POLITECNICO DI TORINO

Master's Degree in Electronics Engineering

Master's Degree Thesis

LoRaWAN for Air Quality Monitoring System

Prof. MAURIZIO REBAUDENGO **Amir Nagah ELGHONAIMY**

March 2021

Summary

This Thesis is an extension development work to the project that exists in[1] which aims to measure the air quality using some distributed sensor nodes that depend on Particulate Matter, Temperature, Relative Humidity, and Pressure sensor.

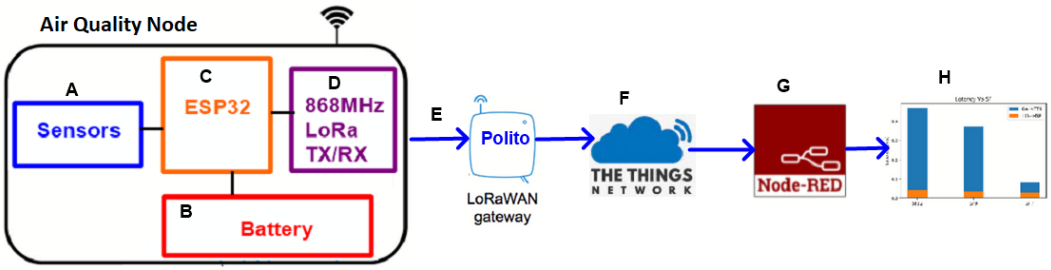


Figure 1: The System Block Diagram

Looking at figure 1, this study project focus mainly on Parts C, D, and E which are the physical and MAC layer's properties and implementation, also we will give attention to the practical implementation of F(The Things Network(TTN) configurations), G(Data Storage) and H(Data Visualization).

The sensor node in the meantime is transmitting its data (block D) over a "WiFi" connection which has a high throughput (supports real-time data monitoring) but it suffers from relatively short-range communications and it consumes a lot of the battery power of the station. The Thesis is a study for an alternative Wireless technology that can replace the "WiFi" efficiently by extending the transmission range and still able to have a sufficient throughput, latency, and consumes lower power. We choose the LoRaWAN technology for our project after examining the available LPWAN technologies in the market. LoRaWAN satisfies our system-critical parameters which are: A maximum communication range that can reach 0.6Km. The Maximum update time for the sensor data is 5 minutes(300 seconds). Technology's latency should not push the transmission to violate the update time. and advantage of LoRaWAN over other competitors is that it is working in the free ISM Band.

The hardware that was used is the Pycom-FiPy board which is based on the

ESP32 µController combined with the SFX172 LoRa chip (LoRaWAN stack - Class-A).

The complete system looks as was shown in figure 1, the TTN was used as the web-server for its simplicity and free of charge. Node-Red was used as the application-server to store the data on a CSV file.

A field test was conducted in the city of Turin, the range test was made using two different packet lengths, 20 bytes (assuming all the possible data that can be included) and 51 byte which is the maximum payload that can be used with the TTN network at high-speed transmission Spreading Factor(SF) 7. we also used 3 different SF to examine the behavior of the LoRa modulation characteristics in a real situation.

Four different points were defined on the city of Turin which simulate Line of sight and non Line of sight scenarios. The range measurement was quantified using the Gateway(GW) registered under the name "polito" on the TTN website.

Range

the test showed that LoRa is indeed a long-range modulation (over 2Km) in urban areas, but it also showed that it suffers from the surrounding environment (metal, moving vehicles..etc), whoever this can become over by using more Gateways.

Throughput and update time

TTN restriction was a challenge for our packet since we were only allowed to send 417Msg/Day for 20 bytes, considering this limitation we end up using only SF=7 that achieves a message every 3.5 minutes. Then we suggested some techniques to reduce the packet size like skipping the redundant information (Like ID and Time) and introduced the concept of the Sensors States Byte(SSB) which can be seen in Figure 2, using those techniques we were able to reduce the update time of transmission from 3.5 minuets to 2.2 minutes for TTN web-server (few seconds for a private web-server).

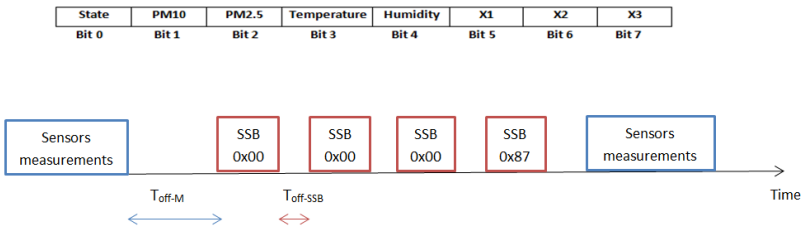


Figure 2: The Sensors States Byte and an example of pattern transmission

Latency

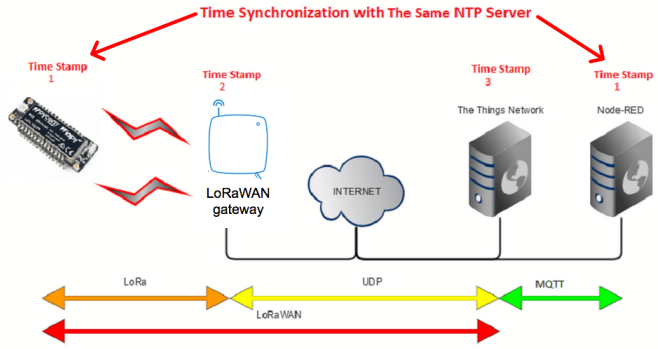


Figure 3: Timing of a LoRaWAN Packet

The system as shown in figure3 using the same Network-Time-Protocol(NTP) server to sync the sensor node and the Node-Red clocks so results become more accurate(since we can't control the time source of the GW and TTN). Subtracting the timestamps of every stage we get the latency. to avoid negative timing we took the average of those measurements.

SF	TOA Latency	Other Latency components
7	100(milliseconds)	0.1(milliseconds)
9	330(milliseconds)	0.35(milliseconds)
12	2400(milliseconds)	0.45(milliseconds)

Table 1: Latency components for every SF

The Conclusion is that LoRaWAN can be used as an LPWAN technology for the Node but using TTN web-server, the transmission must be restricted to only SF=7 with an update time = 2.2 min. changing the web-server to a private one would give a big restriction-free on the duty cycle that will be reduced from 3.3 minutes to 5 seconds.

Acknowledgements

ACKNOWLEDGMENTS

It was tough but it worth it.....
Thank you Wael for all the helps and time, Osama my best friend who keep giving me positive support.Edoardo giusto for assisting me and give me valuable hints about how to organize my thesis, Mohammad ghazivakili for introduce me to Node-Red which made a lot of things easier and Gustavo Ramirez for the technical guidance you gave me when i get stuck. Thanks to Italy for such a wonderful and valuable experience to study in the poleticnico Di Torino .

Table of Contents

List of Tables IX

List of Figures X

Acronyms XIV

1 Introduction 1
1.1 LPWAN . . . 1

2 LoRa and LoRaWAN Technology 4
2.1 LoRa vs. LoRaWAN . . . 4
2.1.1 LoRa Modulation (Physical Layer) . . . 4
2.1.2 LoRa Parameters . . . 5
2.1.3 Coding Rate(CR) . . . 8
2.2 LoRaWAN (MAC Layer) . . . 9
2.2.1 LoRaWAN Architecture . . . 9
2.2.2 LoRaWAN Packet Format . . . 12
2.2.3 Time On Air and Duty Cycle . . . 14
2.2.4 LoRaWAN Power Consumption . . . 15
2.2.5 LoRaWAN Classes . . . 17
2.3 Authentication And Encryption . . . 19
2.3.1 Authentication with the Network Server . . . 19
2.3.2 The Application Serve data Encryption . . . 20
2.3.3 The Full LoRaWAN Frame . . . 20
2.4 Activation of LoRa Devices: ABP or OTAA . . . 21
2.4.1 Activation By Personalization(ABP) . . . 22
2.4.2 Over The Air Activation(OTAA) . . . 22
2.5 LoRaWAN networks and servers . . . 25
2.5.1 The different types of networks . . . 25
2.6 LoRa Limitations . . . 26
2.6.1 ISM Band Limitations . . . 26

2.6.2 LoRa Technology Limitations 26

3 Design and Implementation 28
3.1 The System Design 28
3.1.1 Architecture 28
3.1.2 Components 29
3.1.3 TTN Limitations 37
3.1.4 Node-Red 38
3.1.5 Firmware Flowchart 40

4 Measurements and Results 42
4.1 The System Criteria 42
4.2 Test Scenarios 43
4.2.1 The Size of The Payload 43
4.2.2 Measurements Points Geo-locations 44
4.2.3 Test setup 44
4.2.4 Range Test 48
4.2.5 Throughput of the System 51
4.2.6 Byte-Rate Suitability For The Task 52
4.2.7 Can we do better ? 54
4.2.8 The Latency 58

5 Conclusion and future work 64
5.1 Conclusion 64
5.2 Future Work 65

A A 66

B B 76

Bibliography 82

List of Tables

1 Latency components for every SF iv

1.1 LPWAN Comparison 3

2.1 comparison between a private network and an dedicated network . . 26

4.1 Test Points of the Experiment 45
4.2 Short Payload Packet configurations 48
4.3 Long Payload Packet configurations 49
4.4 The Evaluated bit-rate for different SF 52
4.5 Timing Analysis for 20 byte Payload 54
4.6 Encoding of the Last 3 LSB of the SSB 56
4.7 Transmission of measurement Packet(10+13) and SSB(1+13) 57

List of Figures

1 The System Block Diagram ii
2 The Sensors States Byte and an example of pattern transmission . . iii
3 Timing of a LoRaWAN Packet iv

1.1 Famous wireless Technologies[5] 2

2.1 The LoRa Chirp Signal characteristics[7] 5
2.2 Real LoRa Signal[8] 5
2.3 LoRa Chirps with different SF[7] 6
2.4 An un-modulated LoRa Chirp for BW= 125KHz 6
2.5 Every Chirp is divided into 2^2 chips 7
2.6 Influence of the Coding Rate on the number of added bits[[10]] . . . 9
2.7 LoRa and LoRaWAN on the OSI model [11] 9
2.8 LoRaWAN Architecture [12] 10
2.9 The Role of LoRaWAN Gateway 11
2.10 Network Session Key Authentication 11
2.11 Application Session Key Encryption 12
2.12 Lora Packet format 12
2.13 LoRaWAN Simple Packet format 13
2.14 Spectrum Analyzer view of the Frame[6] 13
2.15 Example of the format for a real LoRaWAN Packet 14
2.16 LoRa Duty Cycle restriction 14
2.17 Table – LoRaWAN spreading factor with TOA for 64-byte payload For a B.W. = 125 kHz for 500-meter distance [11] 15
2.18 the air time and fair access policy for different data rates for empty payload packet 15
2.19 Energy Consumption by Device Class[18] 16
2.20 Class-A transmit/receive profile [19] 17
2.21 Class-B transmit/receive profile[19] 18
2.22 Class-C transmit/receive profile[19] 19
2.23 The MIC and frame authentication by the network server 20
2.24 Using the AppSKey to encrypt/decrypt the Data 20

2.25 Encryption followed by the Authentication process 21
2.26 Configuration of DevAddr, NwkSKey and AppSKey in ABP 22
2.27 Configuration parameters before the Join Request (OTAA) 22
2.28 Configuration parameters after the Join Request (OTAA) 23
2.29 Join Request - Join Accept in OTAA 24
2.30 The Infrastructure of a private LoRaWAN network 25
2.31 Infrastructure of a dedicated LoRaWAN network 26
2.32 LoRaWAN Duty Cycle and EIRP limitations for Up-Link and Down-link [20] 27

3.1 "Polito" Gateway on Google map 29
3.2 The LoRaWAN System Architecture 29
3.3 TTN Map of the registered Gateways (near the test position in Torino City) 30
3.4 FiPy board and its expansion board 31
3.5 FiPy System block diagram[21] 32
3.6 Lora 868MHz Antenna 32
3.7 TTN Console 33
3.8 registering our application in the TTN applications 34
3.9 Payload in hexadecimal representation 34
3.10 Payload decoder function 35
3.11 Payload in human readable format 36
3.12 An encrypted LoRaWAN Packet 36
3.13 The Metadata of the received packet 37
3.14 Node-Red Flow 38
3.15 Node-Red Flow for extracting the up-link messages from TTN . . . 39
3.16 Tracking the timing of the LoRaWAN packet 40
3.17 LoRa Up-Link transmission Flowchart 41

4.1 Map of the Test Points in the city of Turin[Google Earth] 44
4.2 Same Packet received by two gateways 45
4.3 The un-trusted gateway's location w.r.t sensor node 46
4.4 Packet Deliver Rate with different test points 48
4.5 PDR for the Long payload 49
4.6 Bit Rate for SF = 7 51
4.7 Evaluating the Bit-rate for every SF,(BW = 125Khz) 53
4.8 LoRaWAN Timing for 20 bytes Payload packet 54
4.9 An Example of our suggested Sensors State Byte 55
4.10 The 1% duty cycle allows the transmission every 5 seconds for data size = 1byte using SF7 56
4.11 An Example of the transmission pattern of the sensor data 57

4.12 Timing of a LoRaWAN Packet 58
4.13 the data as constructed by Node-Red 59
4.14 Latency algorithm for constructing the Packet 61
4.15 Latency algorithm for analyzing the data of the received packet . . 62
4.16 The TOA contribution part to the latency 63
4.17 Other Latency components 63

Acronyms

ABP
Activation by Personalization

ACK
Acknowledgment

BLE
Bluetooth low Energy

bps
bit per seconds

BW
Band Width

CRC
cyclic redundancy check

CR
Code Rate

CSS
Chirp Spread Spectrum

csv
Comma Separated Values

DevAdd
Device Address

DL

Down-Link

DC

Duty Cycle

EIRP

Effective Isotropic Radiated Power

ETSI

The European Telecommunications Standards Institute

FEC

forward error correction

Km

Kilo Meter

GW

GateWay

LOF

Line of Sight

MSG

Message

npreamble

Number of Preamble symbols

NTC

Negative Temperature Coefficient

NTP

Network Time Protocol

NLOS

Non Line of Sight

MQTT

Message Queuing Telemetry Transport

Min

Minutes

MPD

Messages per day

SFD

start frame delimiter

SNR

Signal to Noise ratio

SF

Spreading Factors

SP

Server Provider

ISM

Industrial, scientific and medical

IoT

Internet of Things

IP

Internet Protocol

The Symbol duration

The Chip duration

TOA

Timer On Air

TTN

The Things Network

TP

Test Point

OTAA

Over the Air Activation

OSI

Open Systems Interconnection

LPWAN

Low Power Wide Area Network

LoRa

Long Range

LOS

Line of Sight

PDR

Packet Deliver Rate

RTC

Real-time clock

UI

User Interface

UL

UP-Link

WS

Weather Station

Chapter 1

Introduction

Recently, the fast spread of COVID-19 has threatened all humanity around the globe. After the start of its first wave (around Nov-2019), a lot of studies were conducted to understand the new pandemic's characteristics, causes, and effects. A study [2] shows a relation between air pollution (NO2 levels) and air currents in Italy with confirmed deaths related to COVID-19, another study from Harvard analyzed air pollution and COVID-19 deaths concluded "A small increase in long-term exposure to PM2.5 leads to a large increase in the COVID-19 death rate." [3, 4]. The advancements that happen in recent years on many aspects of technologies like Embedded systems, Low power wireless technology, and the sensor's design had lead to a new era of systems to appears that are able to sense the environment around and send the data for a long-distance using only a portable battery and those devices are able to work for several years, from this advancements came the LPWAN.

1.1 LPWAN

The modern consistency development of Engineers for new wireless technologies had led to upraise the Low Power Wide Area NetworkLPWAN technology which becomes the workhorse for the Long-range communications for almost all the modern IoT systems for its tremendous capabilities to achieve the communications between sensors that are separated apart with long-distance (that sometimes rise up to hundreds of kilometers) surviving on the available batteries' technologies, this opens the doors for whole new applications and services.

The LPWAN has some attributes that define it from other wireless technologies as mentioned before but it'll summarize here as it's the main characteristic of the systems that are based on this technology [2]:

1. **Long Range**: The operating range of LPWAN technology varies from a few

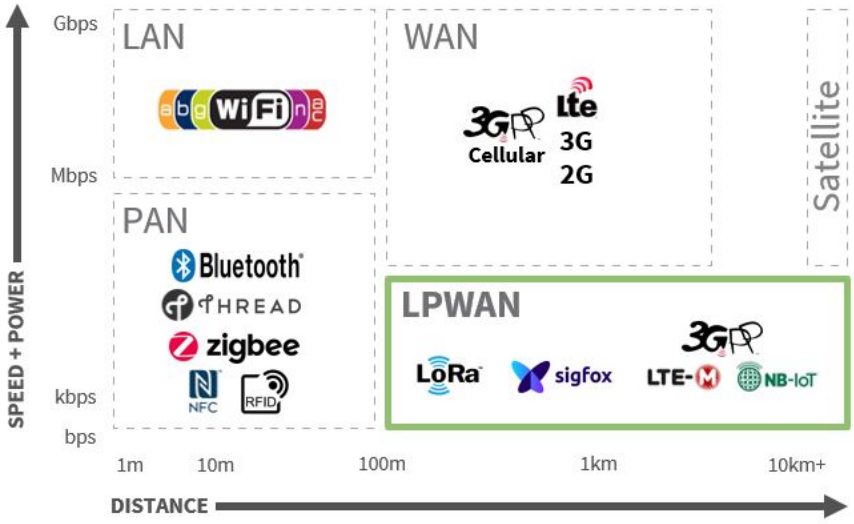


Figure 1.1: Famous wireless Technologies[5]

kilometers in urban areas to over 30 km in rural settings. It can also enable effective data communication in previously infeasible indoor and underground locations.

2. **Low Power**: Optimized for power consumption, LPWAN transceivers can run on small, inexpensive batteries for up to 10 years.

3. **Low Cost**: LPWAN's simplified, lightweight protocols reduce complexity in hardware design and lower device costs. Its long-range combined with a star topology reduces expensive infrastructure requirements, and the use of license-free or licensed bands reduces network costs.

4. **Low Data Rate**: Optimized to communicate for very small data rates, for a few Euro cents a month.

To be able to make a suitable choice we have to compare those technologies in terms of their specifications, we will make a comparison as in Table 1.1.

	NB-IOT	Sigfox	LoRaWAN
Range(Km)	1(urban), 10(rural)	10(urban), 40(rural)	5(urban), 20(rural)
Channel BW(Hz)	200k	100	250k and 125k
Data-Rate(bps)	200K	100	50K
Frequency(MHz)	LTE	433, 868, 915	433, 868, 915
Bidirectional	Yes/Half-duplex	Limited/Half-duplex	Yes/Half-duplex
Max.payload(bytes)	1600	12(UL), 8(DL)	243
Maximum msg/day	Unlimited	140(UL), 4(DL)	depends on web-server

Table 1.1: LPWAN Comparison

Besides the added cost to the NB-IoT, it was logistically not easy to have this service at the time of writing this thesis. We go with the LoRaWAN (only) as it excels over the SigFox in many areas.

The organization of this thesis is as follows: Chapter 2 discusses the characterization and Limitations of the LoRaWAN technology. Chapter 3 describe the testbed and the configurations the web-server and the application-server that will be used during the field-test. Chapter 4 will demonstrate the results and the measurements that had been captured during the field-test and finally Chapter 5 is the conclusion and suggestion for future work to improve the system.

Chapter 2

LoRa and LoRaWAN Technology

In this section, we are going to discuss LoRa and LoRaWAN Technology which are used to establish communications from the Sensor nodes to the Application Server.

2.1 LoRa vs. LoRaWAN

LoRa and LoRaWAN are sometimes get misled, in this section we're going to define each one of them and discuss the main characteristics for those different layers but we can generally say that LoRa is a modulation that is used to send data between a transmitter and a receiver while LoRaWAN protocol allows a LoRaWAN Device to transmit data to a LoRaWAN server.

2.1.1 LoRa Modulation (Physical Layer)

LoRa(Trademark of Semtech) is a Modulation technique that is derived from the Chirp Spread Spectrum(CSS) modulation, hence it describes the physical layer (bits layer implementation) in the OSI Model. The protocol was designed to satisfy the requirements of the IoT battery enabled devices (Low Power Consumption Protocol). One of the known characteristics of the CSS is the ability to establish long-range communication with high resilience to interference. A single Gateway can cover entire cities or hundreds of square Kilometers[6].

Although the technology range mainly depends on the environment and is severely affected by the obstacles[9], However according to Semtech (SX1272/73 DataSheet[20]) LoRa modulation can achieve a link budget of 157 dB. Figure 2.1. shows how a LoRa chirp looks like. LoRa operates in the ISM band over 433, 868 MHz (in Europe), and 915 MHz.

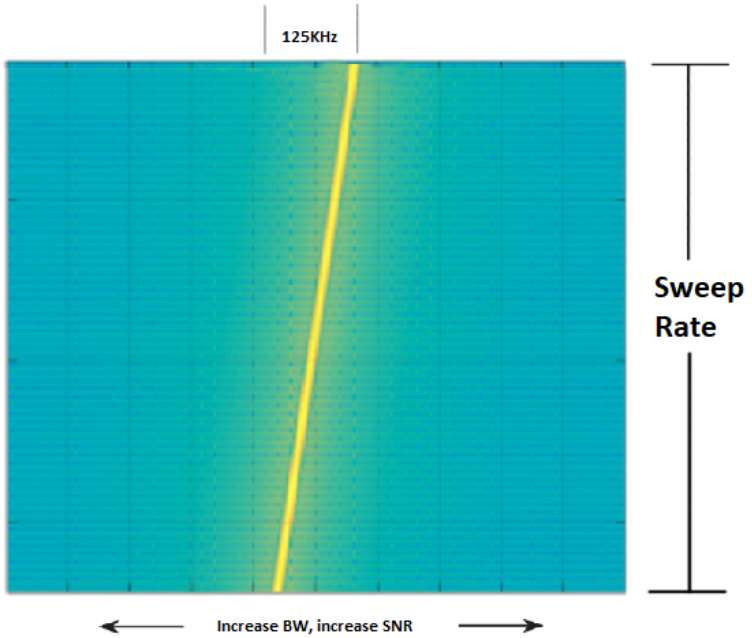


Figure 2.1: The LoRa Chirp Signal characteristics[7]

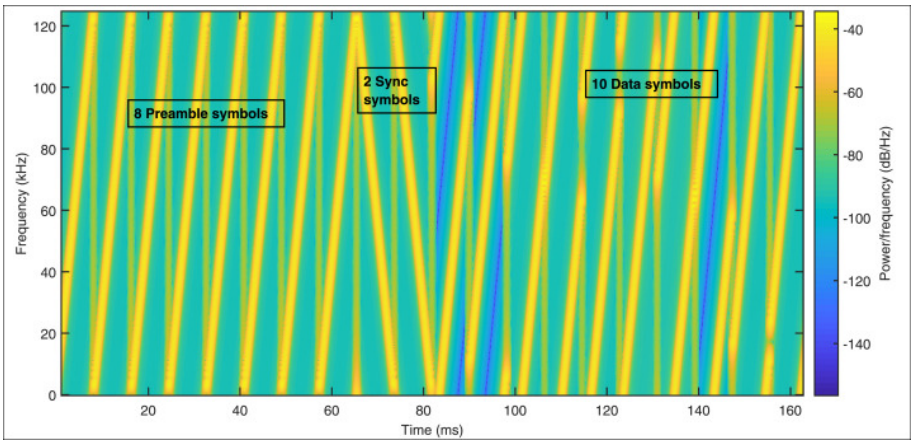


Figure 2.2: Real LoRa Signal[8]

2.1.2 LoRa Parameters

In LoRa, Chirp and symbol (both) are used to refers to the Modulated Signal(which carries the Data). The transmission of the LoRa signal can be characterized by several parameters Like the Spreading Factors SF which are defined to control the bit rate, improve the range and decrease energy consumption, also the Bandwidth(BW), Frequency channel, and the Transmission Power, all those parameters can be

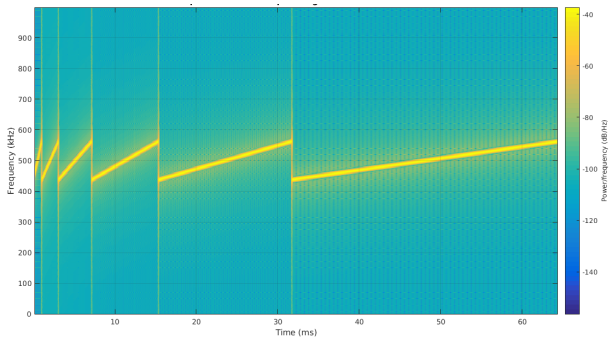


Figure 2.3: LoRa Chirps with different SF[7]

controlled in the physical layer.

Chirp Chips and Spreading Factor

To clarify the two technical terms, Chirp(symbol) is the signal that sweeps through the whole transmission BW of the LoRa Signal.

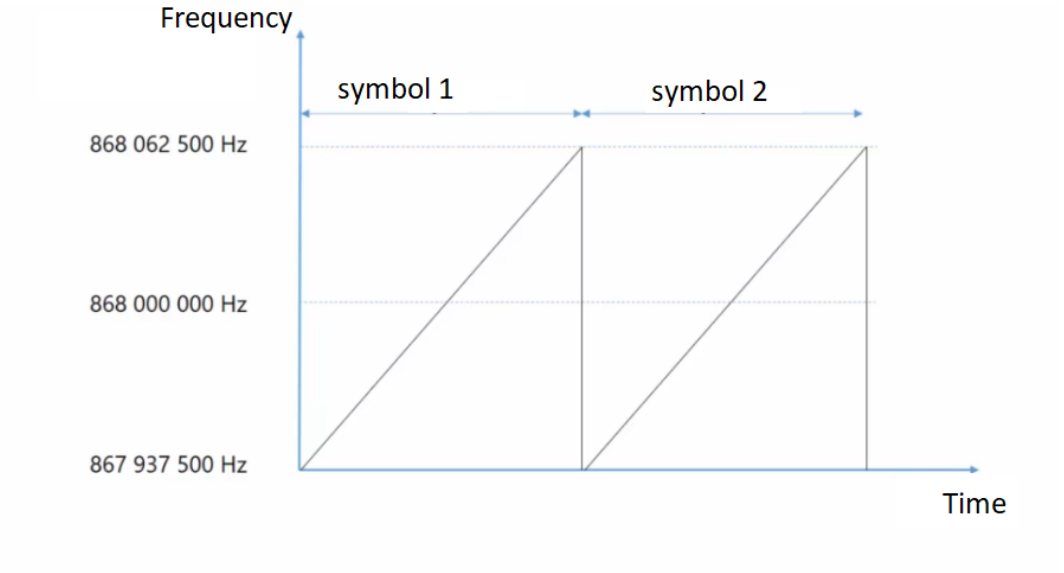


Figure 2.4: An un-modulated LoRa Chirp for BW= 125KHz

The chirp could be an up-Chirp as in 2.1 or a Down-Chirp, depending on the Spreading Factor, a LoRa symbol can hold several bits equal to the SF. If SF=7 this means a LoRa symbol can hold 7 raw bits, for example, 0011010(26 in decimal).

This means that the symbol can be encoded with 127 values(for SF7), or we can say the Chirp is divided into 127 chips.

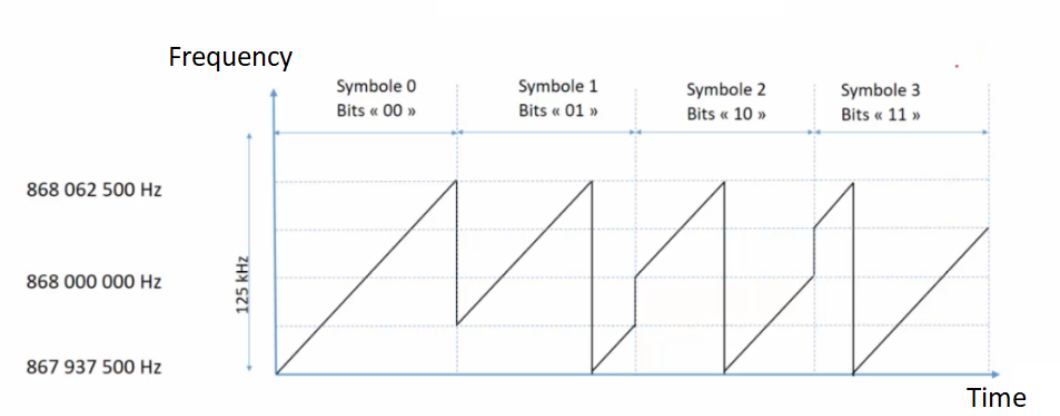


Figure 2.5: Every Chirp is divided into 2^2 chips

Just for simplicity, Figure 2.5 demonstrate a Modulated Up-Chirp with 2 bits which shall make a symbol that holds 4 chips. The minimum LoRa raw bits a symbol can hold is 7 and the maximum is 12.

SF is on the angle of the chirps(how to spread out the chirp would be). It indicates how many bits the chirp includes. It also determines the data-rate and it has a direct effect on the transmission distance (Fixing the transmitting power and B.W), for example, SP=7 means the chirp represents 7 bits hence 2^{SF} number of starting and ending frequency positions. **Increasing the SF by one will result of doubling the time it takes to transmit a chirp**(assume all other parameters remains the same), by doubling the chirp time, receivers will have more time to sample the received signal and hence an increase in the Signal to Noise ratio(SNR) which means better reception and longer distance (but the downside is the power consumption)[9]. Observing a LoRa signal on spectrogram we'll see a signal like in Figure 2.2 which shows a combination of up-chirps and down-chirps (represents the preamble, sync, and the data). LoRa supports six different spreading factors ranging from SF7 to SF12, with SF7 having the highest data-rate, Lowest range, Energy consumption, Time on Air, and highest Sensitivity(For an SF7 signal, the receiver needs a higher SNR).
The Spreading factor defines two values:

1. The Number of Raw bits that can be encoded by that symbol.

2. Each LoRa symbol can hold 2^{SF} chips.

The bit Rate can be calculated by

$$R_b(bit/sec) = SF \times \frac{BW}{2^{SF}} \times \frac{4}{(4+CR)} \tag{2.1}$$

Where:

1. SF: Spreading Factor (7-12)
2. BW: Bandwidth (Hz) or Chips/Sec
3. CR: 1-4 (Explained in2.1.3)

Every Chirp in the LoRa Signal contains 2^{SF} Chip, if we want to know how to calculate the Time a LoRa symbol would take then we can use the following equation:
First, we find the chip duration time

$$T_c(sec) = \frac{1}{BW} \tag{2.2}$$

Then by knowing the SP that we use in the transmission, the Chirp(Symbol) duration would be:

$$T_s(sec) = \frac{2^{SF}}{BW} \tag{2.3}$$

from Eq 2.3 we can see the direct effect of the SF and BW on the symbol timing which will have a big roll in controlling the TOA for a LoRaWAN Packet which is a very important parameter that is considered one of the Limitations of using LoRaWAN Network (TTN in our Case).

2.1.3 Coding Rate(CR)

LoRa Modulation adds forward error correction(FEC) in every symbol transmission. The Coding Rate is a ratio that adds more bits to the transmitted bits for error detection/correction. LoRa Modulation allows the Code rate for the values of CR = 4/5, 4/6, 4/7, or 4/8. The general formula for the coding rate is: $CRC = \frac{4}{(4+CR)}$ where CRC stands for Cyclic coding rate.

The adding of the FEC bits shall increase the signal immunity against noise but it reduces the transmission throughput and increases the transmission time.

CodingRate (RegModemConfig1)	Cyclic Coding Rate	Overhead Ratio
1	4/5	1.25
2	4/6	1.5
3	4/7	1.75
4	4/8	2

Figure 2.6: Influence of the Coding Rate on the number of added bits[[10]]

2.2 LoRaWAN (MAC Layer)

LoRaWAN is a data-link protocol that manages how LoRa devices enter the network and how they exchange data, how much data-rate can be send/receive, the addresses they use, encryption, and other packet configurations. LoRaWAN could be compared to IP protocol while LoRa could be compared with Ethernet(As a physical layer).

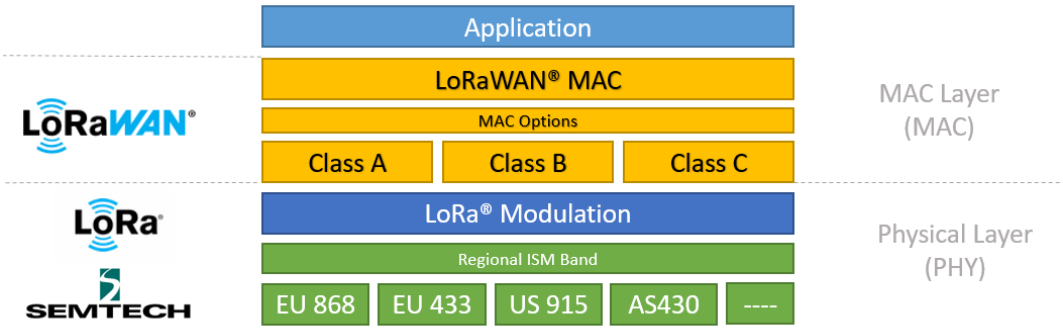


Figure 2.7: LoRa and LoRaWAN on the OSI model [11]

2.2.1 LoRaWAN Architecture

The Architecture of LoRaWAN has a topology of a star of the star (Which enhances a lot the reception of the data, since one packet can be received by many Gateways which increases the "success of packet delivery"). Architecture can be seen as in Fig 2.8.

It is worth to mention that unlike other LPWAN protocols(Like Sigfox), LoRaWAN is a bidirectional protocol that can give a good range of applications (

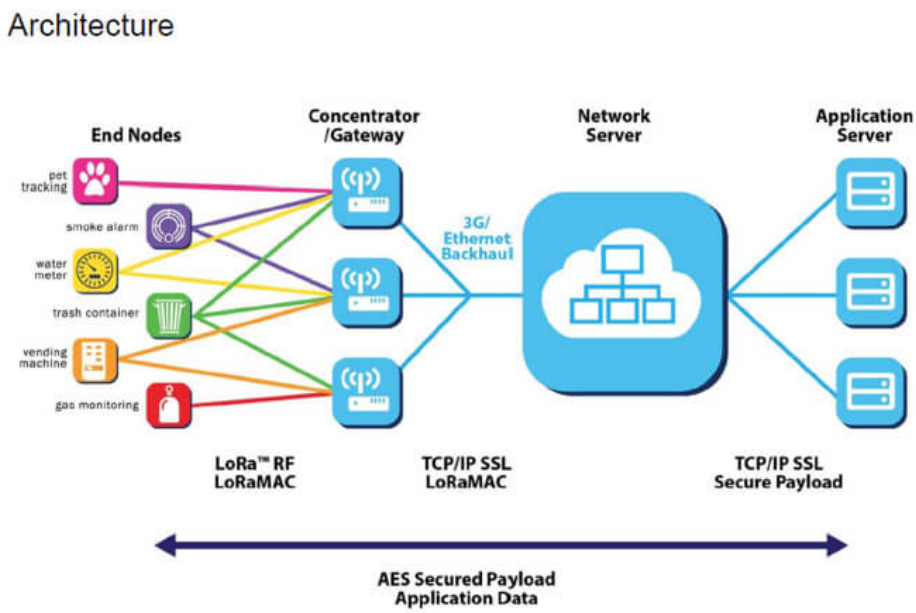


Figure 2.8: LoRaWAN Architecture [12]

and algorithms) that needs two-way communications between the sensor node and the server.

LoRa Devices(End Devices)

They are electronic IoT embedded system that has the features of been low power consumption, small size, and low cost. To transmit through LoRa protocols, those devices must have a LoRa Radio.

LoRa Gateway

They are Electronic devices that have the ability to listen over multiple channels at the same time and on all spreading factors. Once they receive a LoRa frame they transmit its content to the Web-server using the internet. It is the link between the LoRa modulation and IP communication. Each LoRa Gateway has a unique identifier (64-bit EUI). This identifier is useful for register and activates a Gateway on a Network Server (We will talk about that in later chapters) Figure2.9 describes the steps that a LoRa frame went through.

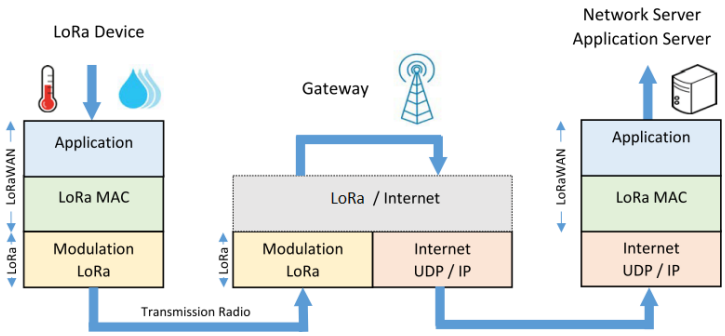


Figure 2.9: The Role of LoRaWAN Gateway

The Network Server

The Network Server receives the message from the Gateways, then it drops the duplicated messages(which came from different Gateways). The LoRaWAN Packet is authenticated using a 128-bit AES key called Network Session Key: NwkSKey.

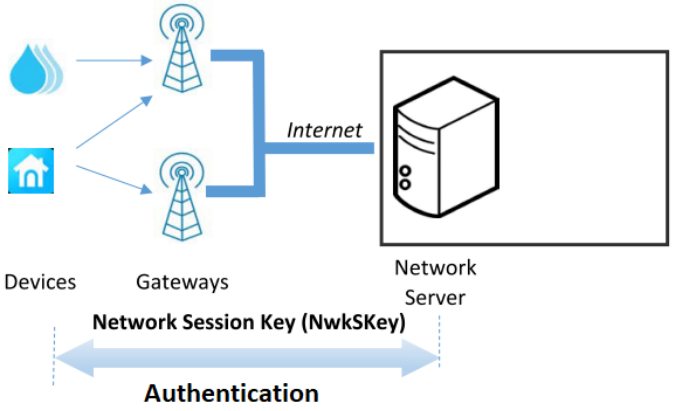


Figure 2.10: Network Session Key Authentication

The Application Server

It separates the applications from each other. The registered LoRa devices can store their data (Frame Payload) using the Application server. The messages here are encrypted using a 128-bit AES key called Application Session Key: AppSKey.

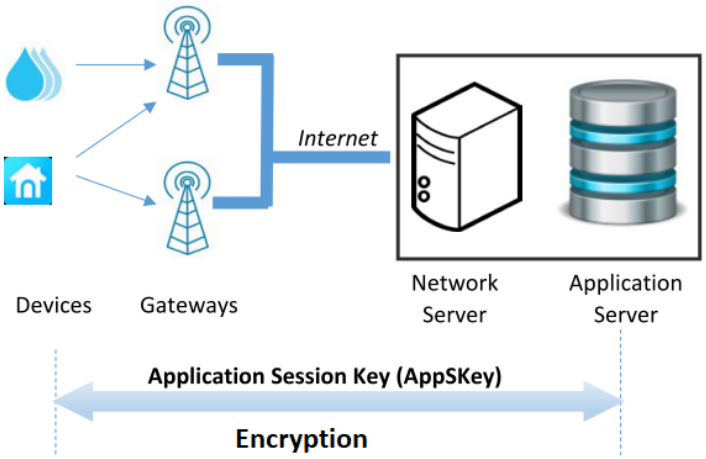


Figure 2.11: Application Session Key Encryption

2.2.2 LoRaWAN Packet Format

We simply don't send the data directly using the LoRa Modulation, instead, there must be a frame for our data. The Frame of the LoRa Packet consists of:

- A preamble to allow the receiver to synchronize.
- Optional header (Used in the explicit mode)
- Data Payload
- CRC fields (checking the integrity of the frame).

The LoRa protocol data is called PHY Payload (physical layer data). a general overview of the LoRa frame would look as in fig2.12 As LoRaWAN is a different

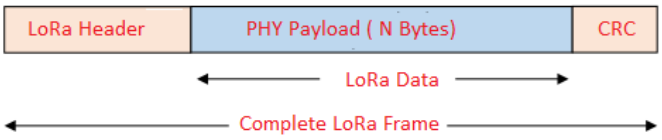


Figure 2.12: Lora Packet format

layer protocol than LoRa then the frame has to be modified. A simple general view for the LoRaWAN frame would look like in fig2.13

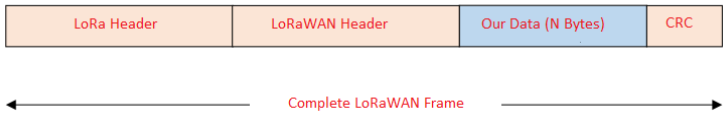


Figure 2.13: LoRaWAN Simple Packet format

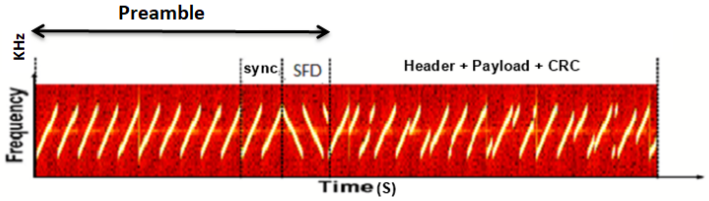


Figure 2.14: Spectrum Analyzer view of the Frame[6]

The Preamble is mandatory for every transmission to synchronize the receiver with the incoming data flow. By default, the packet is configured with 12.25 symbols long sequence[13]:

- 8 configurable symbols (npreamble)
- 2 sync word symbols
- 2.25 SFD symbols

The Structure of the LoRaWAN packet is well-defined in[14], the important thing to know is that sending an empty up-link message (with no payload data) will always have an overhead of 13[1] bytes(2.15) which we must include in our analysis when we want to calculate the Packet size or the Time On Air.

[1]13 bytes due to [MHDR(1) + DevAddr(4) + FCtrl(1) + FCnt(2) + Fport(1) + MIC(4)] in a packet with no options (the payload size is independent from the encryption.)

```
example : 80C02301260021000266EEA76CCE0C1BBC7A36F69F
80      C0230126   00      2100   02      66EEA76CCE0C1BBC   7A36F69F
MTYPE   devaddr    FCtrl   FCnt   FPort   DATA               MIC
```

Figure 2.15: Example of the format for a real LoRaWAN Packet

2.2.3 Time On Air and Duty Cycle

The LoRaWAN standard requires that a LoRa Device does not transmit more than 1% of the time. This is called the Duty Cycle(DC). A Duty Cycle of 1% means that if the device transmits for 1 (time unit), it must stay off for 99% (time unit). This can be represented by $\frac{T_{ON}}{(T_{ON}+T_{OFF})}$ while the Time On-air is the amount of time a packet(or signal) takes during its trip from the transmitter's antenna to the receiver's antenna.

The formulas to calculate the Time on air are:

1. Calculate the Symbol Rate as $R_s = \frac{BW}{2^{SF}}$
2. Calculate the time of one symbol as $T_s = \frac{1}{R_s}$
3. Calculate the Preamble duration as $T_{Preamble} = (n_{Preamble} + 4.25)T_s$
4. Calculate the Number of Payload Symbols as $N_{PayloadSymbols} = 8 + max(ceil(\frac{8PL-4SF+28+16-20H}{4(SF-2DE)})(CR+4),0)$[2]
5. $T_{Packet} = N_{PayloadSymbols} \times T_s$
6. Time on Air for LoRa Packet will be $T_{packet} = T_{preamble} + T_{Packet}$

However we will use the fast approach with the air time calculator in [16]

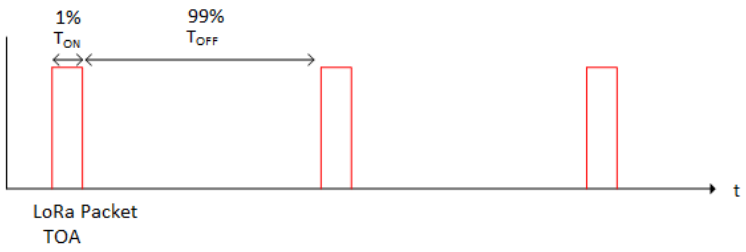


Figure 2.16: LoRa Duty Cycle restriction

[2]for details about the parameters definitions[15, 9].

If the maximum duty-cycle in a sub-band is denoted by "D" and the packet transmission time (Time On Air), each device must be silent in the sub-band for a minimum off-period $T_{off} = TOA(\frac{1}{D} - 1)$. TOA is proportionally affected by the Spreading Factor, longer TOA for higher SF and consequently longer off duration; this can be seen in the table below:

Data Rate (Spreading Factor)	Sensitivity	Time On Air
SF7	-123.0 dBm	41 ms
SF8	-126.0 dBm	72 ms
SF9	-129.0 dBm	144 ms
SF10	-132.0 dBm	288 ms
SF11	-134.5 dBm	577 ms
SF12	-137.0 dBm	991 ms

Figure 2.17: Table – LoRaWAN spreading factor with TOA for 64-byte payload For a B.W. = 125 kHz for 500-meter distance [11]

We're going to use the "Air Time Calculator" in[17] to calculate the TOA concerning its payload size. The TOA for a packet that has no payload (only the overhead = 13 bytes) with different Spreading factors:

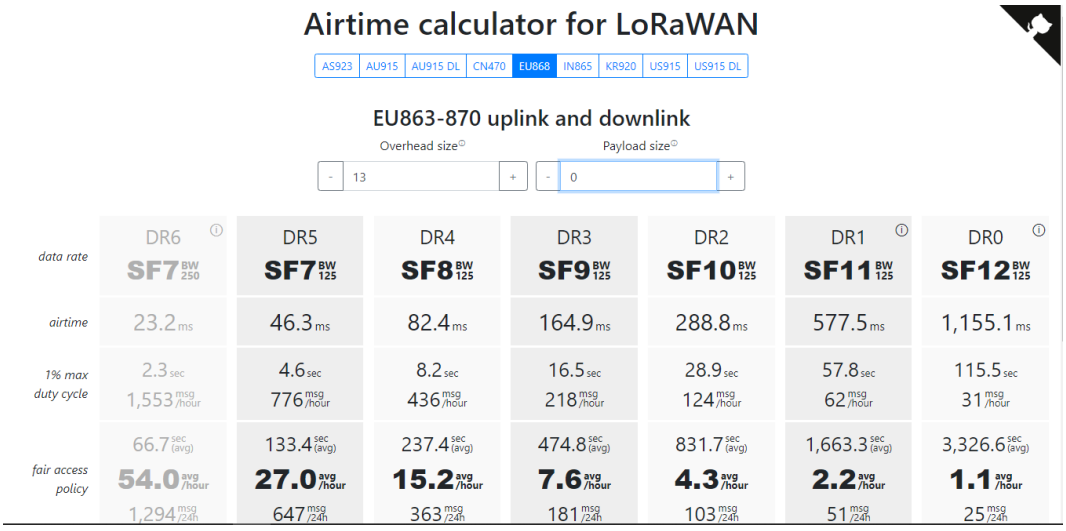


Figure 2.18: the air time and fair access policy for different data rates for empty payload packet

2.2.4 LoRaWAN Power Consumption

The consumption of a LoRa system depends on several parameters:

1. The quantity of data to be sent (Payload).
2. The Spreading Factor.
3. Any collisions during transmission (and therefore retransmission).
4. The request for acknowledgment of the frames sent.
5. The Duty-Cycle.
6. The transmission power of the transceiver.
7. The power consumed in standby between 2 transmissions.

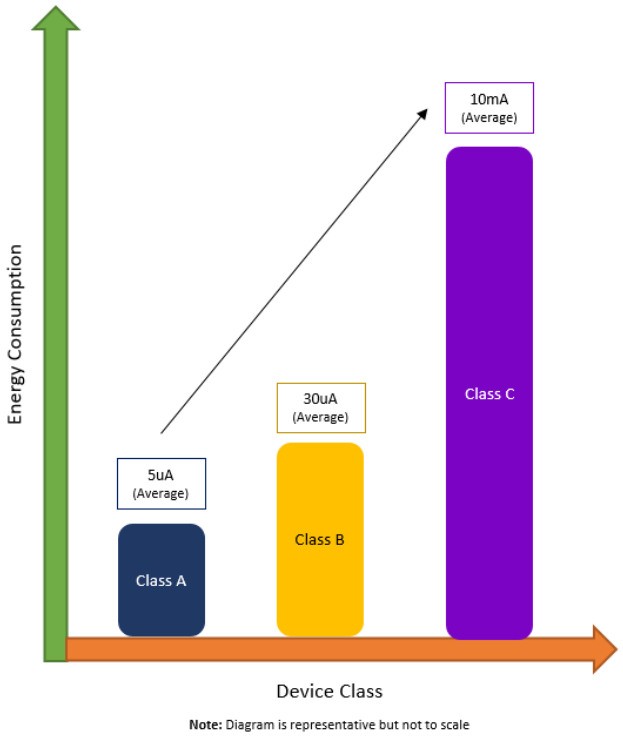


Figure 2.19: Energy Consumption by Device Class[18]

To give a brief idea about the different levels of power consumption for each of the different end-device classes, see Fig 2.19.

2.2.5 LoRaWAN Classes

End Devices in LoRaWAN also come in three classes, based on the profile of each class the reception window for Down-Link messages (from the Network) is defined, and also the Energy consumption(which reflect on the battery's life for the sensor). We are going to talk briefly about Class B, C and give more attention to class-A since it's the default mode for all Sensor Nodes.

Class A (All): Minimal power Application

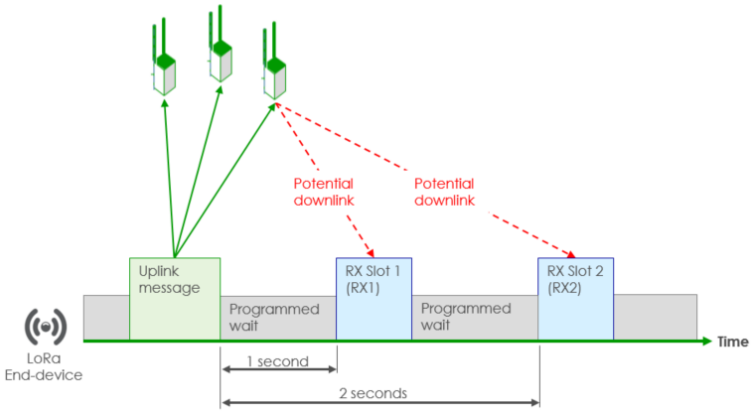


Figure 2.20: Class-A transmit/receive profile [19]

All End devices must support class-A communications (our LoRa module uses that Class for transmission). It follows the "Aloha" transmission scheme which means any device can transmit it is message arbitrarily. The transmission profile of class-A is shown in fig 2.20. Class-A devices are sleeping most of the time, they wake up to transmit their up-link messages after that they start to listen after a programmed waiting time for predefined two consecutive receive windows(Rx1 and Rx2), the waiting time depends on regional parameters and they are defined for Europe to be 1 second for Rx1 and 2 seconds for Rx2. The Gateway can send on Rx1 or Rx2 but not both.

The duration of the Rx windows must be at least the duration of reception of a preamble. A preamble lasts $12.25T_S$(where T_S is the symbol time) and therefore depends on the Data Rate.

First reception window:

1. Slot RX1 is programmed by default at 1 second + / 20 µs after the end of the Uplink transmission.
2. The frequency and the Data Rate (DR) are the same as those chosen during the transmission phase(Uplink).

Second reception window:

1. Slot RX2 is programmed by default at 2 seconds +/- 20 µs after the end of the Uplink transmission.

2. The frequency and the Data Rate (DR) are configurable but also fixed.

The advantage of using Class-A is that it is the most power-efficient class among the three classes but it comes with a downside which is the Downlink messages are queued until the next time an uplink message is received(and an Rx window is open) if the up-link message was not successfully received, also a LoRa Device that only uses Class A cannot receive if it didn't transmit. hence it is not easily reachable.

Class B(Beacon)

Class B Devices behave the same as for Class A Devices, but other Reception windows are scheduled at specific times. To synchronize those Receiving windows, the Gateway must send beacons regularly. A Class B LoRa Device can be contacted regularly without necessarily being an uplink transmission. On the other hand, it consumes more than a class A device.

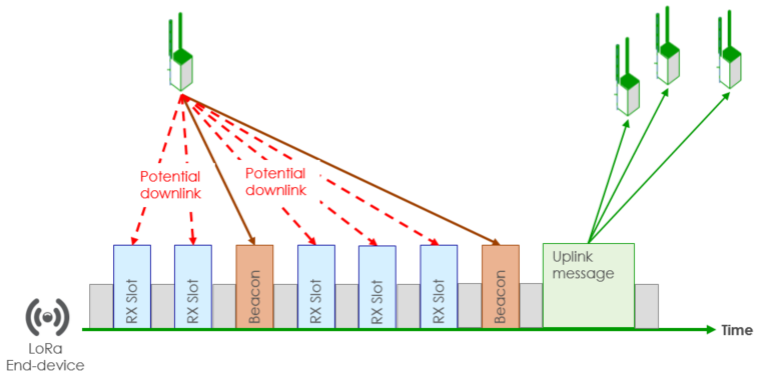


Figure 2.21: Class-B transmit/receive profile[19]

Class C (Continuous)

End devices in Class C mode are used when extremely low power consumption is not an issue, and latency needs to be minimized. The server-side application determines that it is managing class C devices during the join procedure. End devices operating in Class C mode have received windows that are almost always open. These windows close only when the device is transmitting. Because of this,

Class C end devices use more power to operate than Class A or Class B devices. However, in turn, they offer the lowest latency for communication from the server to an end device.

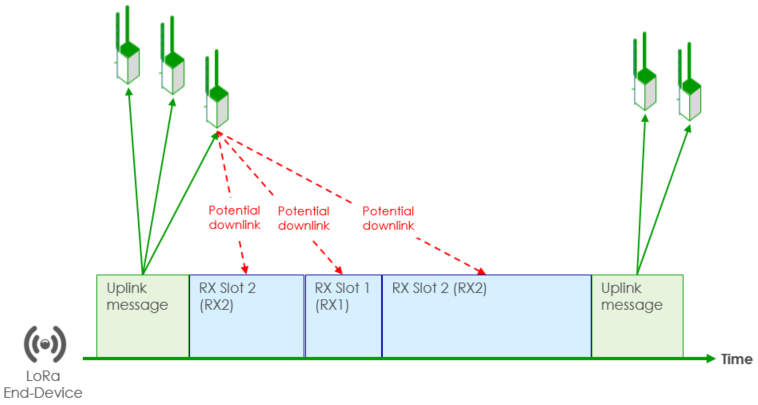


Figure 2.22: Class-C transmit/receive profile[19]

The Gateway used for the Downlink(DL) is the one that received the last message from the LoRa Device. If the LoRa Node never sends an Uplink, it can not ever have a Downlink, regardless of its class A, B, or C.

2.3 Authentication And Encryption

In this section we are going to discuss the issues related to LoRaWAN transmission security.

2.3.1 Authentication with the Network Server

The Network Session Key(NwkSKey) is used to guarantee the authentication between the LoRa Device and the Network Server. To perform this authentication, a MIC field(Message Integrity Control) is added to the frame. It is calculated based on the data transmitted and the NwkSkey. On the receiver, the same calculation is made. If the NwkSkey is equivalent in the Device and the Network Server then the two calculated MICs must match. The process is shown in figure 2.23.

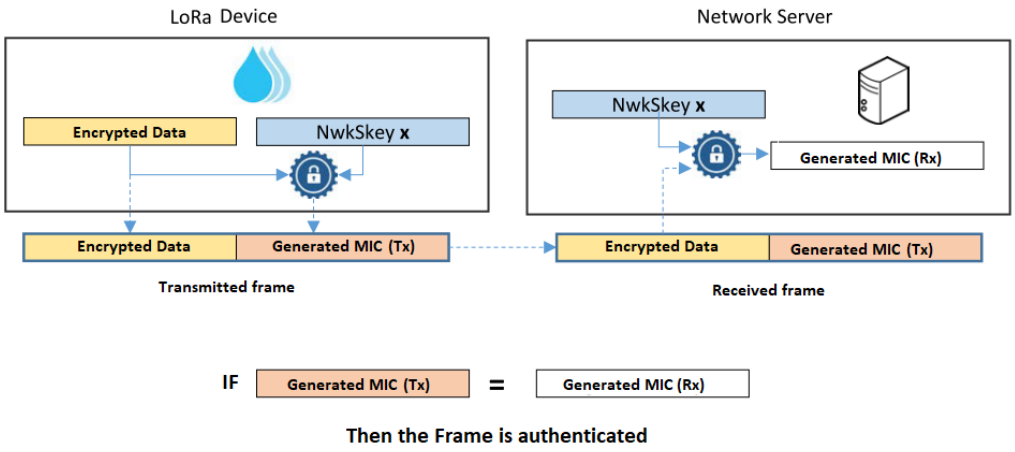


Figure 2.23: The MIC and frame authentication by the network server

2.3.2 The Application Serve data Encryption

The Application Session Key(AppSKey) is used for encrypting the data between the LoRa Device and the Application Server. Only if the data has the same key, it will be decoded. Data encryption/decryption is done as in figure 2.24.

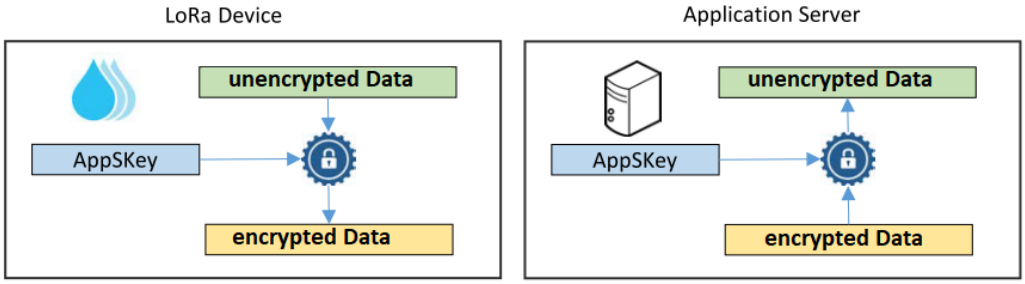


Figure 2.24: Using the AppSKey to encrypt/decrypt the Data

2.3.3 The Full LoRaWAN Frame

The LoRaWAN Frame at the end will look like 2.25.

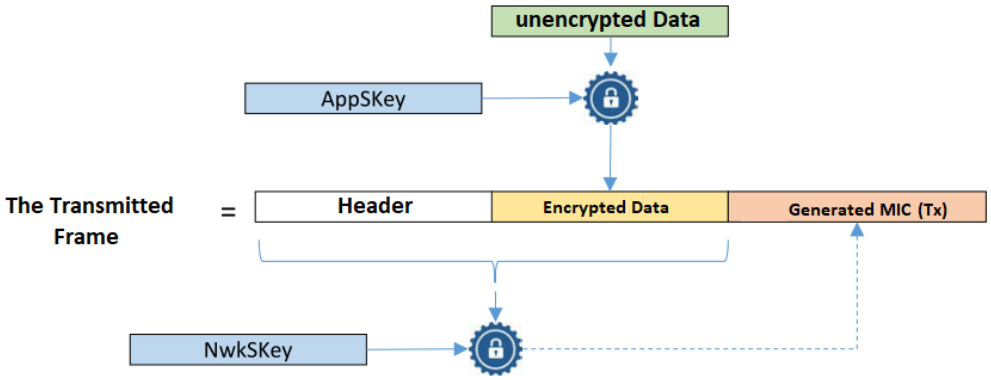


Figure 2.25: Encryption followed by the Authentication process

2.4 Activation of LoRa Devices: ABP or OTAA

In LoRaWAN, there are three essential elements for the communication to happens:

1. The DevAddr for Device identification.
2. NwkSKey for authentication.
3. The AppSKey for encryption.

LoRaWAN offers two methods to share those elements before the data transmission begin:

1. Activation By Personalization(ABP).
2. Over The Air Activation(OTAA).

2.4.1 Activation By Personalization(ABP)

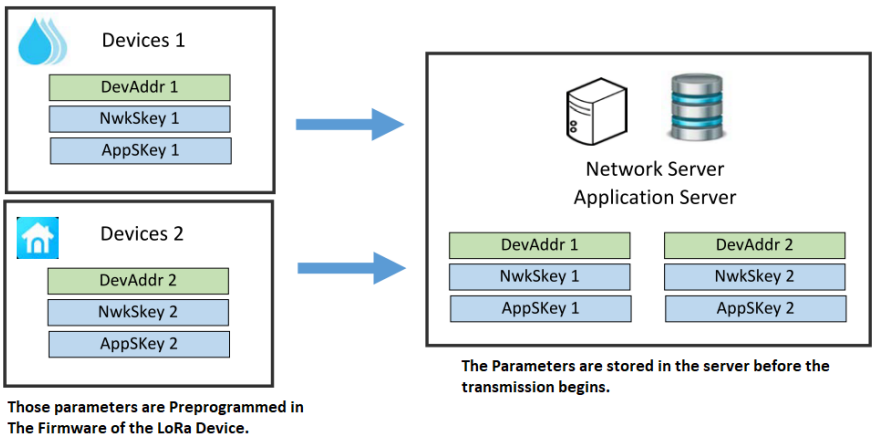


Figure 2.26: Configuration of DevAddr, NwkSKey and AppSKey in ABP

It is the simplest of the two methods. It is very suitable to be used in the case of a LoRaWAN prototype test or communication setup. Its simplicity comes due to the fact that the LoRa Device, The Network server, and the Application Server, both have the DevAddr, the AppSKey, and the NwkSKey before any communication start. This can be shown in figure 2.26

2.4.2 Over The Air Activation(OTAA)

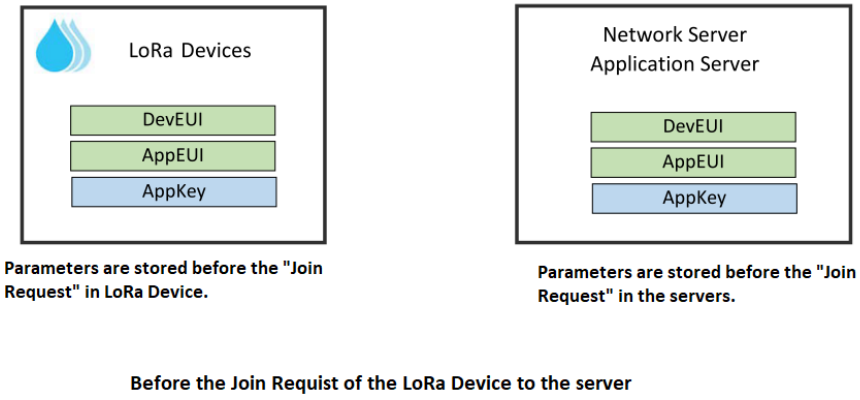


Figure 2.27: Configuration parameters before the Join Request (OTAA)

In the OTAA the DevAddr, the AppSKey, and the NwKSKey will be generated during the join request of the LoRa Device (at the beginning of the transmission)as can be seen in figure2.27.

It happens only once at the 1st communication. But there are other parameters that must be known for both the LoRa Device and the Web Server before the communication starts, those are:

1. DevEUI: It is a unique identifier for the LoRa device Some LoRa Devices already have a factory supplied DevEUI.
2. AppEUI: A unique identifier for the server application.
3. Appkey: An AES 128 key used to generate the MIC (Message Code Integrity) during the Join Request. It is shared with the Network server.

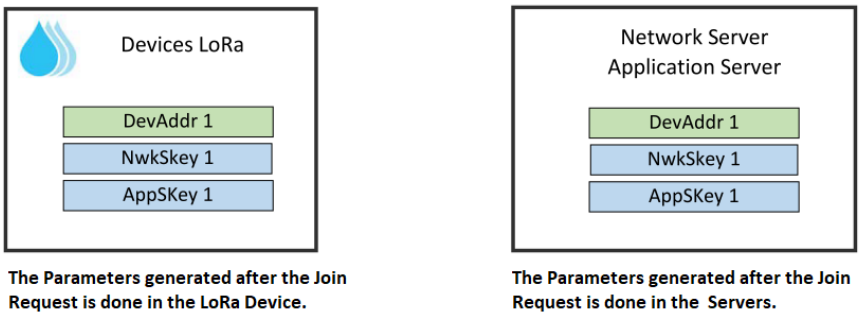


Figure 2.28: Configuration parameters after the Join Request (OTAA)

After the negotiation of the LoRa Device to join the Network Server(Join Request), both the LoRa Device and the server will generate the essential parameters: DevAddr, NwkSKey, and AppSKey as can be seen in figure 2.28.

The generated parameters after the join request are:

1. NwkSKey: Used for authentication with the Network Server.
2. AppSKey: Used for data encryption.
3. DevAddr: Unique 32-bit identifier within a LoRa network.

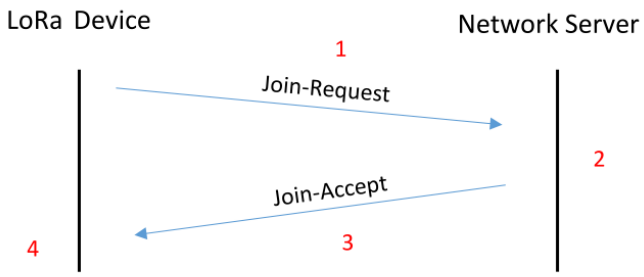


Figure 2.29: Join Request - Join Accept in OTAA

The Scenario of the OTAA joint request as in (2.29) goes like this:

1. The LoRa Device initiates the Join-Request by sending DevEUI, AppEUI, and AppKey information to the server.
2. The Network Server then authenticates the Join Request and validates it. Then it generates a NwkSKey, an AppSKey, and a DevAddr.
3. The Network Server returns the DevAddr, as well as other parameters. Those parameters allow the LoRa Device to generate the same NwkSKey and AppSKey as the Web server.

2.5 LoRaWAN networks and servers

2.5.1 The different types of networks

LoRaWAN networks can be used in three configurations:

1. Using the operated LoRaWAN networks offered by a telecom operators.
2. Using a private LoRaWAN network.
3. Using a dedicated network.

At the meantime there is no telecom operators in Italy that supports LoRaWAN coverage. We are going to demonstrate only the private LoRaWAN networks and the dedicated networks.

Private LoRaWAN networks

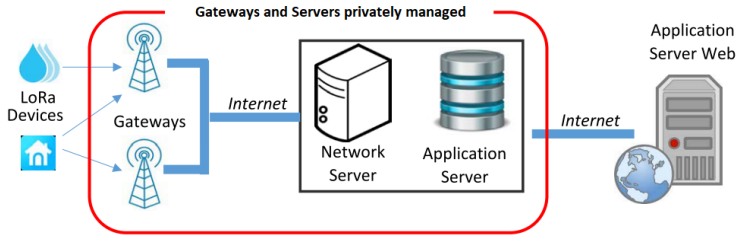


Figure 2.30: The Infrastructure of a private LoRaWAN network

Private networks give the implementation of the Gateway and the servers to the user, so the whole infrastructure to communicate with a LoRa device can be defined by the user. There are some well known open source servers like "Chirp Stack"[`https://www.chirpstack.io/`].

The dedicated LoRaWAN network

In this case, the Infrastructure of the LoRaWAN server is managed by the server provider(SP), while it gives the user the ability to install their own Gateways (and also using other users gateways), it's more like a gateway sharing community. The Things Network(TTN) is an example of that `https://www.thethingsnetwork.org/` and we are going to use it in our test.

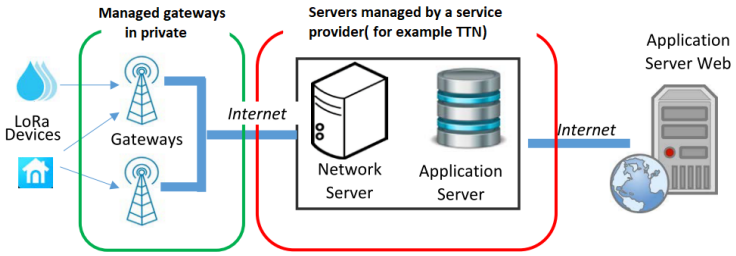


Figure 2.31: Infrastructure of a dedicated LoRaWAN network

	Private networks	Dedicated networks
Infrastructure Cost	GW and servers fees	only GW if user uses his own
Coverage	Optimized to user needs	Depends on the SP
UpLink MSG	Unlimited (but respect 1% DC)	Depends on the SP
DownLink MSG	Unlimited (but respect 1% DC)	Depends on the SP

Table 2.1: comparison between a private network and an dedicated network

2.6 LoRa Limitations

Knowing the Limitations of technology is a key element to define whether or not it's going to satisfy the need or not. And like all other technologies, LoRaWAN has its own pros and cons, and in this section, we're going to mention them.

2.6.1 ISM Band Limitations

According to The European Telecommunications Standards Institute ETSI to work in the ISM Band must follow some rules to avoid the over-crowded and interference when using the band, there are two main restrictions that LoRaWAN must follow for working on this Band which are:

1. 1% Duty Cycle (for the standard sub-Bands)
2. EIRP of 14dBm (25mW) in Europe

2.6.2 LoRa Technology Limitations

1. LoRa has a maximum Data Rate of 11Kb/sec (SF = 7)

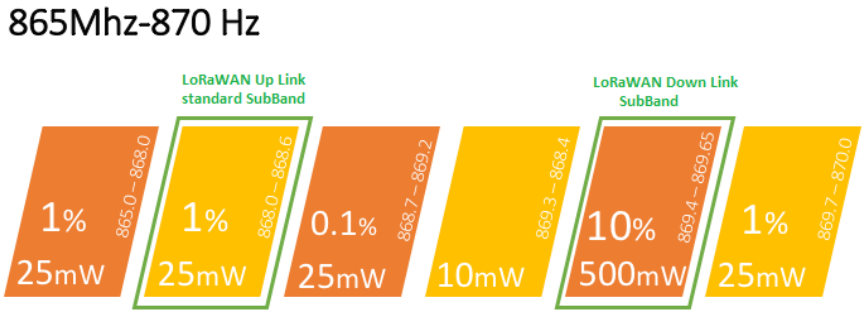


Figure 2.32: LoRaWAN Duty Cycle and EIRP limitations for Up-Link and Down-link [20]

2. Packet size can't exceed 256 bytes/transmission

There are other limitations that are imposed by the network server provider as we will show in the next chapter.

Chapter 3

Design and Implementation

In this section, we're going to describe our System component and their interaction by presenting the system architecture and how the implementation process was done.

3.1 The System Design

We start here by describing the System architecture and the involved components.

3.1.1 Architecture

Our system is composed of:

1. **Sensor Node**: Which is based on the FiPy Board with its Expansion board 3.1 to collect the sensors data and handle the transmission of the packets via LoRaWAN Transceiver (embedded module).

2. **The Gateway**:

   LoRaWAN architecture allows the packets to be received by more than one Gateway, The Gateway "Polito" which is established in the Politecnico Di Torino was considered to be the system gateway for the Tests.

3. **TTN Web Server**: It's a web server that provides open tools and a global, open network to build IoT applications at low cost, featuring maximum security and ready to scale (but that also comes with more limitations).

4. **Node-Red**: It is a visual tool designed for the IoT, It quickly assembles flows of various services. It enables users to stitch together Web services and hardware by replacing common low-level coding tasks, and this can be

Figure 3.1: "Polito" Gateway on Google map

done with a visual drag-drop interface. Various components in Node-RED are connected to create a flow. Most of the code needed is created automatically.

Figure 3.2: The LoRaWAN System Architecture

Figure 3.2 represents the generic interconnection of the elements. The Sensors are sending packets to TTN through the TTN registered gateways (gateways that are registered on the TTN web-server), the locations of the gateways nearby in the city of Turin are shown in figure 3.3, after that, we use the feature that TTN provides which is the integration with the Node-Red through the MQTT Protocol. Once the packet arrives Node-Red platform, it can be stored, analyzed, or visualized with many possibilities.

3.1.2 Components

LoRaWAN Sensors Node

FiPy is the first 5-network development board on the market which is offered by Pycom company. It features WiFi, BLE, LoRa, Sigfox, and LTE-M it's programmed with Micro-python. It's based on the famous ESP32 µConteroller supplied by 3.3V-5.5V, the µConteroller is connected through a high-speed SPI connection to the

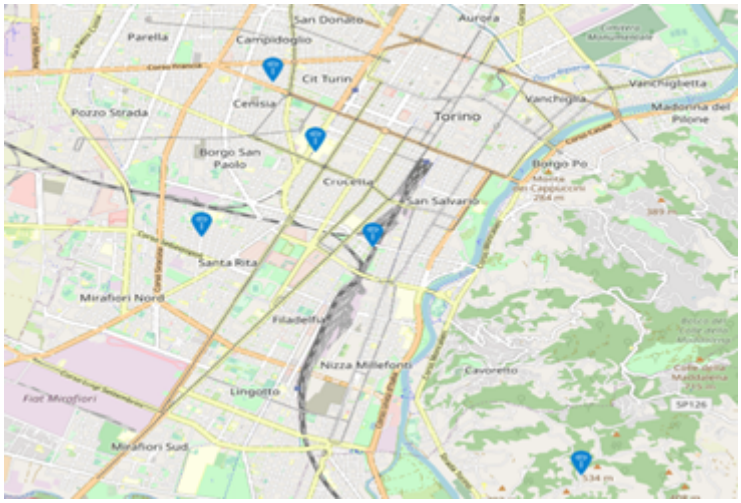


Figure 3.3: TTN Map of the registered Gateways (near the test position in Torino City)

SX1272 Semtech chip which is responsible for performing the LoRa modulation. The block diagram of FiPy is shown in Figure 3.5

The FiPy was used with the Lora 868MHz Band antenna as the experiment is conducted in Europe which is a standard dipole with a gain of 2.15 dbi.

Using the FiPy with its expansion board allows access to the Board using Mini-USB cables directly or via WiFi connection. The development environment that was used is the Pymark plugin using the Visual Studio Code editor.

DHT11–Temperature and Humidity Sensor The DHT11 is a commonly used temperature and humidity sensor. The sensor comes with a dedicated NTC to measure temperature and an 8-bit microcontroller to output the values of temperature and humidity as serial data. The sensor is also factory calibrated and hence easy to interface with other microcontrollers. The sensor measurement range for the temperature starts from 0°C to 50°C with an accuracy of ±2°C and the measurement range for the humidity starts from 20% to 90% with an accuracy of $\pm 5\% RH$ [22]. To extract the sensor data with the FiPy the Library dth.py was used as it features the required methods to do the job, in the following lines we demonstrate a part of the code to initialize and extract the sensor data.

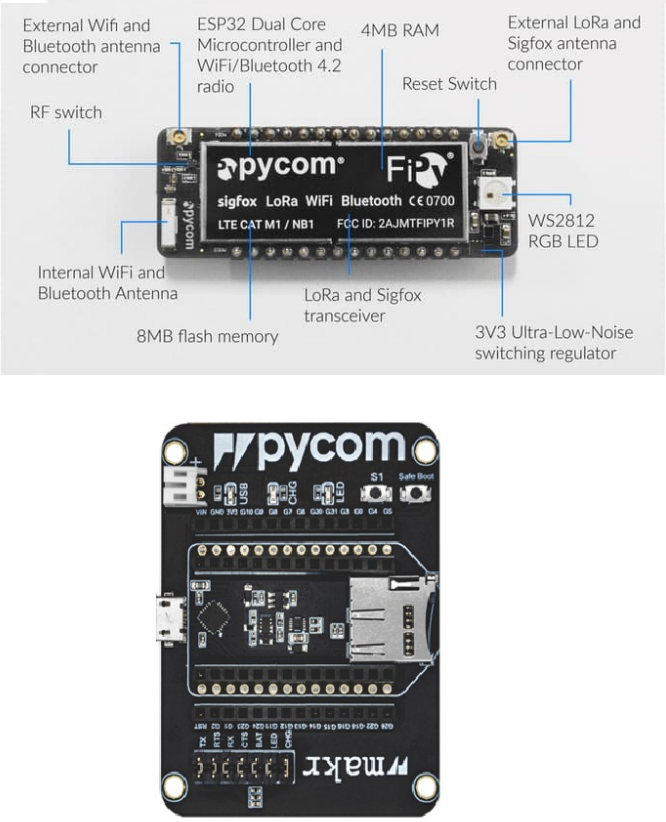


Figure 3.4: FiPy board and its expansion board

```
1 from dth import DTH          # Custom Library of the dHT11 sensor
2
3 # creating DHT11 Object on pin 3 of the Fipy
4 DHT_11 = DTH(Pin('P3', mode=Pin.OPEN_DRAIN),0)
```

Using the dth library to extract the sensor measurements from the DTH11 sensor line 25.

Using the method read() from the dth library to read the Temperature and the humidity from the sensor, if the results were valid data (4 bytes that represent the Temperature and the humidity) then the data will be printed on the Visual studio Console (for debugging). Then the results for both Temperature and humidity were stored in two separate variables to be transmitted using the LoRaWAN socket which we'll demonstrate later in this chapter.

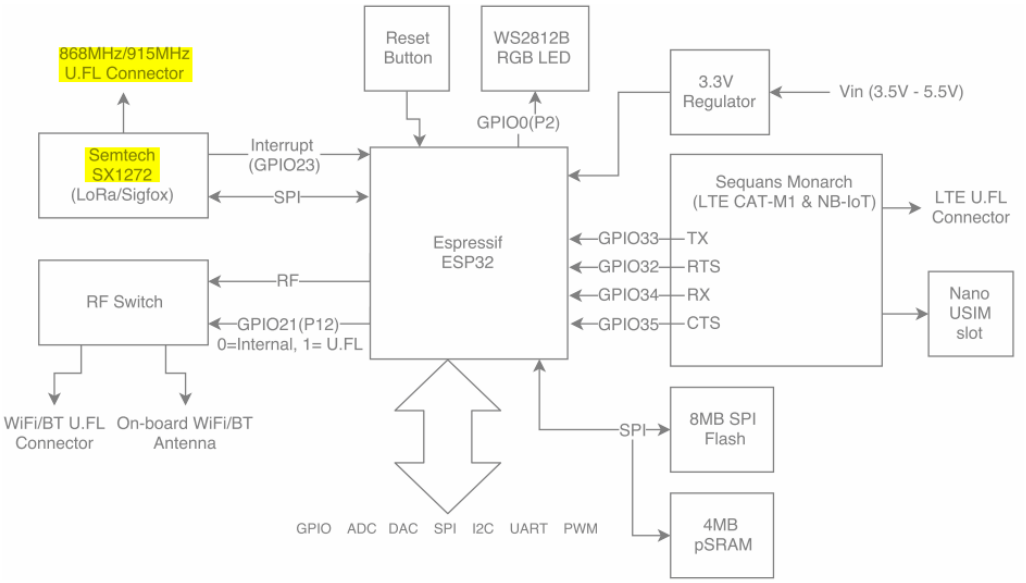


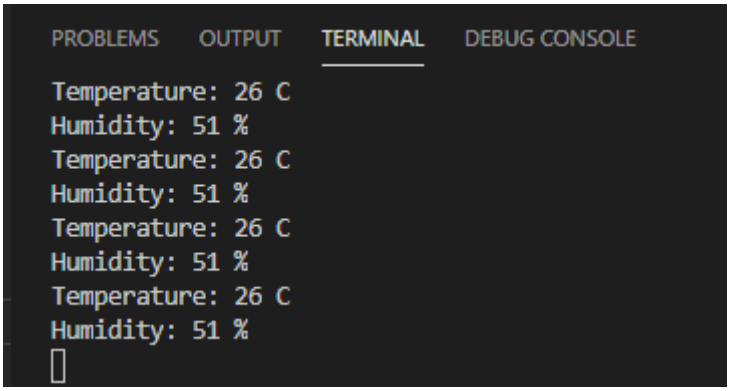
Figure 3.5: FiPy System block diagram[21]

Figure 3.6: Lora 868MHz Antenna

```
result = DHT_11.read()
    if result.is_valid():
    pycom.rgbled(0x001000) # green
        print("Temperature: %d C" % result.temperature)
        print("Humidity: %d %%" % result.humidity)
    hum = result.humidity
    temp = result.temperature
```

Assigning the Temperature and the humidity values line 56.

The console output will be like: So at this point, we're sure that our sensor is



sending the data to the FiPy which collects this data properly.

The Things Network (TTN)

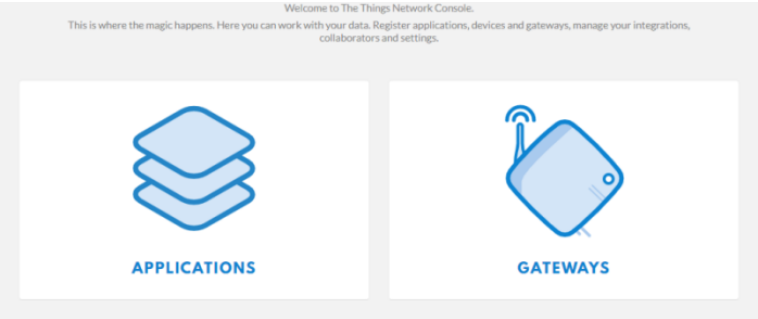


Figure 3.7: TTN Console

As previously described TTN is a dedicated web server that's dedicated to the LoRaWAN IoT Applications by allowing the users to configure and monitor their LoRaWAN applications and also their gateways through the TTN Console as shown in figure 3.7.

We configured our application using the necessary values for the ABP authentication because as we mentioned in 2.4 it is simpler and faster for prototyping purpose and testing. The configuration window is shown in figure 3.8.

Payload format The FiPy was programmed to concatenate the Temperature and humidity readings and send them using the "ustruct" library which returns several bytes needed to store a given data input, this was done due to two facts, 1st

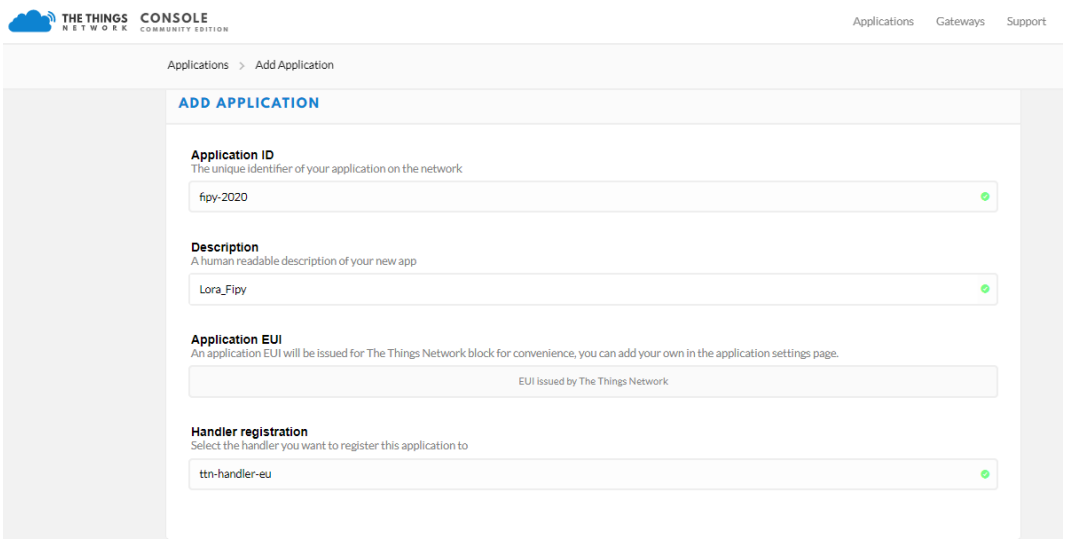


Figure 3.8: registering our application in the TTN applications

the “send” socket of the LoRaWAN requires the input argument to be in the "bytes" format, 2nd TTN strongly advice the users to always use the binary encoding for their payload data [23].

```
1 Con_Data = int(str(temp)+str(hum))
2     Data_11 = ustruct.pack('h',Con_Data)
3     s.send(Data_11)
```

Constructing the Packet byte format in-order to be send using the LoRa socket line 65.

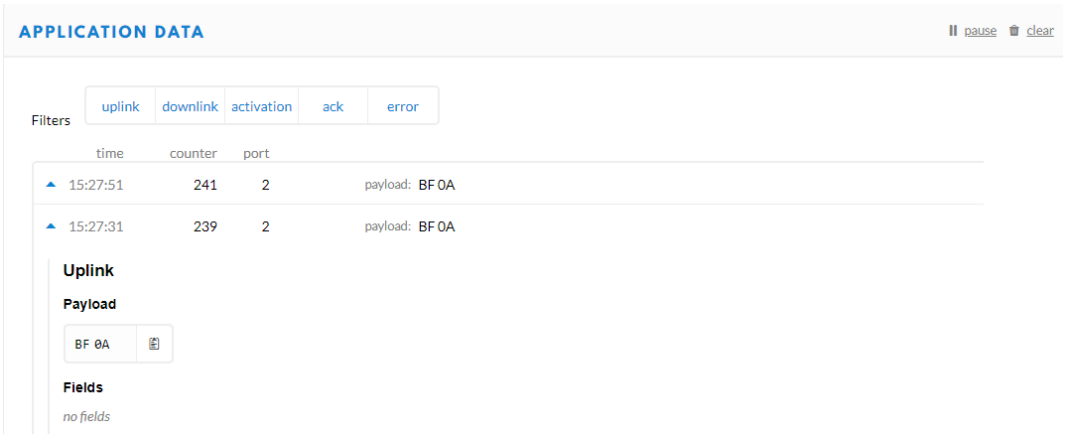


Figure 3.9: Payload in hexadecimal representation

The transmitted packet can be checked on the TTN web server page which also gives many valuable Metadata about the transmission parameters but we'll demonstrate the payload first; it will look as shown in figure 3.9 which is represented in hexadecimal form.

TTN provide a convenient tool that lets the server do payload decoding for the user to see meaningful data on the web UI and this is done by writing a JavaScript decoding function under the section called "payload format". The function can be tested using a payload that's inserted by the user (we used a real payload from the application page).

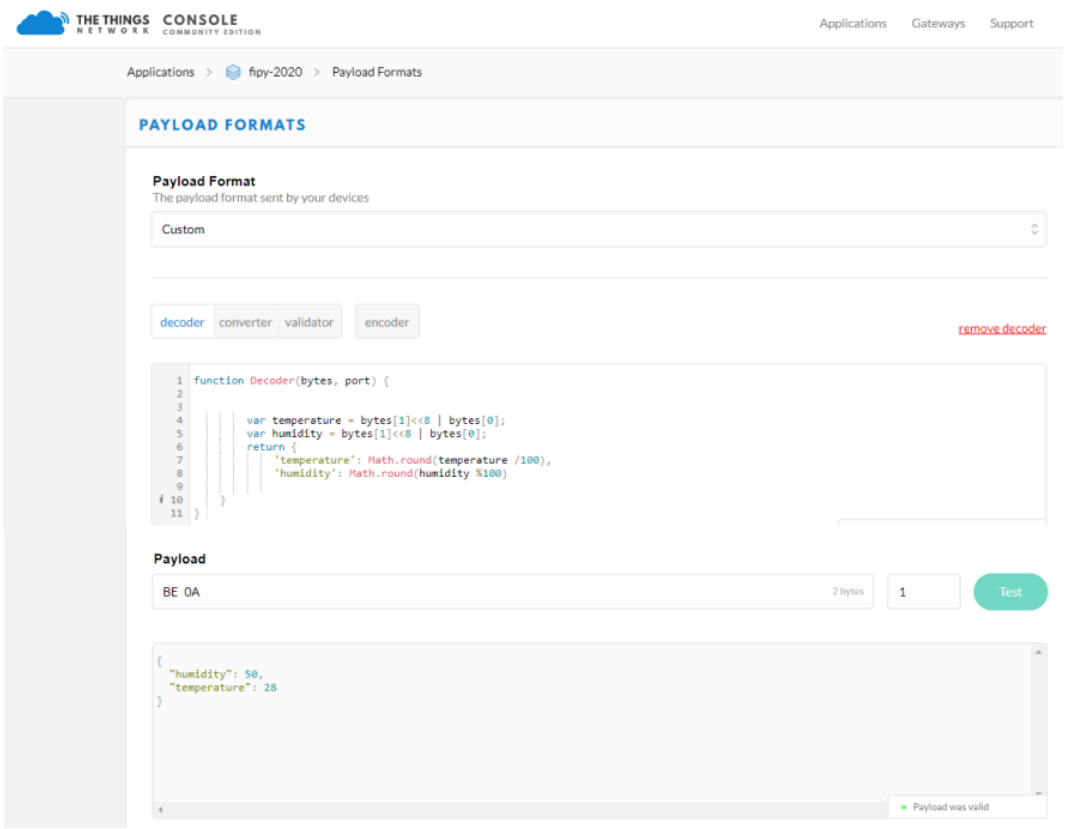


Figure 3.10: Payload decoder function

After that we can see the received data as show in the following figure 3.11. Notice that the payload represented on the TTN application web page is the decrypted payload, if we want to check the encrypted data we can look at the TTN gateway page to find out the size of the encrypted data as it will look like in figure 3.12.

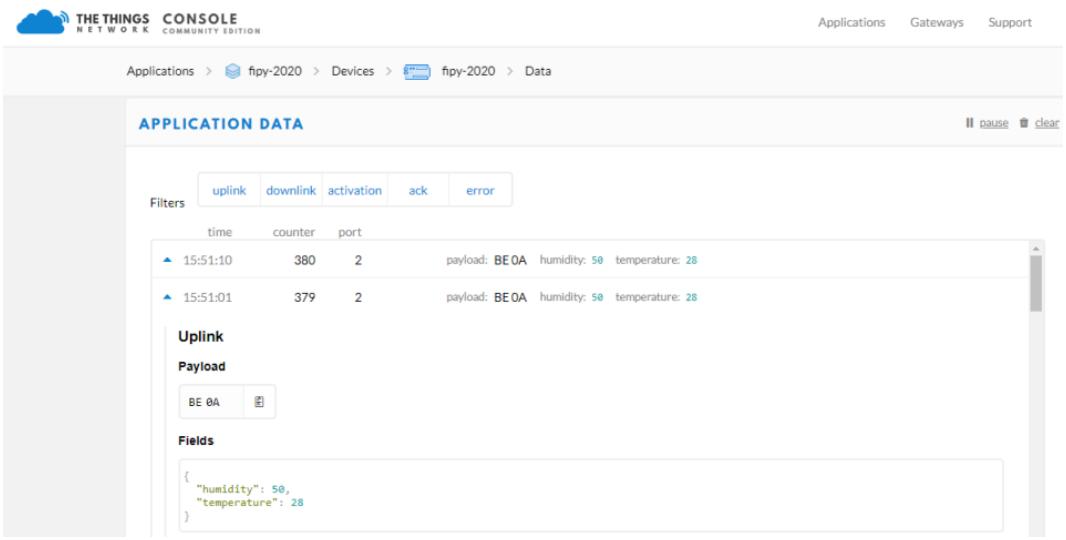


Figure 3.11: Payload in human readable format

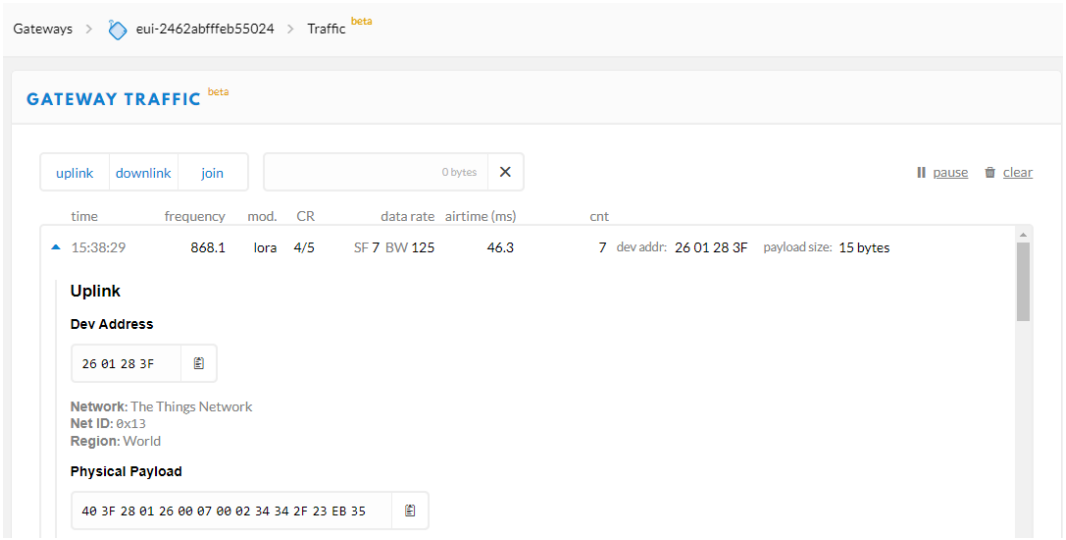


Figure 3.12: An encrypted LoRaWAN Packet

As we mentioned in the LoRaWAN tech. section the overhead is 13 bytes (default) which explains why the Payload on the gateway Console is 15 bytes (since our application uses only 2 bytes for the payload data). **The Payload Metadata**

One of the most valuable information that the user gets from the TTN server is the Metadata which shows many important parameters that contribute to the reception of the data packet Those parameters are shown in figure 3.13 .those are the parameters that we're going to use in our experimental field test of the

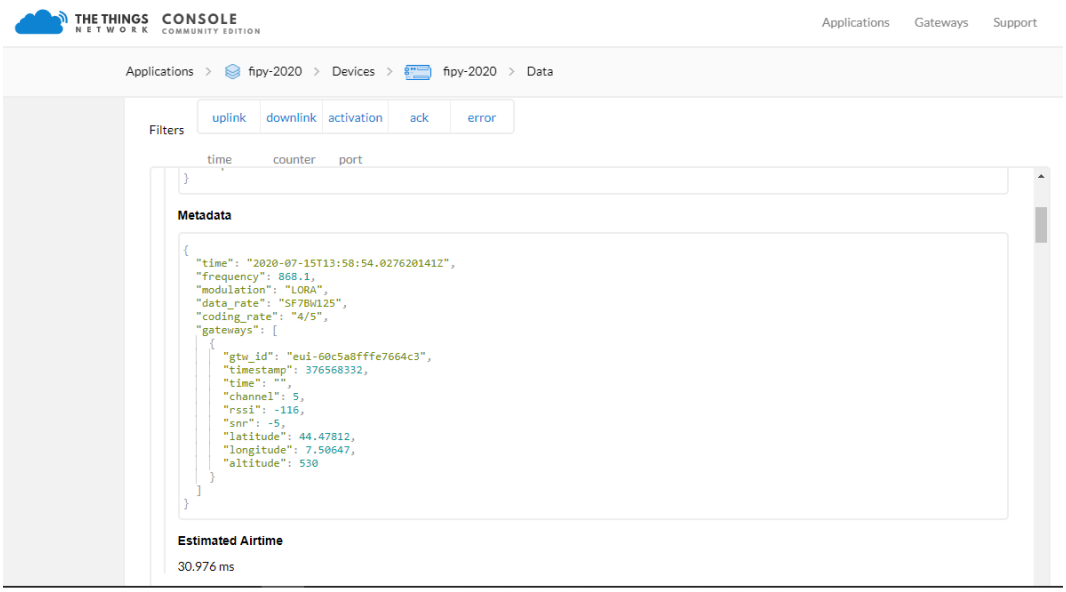


Figure 3.13: The Metadata of the received packet

LoRaWAN technology which will reflect us the range, data rate (throughput) and the delay.

3.1.3 TTN Limitations

We had mentioned before the limitations of using the ISM Band along with the LoRaWAN technology limitations; Now as we're going to use the TTN web-server we must know its limitations and rules as well. According to [24] the TTN limitations which are imposed when using the TTN server (known as the fair access policy for The Things Network's- public community network) states that The Payload data for each end-device can send As :

- An average of 30 seconds up-link TOA, per 24 hours, per device.
- At most 10 downlink messages per 24 hours, including the ACKs for confirmed up-links.
- Devices with fixed hard coded spreading factors of SF12 or SF11 are not allowed to join the network.

3.1.4 Node-Red

Node-RED is a programming tool for wiring together hardware devices, APIs, and online services. Primarily, it is a visual tool designed for the Internet of Things, but it can also be used for other applications to very quickly assemble flows of various services. It enables users to stitch together Web services and hardware by replacing common low-level coding tasks (like a simple service talking to a serial port), and this can be done with a visual drag-drop interface. Various components in Node-RED are connected to create a flow. Most of the code needed is created automatically.

Figure 3.14: Node-Red Flow

Why Node-Red?

The need for extracting and storing the Meta Data of the payload message is the main reason why the researcher used Node-Red since the storage integration capability that's offered by TTN doesn't provide (store) the Metadata of the messages. Also, the web Console of TTN only stores the data until the browser buffer will overflow then all the data will be lost hence the need for an application to store those data. Once we have the data stored we can analyze it as we want.

Node-Red Payload and Metadata flow

Node-Red really makes it easy for us to do our measurements as it provides ready-made Nodes that Emulate the Up-link(or Down-link) of the TTN Nodes (or Gateways). A Flow has been developed to extract the LoRa packet from the TTN web-server and store the data in a .csv file, the flow is shown in figure 3.15.

Describing the flow from left to right, 1st we use the **ttn up-link** which is the connection between Node-red and TTN server, the output of this node is split into two branches one that goes to the **function** Node which is used to extract the

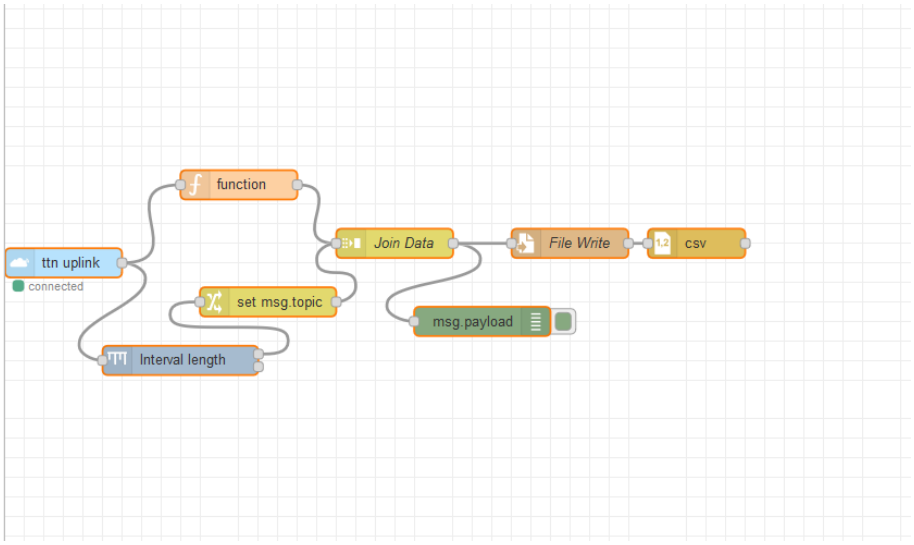


Figure 3.15: Node-Red Flow for extracting the up-link messages from TTN

needed parameters from the received up-link messages, this node is programmed in JavaScript to extract the following parameters:

- Message Counter (Counter)
- Message payload data (PL)
- Data Rate (DataRate)
- RSSI (rssi)
- SNR (snr)
- The time which the gateway received the uplink message (Time)
- Gateway ID (GW_ID)
- Gateway Location (lat , lng , alt)

The JavaScript code for this node is in the appendix. The other branch of the output goes to a Node called **Interval length** which gives the ability to measure the time difference between two consecutive messages[25]. From this node, we can measure the delay that occurs between the up-link messages, Notice that **we can't measure the accurate latency of the system** since we don't control the clock sources that go to the different stages of the system like the TTN and the Gateway and we will discuss this in the next chapter.

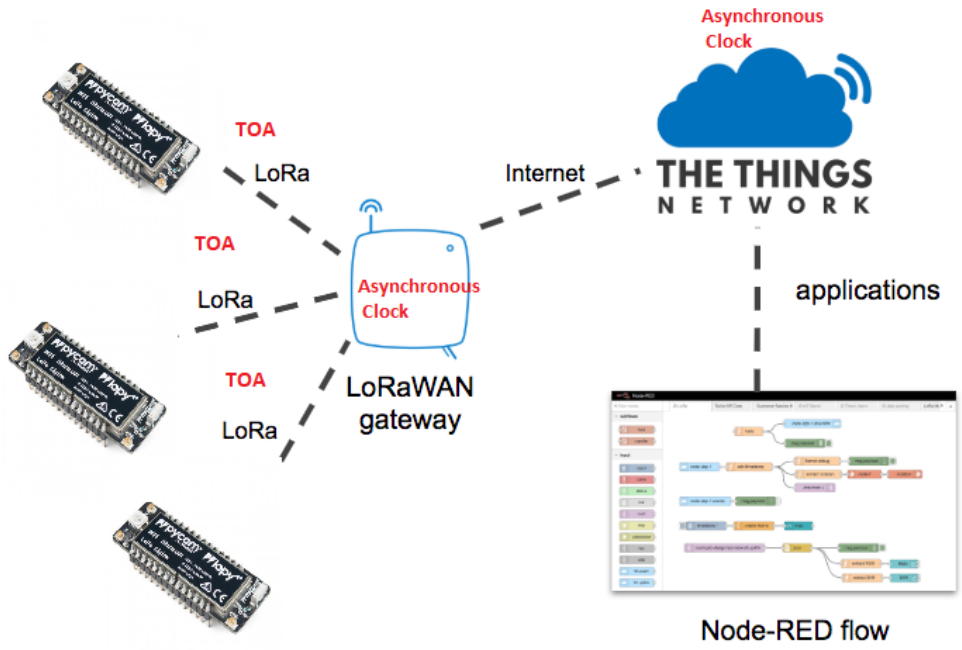


Figure 3.16: Tracking the timing of the LoRaWAN packet

The Extracted data will then be stored in a .csv file to do the analysis later. In the next section, we're going to demonstrate the measurements and the results that had been done.

3.1.5 Firmware Flowchart

As the system components have been described, here we demonstrate the flowchart that the system follows to establish the LoRa connection between the Sensor node and the gateway. The transmission is non-confirmed so the sensor node doesn't wait for any response from the GWs(in the Rx Windows as described in 2.20). The Firmware was cited in Appendix A.

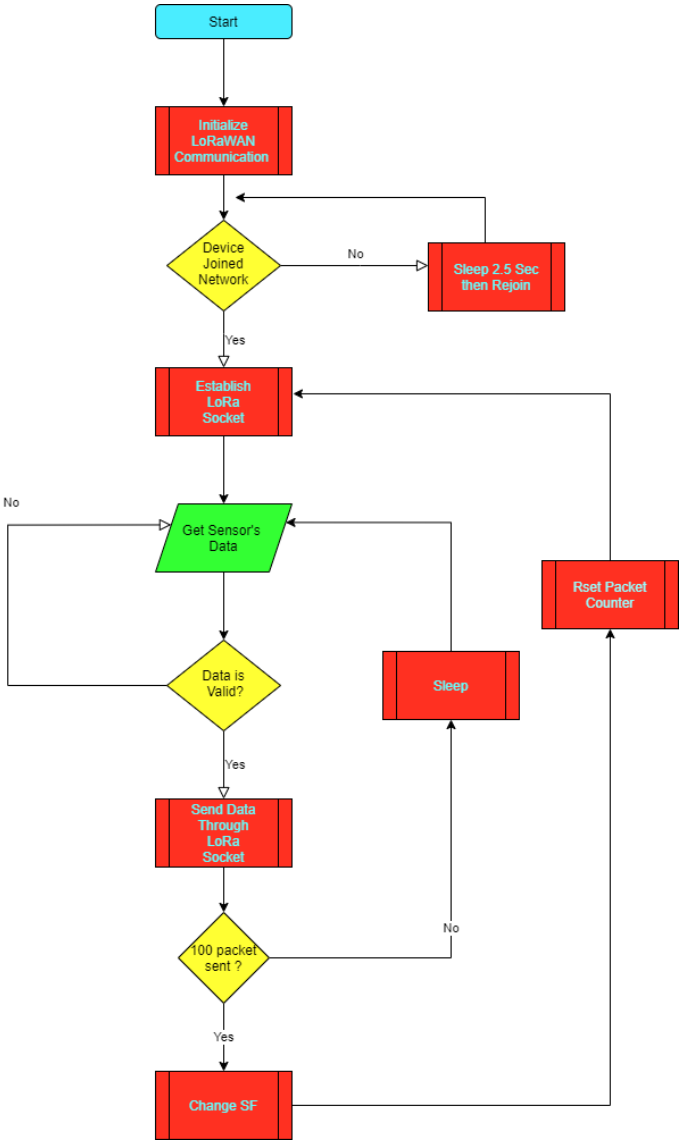


Figure 3.17: LoRa Up-Link transmission Flowchart

Chapter 4

Measurements and Results

After defining the main parts of the proposed design and system methodology along with the main aspects and structure of the testbed that is going to be used to test the system performance and the reliability of the communication scheme prototype in the previous chapter, in this chapter, the measurements and results are going to be discussed. We will focus on the investigation of the scheme performance in different scenarios to examine its capabilities and to check how it behaves in special environments and under special circumstances to show its abilities to fit these particular uses.

4.1 The System Criteria

To Evaluate the system properly we have to have reference parameters that we can compare with the results we get from the real measurements, so we can know what the technology can satisfy and what it can not. The following are the criticalities for the system:

1. Should cover a range of at least 600 meters.

2. The update time for up-link messages should be $<$ 300 Sec

3. Should be able to handle the transmission of 20 bytes of data within every transmission interval.

Hence the objective of the measurements is evaluating the applicability of the LoRaWAN technology to satisfy those criteria or no.

4.2 Test Scenarios

As we want to evaluate the performance of the technology with our critical parameters, The scenarios and the measurement of the experiment will go like this:

1. We will be testing the LoRaWAN coverage distance, this will be done by transmitting 100 LoRaWAN packets from four test points and we will observe the success of the reception of the packets (which indicated by the Packet Deliver Rate PDR) and also observe how the spreading factor(SF) could affect the PDR. We will also examine the effect of the packet size on PDR by using two different packet lengths, one will be 20 bytes and the other will be 51 bytes.

2. Considering the TTN policy and the ETSI duty cycle restriction (for the 868MHz Band) we will quantify the maximum achievable data rate w.r.t a payload data of 20 bytes.

3. After defining the maximum data rate which satisfies the TTN and ETSI constraints, we will measure the actual delay that the packets take through their transmission journey from the Sensor Node until it reaches the Application server(Node-Red).

4.2.1 The Size of The Payload

The short payload size was determined based on the data ingredients of our sensor node which is:

1. ID (2-bytes).

2. 4xPM2.5:(10 bits/Sensor).

3. 4xPM10:(10 bit/Sensor).

4. TEMP:(1-byte).

5. HUM:(1-byte).

6. RTC:(4-byte).

7. GPS:(from 2 to 8 bytes - Future Version)

The Sensor Node Box contains 4 of PM2.5 sensors and 4 of PM10 sensors, which in total produce 80 bits/sec (10 bytes).

4.2.2 Measurements Points Geo-locations

The measurement test was set as following, four test points were chosen in the city of Turin to measure the performance of the technology. As was mentioned in the previous chapter 3.3 that TTN provides a gateways map that shows the distribution of the registered gateways that forward the packets to the TTN server. we had noticed that sometimes an unknown gateway that is not trusted (registered to TTN server with fake locations) receives the packets, to avoid those Gateways the Gateway which is installed in the polytechnic university of Turin (registered under the name Polito) was considered the reference Gateway to the sensor Node's test points. The four test points are chosen to demonstrate different transmission environment, we try to emulate the Line of Sight(LOS), Non-Line of Sight(NLOS), relatively short and long distances.

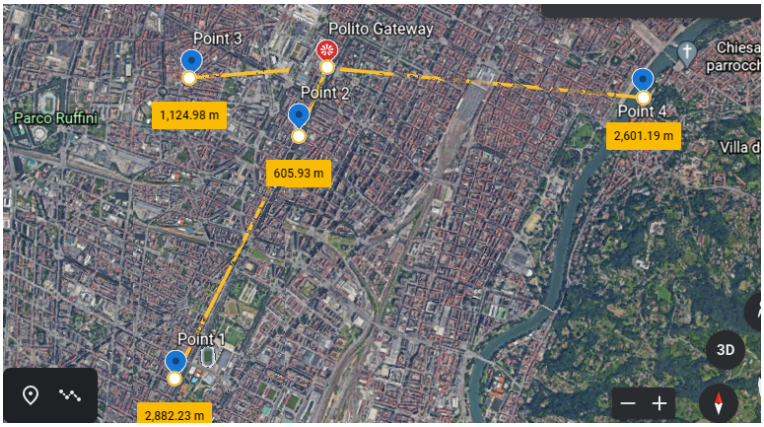


Figure 4.1: Map of the Test Points in the city of Turin[Google Earth]

Point 1 is a far LOS from "Polito" GW with few blocking buildings with height < 100 m across the sight. Point 2 is a short-range LOS with few buildings in the vicinity. Point 3 has a lot of buildings that block the view of the GW. Point 4 is a far distance with an environment that's in front of a water surface (river) where there're moving boats and many other obstacles. The chosen points to do the test are listed in Table 4.1 below

4.2.3 Test setup

The measurement test was set as following, four test points were chosen in the city of Turin to run the tests. The bandwidth will be fixed BW = 125KHz and code rate = 4/5 for all the measurements. The Gateway "Polito" was chosen to be the reference Gateway and its geographical information is shown in Table 4.1.

LoRaWAN architecture as mentioned in 2.2.1 follows "star of star" topology,

Point Number	Distance from Polito Gw(Km)	Latitude	Longitude
1	2.88 LOS	45.039404	7.645806
2	0.6 LOS	45.057617	7.659197
3	1.11 NLOS	45.061521	7.648052
4	2.59 NLOS	45.060447	7.694892
Gw	0	45.064167	7.6597

Table 4.1: Test Points of the Experiment

which allows many GWs to receive the same packet. The TTN GW map tells us about the registered Gateways in the city of Turin as shown in 3.3. But besides those "Registered" GW there are also some gateways that are not registered as "Trusted" in the TTN server which may have non-accurate location data about them. we had to deal with one of them since the TTN web-server some times ignore the reception of the same packet from multiple GW and start to ignore some of the GW which leads to the ignorance of the GW "polito" some times. This gateway which seems that it can received our packet from a distance of 60Km as

Figure 4.2: Same Packet received by two gateways

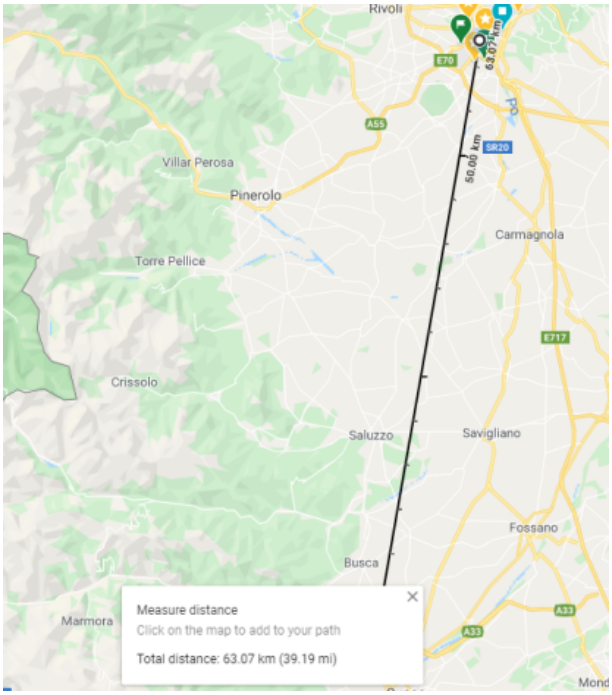


Figure 4.3: The un-trusted gateway's location w.r.t sensor node

shown in figure 4.2 under the GW ID = "EUI-60C5a8fffeb55024" which receives the same packet as GW ID = "polito", taking the coordinates of this GW and put it in Google map we get 4.3 (which is pretty impressive if it was real), we are not denying that LoRa can achieve this long range (there are many tests that had done much longer range[26],but the fact that there are many GW around the sensor node (As shown in 3.3) and the experiment was done in the urban city of Turin, all that create suspicious about the location of this gateway and hence in order to quantify the measurements of the PDR we are going to consider only the packets received by the Gateway "Polito" since it is defined by TTN as "registered Gateway" which means that its location information is accurate and hence we can be confident about our distance measurements that it represents accurate information about the distance.

$$PDR(\%) = (\frac{R}{T}) \times 100 \tag{4.1}$$

The transmitting power is automatically handled by TTN so it's not one of the developing variable parameters (unless the transmission is only LoRa).
The PDR is a measure of how many packets has been delivered successfully and it can be calculated as in Eq 4.1.

Where:

1. R is the successful received packets
2. T is the Number of the Total transmitted packets

4.2.4 Range Test

Besides the General measurement specifications that had been mentioned in the previous section, the range test will have an additional setup to observe the Packet length on the range, we are going to use two packet lengths, one with the real data of the DHT11 sensor (which is 2 bytes size) and the other one with a fixed payload data that will be relatively big (51 bytes) both will be tested Concerning LoRaWAN parameters and will be given the names *Short Payload* and *Long Payload* respectively. A 100 (unconfirmed) packets will be transmitted from the sensor node to the Gateway.

Data Size	SF	BW	CR	Pkt numbers	MSG Type
2 Bytes	7,9,12	125KHz	4/5	100	Unconfirmed

Table 4.2: Short Payload Packet configurations

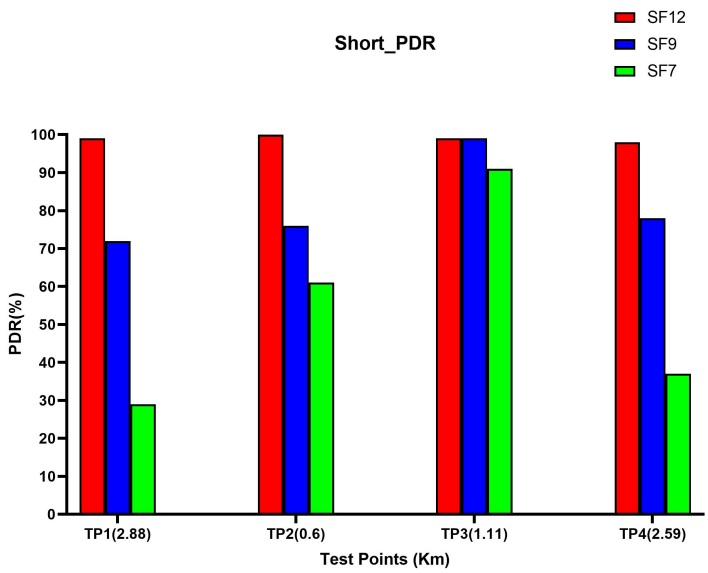


Figure 4.4: Packet Deliver Rate with different test points

Data Size	SF	BW	CR	Pkt numbers	MSG Type
51 Bytes	7,9,12	125KHz	4/5	100	Unconfirmed

Table 4.3: Long Payload Packet configurations

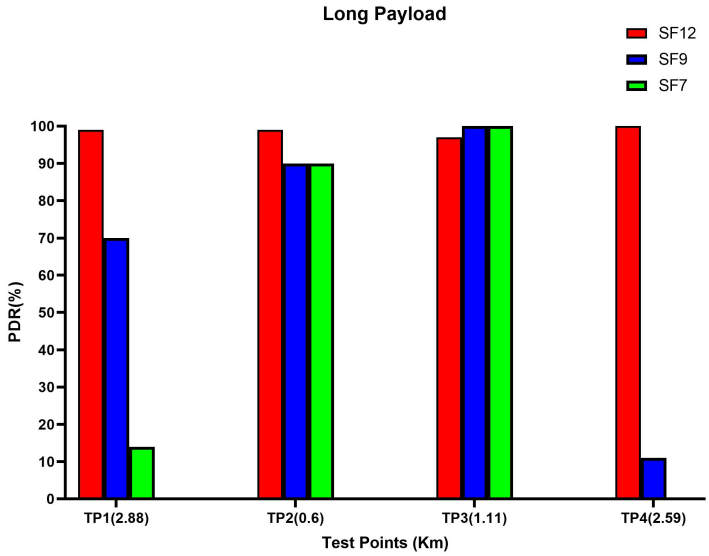


Figure 4.5: PDR for the Long payload

Figures 4.4 and 4.5 demonstrate the results of the PDR for every test point with varying the spreading factor for Short and Long Payload respectively. SF12 achieves almost 100% for all the distances for Short and Long Payloads. SF7 and SF9 showed unstable behaviors regardless of what was expected, In Short Payload it was expected that TP2 will have the highest PDR, since it is the nearest to our Gateway, but the results showed that TP3 was a better performance for those SF. The reason for this behavior is due to the different types of interference that surround the LoRa Device during the experiment time (vehicle, bus movements, and maybe other radio signals) since TP2 is a bus station. If we observe Figure 4.5 where the same experiment was conducted, we can see a better behavior for the same Spreading factors(7 and 9). Although the Payload this time is much bigger than the 1st experiment(more than 90% PDR for TP2). We can also see a clear reduction in the PDR in the Long Payload case as the distance increases (as in TP3, TP4) which comply with the physical description of the SF(2.1.2) as was described in Ch.2. A critical note to consider here, that those measurements PDR were taking only w.r.t. "Polito" GW, however, TTN web-server shows that a

much higher PDR than that for packets reception but it was through other GWs (which we discarded as mentioned previously). also, the range performance can be enhanced by increasing the code-rate, as we used the minimum CR = 4/5 however if we used the CR = 4/8 we can have a longer range (better reception for the same Node) but the downside is that we are going to lose the bit-rate.

4.2.5 Throughput of the System

Now we want to evaluate the performance of LoRaWAN in terms of its maximum rate of data that packets can be transmitted with. General speaking we can theoretically know the data rate that we can get corresponding to every SF by The following Equation[15] :

$$DR = SF \times \frac{BW}{2^{SF}} \times \frac{4}{(4+CR)}$$

Where:

1. DR = Data rate
2. SF = Spreading factor packets (7-12)
3. BW = Bandwidth(Hz)
4. CR = Coding rate (1-4)

Also, a very handy calculator[16] provided by Semtech can calculate the bit rate of the transmission rapidly as shown in Figure 4.6.

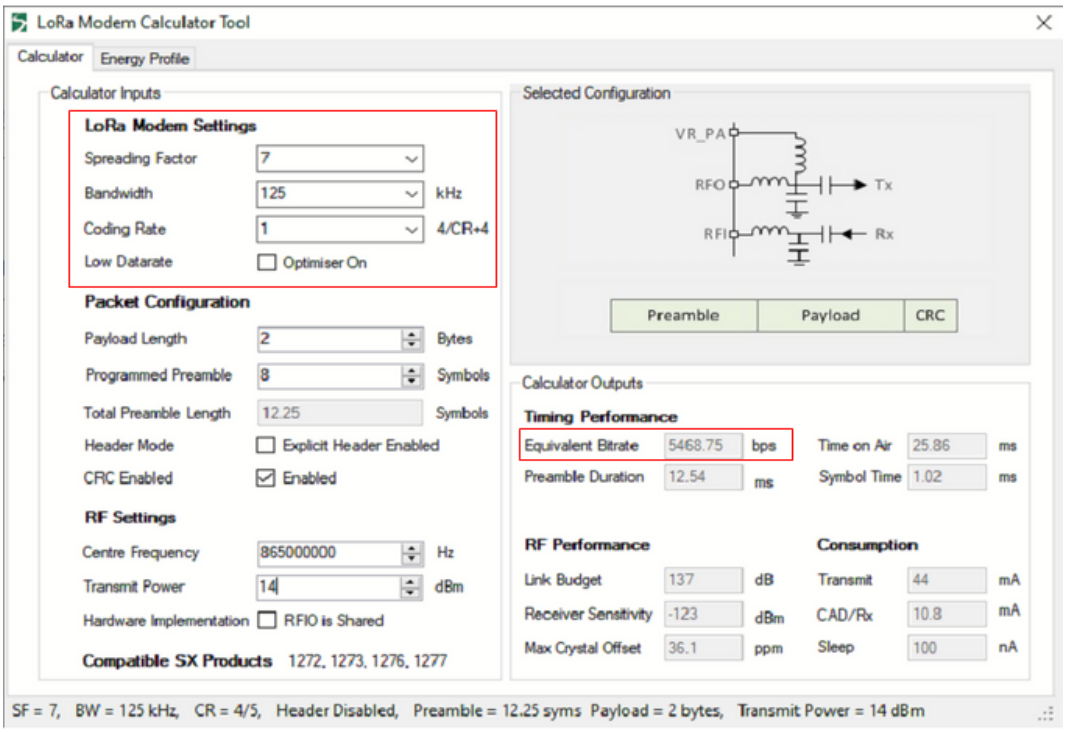


Figure 4.6: Bit Rate for SF = 7

SF	BW(KHz)	CR	Evaluated Bit Rate(bps)
7	125	4/5	5468
8	125	4/5	3125
9	125	4/5	1775
10	125	4/5	976
11	125	4/5	537
12	125	4/5	292

Table 4.4: The Evaluated bit-rate for different SF

We can find out the Bit Rate for the other Spreading factors and set a table for them as shown in Table 4.4. Note that Bandwidth can increase the bit rate but it is only controllable by LoRa transmission (not LoRaWAN).

4.2.6 Byte-Rate Suitability For The Task

We shall begin here to analyze the applicability of the data rate that LoRaWAN support for our system, we will also consider the limitations of the duty cycle and TTN fair access policy and see if the technology can deliver the packets at a reasonable rate or not.

Packet Timing Analysis

We will start our analysis simple by examining the time it takes to transmit a 20-byte payload packet taking into consideration all the restrictions. Knowing that the overhead for LoRaWAN Packet (minimum without payload) is 13 bytes and using the air time calculator [17] we can find out a good approximation of how long our packet shall be on the air TOA, the time the device should be sleep after that and how many packets (message) should be sent during the hour. So the total packet size will be 33 bytes (20 + 13).

From fig 4.8 we can see that SF=12 is the critical case for the data rate with the highest limitations on the duty cycle and the maximum number of a packet per day for transmitting a 20-byte payload LoRaWAN packet on the EU868 Band, analyzing the results of this SF in details would be :

1. For SF=12, TOA = 1810.4 ms for a 33-byte Packet.
2. T_{off} = 181 Seconds between subsequent packets (1% Duty Cycle).

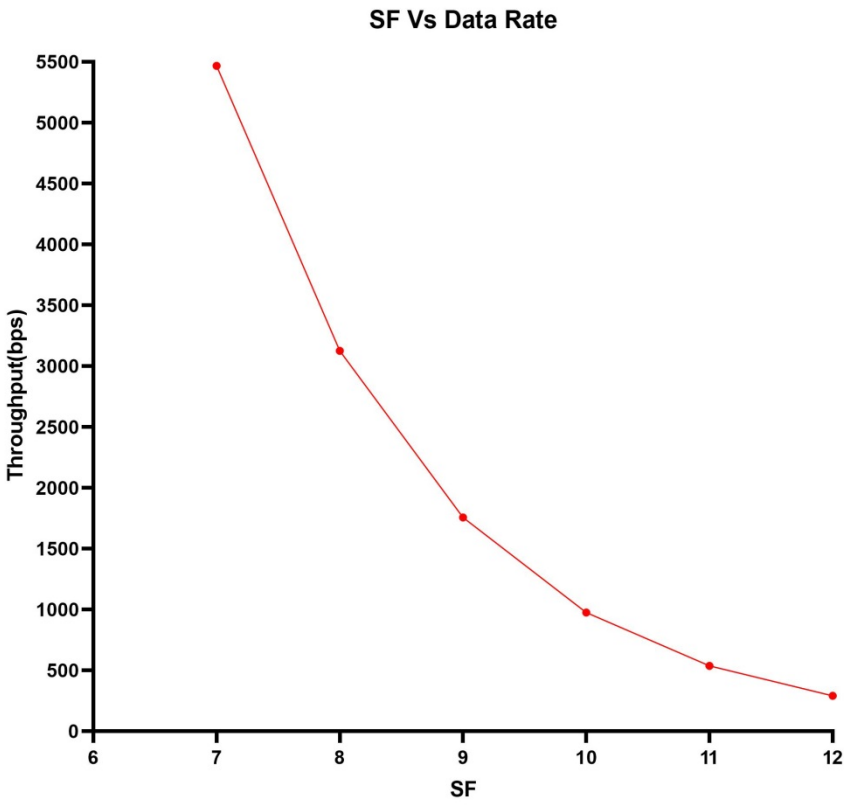


Figure 4.7: Evaluating the Bit-rate for every SF,(BW = 125Khz)

3. TTN Access policy limits the total number of transmitted packets per day to 16 MSG/day (for transmitting the whole day).

So, TTN only allow us to send 16 Message per day using the SF12.
while the Maximum packet rate that the weather station must not exceeds is 5 min. hence using SF12 is not applicable for our system. An important thing to notice here is that the Fair Access policy for TTN does not exist with a private web-servers.

Combining the information of the bit rate with the LoRaWAN transmission restrictions we can now know that the suitable mode for transmission for our system is the SF7 and under the limitations of the TTN web-server, this should imply a bit Rate of 5.5Kbps with a $T_{off} = 3.5 minuets$, also notice that the T_{off} increases or decreases depending on the size of the packet.

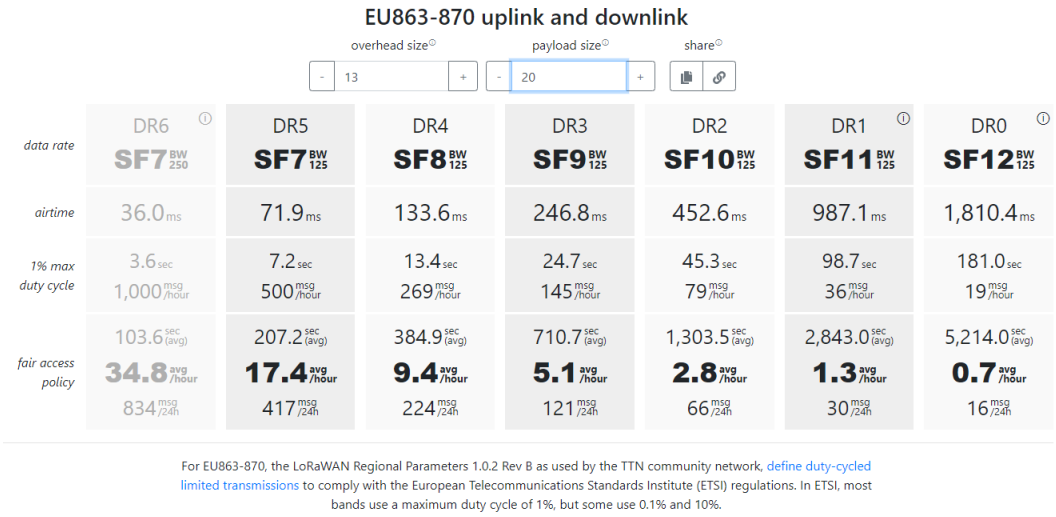


Figure 4.8: LoRaWAN Timing for 20 bytes Payload packet

Packet Size	SF	TOA(ms)	DC 1%(sec)	Sleep Time(min.)	Max. MSG/Day
33	7	71.9	7.2	3.5	417
33	8	133.6	413.4	6.4	224
33	9	246.8	24.7	11.8	121
33	10	452.6	45.3	21.7	66
33	11	987.1	98.7	47.4	30
33	12	1810.4	181	86.9	16

Table 4.5: Timing Analysis for 20 byte Payload

4.2.7 Can we do better ?

A more efficient suggestion for the Air quality station's packet is:

Packet reduction

We can take advantage that the LoRaWAN packet already contains the device ID, so we don't need to waste another 2 bytes to transmit information that is already included(we only need to extract it from the meta-data of the payload). Also using the advantage that LoRaWAN Packet contains the timestamp within can make us save the RTC data which reserve 4 bytes of the packet, Now our packet shall

contain the following:

1. PM2.5:(40 bits)
2. PM10:(40 bits)
3. TEMP:(1-byte)
4. HUM:(1-byte)

now we have a reduction from 20 bytes to only 12 bytes of data which can be send using TTN every 2.7 minutes and every 10 seconds if we are going to use a private web-server.

Data averaging and SSB

The fact that weather data is not a rapidly changing type of data(per seconds), hence we don't need to send the data at the same rate as the sensors produce it, instead, we can use a simple technique like introducing a Sensors States Byte(SSB) which is simply a byte that gives us an indication if there is any change happened in the sensor's readings. and we can use it as the default transmission unless there is a change in the sensor's measurement.

State	PM10	PM2.5	Temperature	Humidity	X1	X2	X3
Bit 0	Bit 1	Bit 2	Bit 3	Bit 4	Bit 5	Bit 6	Bit 7

Figure 4.9: An Example of our suggested Sensors State Byte

The SSB byte could be constructed like in figure 4.9. Where "State" bit is an indication if there is any change happens in any of the measurements of any of the sensors, so at the receiver side the check can only be done on that bit, if it = 0 then no change happens in any of the readings, if it = 1 then the proceeding sensor's bits must be checked to know which sensor(s) measurement had changed. The 3 least significant bits can be used as an indicator to the web-server of the upcoming data packet, we can set our own coding for those 3 bits. For example, we can establish a table of the upcoming measurement that will be sent like:

An algorithm can be set on the sensor node to follow a specific pattern whenever the measurement of any of the sensors changes and/or with a specific period of time.

The normal pattern of transmission now is reduced to 1 byte(14 bytes in total) per transmission, using LoRa air time calculator, this gives us the possibility to

Sensor Measurement	X1	X2	X3	Size(bytes)	TTN T_{off}(min)	1% T_{off}(sec)
PM10	0	0	0	2	2.2	4.6
PM2.5	0	0	1	2	2.2	4.6
PM10+PM2.5	0	1	0	4	2.4	5.1
Temperature	0	1	1	1	2.2	4.6
Humidity	1	0	0	1	2.2	4.6
GPS	1	0	1	4	2.4	5.1
RTC	1	1	0	4	2.4	5.1
All Measurements	1	1	1	10	3.3	6.2

Table 4.6: Encoding of the Last 3 LSB of the SSB

transmit a message every 2.2 minutes using TTN and every 5 seconds if we use a private web-server.

So we can send a Full packet with the sensor's data once every few minutes then keep transmitting only the SSB until a change happens in the measurement of one of the sensors, the difference between the duty cycle of the full sensor measurements and the SSB is one minute for the amount of the sensor data that we have at the moment but as the data may get bigger, this difference becomes larger.

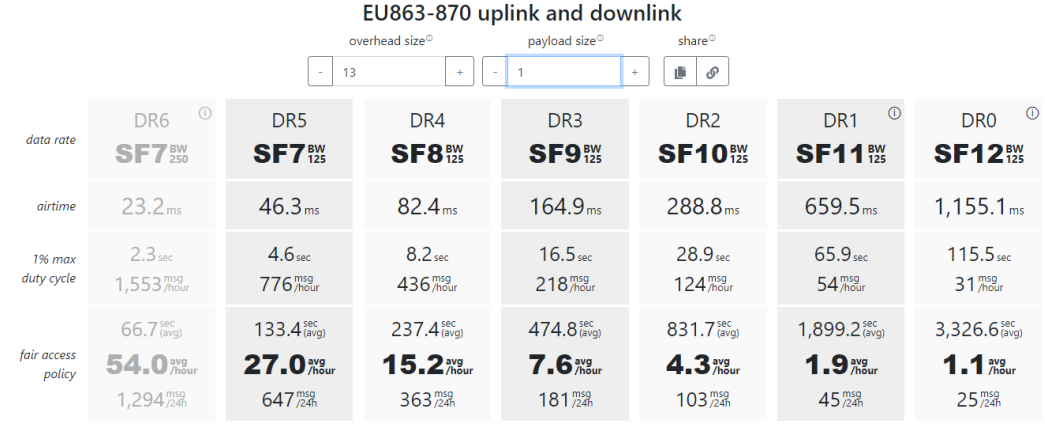


Figure 4.10: The 1% duty cycle allows the transmission every 5 seconds for data size = 1byte using SF7

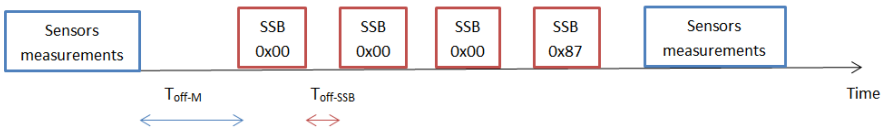


Figure 4.11: An Example of the transmission pattern of the sensor data

	All Measurement Packet	Sensor State Byte
Size(Byte)	10	1
Duty Cycle Limitation	MSG every 5 Sec	MSG every 5 Sec
TTN Limitation	MSG every 3.3min	MSG every 2.2min

Table 4.7: Transmission of measurement Packet(10+13) and SSB(1+13)

The Pros of this technique

The power of those techniques become more important to solve the limitations of the TTN web-server not just for the Fair Access Policy but also the very limited Downlink per day for a gateway which is only 30 MSG/day, so now we can make the transmission smarter to track the changes on the measurements and give the sensor node the intelligence to use the possibility to transmit a much higher number of Uplink over the LoRaWAN technology (For TTN).

The con of this method is that the user can't give commands to the sensor node to change its pattern unless they changed the Firmware.

4.2.8 The Latency

In this section, we are going to demonstrate the Latency that a LoRaWAN packet may encounters during its journey starting from the Node sensor until it reaches the application server successfully.

The Latency Test Configurations

The analysis will focus on the LoRaWAN class-A investigating the impact of different parameters on the latency, this will be done by fixing the bandwidth and the code rate to 125KHz and 4/5 respectively while using the different spreading factor(7,9,12). Setup a single sensor node (FiPy) and a single Gateway, we will run the test. Looking at figure 3.16, the LoRaWAN packet leaves the sensor node to the gateway as it is modulated using the LoRa protocol, the gateway then forwards this packet to a single network server (TTN in our case) over an IP-based network, the network server can be connected to an application server which demonstrates the data as the user wants, every stage during this journey costs the packet an amount of delay which we demonstrate it on the Packet timing diagram in Fig 4.8.

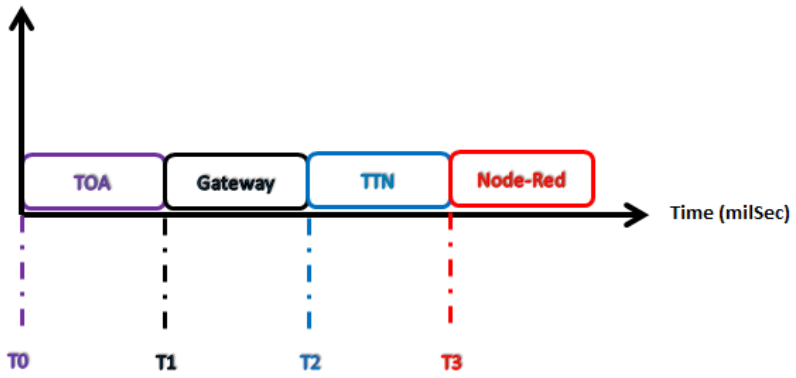


Figure 4.12: Timing of a LoRaWAN Packet

Where :

1. T0 = The RTC initial time in the Sensor Node.
2. T1 = The Packet spends TOA then the GW register its received time.
3. T2 = TTN registers its Timestamp after receiving the packet.

4. T3 = after the TTN decoding and processing and transmission, finally the packet reaches Node-Red which has its own timestamp.

The analysis gives an estimation of the latency. As the Synchronization only occurs between the Fipy and Node-Red. We can't know the clock source of TTN and as we don't use a private Gateway we can't also control it's clock source. The RTC in the Fipy is syncing its time with an NTP server which is "time.inrim.it" for Italy. We also modified the clock reference for our Windows PC (that runs Node-red) to sync its clock from the same source. In this way we reduced the error that could happens due to asynchronous clock between different stages. figure 3 shows the block diagram for clock synchronization between the Node and the application server. We then embed the RTC time information inside the LoRa Packet as the MAC payload and transmitted at the time (T_0). It takes an amount of time TOA which has a direct relation with the LoRa physical parameter from the relation: $TOA \propto \frac{2^{SF}}{BW}$ until it reaches the gateway(T_1), the best(lowest) TOA can be achieved by using the lowest SF and the highest BW. The packet spends an amount of time during processing then it is re-transmitted to the web-server using a network connection to reaches TTN at (T_2). It is then get processed again and decoded then transmitted to arrive at (T_3) at Node-Red which is installed on the same PC that is connected to the sensor node. The Node-Red then generates a .csv file that contains the time stamp for every stage the packet arrives.

A comparison will be done based on the time that had been recorded at each of those stages. an accuracy error of ±1 Seconds will be considered due to the asynchronous global clocks between Gateway, TTN, and the Node-Red where in this Node-red flow we used the "moment" Node to measure the time(in milliseconds) at which Node-red receives the Packet. The output .csv file should look like in fig

	B	C	D	E	F	G	H	I	J	K	L	M
249	Node_Time:"[2020	10	7	14	14	42	456943	Air_Time:"102656000"	GwTime:"2020-10-07T14:14:42.64634361Z"	TTN_Time:"2020-10-07T14:14:42.667389795Z"	NR_Time:"2020-10-07T14:14:43.041Z"}	DataRate:"SF7BW125"}]}
250	Node_Time:"[2020	10	7	14	14	52	458943	Air_Time:"102656000"	GwTime:"2020-10-07T14:14:52.646276873Z"	TTN_Time:"2020-10-07T14:14:52.667077587Z"	NR_Time:"2020-10-07T14:14:52.793Z"}	DataRate:"SF7BW125"}]}
251	Node_Time:"[2020	10	7	14	15	2	460952	Air_Time:"102656000"	GwTime:"2020-10-07T14:15:02.646362912Z"	TTN_Time:"2020-10-07T14:15:02.667374981Z"	NR_Time:"2020-10-07T14:15:02.920Z"}	DataRate:"SF7BW125"}]}
252	Node_Time:"[2020	10	7	14	15	12	462946	Air_Time:"102656000"	GwTime:"2020-10-07T14:15:12.645638156Z"	TTN_Time:"2020-10-07T14:15:12.667405074Z"	NR_Time:"2020-10-07T14:15:12.748Z"}	DataRate:"SF7BW125"}]}
253	Node_Time:"[2020	10	7	14	15	22	465990	Air_Time:"102656000"	GwTime:"2020-10-07T14:15:22.655853088Z"	TTN_Time:"2020-10-07T14:15:22.678901912Z"	NR_Time:"2020-10-07T14:15:22.888Z"}	DataRate:"SF7BW125"}]}
254	Node_Time:"[2020	10	7	14	15	32	468943	Air_Time:"102656000"	GwTime:"2020-10-07T14:15:32.655511277Z"	TTN_Time:"2020-10-07T14:15:32.676461795Z"	NR_Time:"2020-10-07T14:15:32.822Z"}	DataRate:"SF7BW125"}]}
255	Node_Time:"[2020	10	7	14	15	42	470945	Air_Time:"102656000"	GwTime:"2020-10-07T14:15:42.655255273Z"	TTN_Time:"2020-10-07T14:15:42.676276571Z"	NR_Time:"2020-10-07T14:15:42.718Z"}	DataRate:"SF7BW125"}]}
256	Node_Time:"[2020	10	7	14	15	52	472945	Air_Time:"102656000"	GwTime:"2020-10-07T14:15:52.665003508Z"	TTN_Time:"2020-10-07T14:15:52.686047038Z"	NR_Time:"2020-10-07T14:15:52.791Z"}	DataRate:"SF7BW125"}]}
257	Node_Time:"[2020	10	7	14	16	2	474944	Air_Time:"102656000"	GwTime:"2020-10-07T14:16:02.6648394812"	TTN_Time:"2020-10-07T14:16:02.686519769Z"	NR_Time:"2020-10-07T14:16:02.826Z"}	DataRate:"SF7BW125"}]}
258	Node_Time:"[2020	10	7	14	16	12	476942	Air_Time:"102656000"	GwTime:"2020-10-07T14:16:12.6646788992"	TTN_Time:"2020-10-07T14:16:12.685792596Z"	NR_Time:"2020-10-07T14:16:12.765Z"}	DataRate:"SF7BW125"}]}
259	Node_Time:"[2020	10	7	14	16	22	478951	Air_Time:"102656000"	GwTime:"2020-10-07T14:16:22.6642698822"	TTN_Time:"2020-10-07T14:16:22.68468122Z"	NR_Time:"2020-10-07T14:16:22.776Z"}	DataRate:"SF7BW125"}]}
260	Node_Time:"[2020	10	7	14	16	32	480945	Air_Time:"102656000"	GwTime:"2020-10-07T14:16:32.6743184352"	TTN_Time:"2020-10-07T14:16:32.695788993Z"	NR_Time:"2020-10-07T14:16:32.826Z"}	DataRate:"SF7BW125"}]}
261	Node_Time:"[2020	10	7	14	16	42	482942	Air_Time:"102656000"	GwTime:"2020-10-07T14:16:42.67388267Z"	TTN_Time:"2020-10-07T14:16:42.696042523Z"	NR_Time:"2020-10-07T14:16:42.757Z"}	DataRate:"SF7BW125"}]}
262	Node_Time:"[2020	10	7	14	16	52	485990	Air_Time:"102656000"	GwTime:"2020-10-07T14:16:52.67398677Z"	TTN_Time:"2020-10-07T14:16:52.695119649Z"	NR_Time:"2020-10-07T14:16:52.796Z"}	DataRate:"SF7BW125"}]}
263	Node_Time:"[2020	10	7	14	17	2	488942	Air_Time:"102656000"	GwTime:"2020-10-07T14:17:02.6734793482"	TTN_Time:"2020-10-07T14:17:02.694552276Z"	NR_Time:"2020-10-07T14:17:02.827Z"}	DataRate:"SF7BW125"}]}
264	Node_Time:"[2020	10	7	14	17	12	490942	Air_Time:"102656000"	GwTime:"2020-10-07T14:17:12.6833316042"	TTN_Time:"2020-10-07T14:17:12.704330743Z"	NR_Time:"2020-10-07T14:17:12.757Z"}	DataRate:"SF7BW125"}]}
265	Node_Time:"[2020	10	7	14	17	22	492945	Air_Time:"102656000"	GwTime:"2020-10-07T14:17:22.6868702122"	TTN_Time:"2020-10-07T14:17:22.707736386Z"	NR_Time:"2020-10-07T14:17:22.796Z"}	DataRate:"SF7BW125"}]}
266	Node_Time:"[2020	10	7	14	17	32	494947	Air_Time:"102656000"	GwTime:"2020-10-07T14:17:32.6829205342"	TTN_Time:"2020-10-07T14:17:32.7038817Z"	NR_Time:"2020-10-07T14:17:33.076Z"}	DataRate:"SF7BW125"}]}
267	Node_Time:"[2020	10	7	14	17	42	496951	Air_Time:"102656000"	GwTime:"2020-10-07T14:17:42.68272737Z"	TTN_Time:"2020-10-07T14:17:42.705331035Z"	NR_Time:"2020-10-07T14:17:42.894Z"}	DataRate:"SF7BW125"}]}
268	Node_Time:"[2020	10	7	14	17	52	498942	Air_Time:"102656000"	GwTime:"2020-10-07T14:17:52.6924305072"	TTN_Time:"2020-10-07T14:17:52.713229795Z"	NR_Time:"2020-10-07T14:17:52.772Z"}	DataRate:"SF7BW125"}]}
269	Node_Time:"[2020	10	7	14	18	2	500944	Air_Time:"102656000"	GwTime:"2020-10-07T14:18:02.692438052"	TTN_Time:"2020-10-07T14:18:02.713740126Z"	NR_Time:"2020-10-07T14:18:03.155Z"}	DataRate:"SF7BW125"}]}
270	Node_Time:"[2020	10	7	14	18	12	502942	Air_Time:"102656000"	GwTime:"2020-10-07T14:18:12.6919587362"	TTN_Time:"2020-10-07T14:18:12.712652851Z"	NR_Time:"2020-10-07T14:18:12.981Z"}	DataRate:"SF7BW125"}]}
271	Node_Time:"[2020	10	7	14	18	22	504942	Air_Time:"102656000"	GwTime:"2020-10-07T14:18:22.6920457722"	TTN_Time:"2020-10-07T14:18:22.713336183Z"	NR_Time:"2020-10-07T14:18:22.786Z"}	DataRate:"SF7BW125"}]}
272	Node_Time:"[2020	10	7	14	18	32	506946	Air_Time:"102656000"	GwTime:"2020-10-07T14:18:32.7017457772"	TTN_Time:"2020-10-07T14:18:32.723826853Z"	NR_Time:"2020-10-07T14:18:33.857Z"}	DataRate:"SF7BW125"}]}
273	Node_Time:"[2020	10	7	14	18	42	508944	Air_Time:"102656000"	GwTime:"2020-10-07T14:18:42.7014215832"	TTN_Time:"2020-10-07T14:18:42.734344623Z"	NR_Time:"2020-10-07T14:18:43.070Z"}	DataRate:"SF7BW125"}]}
274	Node_Time:"[2020	10	7	14	18	52	510942	Air_Time:"102656000"	GwTime:"2020-10-07T14:18:52.70109641Z"	TTN_Time:"2020-10-07T14:18:52.721982304Z"	NR_Time:"2020-10-07T14:18:53.427Z"}	DataRate:"SF7BW125"}]}
275	Node_Time:"[2020	10	7	14	19	2	512951	Air_Time:"102656000"	GwTime:"2020-10-07T14:19:02.7007911662"	TTN_Time:"2020-10-07T14:19:02.721632732Z"	NR_Time:"2020-10-07T14:19:03.329Z"}	DataRate:"SF7BW125"}]}
276	Node_Time:"[2020	10	7	14	19	12	514945	Air_Time:"102656000"	GwTime:"2020-10-07T14:19:12.7105880992"	TTN_Time:"2020-10-07T14:19:12.731193198Z"	NR_Time:"2020-10-07T14:19:13.155Z"}	DataRate:"SF7BW125"}]}
277	Node_Time:"[2020	10	7	14	19	22	516944	Air_Time:"102656000"	GwTime:"2020-10-07T14:19:22.7105712582"	TTN_Time:"2020-10-07T14:19:22.731176228Z"	NR_Time:"2020-10-07T14:19:23.495Z"}	DataRate:"SF7BW125"}]}
278	Node_Time:"[2020	10	7	14	19	32	518942	Air_Time:"102656000"	GwTime:"2020-10-07T14:19:32.7103021662"	TTN_Time:"2020-10-07T14:19:32.731183757Z"	NR_Time:"2020-10-07T14:19:32.824Z"}	DataRate:"SF7BW125"}]}

Figure 4.13: the data as constructed by Node-Red

4.13. A python script was developed specifically to read the data from the file and calculate the average latency for every SF.

Fig 4.10 shows the flowchart of the algorithm that had been followed inside the Node to get the timestamp data from the NTP-server, while Fig 4.11 shows the flowchart that had been followed by both Node-Red to extract the timing information for every stage then write those data into a .csv file, then a python script has been developed to calculate the average than draw the Data.

The python script for the Sensor Node, Node-red Nodes, extracting the data from the .csv file, calculate and draw the average Latency can be found in B.

fig 4.12 and fig 4.13 demonstrate the components that contribute to the latency from every stage.

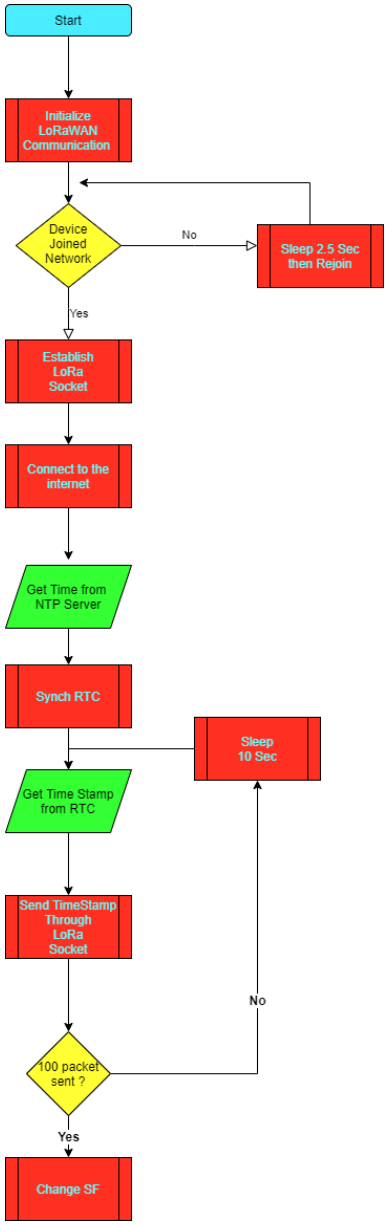


Figure 4.14: Latency algorithm for constructing the Packet

The Latency results

As can be seen in Fig 4.16 and Fig 4.17 the TOA which is a LoRa protocol characteristic is dominating over all the other latency components As TOA =2400 milliseconds for SF12 while it is only 100 milliseconds for SF7 however it is still far larger than the latency that occurs from the Gateway processing, TTN decoding

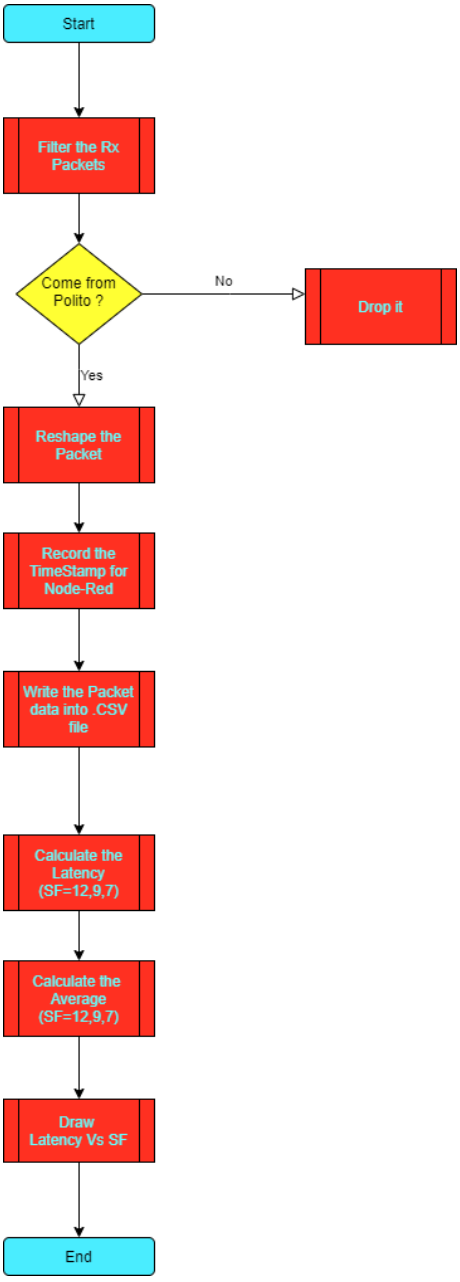


Figure 4.15: Latency algorithm for analyzing the data of the received packet

and transmitting the LoRa Packet which is as can be seen a fraction of a millisecond.

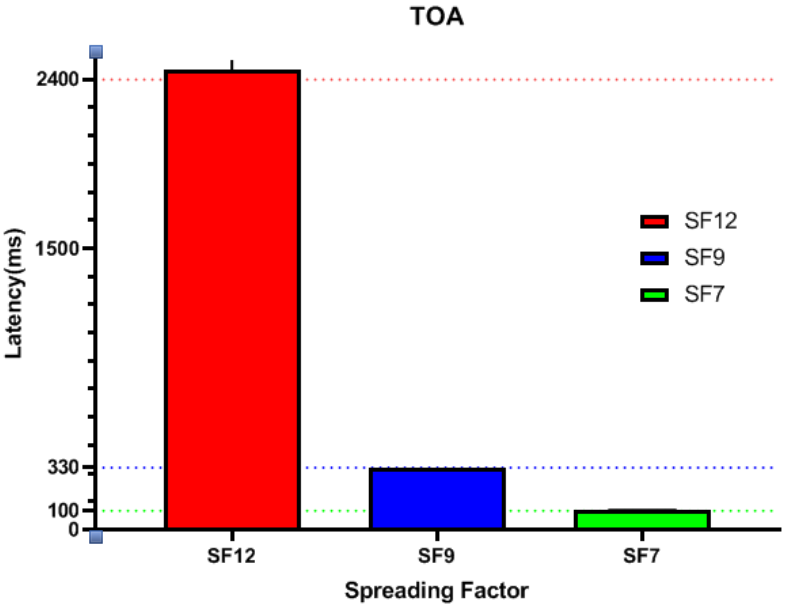


Figure 4.16: The TOA contribution part to the latency

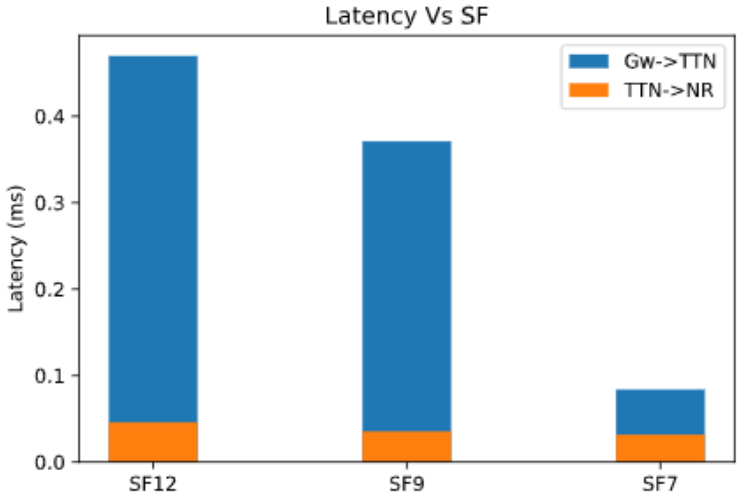


Figure 4.17: Other Latency components

Chapter 5

Conclusion and future work

5.1 Conclusion

We had demonstrated the applicability of LoRaWAN technology to use it as the Communication protocol for the Weather station IoT system, the measurements showed that LoRaWAN can achieve acceptable ranges of data communications which were 600 meters, but this also should be chosen carefully concerning the environment(in the case where there is only one gateway in the area) since the transmission is severely affected by the surroundings as we demonstrated in the Test point 2. In our experiments we used a foreign web-server like the things network which add some more additional constrains above those who already exist for using the ISM Band; that limits the update time for packets transmission and the number of messages per day which limited the usage of the LoRa Protocol in our system case to only one suitable spreading factor which is SF7, fortunately, SF7 can satisfy the data-Rate of 5.4Kbps, and it has a minimum latency of around 100 milliseconds with ability to transmit 417 packets per day (packet update time every 3.5 minutes) with a packet latency of around 100 milliseconds as the measurements showed. We then introduced a suggestion to improve this situation by removing the redundant information like the ID and the time from the packet, hence we were able to reduce the packet size to only 10 bytes which has an update time of 2.7 minutes, then we introduced the "Sensors States Byte" which is a byte that we are going to send instead of sending the whole data every transmission which reduced the update time to 2.2 minutes which give us more reliability transmit our information under the TTN restrictions.

5.2 Future Work

Future work could be considered is considering using a less constrained web-server that has fewer restrictions on the number of packets per day like the open-source web-server "chirpstack" which is Linux based web-server. Also using a gateway that allows being synchronized with the system like for example the one based on the Raspberry Pi(using The concentrator module iC880A)[27] this shall give more accurate timing analysis and experiment parameters control one good latency end to end analysis had been done on [28] where the researcher was examining many different parameters on the latency including the IP connection used to transmit the data from the gateway to the web-server. Also one of the interesting features that can be included in future works is to take the advantage of the fact that the Fipy is based on the ESP32 which has 2 interesting features that can significantly reduce the power usage of the Node which are the Ultra-Low power feature(ULP) and the Dual-core that can be used efficiently with the freeRtos that is already supported by this µController.

Appendix A

A

The **boot.py** to include Libraries and define the variables

```
#########################################################################
# Importing the Required Liberaries
#########################################################################
from network import LoRa
from machine import UART
#from network import WLAN
from dth import DTH          # Custom Library of the dHT11 sensor
import machine
from machine import Pin
import socket                # To do the connections
import ubinascii             # To do the hexdecimal Convesions
import struct                #
import ustruct
import time
import ujson
import pycom                 # To add the hardware



#########################################################################
# Defning Variables
#########################################################################

# Creating DHT11 Object on pin 3 of the Fipy
DHT_11 = DTH(Pin('P3', mode=Pin.OPEN_DRAIN),0)

# Parameters
SF7_DR5 = 5
SF8_DR4 = 4
SF9_DR3 = 3
```

```
    SF10_DR2= 2
    SF11_DR1= 1
    SF12_DR0= 0

    # Channel's Freq.
    Ch_0 = 868100000
    Ch_1 = 868300000
    Ch_2 = 868500000

    SF_Array = [SF12_DR0,SF11_DR1,SF10_DR2,SF9_DR3,SF8_DR4,SF7_DR5]
```

The **main.py** file

```

##############################################################################
# Importing the Required Liberaries (done in boot.py)

##############################################################################
def LoRa_Intilize():
    ##########################################################################
    #  LoraWan Configurations
    ##########################################################################
    # Initialise LoRa in LORAWAN mode.
    lora = LoRa(mode=LoRa.LORAWAN)           # Lora Constructor - Creat lora Object from the Lora Class
    #lora = LoRa(mode=LoRa.LORAWAN, adr=True, tx_retries=1)

    # create an ABP authentication paramsz
    # Lora MAC Address ---> 70B3D5499800DEB4
    dev_addr  = struct.unpack( ">l" , ubinascii.unhexlify( '260117BD' ))[ 0 ] # these settings can be found from TTN
    nwk_swkey = ubinascii.unhexlify( '9E81780F9C179F3D559942DB692CA8D8' ) # these settings can be found from TTN
    app_swkey = ubinascii.unhexlify( 'E6B0790498B89248FF46EB374CDC8B76' ) # these settings can be found from TTN

    # set the 3 default channels to the same frequency ( EU Configurations)
    lora.add_channel( 0 , frequency=868100000 , dr_min=0 , dr_max=5 ) # dr = 0 ---> S.F. = 12 (Slower Transmission Time but Longer Range)
    lora.add_channel( 1 , frequency=868100000 , dr_min=0 , dr_max=5 ) # dr = 5 ---> S.F. = 7 (Faster but Lower Range)
    lora.add_channel( 2 , frequency=868100000 , dr_min=0 , dr_max=5 )

    for i in range( 3 , 16 ):
```

```
26          lora.remove_channel(i)
27
28      # join a network using ABP (Activation By Personalisation)
29      lora.join(activation=LoRa.ABP, auth=(dev_addr, nwk_swkey,
        app_swkey), timeout=0)     # timeout=0 --> if it doesn't connect
        instantinusly, Continue the Code
30
31
32      # wait until  the module  has joined the network
33      while not lora.has_joined():
34          pycom.rgbled(0xff0000)       # Led Red -----> No connected to
        Lora Network yet
35          time.sleep(2.5)
36          print('Not joined yet...')
37
38      print('joined ')
39      #pycom.rgbled(0x00ff00)           # Led Green ---> It's Connected
        now
40
41  def LoRa_Scocket(SF):
42      # create a LoRa socket
43      LoRa_S = socket.socket(socket.AF_LORA, socket.SOCK_RAW)
44      LoRa_S.setsockopt(socket.SOL_LORA, socket.SO_DR, SF )
45      # make the socket non-blocking       -- We can send & Receive data
        w/o having to hold on the rest of our code
46      LoRa_S.setblocking( False )
47      return LoRa_S
48
49
50  def Sensor_Data():
51      ######################################################################
52      #  DHT11 Data   - Collect Sensor Data
53      ######################################################################
54      pycom.heartbeat(False)
55      pycom.rgbled(0x000008) # blue
56      result = DHT_11.read()
57      if result.is_valid():
58          pycom.rgbled(0x001000) # green
59          print("Temperature: %d C" % result.temperature)
60          print("Humidity: %d %%" % result.humidity)
61
62      hum = result.humidity
63      temp = result.temperature
64     ##################### Payload Short#####################
65      Con_Data = int(str(temp)+str(hum))
66      return  ustruct.pack('h',Con_Data)
67
68  def Fixed_Data():
69   ################### Payload Long#########################
```

```
70      Id = 'A1'
71      pm2_5 = 56
72      pm10 = 61
73      temp = 22
74      humidity = 48
75      #GPS  = (44.47812,7.50647)                  # Tuple Lat. and Long.
76      data = [Id,pm2_5,pm10,temp,humidity]
77      return ujson.dumps(data).encode()     # PayLoad_Long = Fixed_Data
        ()
78
79  LoRa_Intilize()
80  LoRa_S = LoRa_Scocket(SF12_DR0)
81  Count_Tx_Pack = 0
82  Count_SF = 0
83  Keep_transmission = 1
84  #print("Now Count_SF = ", str(Count_SF))                              #
        Debug SF value now
85  while Keep_transmission:
86      ##########################################################################
87      PayLoad_Long = Fixed_Data()
88      LoRa_S.send(PayLoad_Long)
89      pycom.heartbeat(False)
90      Count_Tx_Pack = Count_Tx_Pack + 1
91      #print("Count_Tx_Pack = ", str(Count_Tx_Pack))                       #
        Debug How many packet was sent
92      if Count_Tx_Pack == 100:            # 100 Packet for UL
93          Count_Tx_Pack = 0
94          if Count_SF < 5:
95              Count_SF = Count_SF + 1
96             # print("Now Count_SF = ", str(Count_SF))                     #
        Debug the changing of SF
97              LoRa_S  = LoRa_Scocket(int(SF_Array[Count_SF]))
98          else:
99              Keep_transmission = 0
100             #print("Transmission Ends")                                 # If
        packet still < 100 keep using same SF
101     time.sleep(10)                                                      #
        Should ba change according to the Fair access policy.
```

The **DH11 Library** to extract valid data from the sensor

```
1   import time
2   from machine import enable_irq, disable_irq,  Pin
3
4
5   class DTHResult:
6       'DHT sensor result returned by DHT.read() method'
7
```

```
8       ERR_NO_ERROR = 0
9       ERR_MISSING_DATA = 1
10      ERR_CRC = 2
11
12      error_code = ERR_NO_ERROR
13      temperature = -1
14      humidity = -1
15
16      def __init__(self, error_code, temperature, humidity):
17          self.error_code = error_code
18          self.temperature = temperature
19          self.humidity = humidity
20
21      def is_valid(self):
22          return self.error_code == DTHResult.ERR_NO_ERROR
23
24
25  class DTH:
26      'DHT sensor (dht11, dht21,dht22) reader class for Pycom'
27
28      #__pin = Pin('P3', mode=Pin.OPEN_DRAIN)
29      __dhttype = 0
30
31      def __init__(self, pin, sensor=0):
32          self.__pin = pin
33          self.__dhttype = sensor
34          self.__pin(1)
35          time.sleep(1.0)
36
37
38      def read(self):
39          #time.sleep(1)
40
41          # send initial high
42          #self.__send_and_sleep(1, 0.025)
43
44          # pull down to low
45          self.__send_and_sleep(0, 0.019)
46
47          # collect data into an array
48          data = self.__collect_input()
49          #print(data)
50          # parse lengths of all data pull up periods
51          pull_up_lengths = self.__parse_data_pull_up_lengths(data)
52          # if bit count mismatch, return error (4 byte data + 1 byte
    checksum)
53          #print(pull_up_lengths)
54          #print(len(pull_up_lengths))
55          if len(pull_up_lengths) != 40:
```

```
56              return DTHResult(DTHResult.ERR_MISSING_DATA, 0, 0)
57
58          # calculate bits from lengths of the pull up periods
59          bits = self.__calculate_bits(pull_up_lengths)
60
61          # we have the bits, calculate bytes
62          the_bytes = self.__bits_to_bytes(bits)
63          #print(the_bytes)
64          # calculate checksum and check
65          checksum = self.__calculate_checksum(the_bytes)
66          if the_bytes[4] != checksum:
67              return DTHResult(DTHResult.ERR_CRC, 0, 0)
68
69          # ok, we have valid data, return it
70          [int_rh, dec_rh, int_t, dec_t, csum] = the_bytes
71          if self.__dhttype==0:       #dht11
72              rh = int_rh          #dht11 20% ~ 90%
73              t = int_t    #dht11 0..50 C
74          else:               #dht21,dht22
75              rh = ((int_rh * 256) + dec_rh)/10
76              t = (((int_t & 0x7F) * 256) + dec_t)/10
77              if (int_t & 0x80) > 0:
78                  t *= -1
79          return DTHResult(DTHResult.ERR_NO_ERROR, t, rh)
80
81      def __send_and_sleep(self, output, mysleep):
82          self.__pin(output)
83          time.sleep(mysleep)
84
85      def __collect_input(self):
86          # collect the data while unchanged found
87          unchanged_count = 0
88          # this is used to determine where is the end of the data
89          max_unchanged_count = 100
90          last = -1
91          data = []
92          m = bytearray(800)          # needs long sample size to grab
    all the bits from the DHT
93          irqf = disable_irq()
94          self.__pin(1)
95          for i in range(len(m)):
96              m[i] = self.__pin()       ## sample input and store value
97          enable_irq(irqf)
98          for i in range(len(m)):
99              current = m[i]
100             data.append(current)
101             if last != current:
102                 unchanged_count = 0
103                 last = current
```

```
104             else:
105                 unchanged_count += 1
106                 if unchanged_count > max_unchanged_count:
107                     break
108         #print(data)
109         return data
110 
111     def __parse_data_pull_up_lengths(self, data):
112         STATE_INIT_PULL_DOWN = 1
113         STATE_INIT_PULL_UP = 2
114         STATE_DATA_FIRST_PULL_DOWN = 3
115         STATE_DATA_PULL_UP = 4
116         STATE_DATA_PULL_DOWN = 5
117 
118         state = STATE_INIT_PULL_UP
119 
120         lengths = [] # will contain the lengths of data pull up
    periods
121         current_length = 0 # will contain the length of the previous
    period
122 
123         for i in range(len(data)):
124 
125             current = data[i]
126             current_length += 1
127 
128             if state == STATE_INIT_PULL_DOWN:
129                 if current == 0:
130                     # ok, we got the initial pull down
131                     state = STATE_INIT_PULL_UP
132                     continue
133                 else:
134                     continue
135             if state == STATE_INIT_PULL_UP:
136                 if current == 1:
137                     # ok, we got the initial pull up
138                     state = STATE_DATA_FIRST_PULL_DOWN
139                     continue
140                 else:
141                     continue
142             if state == STATE_DATA_FIRST_PULL_DOWN:
143                 if current == 0:
144                     # we have the initial pull down, the next will be
     the data pull up
145                     state = STATE_DATA_PULL_UP
146                     continue
147                 else:
148                     continue
149             if state == STATE_DATA_PULL_UP:
```

```
150                 if current == 1:
151                     # data pulled up, the length of this pull up will
    determine whether it is 0 or 1
152                     current_length = 0
153                     state = STATE_DATA_PULL_DOWN
154                     continue
155                 else:
156                     continue
157             if state == STATE_DATA_PULL_DOWN:
158                 if current == 0:
159                     # pulled down, we store the length of the
    previous pull up period
160                     lengths.append(current_length)
161                     state = STATE_DATA_PULL_UP
162                     continue
163                 else:
164                     continue
165 
166         return lengths
167 
168     def __calculate_bits(self, pull_up_lengths):
169         # find shortest and longest period
170         shortest_pull_up = 1000
171         longest_pull_up = 0
172 
173         for i in range(0, len(pull_up_lengths)):
174             length = pull_up_lengths[i]
175             if length < shortest_pull_up:
176                 shortest_pull_up = length
177             if length > longest_pull_up:
178                 longest_pull_up = length
179 
180         # use the halfway to determine whether the period it is long
    or short
181         halfway = shortest_pull_up + (longest_pull_up -
    shortest_pull_up) / 2
182         bits = []
183 
184         for i in range(0, len(pull_up_lengths)):
185             bit = False
186             if pull_up_lengths[i] > halfway:
187                 bit = True
188             bits.append(bit)
189 
190         return bits
191 
192     def __bits_to_bytes(self, bits):
193         the_bytes = []
194         byte = 0
```

```
195
196         for i in range(0, len(bits)):
197             byte = byte << 1
198             if (bits[i]):
199                 byte = byte | 1
200             else:
201                 byte = byte | 0
202             if ((i + 1) % 8 == 0):
203                 the_bytes.append(byte)
204                 byte = 0
205         #print(the_bytes)
206         return the_bytes
207
208     def __calculate_checksum(self, the_bytes):
209         return the_bytes[0] + the_bytes[1] + the_bytes[2] + the_bytes
    [3] & 255
```

The Node-Red **Function Node**

```
1
2  msg.topic = "Data"
3  var gateways = msg.metadata.gateways;
4  msg.payload = {
5                  Counter: msg.counter,
6                  PL: msg.payload,
7                  DataRate: msg.metadata.data_rate,
8                  rssi: gateways.map(gw => gw.rssi),
9                  snr: gateways.map(gw => gw.snr),
10                 //Gw: gateways.map(gw => gw.gtw_id),
11                 Time: msg.metadata.time,
12                 gateways: gateways.map(gw => {
13                 return {
14                   GW_ID: gw.gtw_id,
15                   location: {
16                        lat: gw.latitude,
17                        lng: gw.longitude,
18                        alt: gw.altitude
19                          },
20                    }
21                 })
22
23 }
24
25 return msg
```

$\mathcal{O}(n \log n)$

`numpy`

Appendix B

B

Latency Python Script **Sensor Node**

```
1
2
3 ##########################################################################
4 #            Reforming Data to the Correct Form
5 ##########################################################################
6
7 import re
8 from datetime import datetime, time, date
9
10 def replace_bad_chars(time_string):
11     time_string = re.sub(r'[\"ZT]', '', time_string)
12     return time_string.split(":")
13
14
15 def convert_to_proper_time(time_array, bad_chars=None, cdate=None):
16     if bad_chars is None:
17         assert len(time_array) == 4
18         [hr, min, sec, msec] = time_array
19     else:
20         #assert len(time_array) == 3
21         [hr, mint, secs] = list(replace_bad_chars(time_array))
22         sec =  int(secs.split('.')[0])
23         try:
24             msec =  int(secs.split('.')[1])
25         except:
26             msec = 0
27     if cdate is None:        # if data is in form
28         return datetime.combine(date.today(), time(int(hr), int(mint)
    , sec, msec))
29     else:
```

```
30          return datetime.combine(cdate, time(int(hr), int(mint), sec,
    msec))
31
32  def delet_mult_qot(number_string):              # Delet the " " from
    the Data
33      return re.sub('"', '', number_string)
34
35
36
37
38
39  ##########################################################################
40  #Read the Data from the .csv file then categorize it according to SF
41  ##########################################################################
42
43  def build_record(data_row):
44      [counter, year, month, day, node_hr, node_min, node_sec,
    node_msec, air_time, gw_time, ttn_time, nr_time, delay, data_rate]
     = data_row.strip().split(',')       # assign the row
45      cdate = date(int(year), int(month), int(day))
46
47      return [convert_to_proper_time([int(node_hr), int(node_min), int(
    node_sec), int(node_msec)], bad_chars=None, cdate=cdate),
48              float(delete_mult_quot(air_time))/1000000.0,
49              convert_to_proper_time(gw_time, bad_chars=True, cdate=
    cdate),
50              convert_to_proper_time(ttn_time, bad_chars=True, cdate=
    cdate),
51              convert_to_proper_time(nr_time, bad_chars=True, cdate=
    cdate),
52              (float(delete_mult_quot(delay))/1000.0) - 10,
53              delete_mult_quot(data_rate)
54              ]
55
56  data0FLAG = "SF12BW125"
57  data1FLAG = "SF9BW125"
58  data2FLAG = "SF7BW125"
59
60  data0 = []
61  data1 = []
62  data2 = []
63  #with open("L_poli.csv", 'r') as csvfile:
64  with open("Test_Latency.csv", 'r') as csvfile:
65      data = csvfile.readlines()      # Read a line from the .csv file
66      for i in range(1,len(data)):    # ignore the Counter and start
    read after until SF
67          record = build_record(data[i])
68          #print(record)
69          if record[-1] == data0FLAG:
```

```
70              data0.append(record)
71          elif record[-1] == data1FLAG:
72              data1.append(record)
73          else:
74              data2.append(record)
75 ##########################################################################
76 #           Test
77 #           len(data0) = 98
78 #           data0[0] =  [datetime.datetime(2020, 9, 27, 17, 28, 25,
       843596),
79 #  2.465792,
80 #  datetime.datetime(2020, 9, 27, 17, 28, 28, 392),
81 #  datetime.datetime(2020, 9, 27, 17, 28, 28, 413),
82 #  datetime.datetime(2020, 9, 27, 17, 28, 28, 488),
83 #  0.01800049999999942,
84 #  'SF12BW125']
85 ##########################################################################
86
87
88
89 ##########################################################################
90 #           Calculate the Latency ( subtract the timing )
91 ##########################################################################
92
93 def calculate_delays(record):
94     node_time, air_time, gw_time, ttn_time, nr_time, ref_delay, _ =
       record
95     time_deltas = [gw_time - node_time, ttn_time - gw_time, nr_time -
       ttn_time ]
96     return  list(map(lambda x: x.total_seconds(), time_deltas))
97
98
99 ##############################################################
100 #    Assign the Latency of each SF to an Array , data0 = SF12
101 ##############################################################
102
103 data0_delays =[]
104 for record in data0:
105     data0_delays.append(calculate_delays(record))
106
107 data1_delays =[]
108 for record in data1:
109     data1_delays.append(calculate_delays(record))
110
111 data2_delays =[]
112 for record in data2:
113     data2_delays.append(calculate_delays(record))
114
115
```

```
116 ##########################################################
117 #            Calculate each latency component alone
118 ##########################################################
119
120
121 SF12_node_gw_delays = [delay[0] for delay in data0_delays]
122 SF12_node_gw_delay_average = np.mean(SF12_node_gw_delays)*1000
123 SF12_node_gw_delay_std = np.std(SF12_node_gw_delays)
124
125 SF12_gw_ttn_delays = [delay[1] for delay in data0_delays]
126 SF12_gw_ttn_delay_average = np.mean(SF12_gw_ttn_delays)*1000
127 SF12_gw_ttn_delay_std = np.std(SF12_gw_ttn_delays)
128
129 SF12_ttn_tnr_delays = [delay[2] for delay in data0_delays]
130 SF12_ttn_tnr_delay_average = np.mean(SF12_ttn_tnr_delays)*1000
131 SF12_ttn_tnr_delay_std = np.std(SF12_ttn_tnr_delays)
132
133 SF9_node_gw_delays = [delay[0] for delay in data1_delays]
134 SF9_node_gw_delay_average = np.mean(SF9_node_gw_delays)*1000
135
136 SF9_node_gw_delay_std = np.std(SF9_node_gw_delays)
137
138 SF9_gw_ttn_delays = [delay[1] for delay in data1_delays]
139 SF9_gw_ttn_delay_average = np.mean(SF9_gw_ttn_delays)*1000
140 SF9_gw_ttn_delay_std = np.std(SF9_gw_ttn_delays)
141
142 SF9_ttn_tnr_delays = [delay[2] for delay in data1_delays]
143 SF9_ttn_tnr_delay_average = np.mean(SF9_ttn_tnr_delays)*1000
144
145 SF9_ttn_tnr_delay_std = np.std(SF9_ttn_tnr_delays)
146
147
148 SF7_node_gw_delays = [delay[0] for delay in data2_delays]
149 SF7_node_gw_delay_average = np.mean(SF7_node_gw_delays)*1000
150
151 SF7_node_gw_delay_std = np.std(SF7_node_gw_delays)
152
153 SF7_gw_ttn_delays = [delay[1] for delay in data2_delays]
154 SF7_gw_ttn_delay_average = np.mean(SF7_gw_ttn_delays)*1000
155 SF7_gw_ttn_delay_std = np.std(SF7_gw_ttn_delays)
156
157 SF7_ttn_tnr_delays = [delay[2] for delay in data2_delays]
158 SF7_ttn_tnr_delay_average = np.mean(SF7_ttn_tnr_delays)*1000
159
160 SF7_ttn_tnr_delay_std = np.std(SF7_ttn_tnr_delays)
161
162
163
164 ##########################################################
```

```
#           Plotting the results
#############################################################
import matplotlib.pyplot as plt
import numpy as np

labels = ['SF12', 'SF9', 'SF7']
node_gw_delay_means = [g1_node_gw_delay_average,
    g2_node_gw_delay_average, g3_node_gw_delay_average ]
gw_ttn_delay_means = [g1_gw_ttn_delay_average,
    g2_gw_ttn_delay_average, g3_gw_ttn_delay_average ]
ttn_tnr_delay_means = [g1_ttn_tnr_delay_average,
    g2_ttn_tnr_delay_average, g3_ttn_tnr_delay_average ]

node_gw_delay_std = [SF12_node_gw_delay_std, SF9_node_gw_delay_std,
    SF7_node_gw_delay_std]
gw_ttn_delay_std = [SF12_gw_ttn_delay_std, SF9_gw_ttn_delay_std,
    SF7_gw_ttn_delay_std]
ttn_tnr_delay_std = [SF12_ttn_tnr_delay_std, SF9_ttn_tnr_delay_std,
    SF7_ttn_tnr_delay_std]

width = 0.35       # the width of the bars: can also be len(x)
    sequence

fig, ax = plt.subplots()

ax.bar(labels, node_gw_delay_means, width, yerr=node_gw_delay_std,
    label='TOA')
#ax.semilogy(labels, node_gw_delay_means, width, yerr=
    node_gw_delay_std, label='TOA')
#ax.bar(labels, gw_ttn_delay_means, width,
                    #yerr=gw_ttn_delay_std,
#                     label='Gw->TTN')
#ax.bar(labels, ttn_tnr_delay_means, width,
                    #yerr=ttn_tnr_delay_std,
 #                    label='TTN->NR')

ax.legend()
ax.set_ylabel('Latency (ms)')
ax.set_title('Latency Vs SF')
ax.legend()

plt.show()
```

The Node-Red **Latency Function Node**

```
1
2
3  msg.topic = "Data"
4  var gateways = msg.metadata.gateways;
5  msg.payload = {
6                  Counter: msg.counter,
7                  PL: msg.payload,
8                  DataRate: msg.metadata.data_rate,
9                  gateways: gateways.map(gw => {
10                 return {
11                   GW_ID:      gw.gtw_id,
12                   TOA:        msg.metadata.airtime,
13                   Gw_Time:    msg.time,
14                   TTN_Time:   msg.metadata.time,
15                     }
16                 })
17 }
18
19 return msg
```

Bibliography

[1] Edoardo Giusto, Renato Ferrero, Filippo Gandino, Bartolomeo Montrucchio, Maurizio Rebaudengo, and Mingyang Zhang. «Particulate Matter Monitoring in Mixed Indoor/Outdoor Industrial Applications: A Case Study». In: *IEEE International Conference on Emerging Technologies and Factory Automation, ETFA*. Vol. 2018-September. Institute of Electrical and Electronics Engineers Inc., Oct. 2018, pp. 838–844. ISBN: 9781538671085. DOI: 10.1109/ETFA.2018.8502644 (cit. on p. ii).

[2] Yaron Ogen. «Assessing nitrogen dioxide (NO2) levels as a contributing factor to coronavirus (COVID-19) fatality». In: *Science of the Total Environment* (2020). ISSN: 18791026. DOI: 10.1016/j.scitotenv.2020.138605 (cit. on p. 1).

[3] Xiao Wu, Rachel Nethery, Benjamin Sabath, Danielle Braun, and Francesca Dominici. «Exposure to air pollution and COVID-19 mortality in the United States: A nationwide cross-sectional study». In: *medRxiv : the preprint server for health sciences* (Apr. 2020), p. 2020.04.05.20054502. DOI: 10.1101/2020.04.05.20054502. URL: https://doi.org/10.1101/2020.04.05.20054502 (cit. on p. 1).

[4] Silvia Comunian, Dario Dongo, Chiara Milani, and Paola Palestini. «Air Pollution and COVID-19: The Role of Particulate Matter in the Spread and Increase of COVID-19's Morbidity and Mortality». In: *International Journal of Environmental Research and Public Health* 17.12 (June 2020), p. 4487. ISSN: 1660-4601. DOI: 10.3390/ijerph17124487. URL: https://www.mdpi.com/1660-4601/17/12/4487 (cit. on p. 1).

[5] wikipedia. *LPWAN*. Oct. 2020 (cit. on p. 2).

[6] LoRa™ Alliance. «LoRaWAN™ Regional Parameters v1.1rA». In: *LoRaWAN™ 1.1 Specif.* (2017) (cit. on pp. 4, 13).

[7] Sakshama Ghoslya. *LoRa: Symbol Generation* (cit. on pp. 5, 6).

[8] Noman Aftab, Syed Ali Raza Zaidi, and Des McLernon. «Scalability analysis of multiple LoRa gateways using stochastic geometry». In: *Internet of Things* 9 (Mar. 2020), p. 100132. ISSN: 25426605. DOI: 10.1016/j.iot.2019.100132 (cit. on p. 5).

[9] Dmitry Bankov, Evgeny Khorov, and Andrey Lyakhov. «On the limits of LoRaWAN channel access». In: *Proceedings - 2016 International Conference on Engineering and Telecommunication, EnT 2016*. 2017. ISBN: 9781509045532. DOI: 10.1109/EnT.2016.9 (cit. on pp. 4, 7, 14).

[10] Semtech. *LoRa Modulation Basics AN1200.22*. 2015 (cit. on p. 9).

[11] Simtech. *What are LoRa® and LoRaWAN®?* (Cit. on pp. 9, 15).

[12] Ellen Li. «LoRapedia, an Introduction of LoRa and LoRaWAN Technology». In: *https://www.seeedstudio.com/blog/2020/08/03/lorapedia-an-introduction-of-lora-and-lorawan-technology/* (Aug. 2020) (cit. on p. 10).

[13] Semtech. *LoRa Modulation Basics AN1200.22*. 2015 (cit. on p. 13).

[14] Eduardo Ruano, Bernard Tourancheau, and Olivier Alphand. *LoRa TM protocol Evaluations, limitations and practical test*. Tech. rep. 2016 (cit. on p. 13).

[15] Semtech. *WIRELESS & SENSING Revision 2 May 2015 © 2015 Semtech Corporation 1 SX1272/3/6/7/8 LoRa Modem Design Guide*. Tech. rep. 2015 (cit. on pp. 14, 51).

[16] *Download SX1272 LoRa calculator by Semtech SA*. URL: https://sx1272-lora-calculator.software.informer.com/download/ (cit. on pp. 14, 51).

[17] Arjan Avbentem. *Airtime calculator for LoRaWAN Retrieved from https://avbentem.github.io/airtime-calculator/ttn/eu868*. Sept. 2020 (cit. on pp. 15, 52).

[18] Simtech. *An In-depth look at LoRaWAN® Class A Devices* (cit. on p. 16).

[19] Lennart Nordin. *LoRaWAN Device Classes: A, B and C*. 2018 (cit. on pp. 17–19).

[20] Diogo Ferreira. «Mockups, Authentication Service Completed & more». In: *https://medium.com/electric-cars-charging-system/week-3-mockups-authentication-service-completed-more-41f1d865de0d* (Oct. 2018) (cit. on p. 27).

[21] Pycom. *Pycom_002_Specsheets_FiPy_v2*. Tech. rep. (cit. on p. 32).

[22] *DHT11 Humidity & Temperature Sensor*. Tech. rep. URL: www.droboticsonline.com (cit. on p. 30).

[23] ETSI. *EN 300 220-2 - V3.2.1 - Short Range Devices (SRD) operating in the frequency range 25 MHz to 1 000 MHz; Part 2: Harmonised Standard for access to radio spectrum for non specific radio equipment*. Tech. rep. 2018. URL: https://portal.etsi.org/TB/ETSIDeliverableStatus.aspx (cit. on p. 34).

[24] Arjanvanb Arjan. *Best practices to limit application payloads*. Apr. 2016 (cit. on p. 37).

[25] Node-Red. *Interval length Node*. Oct. 2019 (cit. on p. 39).

[26] The Things Network Global Team. *LoRaWAN® distance world record broken, twice. 766 km (476 miles) using 25mW transmission power*. July 2019 (cit. on p. 46).

[27] Gonzalo Casas. *From zero to LoRaWAN in a weekend*. Nov. 2018 (cit. on p. 65).

[28] Albert Potsch and Florian Hammer. «Towards End-to-End Latency of LoRaWAN: Experimental Analysis and IIoT Applicability». In: *IEEE International Workshop on Factory Communication Systems - Proceedings, WFCS*. 2019. ISBN: 9781728112688. DOI: 10.1109/WFCS.2019.8758033 (cit. on p. 65).