size of 96 Bytes, as computed before, region EU868 and bandwidth 125 kHz. The API says that the maximum payload size for EU868 with SF from 10 to 12 is 51 Bytes; this means that we can evaluate SF values starting from 9 and lowering the SF until we find an airtime smaller than 214.005 ms. The values of airtime corresponding to the SF are report in the following table.

Spreading Factor	**Airtime**
SF9	594.9 ms
SF8	328.2 ms
SF7	184.6 ms

Table 1.1: Airtime based on SF

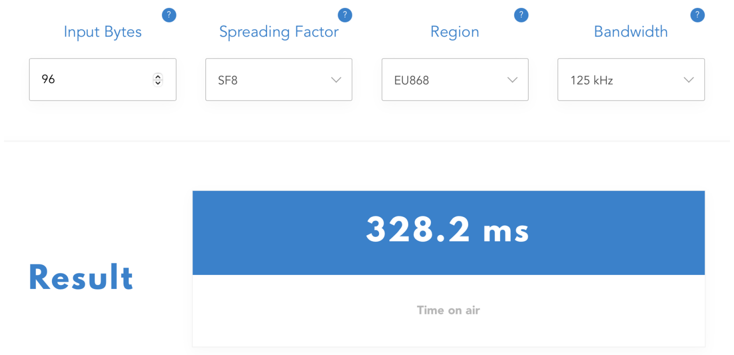
The only value of SF that leads to an airtime smaller than 214.005 ms and a success rate greater than 70% is SF7.



Input Bytes: 96
Spreading Factor: SF9
Region: EU868
Bandwidth: 125 kHz

Result
594.9 ms
Time on air

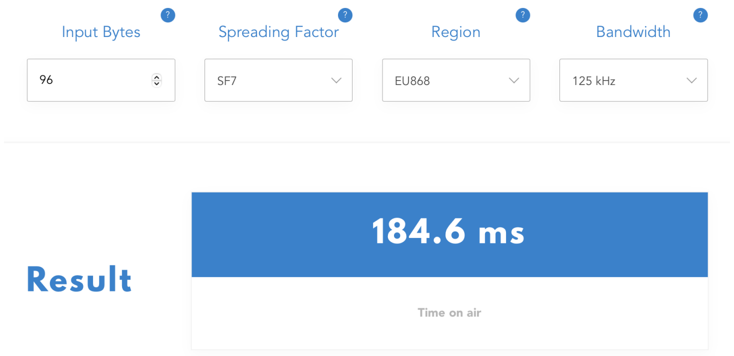
Figure 1.1: Airtime with SF9



Input Bytes: 96
Spreading Factor: SF8
Region: EU868
Bandwidth: 125 kHz

Result
328.2 ms
Time on air

Figure 1.2: Airtime with SF8



Input Bytes: 96
Spreading Factor: SF7
Region: EU868
Bandwidth: 125 kHz

Result
184.6 ms
Time on air

Figure 1.3: Airtime with SF7

2 | EQ2 - LoRaWAN system design

Hardware: âĂć Arduino MKR WAN 1310 Already includes a LPWAN module called Murata CMWX1ZZABZ Need to add an external antenna, the Arduino documentation suggests Dipole Pentaband Waterproof Antenna (https://store.arduino.cc/products/dipole-pentaband-waterproof-antenna). âĂć DHT22 sensor: connected by a digital pin to the Arduino âĂć LoRaWAN Gateway: we can search if we have a neraby gateway from The Things Network at https://www.thethingsnetwork.org/map. If thatâĂŹs not the case, we will need to implement our own gatway. You can buy a LoRa Gateway online for indoor or outdoor usage, based on your needs, e.g. indoor (Mikrotik wAP LR8) outdoor (TEKTELIC KONA MACRO OUTDOOR GATEWAY). Otherwise, you can build your own gateway, e.g. by using a Raspberry Pi 4, a LoRa concentrator board, such as a RAK2245 Pi Hat, and an antenna. âĂć Network Sever The Things Network âĂć ThingSpeak DescribeâĂę

Networking: Arduino and DHT22 are phisically connected. Arduino communicates with the Gateway through LoRaWAN. Gateway to TTN The Gateway can communicate with The Things Network with the gateway connector protocol, that uses as network protocol gRPC or MQTT. TTN communicates the data to ThingSpeak through the built in integration, using the HTTP API exposed by ThingSpeak.

Software: âĂć Codice Arduino âĂć Console TTN âĂć Creazione channel

Arduino code description We include libraries to use LoRaWAN to communicate readings values and to use DHT22 sensor. We define the digital PIN to which the DHT22 sensor is connected and the DHTTYPE, representing the sensor model. The code is formed by two functions: setup and loop. In setup, we initialize both the DHT22 and the LoRa communication, setting the Carrier Frequency to 868 MHz. Moreover, we join the network server thanks to the joinOTAA function. In the loop function, instead, we perform sensor readings for temperature and humidity, encode these values and send them over LoRaWAN netwok. Finally, we insert a delay of 1 minute to wait for the next reading and message.

Console TTN + ThingSpeak Channel - Dalla console andare su create Application e scelgo un ID univoco e un nome per lâĂŹapplicazione e aggiungo una descrizione. DopodichĂĺ schiaccio su create application. - vai su API keys e aggiungi una API Key, scelgo un nome e una data di scadenza, dopodichĂĺ la aggiungo. - Vado su End Devices e faccio register end device. seleziono brand e modello dellâĂŹend device, selezionando hardware version e firmware version. Seleziono il frequency plan raccomandato. Scelgo arbitrariamente JoinEUI, e generato DevEUI. - copio su Arduino le chiavi e i codici che ho generato su TTN. - vado su Payload formatter âĘŠ nella sezione upLink seleziono come formatter type: Custom JavaScript formatter e incollo funzione CHATGPT. - downLink non lo tocchiamo perchĂĺ il flusso non prevede messaggi dal network sensor âĘŠ end device. - creare un nuovo channel su ThingSpeak con 2 fields: Humidity, Temperature - annoto read e write API keys dal canale ThingSpeak - dallâĂŹapplication TTN vado su webhook e aggiungo ThingSpeak inserendo ChannelID e API Key copiati precedentemente da ThingSpeak.

3 | EQ3 - Use LoRaSim to replicate simulations

In this chapter we use the paper "Do LoRa Low-Power Wide-Area Networks Scale?" by M. Bor et al. and the LoRa simulator LoRaSim to reproduce the figures reporting the results of two Experiments Sets from the paper. LoRaSim is a simulator based on SimPy for simulating collisions in LoRa networks and consists in four Python scripts simulating different types of network, that can be run setting some parameters. In order to replicate the two figures, we need to understand which transmitter configuration was user for every configuration SN^i and use the correct simulator with the correct parameters.

3.1. Figure 5: Experiment Set 2

The Experiment Set 2 evaluates the impact of dynamic communication parameter selection on the Data Extraction Rate (DER) and compares three transmitter configurations, called SN^3, SN^4 and SN^5 in the paper.
First of all, we need to choose the correct Python script in the simulator; the paper says that nodes transmit to a single sink (M=1), so we choose the script loraDir.py. From the LoRaSim documentation, we check the parameters used from the selected script, that are reported here:

```
./loraDir.py <NODES> <AVGSEND> <EXPERIMENT> <SIMTIME> [COLLISION]
```

For all experiments, we choose:

- The number of nodes, <NODES>, is chosen from a list built based on the data point of Figure 5 from the paper.
- The average sending interval in milliseconds, <AVGSEND>, is set to 1 million of milliseconds, i.e. 16.7 minutes.
- The simulation time, <SIMTIME>, is not specified from the paper; we choose a simulation time of 58 days, the same as the one used in Experiment Set 1.

- We set <COLLISION> to 1 to enable the full collision check.

The key difference between the different configurations is represented by the <EXPERIMENT> parameter that determines with which radio settings the simulation is run. We are going to choose the experiment looking at how the Experiment Set is described and at the LoRaSim documentation. The paper says that SN^3 uses the settings used by common LoRaWAN deployments, which refers to <EXPERIMENT> = 4. SN^4, instead, is the configuration that minimizes the airtime for each node, by setting the BW, SF and CR, with constant TP; this description corresponds to <EXPERIMENT> = 3. Finally, SN^5 minimizes first airtime and then Transmission Power, as done by the simulator with <EXPERIMENT> = 5.

The complete code used to simulate the network with LoRaSim and plot the DER is reported here.

```python
import os
import subprocess
import math
import pandas as pd
import matplotlib
import matplotlib.pyplot as plt

def simulate(n_nodes, tx_rate, exp, duration):
    env = os.environ.copy()
    env["MPLBACKEND"] = "Agg"
    # Use subprocess.run to execute the command and capture output
    result = subprocess.run(
        [
            "python2",
            "lorasim/loraDir.py",
            str(int(n_nodes)),
            str(int(tx_rate)),
            str(int(exp)),
            str(int(duration)),
            str(int(1))
        ],
        env=env,
        capture_output=True,
        text=True,  # Capture output as text
    )

# Der in aloha defined as S/G = e^(-2G)
def aloha_der(n_nodes,t):
    rate = 1e-6
    return math.exp(-2 * n_nodes * rate * t)

def main():
    duration = 58 * 86400000
    tx_rate = 1e6

    for n_nodes in list(range(1,10)) + list(range(10,100,10)) + list(
                                    range(100,1000,100)) + list(
                                    range(1000,1601,200)):
        print(f"Simulating {n_nodes} nodes")
        simulate(n_nodes, tx_rate, 4, duration)
        simulate(n_nodes, tx_rate, 3, duration)
        simulate(n_nodes, tx_rate, 5, duration)

    data_sn3 = pd.read_csv("exp4.dat", sep=" ")
    data_sn4 = pd.read_csv("exp3.dat", sep=" ")
    data_sn5 = pd.read_csv("exp5.dat", sep=" ")
    data_sn3["der"] = (data_sn3["nrTransmissions"] - data_sn3["
                                    nrCollisions"]) / data_sn3["
                                    nrTransmissions"]
    data_sn4["der"] = (data_sn4["nrTransmissions"] - data_sn4["
                                    nrCollisions"]) / data_sn4["
                                    nrTransmissions"]
    data_sn5["der"] = (data_sn5["nrTransmissions"] - data_sn5["
                                    nrCollisions"]) / data_sn5["
                                    nrTransmissions"]

    plt.plot(data_sn3["#nrNodes"], data_sn3["der"], marker = 'o', label=
                                    "SN3")
    plt.plot(data_sn4["#nrNodes"], data_sn4["der"], marker = 'o', label=
                                    "SN4")
    plt.plot(data_sn5["#nrNodes"], data_sn5["der"], marker = 'o', label=
                                    "SN5")
    plt.title("Success Rate (%)")
    plt.xlabel("Number of nodes")
    plt.ylabel("Rate")
    plt.legend()
    plt.grid()

    plt.savefig("figure5.pdf")
    plt.show()

if __name__ == '__main__':
    main()
```

The plot that corresponds to Figure 5 from the paper is reported here.

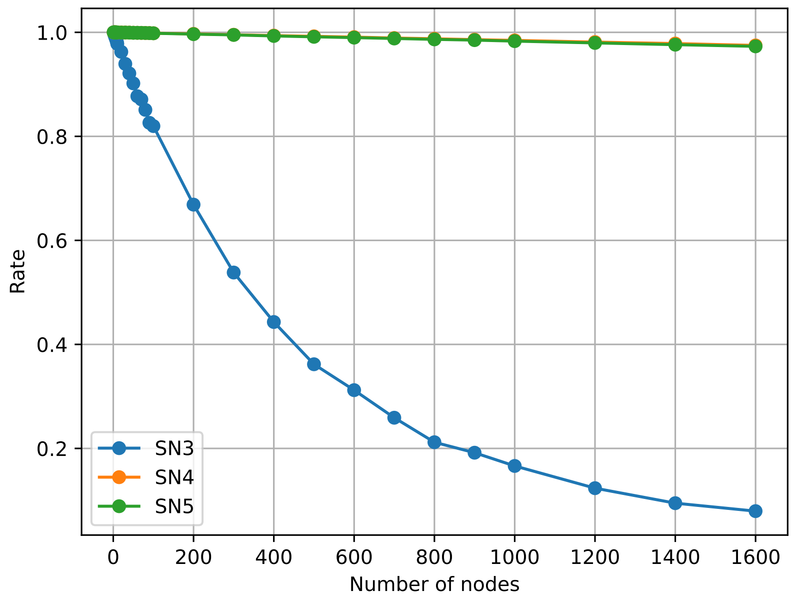


Figure 3.1: Plot corresponding to Figure 5: Experiment Set 2

3.2. Figure 7: Experiment Set 3

The Experiment Set 3 analyzes the impact of the number of sinks M on the network performance, while Figure 7 focuses on the impact on the DER. The experiment uses the configuration called SN[1], with SF = 12, BW = 125 kHz and CR = 4/8, and tests different numbers of sinks (1, 2, 3, 4, 8, 24).
The Python script of the LoRaSim simulator that allows to test multiple sinks is loraDirMulBs.py and takes the following parameters:

```
./loraDirMulBS.py <NODES> <AVGSEND> <EXPERIMENT> <SIMTIME> <BASESTATIONS>
[COLLISION]
```

The only new parameter with respect to the previous Experiment Set is <BASESTATIONS> which is the number of gateways we have in the network. The parameters that are common to Experiment Set 2 are:

- Parameter <NODES>, chosen from the same list of numbers.
- <AVGSEND> is set to 1 million of milliseconds.
- <COLLISION> is set to 1.

We modified the <SIMTIME> to 1 day, because it is not specified by the paper and Google Colab can't handle a very long simulation time for this Experiment Set, since it saturates the available RAM. The paper says that the used setting is the one of SN[1], which uses the most robust LoRa transmitter settings leading to transmissions with the longest possible airtime and with fixed Carrier Frequency. This description refers to <EXPERIMENT> = 0, which uses a constant frequency. The number of gateways, contained in the parameter <BASESTATIONS> changes among the different simulations and is set to 1, 2, 3, 4, 8 and 24, as done in the paper. The complete code used to simulate the network with LoRaSim and plot the DER is reported here.

```
import os
import subprocess
import math
import pandas as pd
import matplotlib
import matplotlib.pyplot as plt

def simulate(n_nodes, tx_rate, exp, duration):
    env = os.environ.copy()
    env["MPLBACKEND"] = "Agg"

    # Use subprocess.run to execute the command and capture output
    result = subprocess.run(
        [
            "python2",
            "lorasim/loraDir.py",
            str(int(n_nodes)),
            str(int(tx_rate)),
            str(int(exp)),
            str(int(duration)),
            str(int(1))
        ],
        env=env,
        capture_output=True,
        text=True,  # Capture output as text
    )

# Der in aloha defined as S/G = e^(-2G)
def aloha_der(n_nodes,t):
    rate = 1e-6
    return math.exp(-2 * n_nodes * rate * t)

def main():
    duration = 30 * 86400000
    tx_rate = 1e6
```

```
for n_nodes in list(range(1,10)) + list(range(10,100,10)) + list(
                                    range(100,1000,100)) + list(
                                    range(1000,1601,200)):
    print(f"Simulating {n_nodes} nodes")
    simulate(n_nodes, tx_rate, 0, duration, 1)
    simulate(n_nodes, tx_rate, 0, duration, 2)
    simulate(n_nodes, tx_rate, 0, duration, 3)
    simulate(n_nodes, tx_rate, 0, duration, 4)
    simulate(n_nodes, tx_rate, 0, duration, 8)
    simulate(n_nodes, tx_rate, 0, duration, 24)

data_bs_1 = pd.read_csv("exp0BS1.dat", delim_whitespace=True,
                                    comment="#", names=["nrNodes", "
                                    DER"])
data_bs_2 = pd.read_csv("exp0BS2.dat", delim_whitespace=True,
                                    comment="#", names=["nrNodes", "
                                    DER"])
data_bs_3 = pd.read_csv("exp0BS3.dat",delim_whitespace=True, comment
                                    ="#", names=["nrNodes", "DER"])
data_bs_4 = pd.read_csv("exp0BS4.dat", delim_whitespace=True,
                                    comment="#", names=["nrNodes", "
                                    DER"])
data_bs_8 = pd.read_csv("exp0BS8.dat", delim_whitespace=True,
                                    comment="#", names=["nrNodes", "
                                    DER"])
data_bs_24 = pd.read_csv("exp0BS24.dat", delim_whitespace=True,
                                    comment="#", names=["nrNodes", "
                                    DER"])

plt.plot(data_bs_1["nrNodes"], data_bs_1["DER"], marker = 'o', label
                                    ="1 sink")
plt.plot(data_bs_2["nrNodes"], data_bs_2["DER"], marker = 'o', label
                                    ="2 sink")
plt.plot(data_bs_3["nrNodes"], data_bs_3["DER"], marker = 'o', label
                                    ="3 sink")
plt.plot(data_bs_4["nrNodes"], data_bs_4["DER"], marker = 'o', label
                                    ="4 sink")
plt.plot(data_bs_8["nrNodes"], data_bs_8["DER"], marker = 'o', label
                                    ="8 sink")
plt.plot(data_bs_24["nrNodes"], data_bs_24["DER"], marker = 'o',
                                    label="24 sink")
plt.title("Success Rate (%)")
plt.xlabel("Number of nodes")
plt.ylabel("Rate")
```

```
    plt.legend()
    plt.grid()
    plt.savefig("figure7.pdf")
    plt.show()

if __name__ == '__main__':
    main()
```

The plot that corresponds to Figure 7 from the paper is reported here.

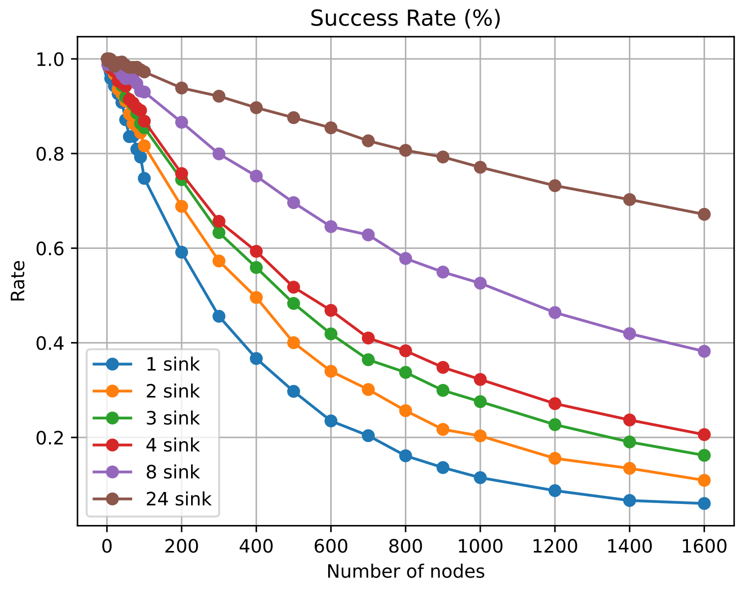


Figure 3.2: Plot corresponding to Figure 7: Experiment Set 3

List of Figures

List of Tables