POLITECNICO
MILANO 1863

SCUOLA DELL'INGEGNERIA INDUSTRIALE E DELL'INFORMAZIONE

COMPUTER SCIENCE AND ENGINEERING

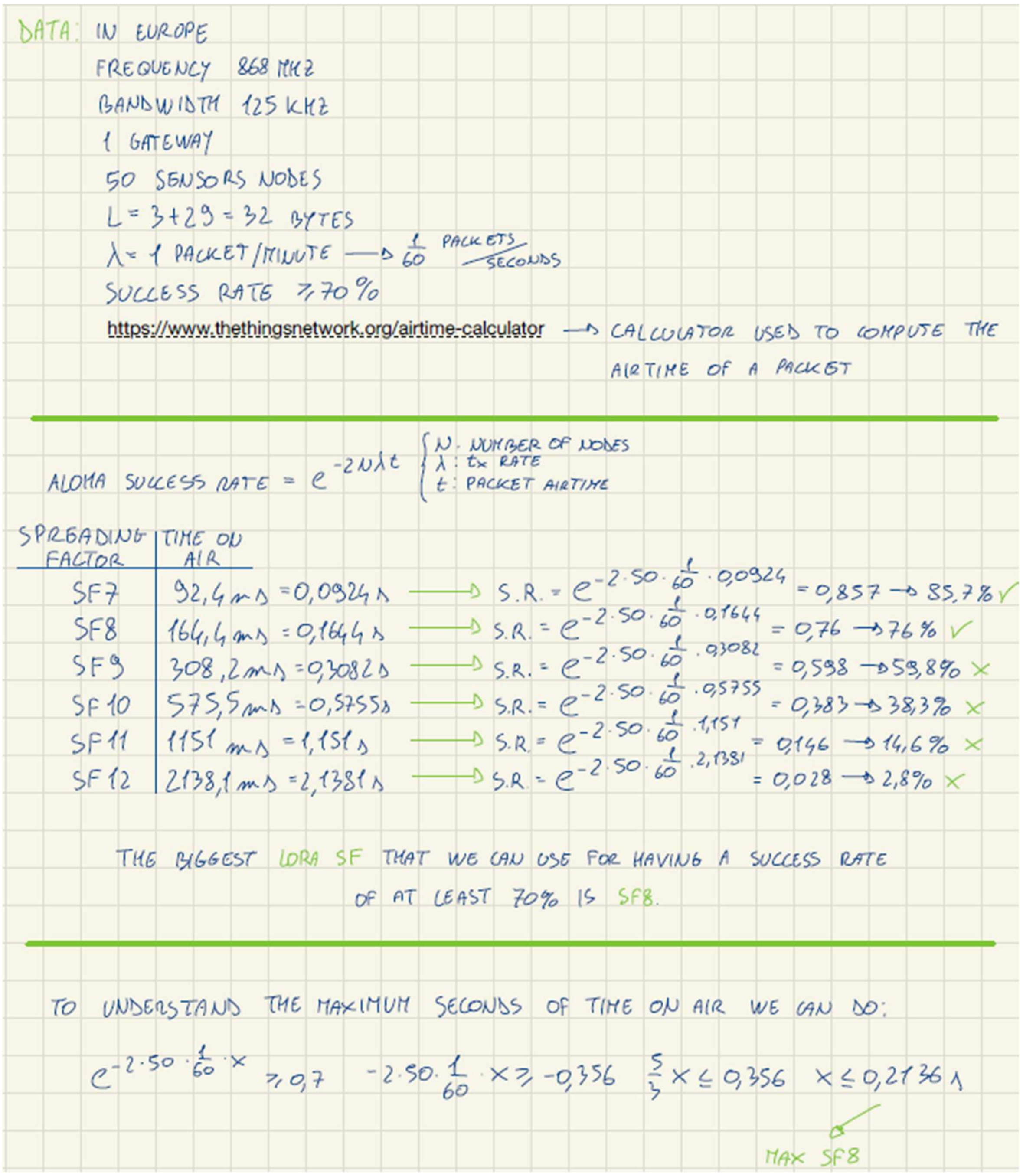
Internet Of Things
3rd challenge

Simone Pio Bottaro - 10774229

Gabriele Lorenzetti – 10730455

A.Y 2024/2025

EQ1



DATA: IN EUROPE

FREQUENCY 868 MHZ

BANDWIDTH 125 KHZ

1 GATEWAY

50 SENSORS NODES

$L = 3+29 = 32$ BYTES

$\lambda = 1$ PACKET/MINUTE $\longrightarrow \frac{1}{60} \frac{\text{PACKETS}}{\text{SECONDS}}$

SUCCESS RATE $\geq 70\%$

https://www.thethingsnetwork.org/airtime-calculator $\longrightarrow$ CALCULATOR USED TO COMPUTE THE AIRTIME OF A PACKET

ALOHA SUCCESS RATE $= e^{-2N\lambda t}$ $\begin{cases} N: \text{NUMBER OF NODES} \\ \lambda: \text{tx RATE} \\ t: \text{PACKET AIRTIME} \end{cases}$

SPREADING FACTOR	TIME ON AIR		
SF7	92,4 ms = 0,0924 s	$\longrightarrow$ S.R. $= e^{-2\cdot 50\cdot \frac{1}{60}\cdot 0{,}0924} = 0{,}857 \longrightarrow 85{,}7\%$	✓
SF8	164,4 ms = 0,1644 s	$\longrightarrow$ S.R. $= e^{-2\cdot 50\cdot \frac{1}{60}\cdot 0{,}1644} = 0{,}76 \longrightarrow 76\%$	✓
SF9	308,2 ms = 0,3082 s	$\longrightarrow$ S.R. $= e^{-2\cdot 50\cdot \frac{1}{60}\cdot 0{,}3082} = 0{,}598 \longrightarrow 59{,}8\%$	✗
SF10	575,5 ms = 0,5755 s	$\longrightarrow$ S.R. $= e^{-2\cdot 50\cdot \frac{1}{60}\cdot 0{,}5755} = 0{,}383 \longrightarrow 38{,}3\%$	✗
SF11	1151 ms = 1,151 s	$\longrightarrow$ S.R. $= e^{-2\cdot 50\cdot \frac{1}{60}\cdot 1{,}151} = 0{,}146 \longrightarrow 14{,}6\%$	✗
SF12	2138,1 ms = 2,1381 s	$\longrightarrow$ S.R. $= e^{-2\cdot 50\cdot \frac{1}{60}\cdot 2{,}1381} = 0{,}028 \longrightarrow 2{,}8\%$	✗

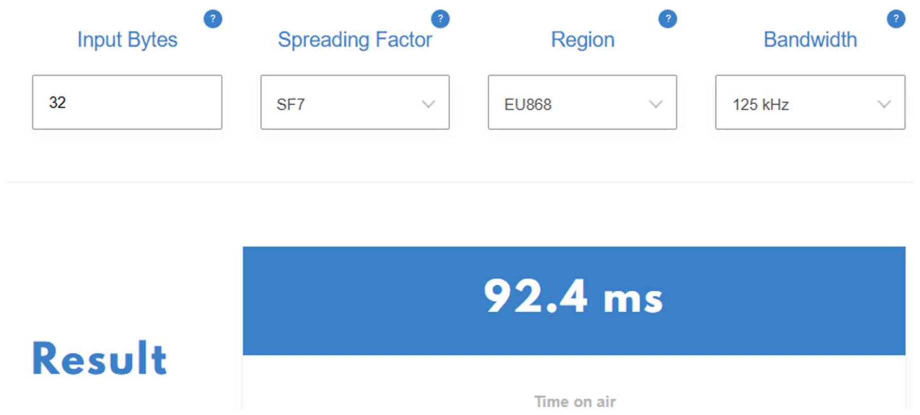
THE BIGGEST LORA SF THAT WE CAN USE FOR HAVING A SUCCESS RATE OF AT LEAST 70% IS SF8.

TO UNDERSTAND THE MAXIMUM SECONDS OF TIME ON AIR WE CAN DO:

$$e^{-2\cdot 50\cdot \frac{1}{60}\cdot x} \geq 0{,}7 \qquad -2\cdot 50\cdot \frac{1}{60}\cdot x \geq -0{,}356 \qquad \frac{5}{3}x \leq 0{,}356 \qquad x \leq 0{,}2136\ s$$

MAX SF8

Example of a time on air calculation



Input Bytes	Spreading Factor	Region	Bandwidth
32	SF7	EU868	125 kHz

Result

92.4 ms

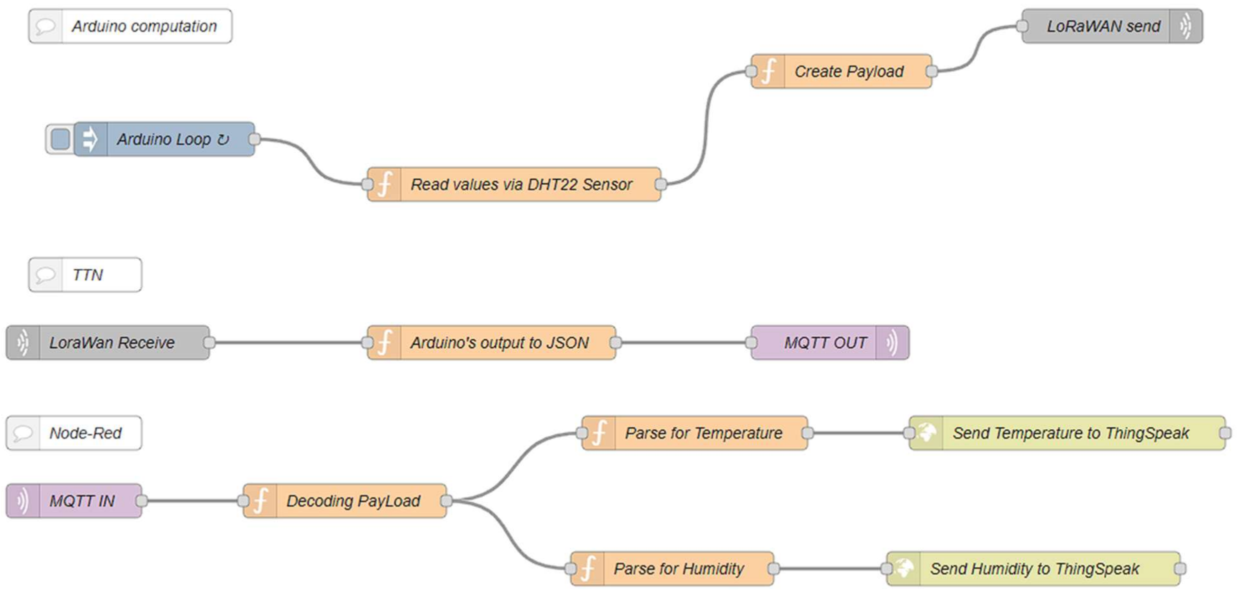
Time on air

EQ2

Introduction

Using an Arduino MKR WAN 1310 connected with a DHT22 sensor, has been designed a concept of system which sends data to a ThingSpeak channel via LoRaWAN. For the implementation of system has been used Node-Red to create a sketch of the block diagram.

LoRaWAN (Long Range Wide Area Network) is a wireless communication protocol designed for low-power devices, typically used in Internet of Things (IoT) applications. It is ideal for devices that need to transmit small amounts of data over long distances while consuming minimal energy. LoRaWAN works by using a modulation technique called LoRa, which enables long-range communication, making it well-suited for rural or large-area networks.



Each of these steps will be analysed in the following pages.

Arduino Computation

This initial phase involves setting up the sensor and communication settings and ensuring that the Arduino can securely send data over LoRaWAN.

The **Arduino MKR WAN 1310** is a development board that integrates a LoRa radio, allowing communication via LoRaWAN. To use it for sending data, such as temperature and humidity from a DHT22 sensor, the first step is to set up the hardware and configure the Arduino.

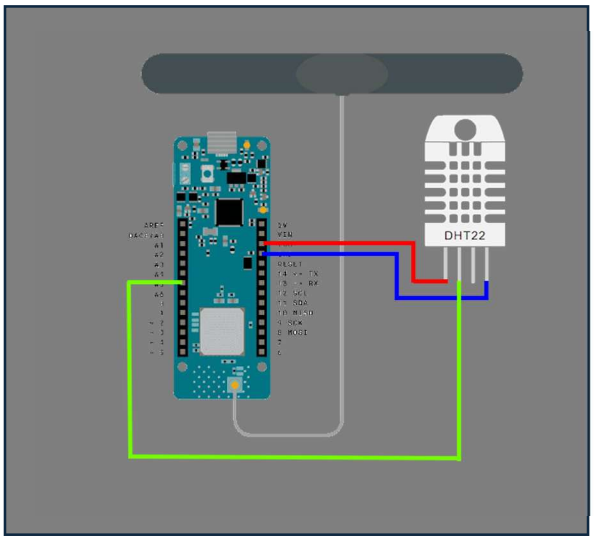
The **DHT22** sensor is connected to the Arduino board, and its data is read in the code.

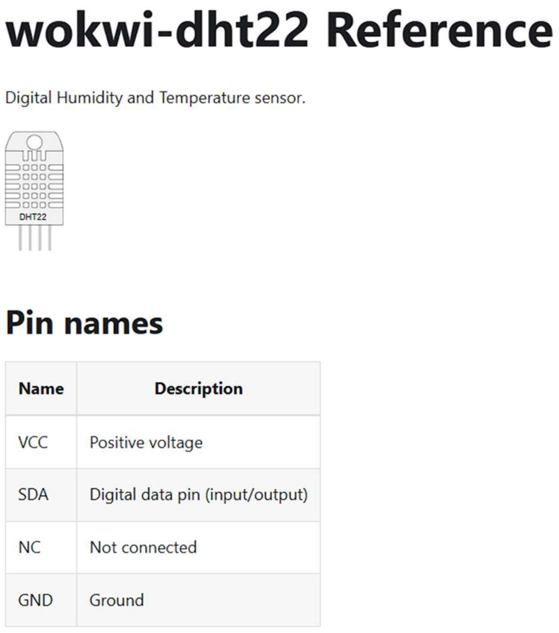
To send the data using LoRaWAN, the Arduino needs to be registered on **The Things Network (TTN)** creating a new account on website. During this process, the Arduino's **Extended Unique Identifier**, and **AppKey** are configured to ensure secure communication.

Once the device is registered, the Arduino can send data over LoRaWAN to a gateway, which forwards the data to a server.

In the case of this project, the Arduino collects temperature and humidity data from the DHT22 sensor. The data is then packaged into a payload and transmitted via LoRaWAN to TTN.

From TTN, the data can be forwarded to other platforms, such as Node-RED or, potentially, directly ThingSpeak for further processing and analysis.



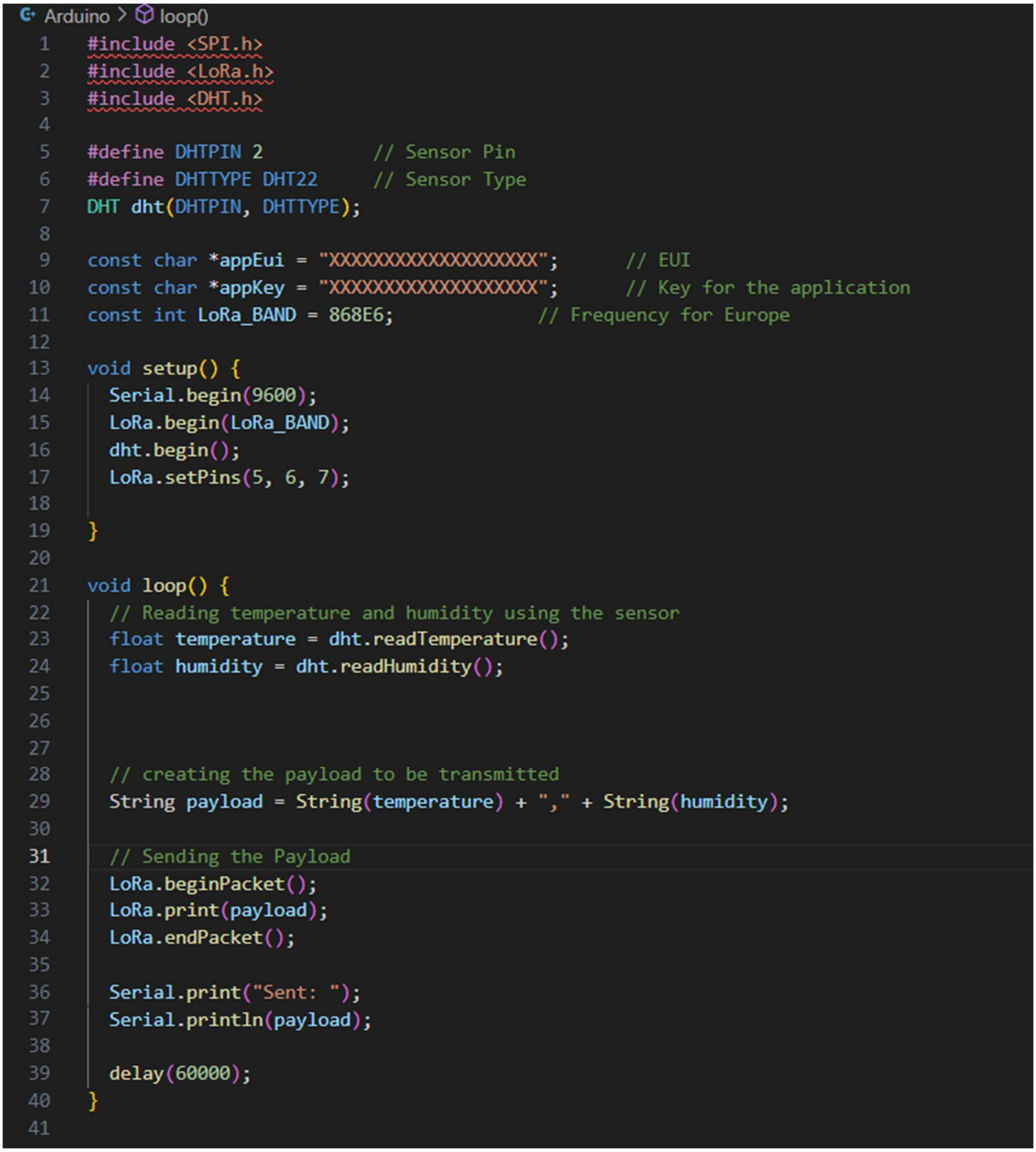

wokwi-dht22 Reference

Digital Humidity and Temperature sensor.

Pin names

Name	Description
VCC	Positive voltage
SDA	Digital data pin (input/output)
NC	Not connected
GND	Ground

Here is showed a simple example of how the code should be to implement the system



```
#include <SPI.h>
#include <LoRa.h>
#include <DHT.h>

#define DHTPIN 2          // Sensor Pin
#define DHTTYPE DHT22     // Sensor Type
DHT dht(DHTPIN, DHTTYPE);

const char *appEui = "XXXXXXXXXXXXXXXXXX";       // EUI
const char *appKey = "XXXXXXXXXXXXXXXXXX";       // Key for the application
const int LoRa_BAND = 868E6;                // Frequency for Europe

void setup() {
  Serial.begin(9600);
  LoRa.begin(LoRa_BAND);
  dht.begin();
  LoRa.setPins(5, 6, 7);

}

void loop() {
  // Reading temperature and humidity using the sensor
  float temperature = dht.readTemperature();
  float humidity = dht.readHumidity();



  // creating the payload to be transmitted
  String payload = String(temperature) + "," + String(humidity);

  // Sending the Payload
  LoRa.beginPacket();
  LoRa.print(payload);
  LoRa.endPacket();

  Serial.print("Sent: ");
  Serial.println(payload);

  delay(60000);
}

```

The Things Network

When the Arduino MKR WAN 1310 sends data via LoRaWAN, the message is picked up by a nearby LoRaWAN gateway and forwarded to The Things Network (TTN). TTN acts as a central server that receives and processes LoRaWAN packets. If configured, TTN can decode the payload into readable values.

Once the message is processed, TTN automatically publishes it to its MQTT broker. Each device in TTN has a unique MQTT topic, and any new data sent by the device is published under that topic in **JSON** format. This JSON includes useful information such as the decoded sensor values, the timestamp and data about the transmission.

Node-Red

The process starts with the MQTT input node, which is configured to subscribe to a specific topic used by TNN to publish messages from Arduino device. Next, the data flows into a **function node** that decodes the payload. The payload received from TTN is encoded in a specific format (often as a hexadecimal string or Base64).

Once decoded, the data is sent to two other **function nodes** that formats it for ThingSpeak. ThingSpeak requires data in a specific structure, usually with field names like ***field1*** and ***field2***, and a valid API key. These nodes prepare the HTTP requests with these parameters.

The formatted data is then passed to two **HTTP request nodes**, which perform a POST request each to ThingSpeak's API. These nodes send the temperature and humidity readings directly to the ThingSpeak channel, updating the corresponding fields in real time.

EQ3

Introduction

Using the paper "Do LoRa Low-Power Wide-Area Network Scale?" and the LoRa simulator, reproduce Figure 5 and Figure 7 from the paper

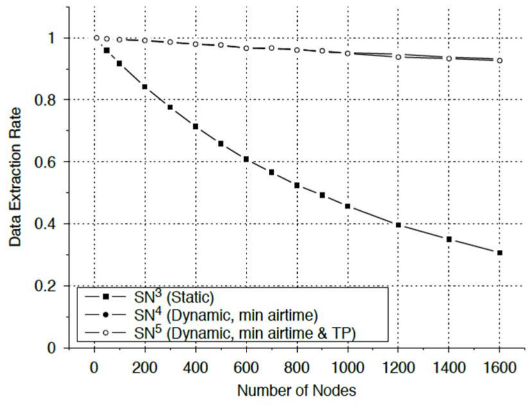


Figure 5: Experiment Set 2 – Dynamic Parameters: Lines for SN^4 and SN^5 overlap. When optimising transmission parameters for minimal airtime (or airtime and power) network capacity greatly improves. With minimised airtime (SN^4) and $DER > 0.9$, well over $N = 1600$ nodes can be supported (compared to $N = 120$ nodes with static settings).

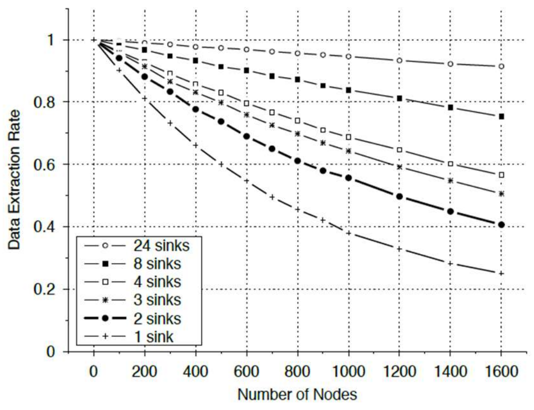


Figure 7: Experiment Set 3 – Multiple Sinks: multiple sink can significantly increase the DER.

In the paper is analysed the scalability limitations of LoRa networks for large-scale IoT applications such as smart cities and environmental monitoring.

Experiment Set 2: Impact of Dynamic Communication Parameters

The first experiment investigates how the performance of a LoRa network can be improved by dynamically optimizing communication parameters for each node, rather than using a fixed, static configuration.

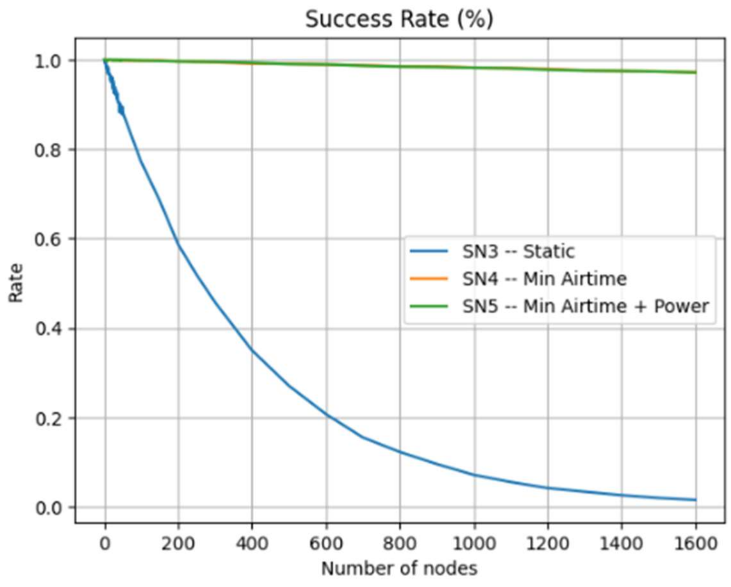
In this setup, a single LoRa sink is placed at the center of the network, and a varying number of nodes (from 0 up to 1600) are randomly distributed around it. Each node periodically transmits a 20-byte packet every 16.7 minutes. Three different communication strategies are compared:

- **SN3 (Static Parameters)**: All nodes use a typical LoRaWAN configuration with high robustness but long airtime (SF12, BW 125 kHz, CR 4/5).
- **SN4 (Minimized Airtime)**: Each node dynamically selects its transmission parameters (Spreading Factor, Bandwidth, Coding Rate) to minimize the airtime, aiming to reduce collision probability and improve network throughput.
- **SN5 (Minimized Airtime + Power)**: Nodes first minimize airtime and then further optimize transmission power to reduce energy consumption, while maintaining connectivity.

For each scenario, the **Data Extraction Rate (DER)**, defined as the ratio of successfully received packets to transmitted packets, is computed. The results show that dynamic optimization dramatically increases the number of supported nodes while maintaining a high DER. Specifically, dynamic parameter selection (SN4/SN5) enables the network to support over 1600 nodes with a DER greater than 0.9, compared to only 120 nodes with static settings (SN3).

In order to replicate the experiment, has been used the ***lorDir.py*** as subprocess, in order to replicate communication strategies SN3, SN4 and SN5 by using values respectively values 3, 4 and 5 for exp value expected by the function ***simulate*** present in the simulator. All the code is available inside the ***LoRasim.ipynb*** uploaded.

Here is showed the plot obtained



Experiment Set 3: Impact of Deploying Multiple Sinks

The second experiment evaluates the effect of increasing the number of sink nodes on the scalability and reliability of a LoRa network.

Similar to the first experiment, nodes periodically send packets to the network, but here the number of sink nodes is varied between **1, 2, 3, 4, 6, 8, and 24**. Nodes are distributed over a larger rectangular area to reflect the extended coverage required when multiple sinks are deployed. Each sink is placed strategically along predefined lines within the deployment area to maximize network coverage.

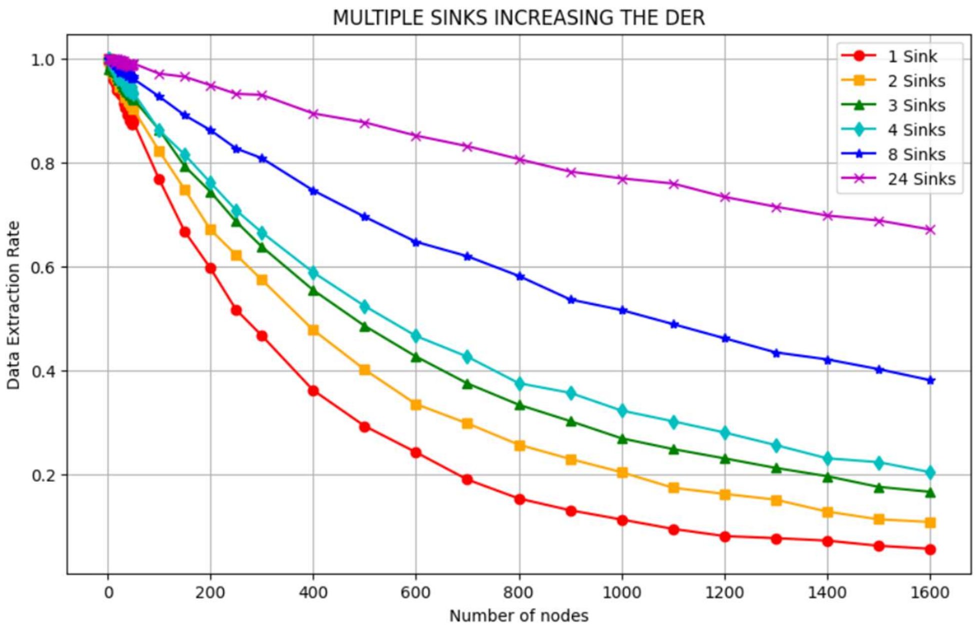
All nodes use the same static transmission parameters, so the communication range remains fixed. The DER is again computed for different numbers of nodes and sinks.

The results reveal that adding more sinks significantly increases the DER, even without changing individual node configurations. With one sink, the DER quickly degrades as the number of nodes increases. However, with three sinks, the network can maintain a DER greater than 0.9 even with 200 nodes. With eight sinks, more than 600 nodes achieve a DER above 0.9. The findings highlight that multiple sinks not only reduce congestion but also exploit the capture effect: when a collision occurs, having several sinks increases the chance that at least one sink will successfully receive a packet.

In order to reproduce the plot of the experiment has been used as subprocess the ***loraDirMulBS.py***

In this case the function provided in the simulator wasn't enough. We had to rewrite it to be capable of compute different simulations for each number of sinks. We used **exp = 0** configuration, that means that nodes transmit according to the basic parameters without any dynamic adjustment like minimizing airtime or power. We also added collision = 1 parameter in order to consider possible collision between packets during the communication. Also this code can be found inside the ***LoRasim.ipynb*** uploaded.

Here is showed the plot obtained


MULTIPLE SINKS INCREASING THE DER
Data Extraction Rate
Number of nodes
1 Sink
2 Sinks
3 Sinks
4 Sinks
8 Sinks
24 Sinks