POLITECNICO
MILANO 1863

Internet of Things
Challenge 3

Andrea Pesciotti
10715428

Teachers:
Redondi Alessandro Enrico Cesare
Fabio Palmese
Boiano Antonio

April 2025

1 EQ1: LoRaWAN Spreading Factor Calculation

1.1 Task Summary

The objective of this task is to determine the largest possible LoRaWAN Spreading Factor (SF) that allows a network of 50 sensor nodes in Europe (868 MHz, 125 kHz bandwidth) to achieve a packet transmission success rate of at least 70

1.2 Input Parameters

The following parameters were provided or derived for the calculation:

- Network Type: LoRaWAN
- Region / Frequency: Europe / 868 MHz
- Bandwidth (BW): 125 kHz
- Number of Nodes (N): 50
- Transmission Intensity per Node (λ_{node}): 1 packet/minute
- Target Success Probability ($P_{success\_target}$): ≥ 0.70
- Personal Code ending (XY): 28
- Calculated Payload Size ($L = 3 + XY$): $3 + 28 = 31$ bytes
- Total Network Arrival Rate ($\lambda_{total} = N \times \lambda_{node}$): $50 \times 1 = 50$ packets/minute
- Total Network Arrival Rate (per second): $50/60 \approx 0.8333$ packets/second

1.3 Time on Air (ToA) Values

The following Time on Air values for a 31-byte payload at 125 kHz bandwidth were obtained using the specified online calculator (https://www.thethingsnetwork.org/airtime-calculator) for different Spreading Factors:

Spreading Factor (SF)	ToA (ms)	ToA (s)
SF12	2138.1 ms	2.1381 s
SF11	1151.0 ms	1.1510 s
SF10	534.5 ms	0.5345 s
SF9	287.7 ms	0.2877 s
SF8	164.4 ms	0.1644 s
SF7	92.4 ms	0.0924 s

1.4 Calculation Protocol

The calculation employed the Pure ALOHA model to estimate the probability of successful packet transmission ($P_{success}$). The formula used is: $P_{success} = e^{-2G}$ where G is the offered channel traffic, calculated as $G = N \times \lambda \times T$.

- N is the total number of nodes (50).
- λ is the transmission rate per node in packets per second (1/60).
- T is the Time on Air (T_{oA}) of a single packet in seconds, which varies with the SF.

1.5 Python Implementation

The following Python code was used to iterate through the Spreading Factors (from highest to lowest), calculate the success probability for each using the provided ToA values and the ALOHA formula, and determine the largest SF meeting the $\geq 70\%$ success rate requirement.

```
import math

N_nodes = 50
lambda_min = 1
target_p_success = 0.70
xy = 28

payload_l = 3 + xy
lambda_sec = lambda_min / 60.0

toa_values_s = {
    12: 2.1381,
    11: 1.1510,
    10: 0.5345,
    9:  0.2877,
    8:  0.1644,
    7:  0.0924,
}

best_sf = None
spreading_factors = sorted(toa_values_s.keys(), reverse=True)

for sf in spreading_factors:
    toa_s = toa_values_s[sf]

    exp = -2 * N_nodes * lambda_sec * toa_s
    p_success = math.exp(exp)

    if p_success >= target_p_success:
        if best_sf is None:
            best_sf = sf
            break
```

1.6 Calculated Success Rates

The Python script calculated the following success probabilities for each Spreading Factor using the provided Time on Air values:

SF	**ToA (s)**	**Calculated $P_{success}$**
SF12	2.1381	0.0280 (2.8%)
SF11	1.1510	0.1469 (14.7%)
SF10	0.5345	0.4099 (41.0%)
SF9	0.2877	0.6192 (61.9%)
SF8	0.1644	0.7604 (76.0%)
SF7	0.0924	0.8573 (85.7%)

Result: Therefore, the biggest (largest number) Spreading Factor that achieves the target success rate of at least 70% is:

$$\boxed{SF8}$$

2 EQ2: LoRaWAN Sensor System Design (Arduino to ThingSpeak)

2.1 Task Summary

The objective of this task is to design a system utilizing an Arduino MKR WAN 1310 board and a DHT22 sensor to collect temperature and humidity data and transmit it wirelessly via LoRaWAN to the ThingSpeak platform for storage and visualization. This involves outlining the complete system architecture, sketching the data processing flow within Node-RED, and detailing the necessary steps for implementation from hardware setup to cloud integration.

2.2 System Block Diagram

The end-to-end system involves several key components communicating in sequence:

1. **Sensor Node:** This is the edge device responsible for interacting with the physical world. *Functionally, it involves:* A microcontroller board (e.g., Arduino MKR WAN 1310) equipped with sensors (e.g., DHT22 for temperature and humidity) and a LoRa radio module. The node requires initial hardware setup, connecting the sensor and antenna. Its firmware is programmed to periodically read data from the sensors, process this data if necessary, securely store unique device identifiers and network security keys, manage the LoRaWAN network connection (including joining the network via an activation process like OTAA), efficiently encode the sensor readings into a compact binary format suitable for LoRaWAN's limited payload size, initiate the transmission process, and employ power-saving strategies (like deep sleep modes) between transmissions to extend battery life.

2. **LoRaWAN Radio Transmission:** This block represents the wireless communication link. The sensor node uses its LoRa radio to transmit the encoded data packets over a long range using the LoRaWAN protocol, typically operating in license-free ISM bands (like 868 MHz in Europe). This transmission is characterized by low power consumption but also low data rates and duty cycle limitations. The focus is on efficiently sending small amounts of data reliably over distance.

3. **LoRaWAN Gateway:** Gateways act as bridges between the low-power wireless domain (LoRaWAN) and standard IP networks (like the Internet). *Their role is to:* Listen for LoRaWAN packets transmitted by nearby sensor nodes, receive these packets, and forward them unaltered to a central LoRaWAN Network Server via a stable internet connection (e.g., Ethernet, Wi-Fi, Cellular). Gateways do not typically interpret the sensor data itself but handle the radio-to-IP conversion and communication with the LNS. Multiple gateways can receive the same packet, providing redundancy.

4. **LoRaWAN Network Server (LNS):** The LNS (e.g., The Things Network) is the central intelligence of the LoRaWAN network. *Its responsibilities in-*

clude: Managing the network infrastructure (gateways and devices), authenticating sensor nodes attempting to join or communicate (validating device identifiers and security keys), decrypting the encrypted data payloads sent by the nodes, handling LoRaWAN protocol specifics (like acknowledgments, data rate adaptation), optionally decoding the application-specific binary payload into meaningful data structures (using pre-configured decoders if provided), and securely routing the processed application data towards the designated application integration endpoint. Device registration and credential management are key configuration tasks performed on the LNS.

5. **Application Layer (Middleware/Integration Platform):** This layer receives the sensor data from the LNS, typically via standard IoT protocols like MQTT or HTTP webhooks. Platforms like Node-RED often fulfill this role. *Its function is to:* Subscribe to the data feed from the LNS, parse the incoming messages, extract the relevant sensor values (e.g., temperature, humidity), potentially perform further processing, logic, or data transformation, and prepare the data for forwarding to its final destination (e.g., a database or cloud platform). This layer often uses API credentials provided by the LNS for secure communication.

6. **Data Forwarding to Cloud:** This conceptual step, often handled within the Application Layer, involves formatting the processed sensor data according to the specific API requirements of the target cloud platform or database. It then securely transmits this data, typically using HTTP requests, authenticating with the cloud platform using specific API keys or tokens associated with the destination channel or service.

7. **Cloud Platform (IoT Platform/Database):** This is the final destination for the sensor data (e.g., ThingSpeak). *Its role is to:* Provide an API endpoint for data ingestion, store the received time-series data persistently, associate it with specific devices or channels, and offer tools for data visualization (dashboards, charts), analysis, and potentially triggering alerts or actions based on the data. Configuration involves setting up data structures (channels/fields) and managing API keys for secure data submission and retrieval.

2.3 Node-RED Flow Sketch

The Node-RED flow acts as the application layer bridge between the LoRaWAN Network Server (specifically TTN in this design) and the ThingSpeak cloud platform. It receives the uplink data, ensures it's correctly formatted, and forwards it to ThingSpeak. It is important to note that this description and the accompanying sketch focus specifically on the configuration and logic within the Node-RED environment itself, representing the core processing steps handled by Node-RED in this communication chain.

Figure 1: Node-RED Flow Sketch for processing TTN LoRaWAN data and updating ThingSpeak.

The flow consists of:

- A `ttn uplink` node: Connects to the specified TTN application/device and receives decoded uplink messages.
- A `Function` node ("Prepare Fields"): Extracts temperature and humidity from the TTN message payload and formats them into the structure required by the ThingSpeak node (e.g., `msg.payload = {field1: temp, field2: hum}`).
- A `thingspeak update` node: Takes the prepared data and uses the configured API key to send the update to the specified ThingSpeak channel fields.

3 EQ3: LoRaSim Paper Figure Reproduction

3.1 Task Summary

The objective of this task is to reproduce Figures 5 and 7 from the paper "Do LoRa Low-Power Wide-Area Networks Scale?" by M. Bor et al., utilizing the LoRaSim simulator. This involves understanding the paper's experimental setup, configuring and running the LoRaSim simulator with parameters matching the original experiments, and presenting the simulation results in a format comparable to the paper's figures. The provided Python notebook `LoRasim.ipynb` serves as a reference for interacting with the simulator.

3.2 Methodology

The reproduction process involved the following steps:

1. **Paper Study:** The relevant sections of the paper by M. Bor et al. were reviewed to identify the parameters used for the simulations corresponding to Figure 5 (Data Extraction Rate vs. Nodes for static and dynamic settings) and Figure 7 (Data Extraction Rate vs. Nodes for multiple sinks).

2. **Simulator Setup:** The LoRaSim environment was set up based on the instructions derived from the reference notebook (`LoRasim.ipynb`) within a Google Colab environment. This involved downloading the simulator, installing Python 2, and installing the required Python dependencies (numpy, simpy, matplotlib) using pip for Python 2.

3. **Simulation Execution:** A Python script, adapted from the reference notebook, was used to run the necessary simulations using the LoRaSim scripts (`loraDir.py` for single sink scenarios and `loraDirMulBS.py` for multiple sink scenarios). Simulations were run for varying numbers of nodes as specified in the paper and notebook.

3.3 Python Implementation for Simulation of Figure 5

The following snippets of a Python script (adapted from `LoRasim.ipynb`) were used within the prepared Colab environment to automate the simulation runs for different node counts and sink numbers, process the output data, and generate the plots corresponding to Figure 5.

```
# --- Simulation Parameters ---
duration_ms = 86400000 # 24 hours in ms
tx_interval_ms = 1e6
node_counts = list(range(1, 10)) + list(range(10, 300, 10)) + list(range(300, 1600, 100))

# --- Simulation Execution for Figure 5 ---
exp_sn3 = 4
exp_sn4 = 3
exp_sn5 = 5

for n_nodes in node_counts:
  simulate(n_nodes, tx_interval_ms, exp_sn3, duration_ms)
  simulate(n_nodes, tx_interval_ms, exp_sn4, duration_ms)
  simulate(n_nodes, tx_interval_ms, exp_sn5, duration_ms)

# --- Data Loading and Processing ---
data_sn3 = pd.read_csv("exp4.dat", sep=" ") # SN3 = exp 4
data_sn3["der"] = (data_sn3["nrTransmissions"] - data_sn3["nrCollisions"]) / data_sn3["nrTransmissions"]
data_sn3.loc[data_sn3["nrTransmissions"] == 0, "der"] = 0.0

data_sn4 = pd.read_csv("exp3.dat", sep=" ") # SN4 = exp 3
data_sn4["der"] = (data_sn4["nrTransmissions"] - data_sn4["nrCollisions"]) / data_sn4["nrTransmissions"]
data_sn4.loc[data_sn4["nrTransmissions"] == 0, "der"] = 0.0

data_sn5 = pd.read_csv("exp5.dat", sep=" ") # SN5 = exp 5
data_sn5["der"] = (data_sn5["nrTransmissions"] - data_sn5["nrCollisions"]) / data_sn5["nrTransmissions"]
data_sn5.loc[data_sn5["nrTransmissions"] == 0, "der"] = 0.0

# --- Plotting for Figure 5 ---
plt.figure(figsize=(10, 6))

# Plot SN3 (Static) - Experiment 4
plt.plot(data_sn3["#nrNodes"], data_sn3["der"], marker='s', linestyle='-', label="SN3 (Static)")

# Plot SN4 (Dynamic, min airtime) - Experiment 3
plt.plot(data_sn4["#nrNodes"], data_sn4["der"], marker='o', linestyle='--', label="SN4 (Dynamic min airtime)")

# Plot SN5 (Dynamic, min airtime & TP) - Experiment 5
plt.plot(data_sn5["#nrNodes"], data_sn5["der"], marker='^', linestyle=':', label="SN5 (Dynamic min airtime & TP)")

plt.title("Figure 5 Reproduction")
plt.xlabel("Number of Nodes")
```

```
plt.ylabel("Data Extraction Rate (DER)")
plt.ylim(0, 1.05)
plt.xlim(left=0)
plt.grid(True)
plt.legend()
plt.show()
plt.close()
```

3.3.1 Figure 5 Reproduction

Figure 2 shows the simulated Data Extraction Rate (DER) for the SN3 (static LoRaWAN settings), SN4 (dynamic minimum airtime), and SN5 (dynamic minimum airtime and transmit power) parameter sets as a function of the number of nodes.

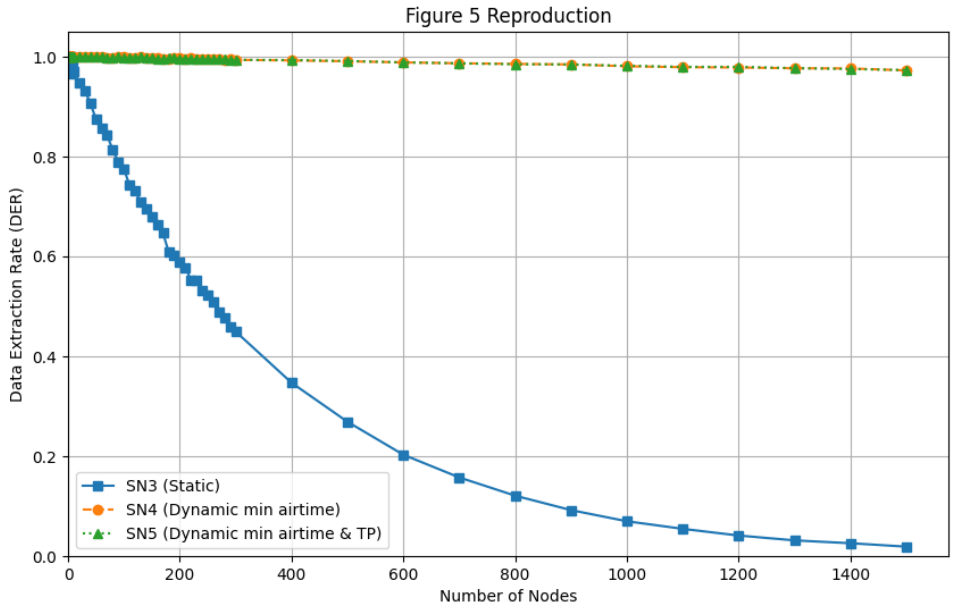

Figure 2: Reproduced Figure 5 of Experiment Set 2.