UNIVERSITÉ SAVOIE MONT BLANC

LoRa® - LoRaWAN® and Internet Of Things

Sylvain MONTAGNY

A LOW POWER, LONG RANGE, WIRELESS TECHNOLOGY

"This book is a tremendous resource for anyone interested in LoRaWAN technology. You will simply discover why LoRaWAN is the premier leading solution for large scale LPWAN deployments. Many thanks to the Savoie Mont Blanc University team on behalf of the LoRa Alliance."

Ms Donna Moore, CEO and Chairwoman of the **LoRa Alliance**

Reviewers

Semtech : Olivier Seller, Damian Gabino Rascon, Joseph Knapp

Université Savoie Mont Blanc : Florent Lorne, Antoine Augagneur, Marie-Line Fournier

About this Book

This book is the result of work carried out by the teaching staff of Savoie Mont Blanc University in the following postgraduate master's degrees:

- Electronics and Embedded Systems
- Telecoms and Computer Networks

All remarks, modifications, improvements or corrections can be proposed on our website contact page: www.univ-smb.fr/lorawan/en/contact/

This book quotes many commercial brands (end-devices, LoRaWAN servers, IoT Platforms…). Even though the university has partners, none of these brands has financed this book and it is with total objectivity that all these chapters have been written.

This book on the Internet of Things and the LoRa / LoRaWAN protocols is updated regularly. On our website www.univ-smb.fr/lorawan you can subscribe to receive an email notification when new information is released. You will also find a set of tools, information and resources on LoRaWAN that you are free to use.

About the Author

Sylvain MONTAGNY

I am an Associate Professor at the Savoie Mont Blanc University since 2006 and I am specialized in microprocessor systems and the Internet of Things. I am in charge of the postgraduate master in Electronics and Embedded Systems where most courses have a close relationship with industry. The new educational trends encourage us to offer high quality on line content. I took up the challenge to write a book with clear and simple information and I hope that this work will give you full satisfaction.

Savoie Mont Blanc University is an institutional member of the LoRa Alliance since 2021.

LoRa Alliance®
Wide Area Networks for IoT

LoRa® and LoRaWAN® course on video

Despite the complete content of this book, a more interactive version is available on video on an e-Learning platform. The LoRa-LoRaWAN video course is available on UDEMY.

This course is composed of 132 videos of approximately 4 minutes each. It includes additional information and details on the configuration of end-devices, gateways and gateways. You can ask your questions on the platform, instructors are there to answer.

One hour of these LoRaWAN videos is available for free.

You can ask a free voucher for educational purpose only.

So far, these videos are only available **in French**. **The English** version will be released shortly.

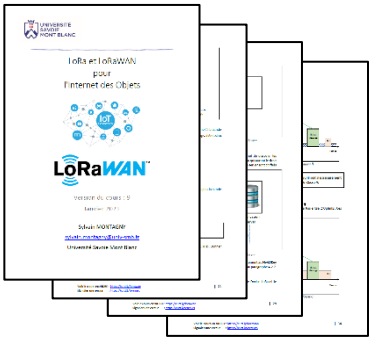
LoRaWAN training session with hands-on (online)

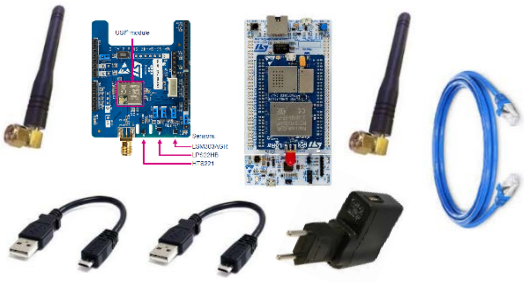
A 2 days 100% online training with instructors is also available in French (in English on demand).

- Equipment provided: We send you a LoRaWAN end-device and a gateway to build your own LoRaWAN network. You keep all the material at the end of the training.
- Reduced number of participants: We wish to personalize the content as much as possible in order to best meet your needs (maximum of 10 participants).
- Personalized follow-up: We are at your disposal throughout the training, and we remain available even after to answer all your questions.

Before the training

- You have access to the e-Learning educational platform
- You receive the teaching kit and have access to servers





During the training

- Follow-up of all participants.
- Set up of a LoRa transmission.
- Test of the transmission parameters (Bandwidth, channels, Spreading Factor).
- Configuration of your gateway, your LoRaWAN server. Device registration.
- Test of ABP and OTAA activation modes.
- Specification of Class A devices, uplink, downlink, confirmed, unconfirmed.
- Adaptive Data Rate (ADR).
- Data export with HTTP and MQTT protocols.
- Creation of your own LoRaWAN server (The Things Stack v3 and Chirpstack).
- Creation of your own IoT Platform.

Legends

List of important information

To be read very carefully

Note

Exercises

Validation of a result

Videos available on our website: www.univ-smb.fr/lorawan

Abbreviations and Acronyms

ABP	**A**ctivation **B**y **P**ersonalization
ADR	**A**daptive **D**ata **R**ate
AS	**A**pplication **S**erver
AppEUI	**A**pplication **E**xtended **U**nique **I**dentifier
AppKey	**A**pplication **K**ey
AppSKey	**A**pplication **S**ession **K**ey
BW	**B**and**w**idth
CDMA	**C**ode **D**ivision **M**ultiple **A**ccess
CHIRP	**C**ompressed **H**igh **I**ntensity **R**adar **P**ulse
CID	**C**ommand **ID**entifier (MAC Command)
CR	**C**oding **R**ate
CRC	**C**heck **R**edundancy **C**ycle
DevAddr	**D**evice **A**ddress
DevEUI	**D**evice **E**xtended **U**nique **I**dentifier
FDM	**F**requency **D**ivision **M**ultiplexing
FFT	**F**ast **F**ourier **T**ransform
HTTP	**H**yper**T**ext **T**ransfer **P**rotocol
HSM	**H**ardware **S**ecurity **M**odule
IoT	**I**nternet **o**f **T**hings
JSON	**J**ava**S**cript **O**bject **N**otation
JoinEUI	**J**oin **E**xtended **U**nique **I**dentifier
JS	**J**oin **S**erver
LoRa	**L**ong **R**ange Device to Cloud platform from Semtech
LoRaWAN	**L**ong **R**ange **W**ide **A**rea **N**etwork
LPWAN	**L**ow **P**ower **W**ide **A**rea **N**etwork.
LTE-M	**L**ong **T**erm **E**volution Cat **M**1
MIC	**M**essage **I**ntegrity **C**ontrol
MQTT	**M**essage **Q**ueuing **T**elemetry **T**ransport
NB-IoT	**N**arrow**B**and **I**nternet of **T**hings
NS	**N**etwork **S**erver
NwkSKey	**N**etwork **S**ession **K**ey
OTAA	**O**ver **T**he **A**ir **A**ctivation
QoS	**Q**uality **o**f **S**ervice
RSSI	**R**eceived **S**ignal **S**trength **I**ndication.
SDR	**S**oftware **D**igital **R**adio
SE	**S**ecure **E**lement
SF	**S**preading **F**actor
SNR	**S**ignal **O**ver **N**oise **R**atio
TDM	**T**ime **D**ivision **M**ultiplexing
TOA	**T**ime **O**ne **A**ir
TTI	**T**he **T**hings **I**ndustrie
TTN	**T**he **T**hings **N**etwork
TTS	**T**he **T**hings **S**tack

Summary

1 Embedded system and IoT

The term IoT (**I**nternet **o**f **T**hings) is recent, but refers to an old usage called Machine to Machine. Machine to Machine (M2M) is a set of wired or wireless network technologies allowing the automatic exchange of information between systems without human intervention. IoT is simply a wider vision of M2M where the devices don't only come from the industrial world, but also from common public usage.

The IoT market increases very significantly around the world and this rapid evolution encourages new players to propose new technologies at each stage: hardware devices, connectivity coverage or cloud service (data storage of visualization platform).

IoT is often presented as the new industrial revolution, and marketing often promises that all use cases are "smart": smart building, smart city, smart healthcare… But making things smart is not always easy and many protocols exist. In this book, we will help you understand one of the main protocols in the IoT world: LoRaWAN.

1.1 The Internet of Things (IoT)

1.1.1 Embedded system in the IoT

Generally speaking, electronic systems can be characterized by their **power consumption**, **computing power**, **size** and **price**. In the specific case of embedded systems used in IoT, we can assign the following weight to each of the characteristics:

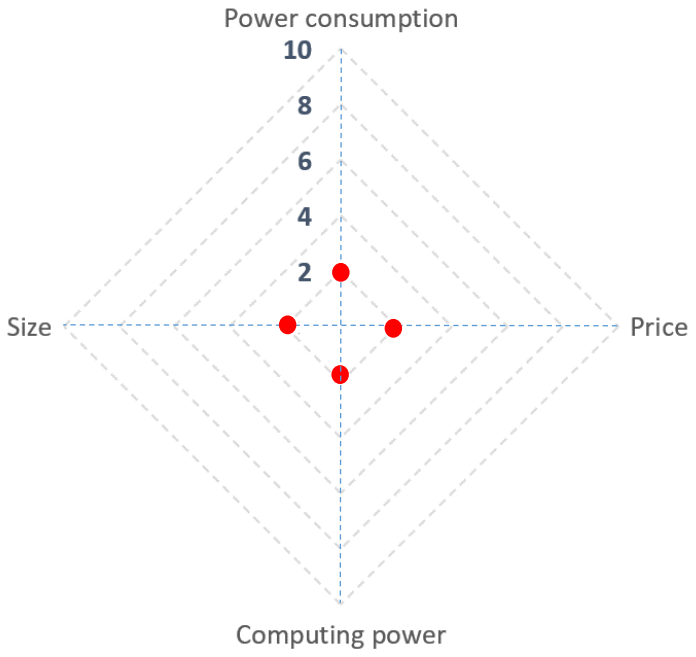


Figure 1: IoT device characteristics

Even if you can obviously find exception to this simple definition, we assume that, compared to other electronic systems, embedded systems used in IoT have:

- Low power consumption
- Low computing power
- A small size
- A low price

The second feature of an IoT device is its ability to communicate data over a wireless network. Many protocols exist and the designer will have a large choice depending on the range, the bandwidth and the low power consumption he wants to reach.

1.1.2 The IoT wireless protocols

In the IoT world, we can find many protocols such as Bluetooth, Zigbee, WiFi, 2G, 3G, 4G, 5G, NFC… We usually classify them according to their bandwidth and range as we can see in Figure 2. As a designer, we are always happy to reach greater range and bandwidth. If you have a quick glance at the figure, we may think that protocols on the top right area are much better than all the others (better range and better bandwidth). Protocol on the bottom left should therefore be the worst. What we are missing here, is that this graphic does not represent the power consumption induced by the protocol. Indeed, NFC is a really low power protocol: It even runs without any energy storage.

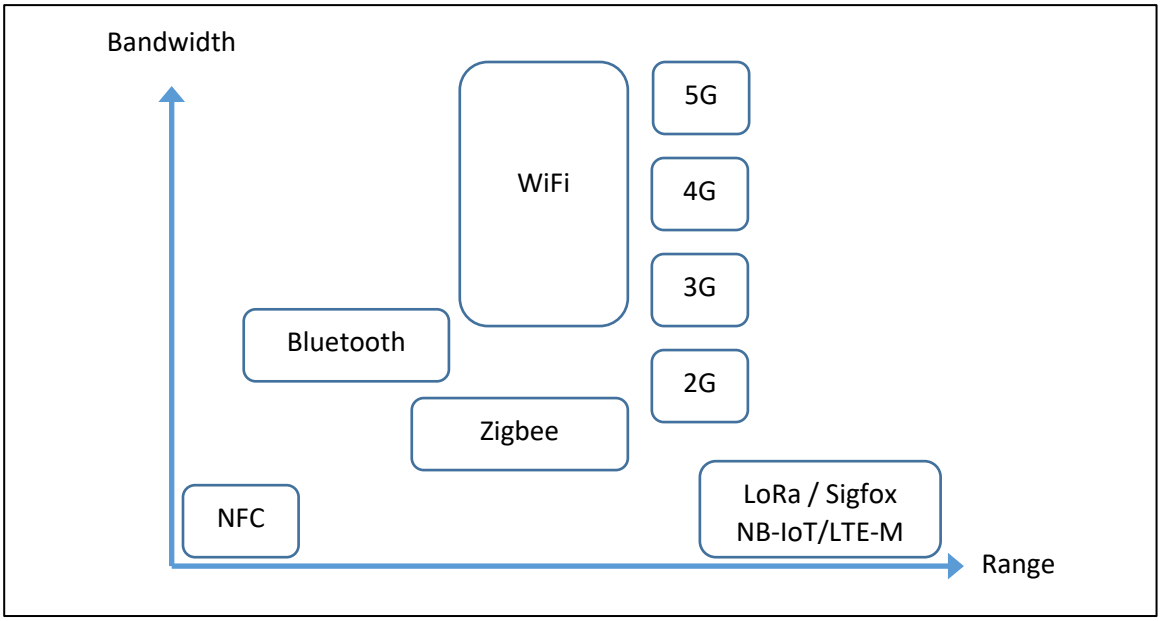


Figure 2: Protocols used in IoT

On the bottom right, we will find low bandwidth, high range, but overall low power protocols like NB-IoT or LTE-M. Sigfox and LoRaWAN are considered as extremely long range and extremely low power. All these networks are called LPWAN: **L**ow **P**ower **W**ide **A**rea **N**etwork.

In this course, we will only focus on **LoRaWAN** (**Lo**ng **Ra**nge **W**ide **A**rea **N**etwork) which is a long range, low data-rate and very low power protocol.

1.1.3 Frequency bands

In Europe, some frequency bands are free to use. This means:

- No need to ask for authorization
- Free of charge

Table1 shows some of these free bands in Europe.

Band	Examples of protocols
13.56 MHz	RFID, NFC
433 MHz	Walkie-talkie, remote control, LoRa
868 MHz	Sigfox, LoRa
2.4 GHz	WiFi, Bluetooth, Zigbee, LoRa
5 GHz	WiFi

Table 1: Free frequency bands

We notice that in Europe, LoRa can use the 433 MHz, 868 MHz or 2.4GHz band but only the 868 MHz band is used for LoRaWAN.

1.2 Media sharing modes

Regardless of the protocol used, the medium for transferring information is the air (all IoT protocols are wireless). The medium must be shared between all transmitters in such a way that each wireless end-device does not interfere one another. For this purpose, we allocate frequency bands for each use case. For example, the FM (**F**requency **M**odulation) radio frequency band in Europe goes from 87.5 MHz to 108 MHz.

Within their frequency band, the end-devices can share the medium in different ways.

1.2.1 FDM (Frequency Division Multiplexing)

Devices use frequency channels to separate their transmissions. **LoRa uses this sharing mode**, i.e. the free 868 MHz band is split into several channels that can be used to transmit information.

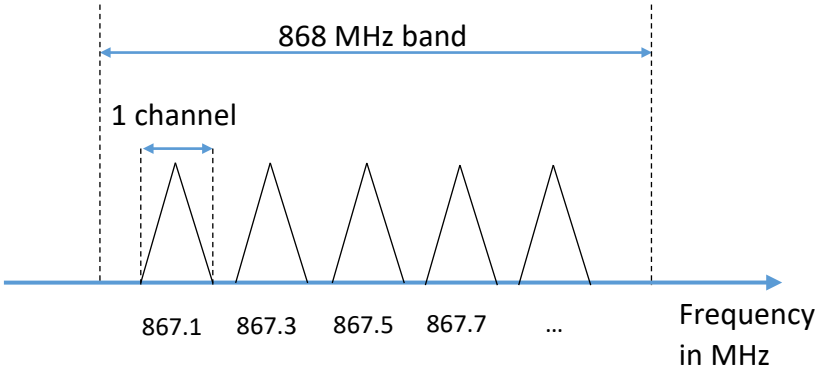


Figure 3: FDMA usage in LoRa

1.2.2 TDM (Time Division Multiplexing)

In this transmission mode, the end-devices transmit intermittently in order to leave the channel free for the others. **LoRa uses this sharing mode**, i.e. LoRa doesn't allow to transmit in continuous mode. However, the end-devices are not synchronized, so collisions can occur.

1.2.3 Spread Spectrum

In this transmission mode, end-devices transmit at **the same time**, **on the same channel**, but with a specific signal structure (with codes or symbols) which let the receiver retrieving the buried signal over the noise. **LoRa uses this sharing mode**.

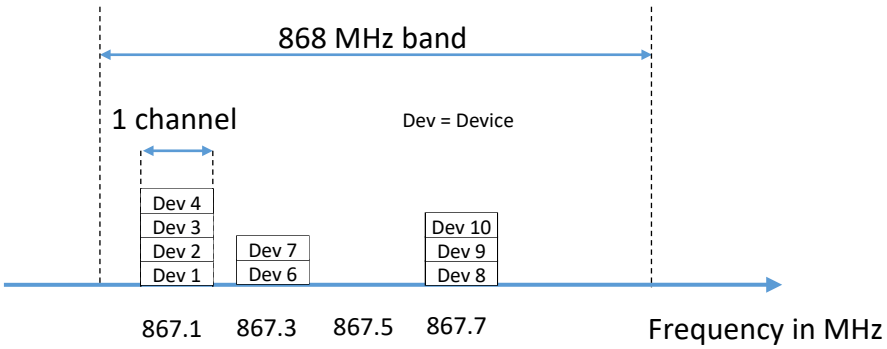


Figure 4: Using Spread Spectrum in LoRa

LoRa end-devices can choose between several channels to transmit. The beauty of LoRa is that even if many end-devices chose the same channel, they will be able to transmit at the same time.

Although, most famous Spread Spectrum modulation methods use "codes" to achieve this performance, LoRa uses symbols instead (called Chirp), thus its name: **C**hirp **S**pread **S**pectrum (**CSS**) modulation.

In order to understand the relevance of this media sharing method, we will validate the concept with code Spread Spectrum in the next section, then, in chapter 3, we will explain in detail the Chirp Spread Spectrum LoRa modulation.

1.3 Spreading spectrum with codes

We will see through an example how it is possible for three end-devices to transmit at the same time on the same channel. We will focus on the validity of the transaction.

The method consists of using codes that have mathematical properties adapted to our objective: transmitting at the same time on the same channel. The matrix below gives four spreading codes (1 per line).

Orthogonal Code User 0	1	1	1	1
Orthogonal Code User 1	1	-1	1	-1
Orthogonal Code User 2	1	1	-1	-1
Orthogonal Code User 3	1	-1	-1	1

Table 2: Hadamart (mathematician) matrix of order 4

The property of these codes (**1 1 1 1** ; **1 -1 1 -1** ; **1 1 -1 -1** ; **1 -1 -1 1**) is called "orthogonal". We don't explain this here but we must keep in mind that it only works if this orthogonal property is respected.

1.3.1 Validation of a single transmission over the air

Each table below represents a transmission, which is independent of the others: we don't care yet if they occur at the same time. We just check that each transmission reach its destination.

Each user has its own orthogonal code from the matrix. Two users cannot have the same code. We often call them "Spreading Code". The method is as follows:

<u>During transmission:</u>

- Each bit of the message is multiplied by its "Spreading Code": (1) x (2)
- The result of the multiplication is transmitted: (3)

<u>During reception:</u>

- Each symbol received (4) is multiplied again by the same "Spreading Code" (2).
- The received message (6) is equal to the sum of the decoded symbols (5), divided by the number of symbols (here we have four symbols).

Each table below represents one user.

1	**User 1 message (bits)**	**1**	**0**	**1**	**0**
2	User 1 "Spreading Code"	1 -1 1 -1	1 -1 1 -1	1 -1 1 -1	1 -1 1 -1
3	Transmitted symbols = (1) x (2)	1 -1 1 -1	0 0 0 0	1 -1 1 -1	0 0 0 0
... transmission ...					
4	Received symbols	1 -1 1 -1	0 0 0 0	1 -1 1 -1	0 0 0 0
5	Decoded symbols = (4) x (2)	1 1 1 1	0 0 0 0	1 1 1 1	0 0 0 0
6	**Message received =** ∑ (5) **/** nbr_Symb	**1**	**0**	**1**	**0**

Table 3: User 1 transmission

We can check that the message received (6) is the same as the user 1 initial message (1).

The last columns of the following tables (User 2 and User 3) are left blank so that you can try the calculation by yourself.

A step by step explanation of this method is explained in video on our website : www.univ-smb.fr/lorawan.

1'	**User 2 message (bits)**	**0**	**1**	**0**	**1**
2'	User 2 "Spreading Code"	1 1 -1 -1	1 1 -1 -1	1 1 -1 -1	1 1 -1 -1
3'	Transmitted symbols = (1') x (2')	0 0 0 0	1 1 -1 -1		
... transmission ...					
4'	Received symbols	0 0 0 0	1 1 -1 -1		
5'	Decoded symbols = (4') x (2')	0 0 0 0	1 1 1 1		
6'	**Message received =** ∑ (5') / nbr_Symb	**0**	**1**		

Table 4: User 2 transmission

1''	**User 3 message (bits)**	**1**	**1**	**0**	**0**
2''	User 3 "Spreading Code"	1 -1 -1 1	1 -1 -1 1	1 -1 -1 1	1 -1 -1 1
3''	Transmitted symbols = (1'') x (2'')	1 -1 -1 1	1 -1 -1 1		
... transmission ...					
4''	Received symbols	1 -1 -1 1	1 -1 -1 1		
5''	Decoded symbols = (4'') x (2'')	1 1 1 1	1 1 1 1		
6''	**Message received =** ∑ (5'') / nbr_Symb	**1**	**1**		

Table 5: User 3 transmission

1.3.2 Simultaneous transmissions

Transmissions now take place simultaneously: User 1, User 2 and User 3 messages are sent at the same time on the same channel. The first column of User 1 is already filled in as an example. The method is as follows:

On the receiver's antennas:

- The signal received is the addition of all symbols transmitted by all users (1, 2, 3) so we add the following lines:

3	Transmitted symbols user 1	1 -1 1 -1	0 0 0 0	1 -1 1 -1	0 0 0 0
3'	Transmitted symbols user 2	0 0 0 0	1 1 -1 -1		
3''	Transmitted symbols user 3	1 -1 -1 1	1 -1 -1 1		
3'''	**∑ of the transmitted symbols (3 + 3' + 3'')**	**2 -2 0 0**	**2 0 -2 0**		

Table 6: Simultaneous transmission of user 1, user 2 and user 3.

Then, on each receiver:

- Each signal received (3''') is multiplied by the user's "Spreading code" (2, 2' or 2'').
- The message (6, 6', 6'') is equal to the sum of the decoded symbols (7, 7', 7''), divided by the number of symbols (here we have four symbols). The message is equivalent to the one sent (1, 1', 1'').

3'''	**Received symbols (3 + 3' + 3'')**	**2 -2 0 0**	**2 0 -2 0**		

2	User 1 "Spreading Code"	1 -1 1 -1	1 -1 1 -1	1 -1 1 -1	1 -1 1 -1
7	Decoded symbols (3''') x (2)	2 2 0 0	2 0 -2 0		
6	**Message received User 1** = ∑ (7)/ nbr_Symb	**1**	**0**		

2'	User 2 "Spreading Code"	1 1 -1 -1	1 1 -1 -1	1 1 -1 -1	1 1 -1 -1
7'	Decoded symbols (3''') x (2')	2 -2 0 0	2 0 2 0		
6'	**Received message User 2** = ∑ (7')/ nbr_Symb	**0**	**1**		

2''	User 3 "Spreading Code"	1 -1 -1 1	1 -1 -1 1	1 -1 -1 1	1 -1 -1 1
7''	Decoded symbols (3''') x (2'')	2 2 0 0	2 0 2 0		
6''	**Message received User 3** = ∑ (7'')/ nbr_Symb	**1**	**1**		

Table 7: Reception of User 1, User 2 and User 3 messages

We can check that the messages received (6, 6', 6'') are the same than the one transmitted (1, 1', 1'').

1.3.3 The LoRa® modulation

LoRa uses a Spread Spectrum method that is different from the one studied above. However, the purpose is the same: being able to transmit **at the same time, on the same channel.** The SX1261 LoRa transceiver can use eight "spreading codes" called "Spreading Factor" [SF5, SF6, SF7, SF8, SF9, SF10, SF11 and SF12]. We can therefore have eight simultaneous transmissions on the same channel. The real SF (**S**preading **F**actor) parameters will be explained in details in chapter 3. In the LoRaWAN protocol, we use only six SF [SF7 to SF12].

2 Radio transmission and propagation

2.1 Units and definitions

2.1.1 The decibel (dB)

When a signal spread along its communication path, the ratio between the power received and the power transmitted can be very different. While the ratio is almost 1 if we use a cable, it can be very high for an amplifier or extremely low for air loss transmission (billionth of a billionth). Dealing with such big and small numbers is not suitable for quickly characterizing a transmission gain or attenuation. Furthermore, multiplying the gain of each transmission block is not convenient.

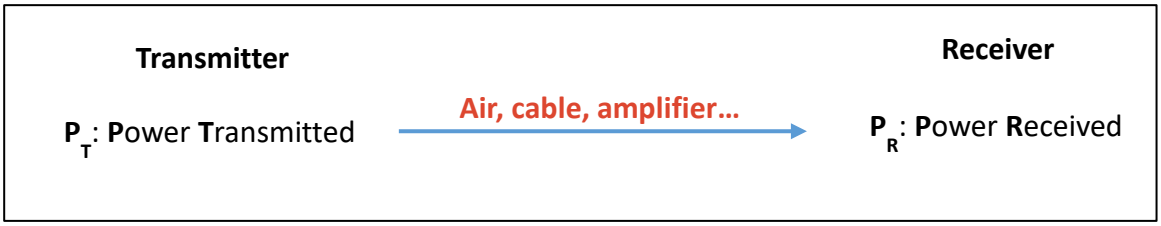


Figure 5 : The power transmission between a transmitter and receiver

dB is a ratio between two powers: the power on the receiver P_R and the power on the transmitter P_T. For the record, the formula for the ratio in dB is:

$$Power\ ratio\ (dB) = 10.log_{10}\left(\frac{P_R}{P_T}\right)$$

If the result is a negative number (-), this is an attenuation. If the result is a positive number (+), this is a gain (+).

If you want the power ratio while you have the power in dB then you can use the following formula:

$$\frac{P_R}{P_T} = 10^{\frac{Power\ ratio\ (dB)}{10}}$$

With these two formulas, you can easily check the Table 8 values.

Power ratio in dB	Power ratio
+ 10 dB	Multiplication by 10
+ 3 dB	Multiplication by about 2
0 dB	Equality
-3 dB	Division by about 2
- 10 dB	Division by 10

Table 8: Power ration calculation

The beauty of using dB is that not only we now deal with number reasonably large, but we also use only the + and – operation for the overall calculation.

Exercise:

In Figure 5, the cable that transmits the signal has a gain of -6dB. What is the power ratio between P_R and P_T?

Answer:

$$\frac{P_R}{P_T} = 10^{\frac{-6}{10}} = 0.25$$

- -6 dB is negative, this is an attenuation.
- 0.25 est less than 1, this is an attenuation.

2.1.2 Power in dBm

The **dBm** is the power in comparison to 1 mW: 0 dBm corresponds to 1 mW. Using the same ratios as in Table 8, we can fill in Table 9 concerning the dBm.

Power in dBm	Power in mW
10 dBm	10 mW
+ 3 dBm	2 mW
0 dBm	1 mW
- 3 dBm	0.5 mW
- 10 dBm	0.1 mW

Table 9: Comparison of power in dB and mW

Exercise: The walkie-talkie has a transmission power of 2 W. Using the Table 9, find the transmission power in dBm.

Answer:

1 mW x 10 x 10 x 10 x 2 = **2 W**

0 dBm + 10 + 10 + 10 + 3 = **33 dBm**

The walkie-talkie has a transmission power of 33 dBm.

For the record, the formula for the power in dBm and the power in Watt are:

$$Power\ (dBm) = 10.log_{10}\left(\frac{Power\ (Watt)}{0{,}001}\right)$$

$$Power\ (Watt) = 0.001\ x\ 10^{\frac{Power\ (dBm)}{10}}$$

2.1.3 RSSI, Sensitivity, SNR, Link Budget

A transmitter transmits a signal with a power P_T. The receiver recovers a fraction of this power (P_R), as well as some noise (P_N).

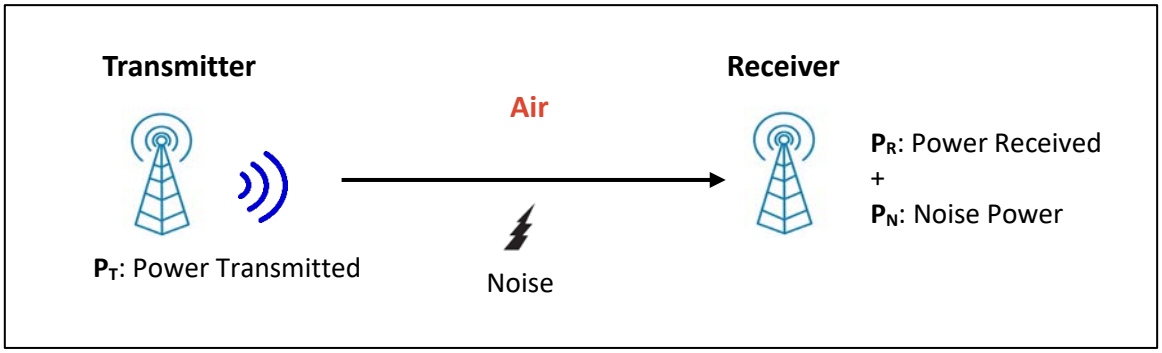


Figure 6: A radio frequency transmission

- The **R**eceived **S**ignal **S**trength **I**ndication (**RSSI)** is the power received: P_R.
- The **Sensitivity** is the minimum P_R power (or minimum RSSI) that must be present at the receiver in order to retrieve the signal. If the received RSSI is below the sensitivity, then the signal is undetectable.
- **SNR** (**S**ignal over **N**oise **R**atio) is the ratio of the received power (P_R) to the noise power (P_N).

All these values (RSSI, Sensitivity, SNR, ...) are given in decibel. A signal can be properly received if the two following conditions are met:

1. The RSSI is greater than the sensitivity of the receiver.
2. The SNR does not fall below a certain threshold that would make the signal impossible to detect on the receiver side.

Exercise: A transmitter provide 13 dBm using an antenna with a gain of 2 dB. The air loss is 60 dB. The receiving antenna has a gain of 2 dB and is connected to a receiver with a sensitivity of -80 dBm. Will the signal be received?

Answer:

13 + 2 – 60 + 2 = - 43	The received power is -43 dBm
- 43 dB is over – 80 dB (sensitivity)	Yes, the signal can be received

The logs below come from a LoRaWAN gateway. It gives an example of RSSI and SNR values measured during a data transmission. The values **"rssi": -13** and **"snr": 9.5** shows that in this example the received signal has high power and has a very good SNR. Indeed, the LoRaWAN device was only a few meters away from the gateway during this test.

```
"gateways":
    {
      "time": "2020-04-29T12:09:45.563621044Z",
      "channel": 0,
      "rssi": -13,
      "snr": 9.5
    }
```

What can we do if the received power (P_R) is below the sensitivity? The first idea would be to increase P_T. This is possible to a certain extent as the transmission power is limited by the specification. The maximum power P_T on the 868 MHz band is 14 dBm (25 mW). The second possibility is to improve the sensitivity of the receiver. That is obviously on what LoRa module designers are working at. In the end, it is the difference between the transmitted power P_T and the

sensitivity of the receiver that matters. This is called the **Link Budget**. In the previous exercice, the link budget available is 93 dB (13 + 80).

- In LoRa, we have a Link Budget of about 157 dB
- In LTE (4G) we have a Link Budget of about 130 dB.

Once you have spent the link budget along the transmission path, the signal is lost.

2.2 Transmission distance in LoRa

The transmitted power (P_T) is attenuated in the air according to the following simplified formula:

$$Loss = 10.log_{10}(Distance^2.Frequency^2.1755)$$

- Loss: in **dB**.
- Distance: in **km**.
- Frequency: in **MHz**.

We can therefore estimate the maximum distance:

$$Distance = \sqrt{\frac{10^{\frac{Loss}{10}})}{1755.Frequency^2}}$$

Since the link budget is the maximum losses that a transmission can withstand, we assume that the budget is spent in the air loss:

$$distance = \sqrt{\frac{10^{\frac{Link\ Budget}{10}})}{1755.frequency^2}}$$

- The LoRa SX1272 transceiver (Link Budget of 157 dB) gives a theoretical distance of 1946 km.
- The LoRa SX1262 transceiver (Link Budget of 170 dB) gives a theoretical distance of 8696 km.

In April 2020, the world record for a LoRa transmission was broken. They reached 832 km in the EU868 band using a power of 25 mW / 14 dBm (maximum power allowed).

2.3 Transceiver documentation

Figure 6 is an extract of SX1262 transceiver documentation.

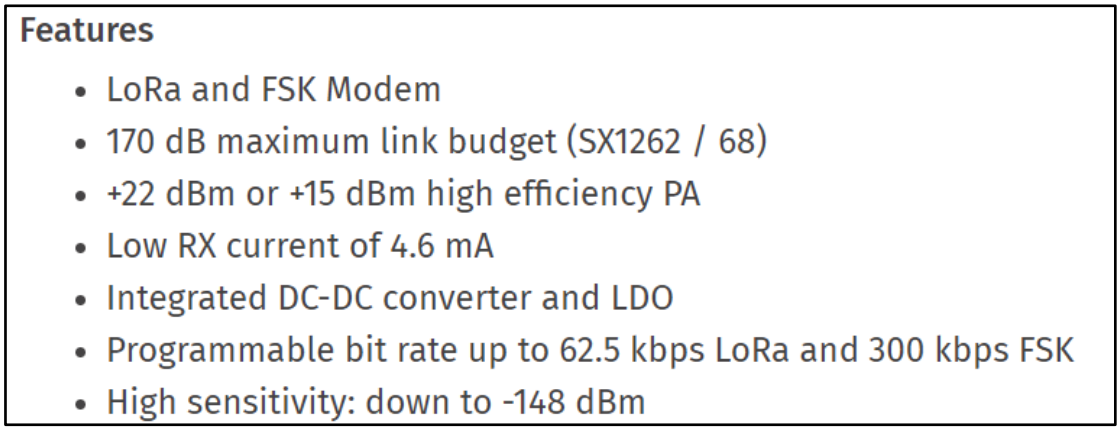


Figure 7: Main features of the SX1262 LoRa Transceiver

Using the **Link Budget** definition (P_T minus receiver sensitivity), we find the 170 dB stated in this documentation (22 dBm + 148 dBm). Nevertheless, we remind that the maximum transmit power in Europe is 14 dBm which reduces the Link Budget to 162 dB (14 dBm + 148 dBm).

In LoRa, the larger the Spreading Factor, the more we are able to transmit in a noisy environment. Table 10 shows the signal-to-noise ratios with which we will be able to carry out a transmission, depending on the Spreading Factor used.

SpreadingFactor (RegModemConfig2)	Spreading Factor (Chips / symbol)	LoRa Demodulator SNR
6	64	-5 dB
7	128	-7.5 dB
8	256	-10 dB
9	512	-12.5 dB
10	1024	-15 dB
11	2048	-17.5 dB
12	4096	-20 dB

Table 10: Influence of the Spreading Factor on the SNR

We notice that with a SF8 transmission, we are able to demodulate the signal with an SNR of -10 dB: we will be able to receive a signal with noise 10 times higher than the signal.

We notice that for a SF12 transmission, we are able to demodulate the signal with an SNR of -20 dB: we will be able to receive with noise 100 times higher than the signal.

However, we also notice that using a higher Spreading Factor, the number of transmitted "chips" increases (2nd column of the table). We will see later what it means exactly, but we can already say that it will obviously affect the transmission time, thus the bit rate.

3 LoRa® modulation (physical layer)

3.1 LoRa® modulation

As we explained earlier, LoRa modulation uses Spread Spectrum to transmit its data. But Instead of using "codes", it will use "Chirps" and that is why it is called Chirp Spread Spectrum modulation. The purpose is always the same: to have several transmissions at the same time in the same channel. The consequence on the spectrum is also the same: it causes a spread of the spectrum on the selected bandwidth.

3.1.1 The Chirp

The signal emitted by the LoRa modulation is a symbol with a basic form represented Figure 8. Its name (Chirp) comes from the fact that this symbol is used in Radar technology (**Chirp**: **C**ompressed **H**igh **I**ntensity **R**adar **P**ulse).

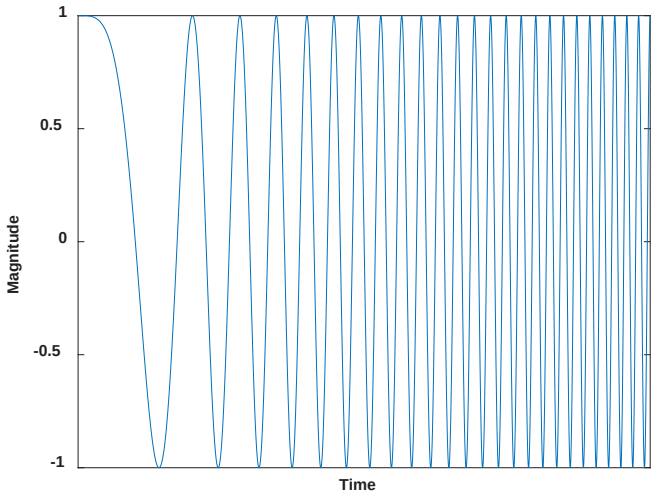


Figure 8: A Chirp (Matlab simulation)

The start frequency is the channel frequency **minus** the bandwidth divided by two. The final frequency is the channel frequency **plus** the bandwidth divided by two. Figure 9 represents the LoRa modulation in the frequency domain.

- The channel ($F_{channel}$) is the center frequency.
- The band occupied around $F_{channel}$ is the bandwidth.

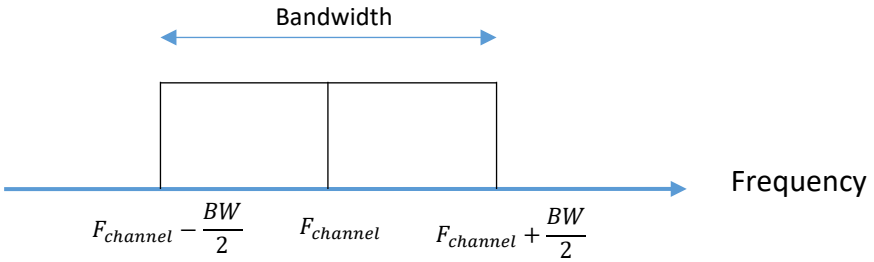


Figure 9: Spectrum of a LoRa transmission

Exercise: A LoRa transmission uses the channel 868,1 MHz with a bandwidth of 125 kHz. Can you give the start and end frequency of the Chirp?

Answer:

- Start frequency: 868 037 500 Hz = 868 100 000 – 125 000/2
- End frequency: 868 162 500 Hz = 868 100 000 + 125 000/2

In order to make the representation easier to understand, we use a Time-Frequency graph (Spectrogram).

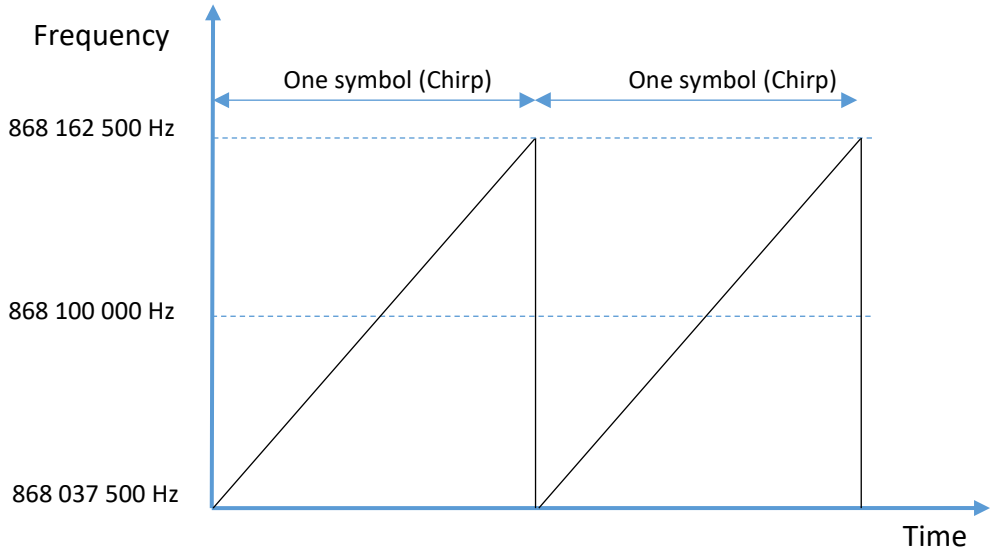


Figure 10: Spectrogram of a LoRa transmission

In LoRa, each symbol represents a number of bit transmitted. The rule is as follows:

Number of bits transmitted in a symbol = Spreading Factor

For example, if the transmission uses SF10, then one symbol (Chirp) represents 10 bits.

During emission, the bits are grouped together in packets of **SF** bits. Then each packet is represented by a particular symbol among 2^{SF} possible forms. Between symbols, the only difference is that they all starts from a specific frequency which represent the packet of bits.

Figure 11 shows a theoretical example of SF2 modulation at 868,1 MHz, with a bandwidth of 125 kHz. Each symbol represents 2 bits.

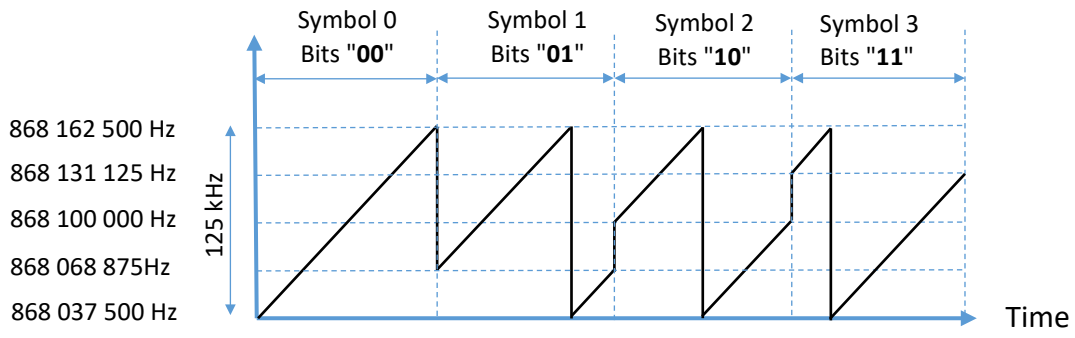


Figure 11: Symbols transmitted in LoRa modulation in SF2 (theoretical case)

Example:

- We consider the following binary sequence: 0 1 0 1 1 1 0 0 0 1 1 0 0 1 0 0 1 1 0 1
- We use SF10

We group the bits in packets of 10. Each packet of 10 bits is represented by a particular symbol. There are 1024 different symbols to encode the 1024 possible binary combinations (2^{10}).

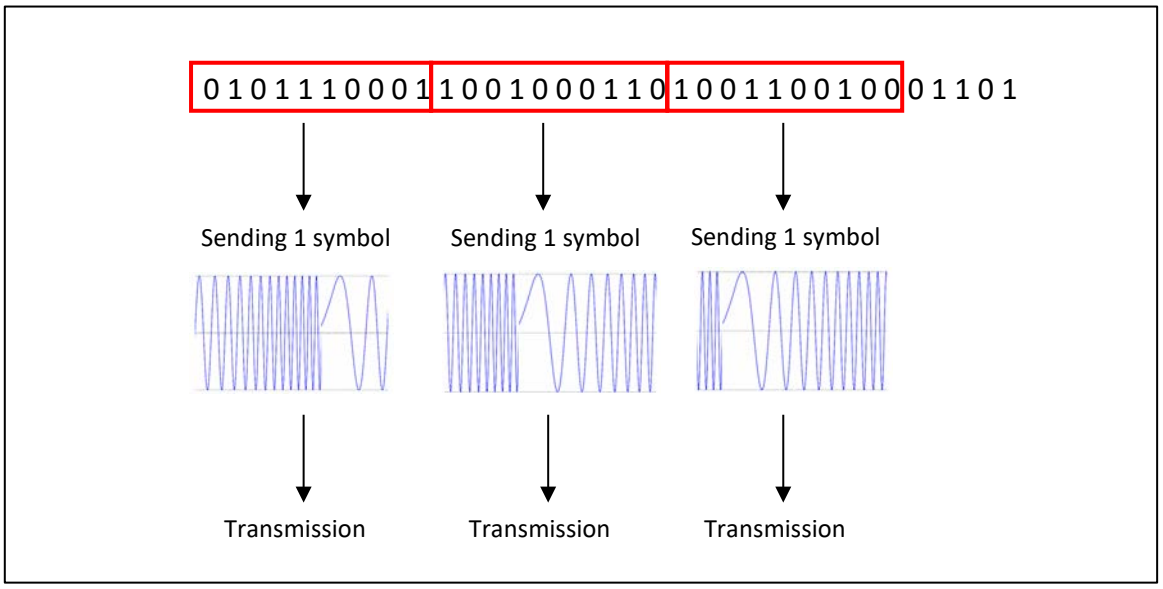


Figure 12: LoRa Chirp transmission

Figure 13 is the spectrogram of a LoRa transmission realised thanks to an ADALM Pluto receiver.

Figure 13: Spectrogram of a LoRa transmission

3.1.2 Symbol transmission time

In LoRa, the transmission time of each symbol (T_{symbol}) depends on the **Spreading Factor** used. The higher the SF, the longer the transmission time. For the same bandwidth, the transmission time of a symbol in SF8 is twice as long as the transmission time of a symbol in SF7. This is the case up to SF12.

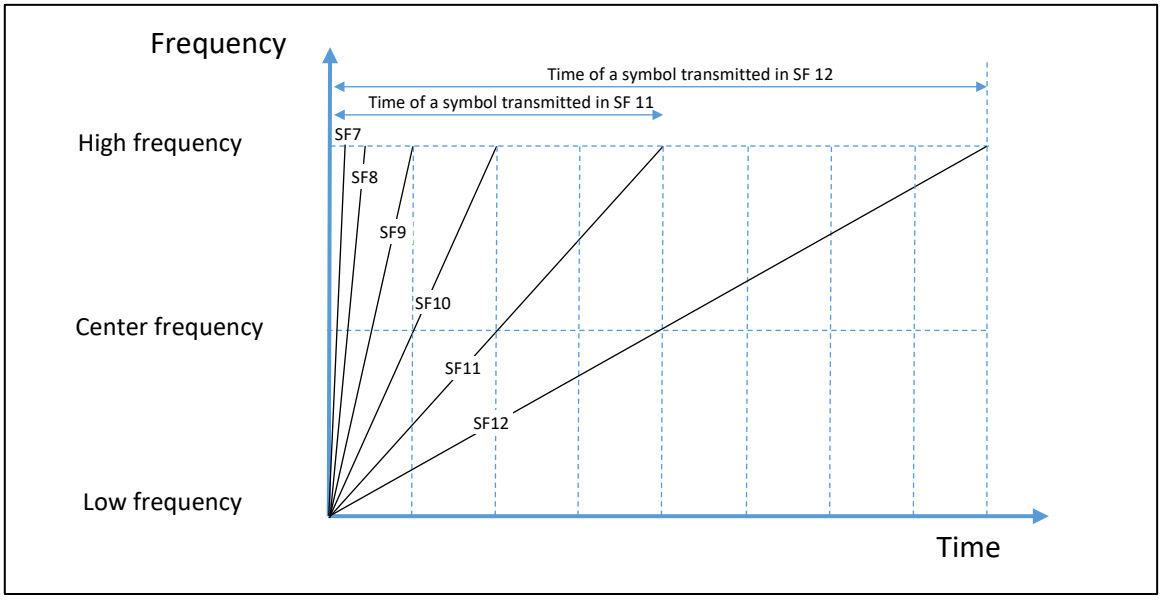


Figure 14: Symbol transmission time

The transmission time of each symbol (T_{symbol}) also depends on the **bandwidth** used. T_{symbol} is inversely proportional to the bandwidth. If we now that into account the **SF** and the **bandwidth**, we have the following expression:

$$T_{symbol} = \frac{2^{SF}}{Bandwidth}$$

As an example, Table 11 shows the transmission time depending on SF, for a bandwidth of 125 KHz.

Spreading Factor	Symbol transmission time
SF7	1.024 ms
SF8	2.048 ms
SF9	4.096 ms
SF10	8.192 ms
SF11	16.384 ms
SF12	32.768 ms

Table 11: Symbol transmission time for BW125

The symbol rate is $\frac{1}{T_{symbol}} = F_{symbol} = \frac{Bandwidth}{2^{SF}}$. Obviously, the higher the bandwidth, the higher the symbol rate.

3.1.3 Time on Air

The Time on Air is the overall duration of a LoRa frame. It depends on the number of symbols presents in the LoRa frame: the payload, the preamble, header and CRC.

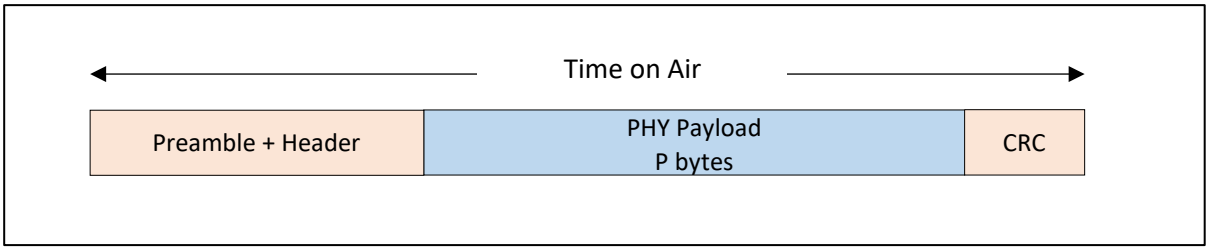


Figure 15: Time on Air of a LoRa frame

$Time\ on\ Air = n_{symbol}.T_{symbol}$, where n_{symbol} is the number of symbols presents in the LoRa frame.

We will see later in course a way to calculate n_{symbol} in order to find the Time on Air of a LoRa transmission.

3.2 LoRa® and LoRaWAN® bit rate

3.2.1 LoRa bit rate

Since each symbol consists of SF bits, the bit rate is:

$$Bit\ Rate = SF.\frac{Bandwidth}{2^{SF}}$$

- **The higher the Spreading Factor, the lower the bit rate.**
- **The higher the bandwidth, the higher the bit rate.**

Exercise: Consider the following two cases: case 1 (SF7, 125 kHz) and case 2 (SF12, 125 kHz). Give the corresponding bit rate.

Answer:

- **Case 1:** For SF7, 125 kHz > Bit Rate = **6,836 bps**
- **Case 2:** For SF12, 125 kHz > Bit Rate = **366 bps**

3.2.2 Influence of the Coding Rate on the bitrate

The Coding Rate is a ratio that increases the number of bits transmitted in order to carry out error detection and correction. In the case of a CR = 4 / 8, there is 8 bits truly transmitted each time we want to transmit 4 bits. In this example, the overhead ratio is 2, which means that there is twice as many transmitted bits.

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

Table 12: Overhead Ratio and Coding Rate

Exercise: Using the previous cases (SF7, 125 kHz) and (SF12, 125 kHz) with a CR of 4/5. Give the corresponding bit rate.

Answer:

- **Case 1:** For SF7, 125 kHz and CR4/5 > Bit Rate = 6.836 kbps / 1.25 = **5469 bps**
- **Case 2:** For SF12, 125 kHz and CR4/5 > Bit Rate = 366 bps / 1.25 = **293 bps**

The documentation of a LoRa transceiver gives the data rates according to the Spreading Factor, the Bandwidth and the Coding Rate. We can check the consistency of the result with your previous calculation: the case 2 has a bit rate of 293 bps.

Bandwidth (kHz)	Spreading Factor	Coding rate	Nominal Rb (bps)	Sensitivity (dBm)
125	12	4/5	293	-136

Table 13: Bit Rate and LoRa transmission parameters

3.2.3 Bit rate simulation with LoRa Modem Calculator

The LoRa Modem Calculator is a small software provided by Semtech to simulate a LoRa transmission according to the configurable parameters: channel, SF, CR, etc...

- The LoRa Modem calculator for SX1272 transceiver is available here.
- The LoRa Modem calculator for SX1261 / SX1262 transceiver is available here.
- An equivalent online LoRa simulator is also available here.

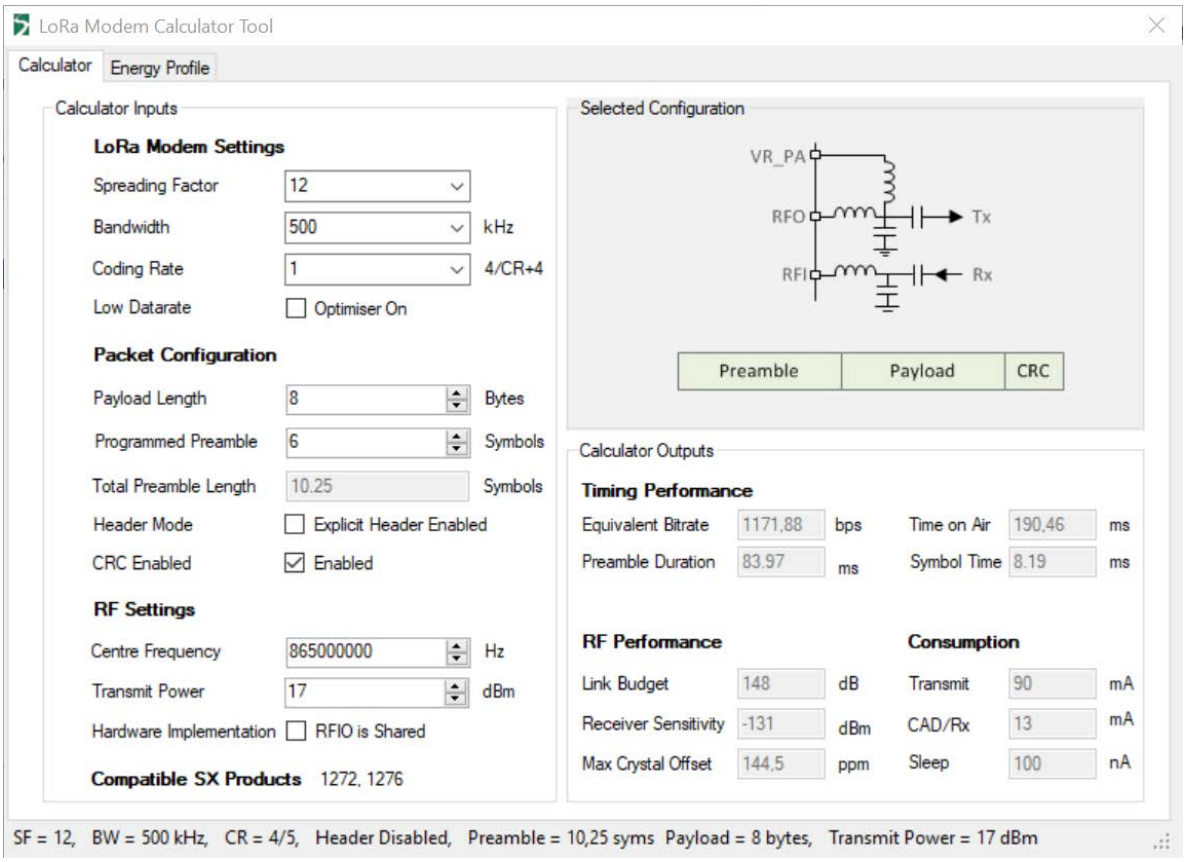


Figure 16: The SX1272 LoRa Calculator software

Exercise: Using the example of the two previous cases [SF7, 125 kHz, CR 4/5] and [SF12, 125 kHz, CR 4/5], check the "Equivalent Bitrate" calculations with the LoRa Calculator software.

Answer:

- **Case 1:** For SF7, 125 kHz and CR4/5> Bit Rate = **5468.75 bps**
- **Case 2:** For SF12, 125 kHz and CR4/5 > Bit Rate = **292.97 bps**

3.2.4 Influence of the LoRa overhead on the bitrate

So far, we only focused on the instantaneous bit rate. In reality, the payload is transmitted in the same frame than :

- A preamble allowing the receiver to synchronize the receiver.
- Optional Headers after the preamble.
- CRC field (frame integrity check) at the end.

The LoRa data are called **PHY Payload.**

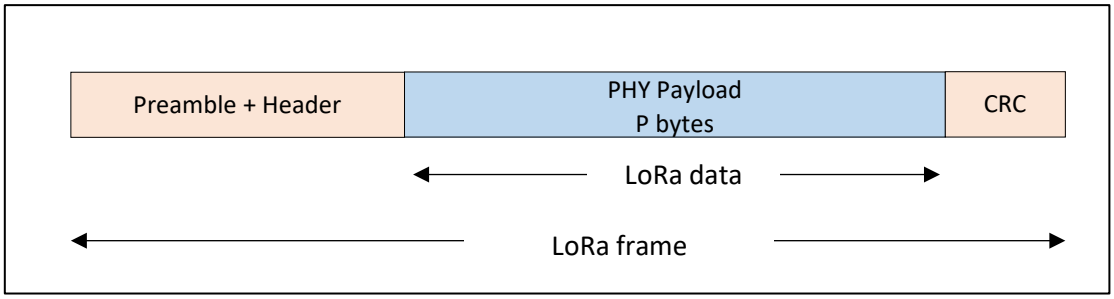


Figure 17: LoRa frame

We are now interested in the PHY Payload bit Rate: how many payload bit per second a user can send/receive? We need to calculate the transmission time of the entire LoRa frame (Time on Air) and for that, we use the "LoRa Modem Calculator" (see chapter 3.2.3). Figure 18 simulates the Time on Air for a SF7, BW125, CR4/5 transmission with 1 byte as a PHY Payload.

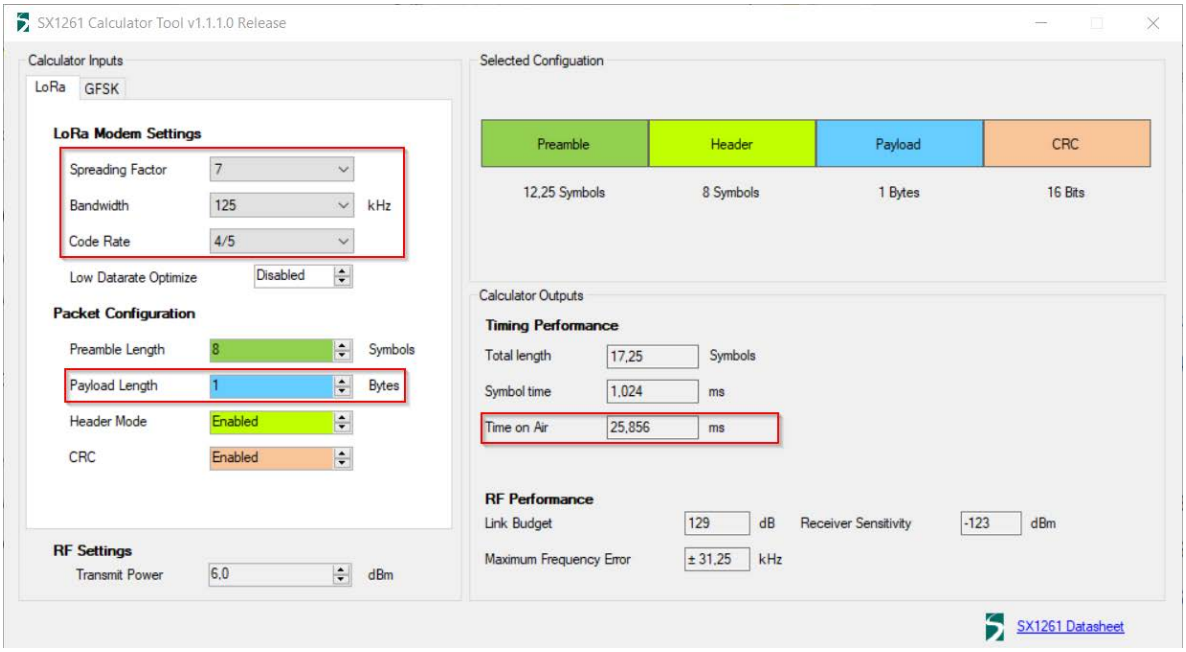


Figure 18: Simulation of Time on Air

Exercise: Check the Time on Air of the two previous cases [SF7, 125 kHz, CR 4/5] and [SF12, 125 kHz, CR 4/5] with 1 byte. Deduce the real bit rate of the PHY Payload in both cases.

Answer:

- Sending one byte (PHY Payload) in SF7 gives a Time on Air of **25.85 ms**
- Sending one byte (PHY Payload) in SF12 gives a Time on Air of **827.39 ms**

- **Case 1:** For SF7, 125 kHz and CR4/5 > Bit $Rate_{LoRa\ Payload}$ = 8 / 25.85 ms = **309.3 bps**
- **Case 2:** For SF12, 125 kHz and CR4/5 > Bit $Rate_{LoRa\ Payload}$ = 8 / 827.39 ms = **9.6 bps**

3.2.5 Influence of the LoRaWAN overhead on the bitrate

The LoRaWAN protocol needs to provide additional information. We will see in paragraph 4.1 the differences between LoRa and LoRaWAN and we will also see in paragraph 6.1.1 the details of each field of the LoRaWAN frame. However, for the time being, we can simply state that LoRaWAN provides an additional service to the LoRa protocol.

Figure 19 represents a simplified LoRaWAN frame. A LoRaWAN header has been added. This LoRaWAN header takes time to be transmitted and therefore increases the overall Time on Air. On the other hand, the user data is still 1 byte.

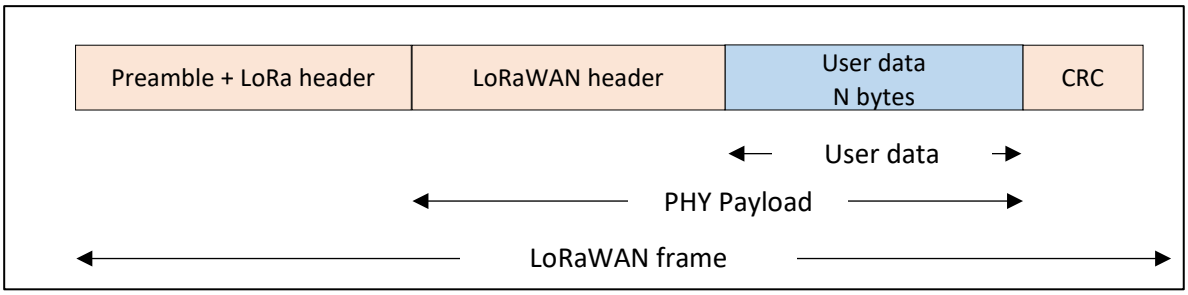


Figure 19: LoRaWAN frame

At the user level, only the amount of useful data transmitted matters. For example, if the user wants to transmit a temperature (1 byte), his only concern is to know how many temperature data he will be able to transmit per seconds / minutes / hours.

We can simulate this real bit rate with the LoRa calculator with the following configuration:

- Spreading Factor: SF7
- Bandwidth: 125 kHz
- Coding Rate: 4/5
- Payload length: The LoRaWAN Header (usually 13 bytes) + the user data (1 byte in our example). The PHY Payload is therefore 14 bytes.

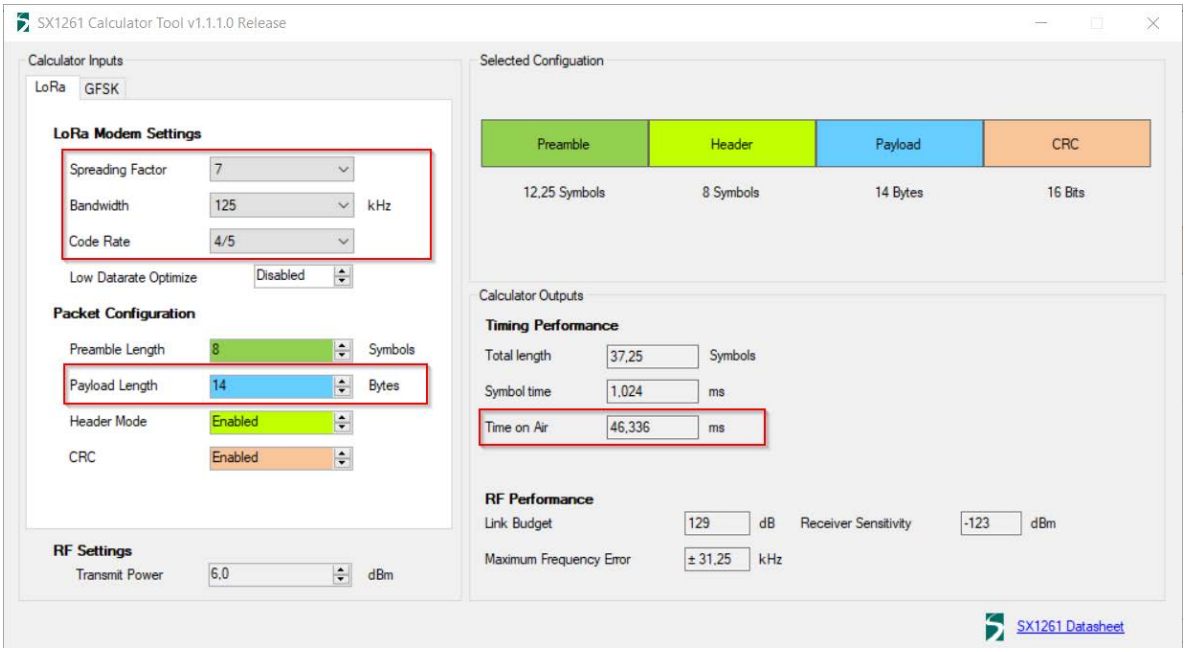


Figure 20: LoRaWAN Time on Air

The LoRa Calculator gives a Time on Air of **46,3 ms** for SF7, 125 kHz and CR4/5.

We can also verify it through a real LoRaWAN transmission from an end-device to a gateway. In this application, we transmit a single byte (a temperature). The gateway saves the transmission information of every LoRa packet received. We collect the following values on the gateway for a transmission in SF7, then for SF12. We are interested in the airtime column (ms).

time	frequency	mod.	CR	data rate	airtime (ms)	cnt		
▲ 09:54:22	868.5	lora	4/5	SF 7 BW 125	46.3	3783	dev addr: 26 01 16 8E	payload size: 14 bytes

time	frequency	mod.	CR	data rate	airtime (ms)	cnt		
▲ 11:07:48	868.3	lora	4/5	SF 12 BW 125	1155.1	1	dev addr: 26 01 16 8E	payload size: 14 bytes

Figure 21: Time on Air for one byte transmitted in SF7 and SF12

We note that:

- Sending one byte (user data) in LoRaWAN with SF7 gives a Time on Air of **46,3 ms**.
- Sending one byte (user data) in LoRaWAN with SF12 gives a Time on Air of **1155.1 ms**.
- The indicated payload (14 bytes) is much higher than 1 byte, which shows that an additional header (LoRaWAN header) has been added (13 additional bytes + 1 payload).

Exercise: Calculate the real user data rate of this transmission for the two cases [SF7, 125 kHz, CR 4/5] and [SF12, 125 kHz, CR 4/5].

Answer:

- **Case 1:** For SF7, 125 kHz and CR4/5 > Date Rate$_{\text{LoRaWAN Payload}}$ = 8 / 46.3 ms = **172.7 bps**
- **Case 2:** For SF12, 125 kHz and CR4/5 > Date Rate$_{\text{LoRaWAN Payload}}$ = 8/ 1155.1 ms = **6.9 bps**

3.2.6 Influence of the Duty-cycle in the EU868 band

The European standard requires that a radiofrequency end-device transmit no more than 1% of the time in the 868 MHz band. This is called the Duty Cycle. For example, a Duty Cycle of 1% means that if an end-device transmit during 1 (no unit), it must stay quiet for 99, regardless of the time unit used.

Example: On the Figure 21, when using SF7, the Time on Air is 46.3 ms. The LoRa device should therefore not transmit for 99 x 46.3ms = 4.58 seconds.

Exercise: Using the previous examples [SF7, 125 kHz, CR 4/5] and [SF12, 125 kHz, CR 4/5], what is the average bit rate if we take into account the Time on Air and the 1% Duty-Cycle of the LoRaWAN standard?

Answer:

- **Case 1:** For SF7, 125 kHz and CR4/5 > Data Rate$_{\text{LoRaWAN Payload 1\%}}$ = 172.7 bps / 100 = **1.73 bps**
- **Case 2:** For SF12, 125 kHz and CR4/5 > Data Rate$_{\text{LoRaWAN Payload 1\%}}$ = 6.9 bps /100 = **0.07 bps**

If we read carefully the specification, things are a little bit more complicated. The idea is the same but in reality, the 1% Duty Cycle applies on each following band:

- 863.0 – 868.0 MHz : 1%
- 868.0 – 868.6 MHz : 1%

This means that if a LoRa end-device sends a LoRa frame on the 867.1 MHz channel (part of the 863.0 - 868.0 MHz) this device is still allowed to send another frame on the 868.1 MHz channel (part of the 868.0 – 868.6 MHz).

3.2.7 Influence of the LoRaWAN server use policy

The 1% Duty-Cycle is a specific parameter which applies on the 868 MHz European band (EU868). Apart from that, the LoRaWAN server can limit the number of messages you are allowed to send.

For example, on The Things Network community edition, a limit prevents an end-device to overload the Network.

> *"The uplink airtime is limited to 30 seconds per day (24 hours) per node and the downlink messages to 10 messages per day (24 hours) per node. If you use a private network, these limits do not apply."*

3.3 Simulation of a LoRa Transmission

3.3.1 Time on Air calculation

The Time on Air depends on the number of symbols sent in a LoRa frame and the time of one symbol.

$Time\ on\ Air = n_{symbol}.T_{symbol}$ where

- n_{symbol} is the number of symbols presents in the LoRa frame.
- $T_{symbol} = \frac{2^{SF}}{Bandwidth}$ is the time of one symbol.

n_{symbol} depends on many LoRa parameters and can be summarised with the following formulae:

$$n_{symbol} = (n_{preamble} + 4{,}25) + 8 + max\left(ceil\left(\frac{8.Payload - 4.SF + 28 + 16 - 20.H}{4(SF - 2.DE)}\right)(CR + 4),0\right)$$

Where:

- Payload is the LoRa PHY payload
- SF is the Spreading Factor
- H=0 when the Header is enable and H=1 otherwise
- DE = 1 when the low data rate optimization is enable, 0 otherwise
- CR is the coding rate from 1 to 4

Exercise: Check the Time on Air value found in Figure 20 (46,3 ms)

Answer: $n_{preamble} = 8$, Payload = 14, SF = 7, H = 0, DE = 0, CR = 1, so

$$n_{symbol} = (8 + 4{,}25) + 8 + ceil\left(\frac{8 * 14 - 4 * 7 + 28 + 16}{4 * 7}\right)(1 + 4) = 45.25\ symbols$$

$$T_{symbol} = \frac{2^{SF}}{Bandwidth} = 1.024ms$$

$Time\ on\ Air = 45.25 * 1.024 = 46.3$ ms

3.3.2 Chirp modulation (Matlab)

The LoRa modulation generates chirp named symbol(t) mixed with a local oscillator signal named channel(t) in order to shift the chirp around the channel frequency ($f_{channel}$) and therefore generate lora(t).

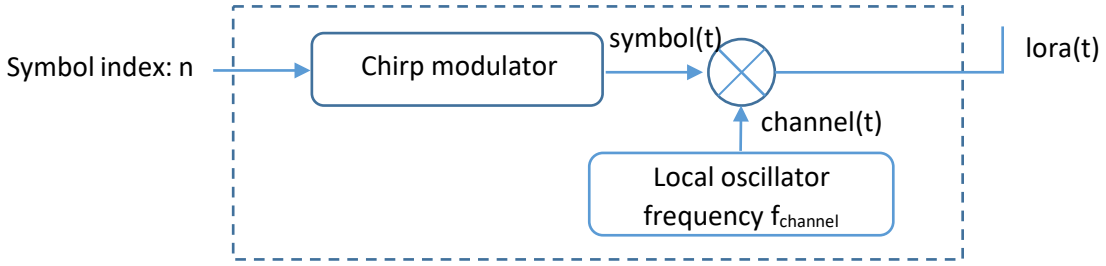


Figure 22: LoRa modulation bloc diagram

Such a modulator can be simulated using Matlab with SF12 and BW125. There is 2^{12} = 4096 different symbols ($n \in [0; 4096]$). Each symbols represents 12 bits.

$$T_{symbol} = \frac{2^{SF}}{Bandwidth} = 32{,}768\ ms$$

Figure 23 represents the spectrogram of symbol(t) for four different index: n=0, 1024, 2048 and 3072.

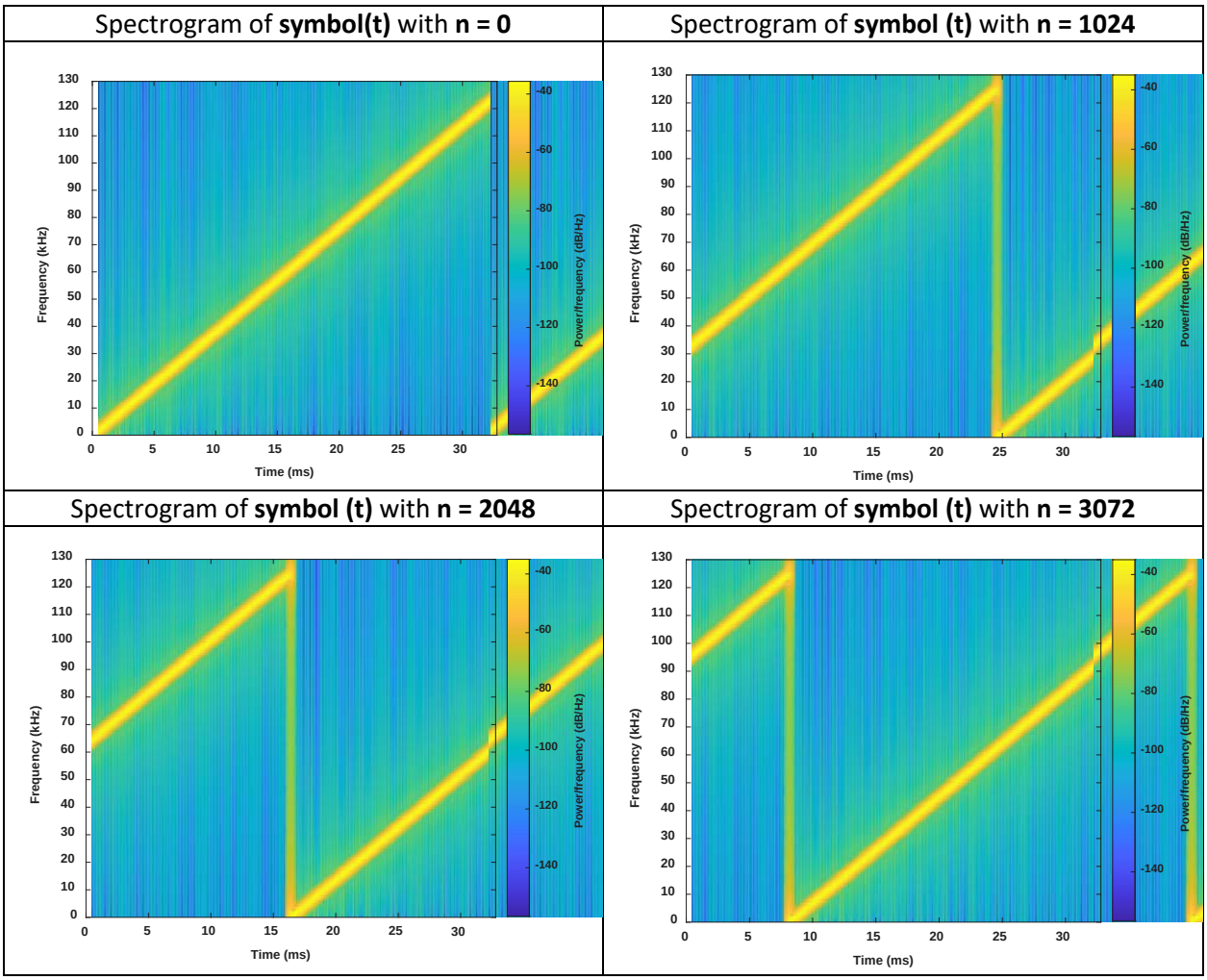


Figure 23: Symbol spectrogram simulation

3.3.3 Chirp demodulation (Matlab)

The first stage of the demodulation process shifts lora(t) in a base-band signal in order to find the symbol transmitted. Than the result is mixed with down(t) which is a down chirp signal. The result demod(t) is filtered and we use FFT to recover the symbol index n, thus the 12 bits .

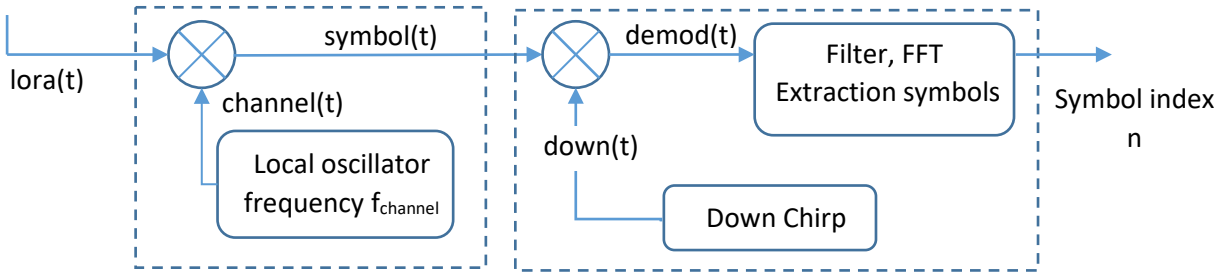


Figure 24: LoRa demodulation bloc diagram

Such a demodulator can be simulated using Matlab with a SF12 and BW125. Here, the first step is to find demod(t) which frequency is an image of the symbol index n.

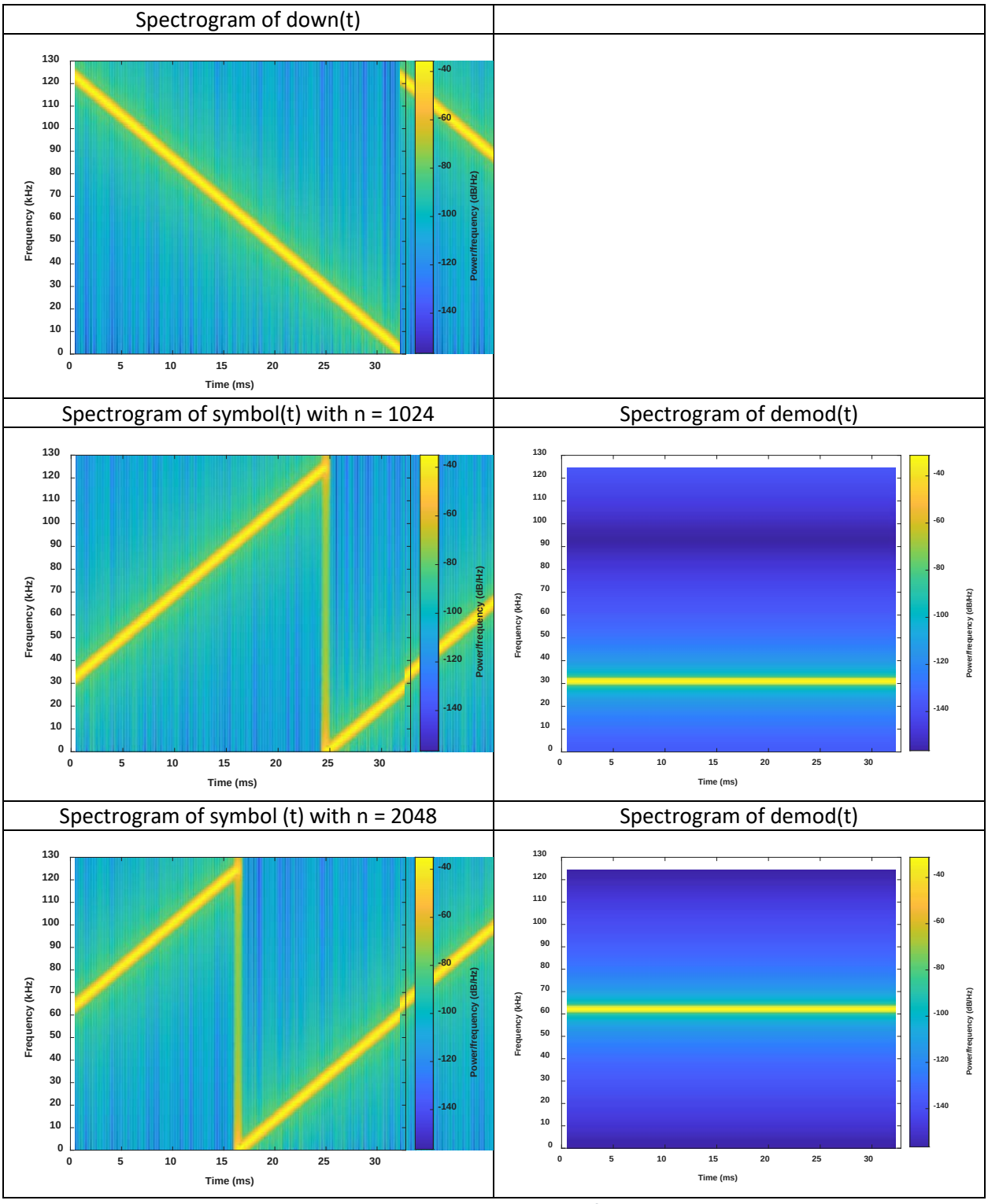


Figure 25: LoRa demodulation spectrogram for n =1024, 2048

Now, we can imagine a real signal received on the demodulator: symbol(t) is composed of 12 symbols with n taking the following random values: 153, 4012, 2122, 251, 947, 3050, 21, 1555, 2954, 84, 454, 812.

Spectrogram of symbol(t) with 12 n values	Spectrogram of demod(t)

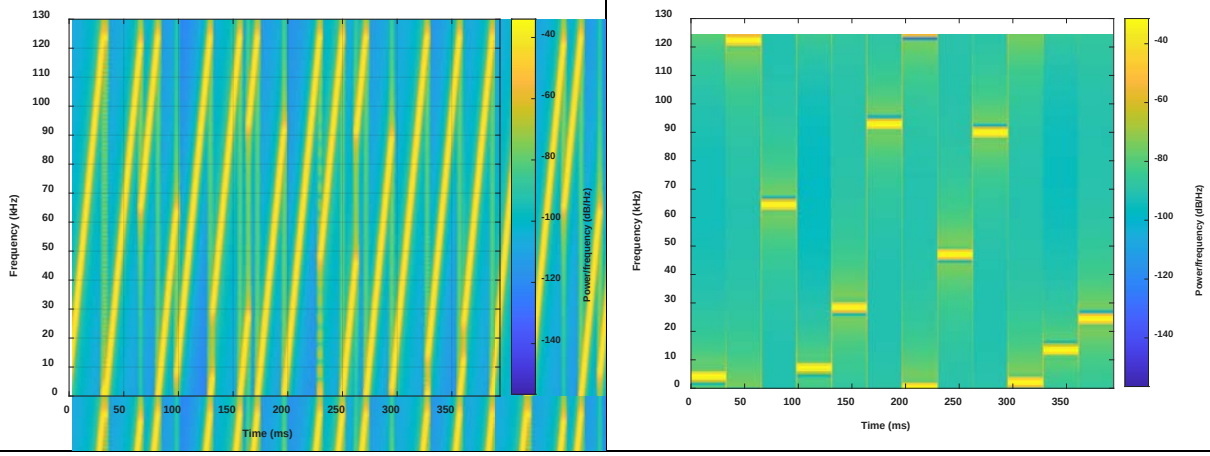


Figure 26: LoRa demodulation spectrogram for 12 n random values

The index can easily be extracted from demod(t) with an inverse FFT algorithm.

3.4 Real test of a LoRa Transmission

For this demonstration, we use a LoRa end-device transmitting a simple Payload with the following parameters:

- Channel: 868,1 Mhz
- Spreading Factor: 12
- Bandwidth : 125 kHz
- Coding Rate: 1 (4/5)
- Number of preamble: 8
- LoRa PHY Payload: "HELLO"

The SDR (**S**oftware **D**efined **R**adio) ADALM-PLUTO is the LoRa receiver. We use SDR Angel [https://github.com/f4exb/sdrangel] to pilot the ADALM PLUTO and display the LoRa modulation.

Figure 27: Analog Device ADALM PLUTO

We can see in the following spectrogram, the 8 up-chirp (preamble) plus 4.25 synchronisation symbols (2 up-chirp and 2.25 down-chirp) indicating the start of the frame.

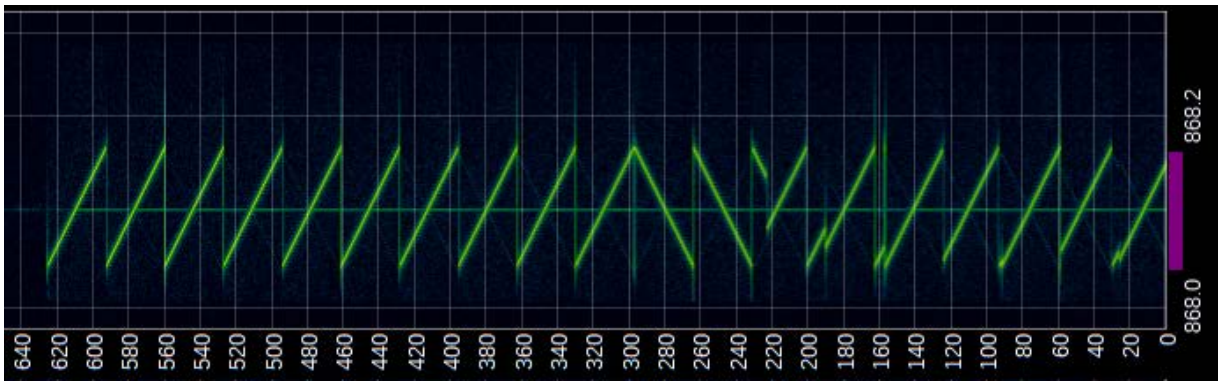


Figure 28: Spectrogram of a LoRa frame

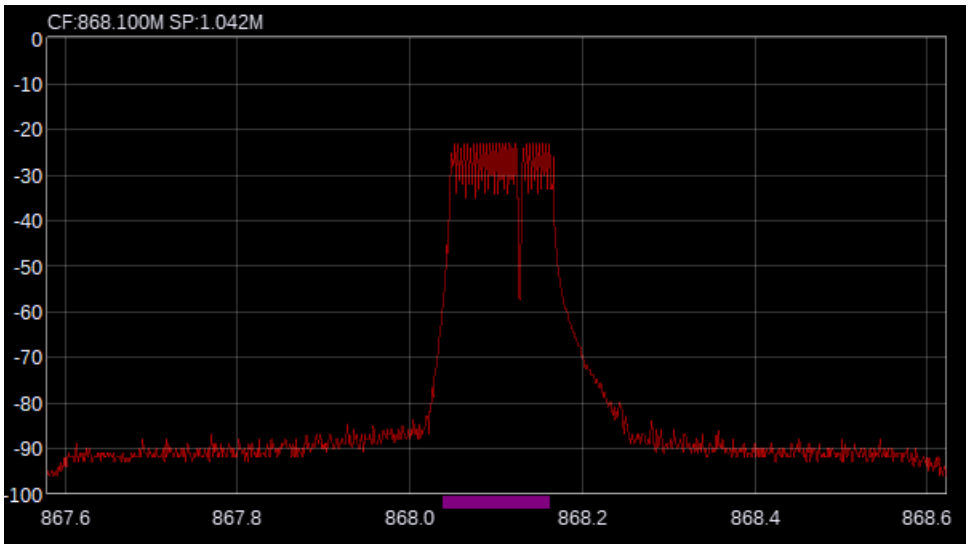


Figure 29: Spectrum around 868.1Mhz band

3.5 Energy consumption

The LoRaWAN protocol targets very low power application. It is common to find LoRaWAN device with several years of battery life. The real consumption of a LoRa system depends on several parameters:

- The amount of data to transmit (Payload).
- The Spreading Factor.
- The possible collisions at the emission (and thus retransmission).
- The request for acknowledgement of the transmitted frames.
- The Duty-Cycle.
- The transmission power of the transceiver.
- The power consumed in standby between two transmissions.

This online simulator gives a first approximation of the consumption and therefore the autonomy of a LoRaWAN end-device.

4 The LoRaWAN® protocol

4.1 LoRa® – LoRaWAN® – LoRa Alliance®

4.1.1 The LoRa Alliance®

The LoRa Alliance is a non-profit organization born in 2015 that aims to develop the technology and the entire LoRaWAN ecosystem. Its members are highly involved companies that invest in the LoRa Alliance to make it grow. Any structure can apply to take part of the LoRa Alliance and participate in the LoRaWAN development.

Savoie Mont Blanc University is part of the LoRa Alliance since 2021.

4.1.2 Protocol versions

One of the main roles of the LoRa Alliance is the specification and evolution of the LoRaWAN protocol. Here are the different versions and their evolution over time.

- **Version 1.0.0** (January 2015): Initial version of the LoRaWAN specification.
- **Version 1.0.1** (February 2016)**:** Addition of new frequency plan for Chinese and Australian. Correction and clarification on many minor points.
- **Version 1.0.2** (July 2016): The physical layer sections is now a separated document called "LoRaWAN Regional Parameters". First stable release.
- **Version 1.1** (October 2017): Improved security and roaming (add new root keys and session keys). New Frame Counters and new MAC Commands). JoinEUI replaces AppEUI. Clarification of Class B and Class C.
- **Version 1.0.3** (July 2018): version 1.0.3 = version 1.0.2 + Class B section of the version 1.1.
- **Version 1.0.4** (October 2020): AppEUI becomes JoinEUI. Many clarifications. Last version 1.0.x.

We notice that very early, a version 1.1 was published, but it was not adopted by the industry. The LoRa Alliance then continued the clarification of version 1.0.x by adding version 1.0.3 and version 1.0.4, which should be the last of the 1.0.x series.

Unless clearly specified, this document deals with LoRaWAN protocol version 1.0.x.

You can find on the LoRa Alliance resource HUB, all the version of the LoRaWAN specification:

- LoRaWAN specification version 1.0
- LoRaWAN specification version 1.0.1
- LoRaWAN specification version 1.0.2
- LoRaWAN specification version 1.0.3
- LoRaWAN specification version 1.0.4
- LoRaWAN specification version 1.1

4.1.3 Regional parameters

When the LoRa Alliance release a specification, it provides another document called the LoRaWAN regional parameters. This companion document describes the LoRaWAN regional parameters for different regulatory regions worldwide. Separating the regional parameters from the protocol specification allows addition or modification of new regions without impacting the latter.

The LoRaWAN specification: This document details the LoRaWAN protocol like the end-device Class, the message format, the frame format, the list of MAC commands, the activation modes (ABP, OTAA), etc…

The regional parameters: This document details the specific parameters for each region (EU868, EU433, US915,…) like the channel frequencies, the Data Rate, the Output Power, the maximum payload size, etc…

4.1.4 Differences between LoRa and LoRaWAN

LoRa is the type of modulation used between two LoRa end-devices or between an end-device and a gateway. When we talk about the whole communication chain (from the end-device to the LoRaWAN server), then we talk about LoRaWAN protocol. LoRaWAN is an extension of the LoRa protocol that gives the capabilities to securely connect the device to a Server in order to provide data to the end user.

- LoRa Physical Layer: Type of modulation (Chirp Spread Spectrum) and the physical frame format used to send data between a transmitter and a receiver.
- LoRaWAN Protocol: Network architecture (end-device, gateways, servers) and a more specific frame format allowing a LoRaWAN end-device to securely transmit data to a LoRaWAN server.

4.2 Structure of a LoRaWAN network

Figure 30 shows the entire LoRaWAN architecture. On the left side, there is the LoRaWAN end-device that transmits data. The user is on the other side and receives the transmitted data over the network.

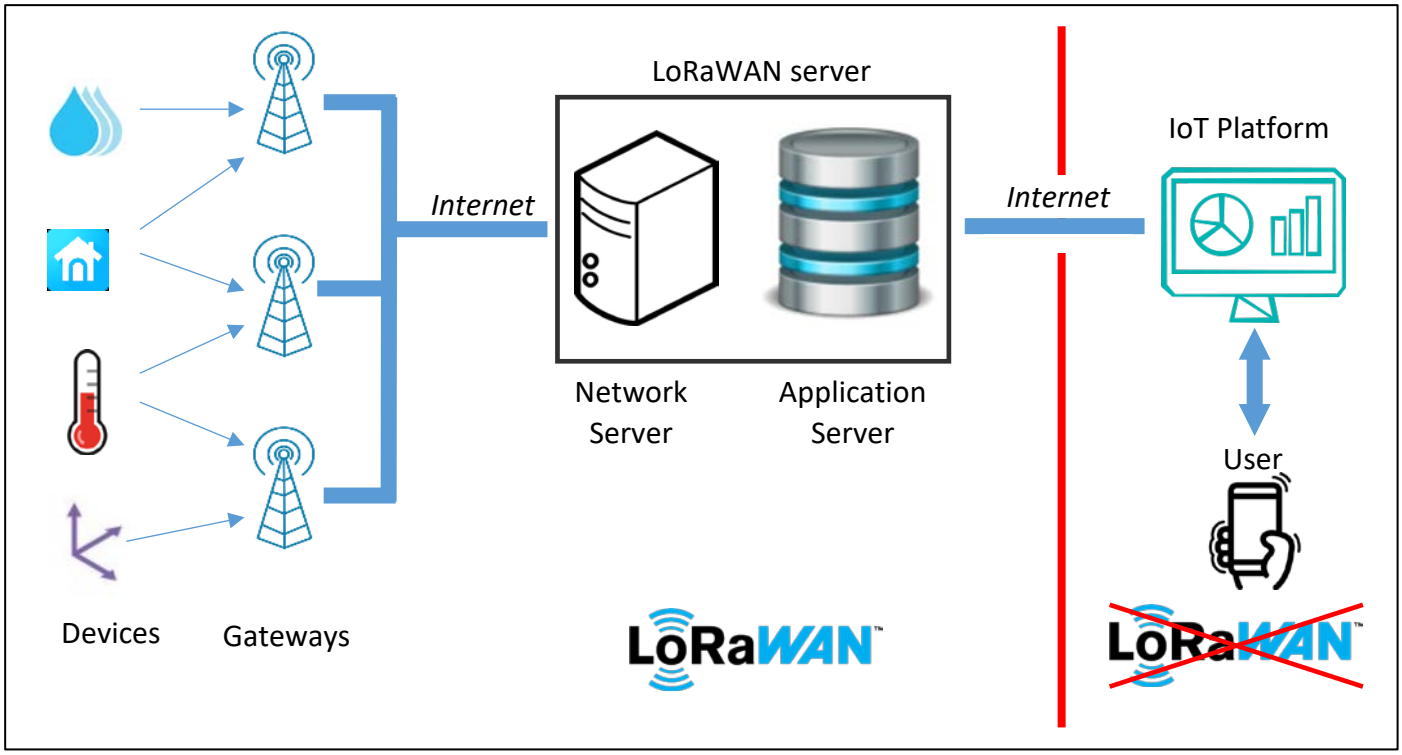


Figure 30: Overall architecture of a LoRaWAN network

The LoRaWAN end-devices, gateways, Network Server and Application Server are the heart of the LoRaWAN architecture, but the IoT platform and the User connection have nothing to do with it as it's just a classic web service.

4.2.1 LoRaWAN end-devices

LoRaWAN end-devices are electronic embedded systems belonging to the IoT world: low power consumption, small size, low power and low cost. They have a LoRa radio transceiver to reach gateways. A LoRaWAN end-device does not specifically address a gateway: all gateways present in the coverage area receive the messages and process them.

There are hundreds of LoRaWAN end-device manufacturers. A very few examples are:

- ATIM [www.atim.com] Designer and manufacturer of wireless data transmission solution since 1996. Pioneer on LPWAN technologies.
- nke-WATTECO [www.nke-watteco.fr] Designer and manufacturer of radiofrequency transmitter for many fields of application.
- adeunis [www.adeunis.com] IoT Sensors specialist. Expert in LPWAN network.
- Abeeway [www.abeeway.com] Provider of powerful and power-efficient geolocation solutions.

4.2.2 LoRaWAN gateways

They listen on all channels, and on all Spreading Factors at the same time. When a LoRa frame is received, it transmits its content over the internet to the Network Server that has been previously configured in the gateway.

On one side, the gateway receives a LoRa modulation on its antenna, and on the other side, it is connected to the internet via any possible backhaul: WiFi, 3G, 4G, 5G, Ethernet, LTE-M…

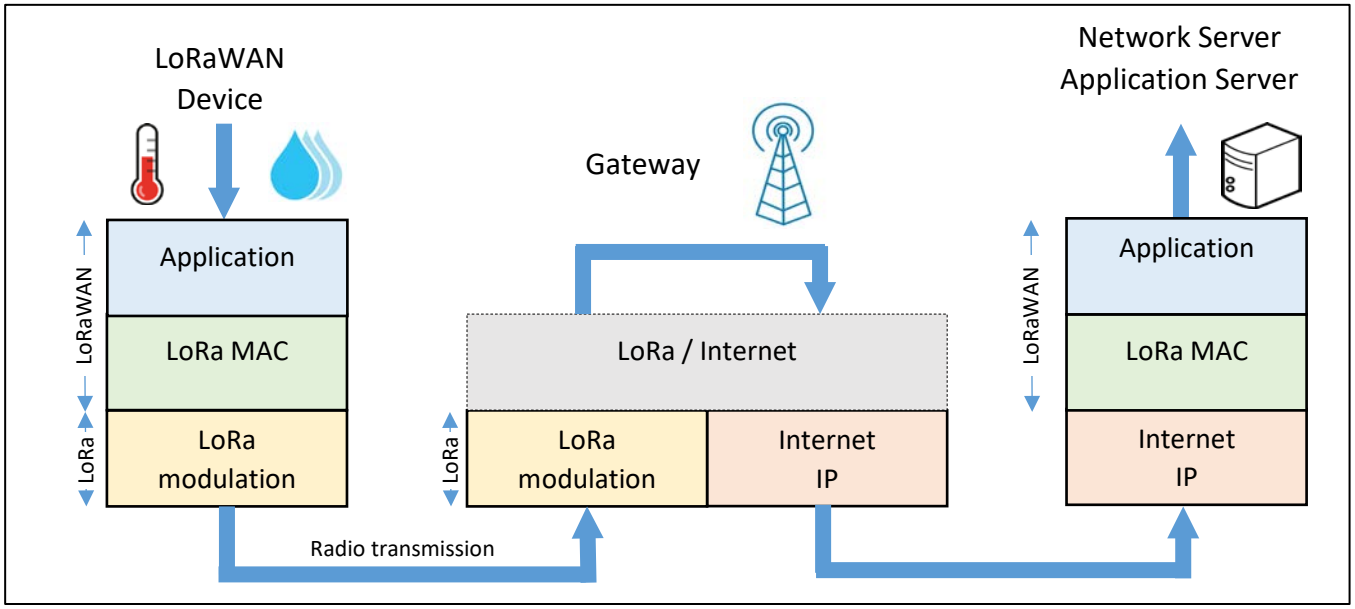


Figure 31: The LoRaWAN gateway

Each LoRaWAN gateway has a unique identifier (64 bits EUI). This ID is useful to register and activate a gateway on a Network Server.

4.2.3 The Network Server

The Network Server receives the messages transmitted by the gateways and removes duplicates packets (several gateways may receive the same message and transmit them to the same Network Server). Then the Network Server authenticate the message thanks to a 128-bit AES key called **NwkSKey** (**N**etwork **S**ession **K**ey). We are talking about authentication, not encryption.

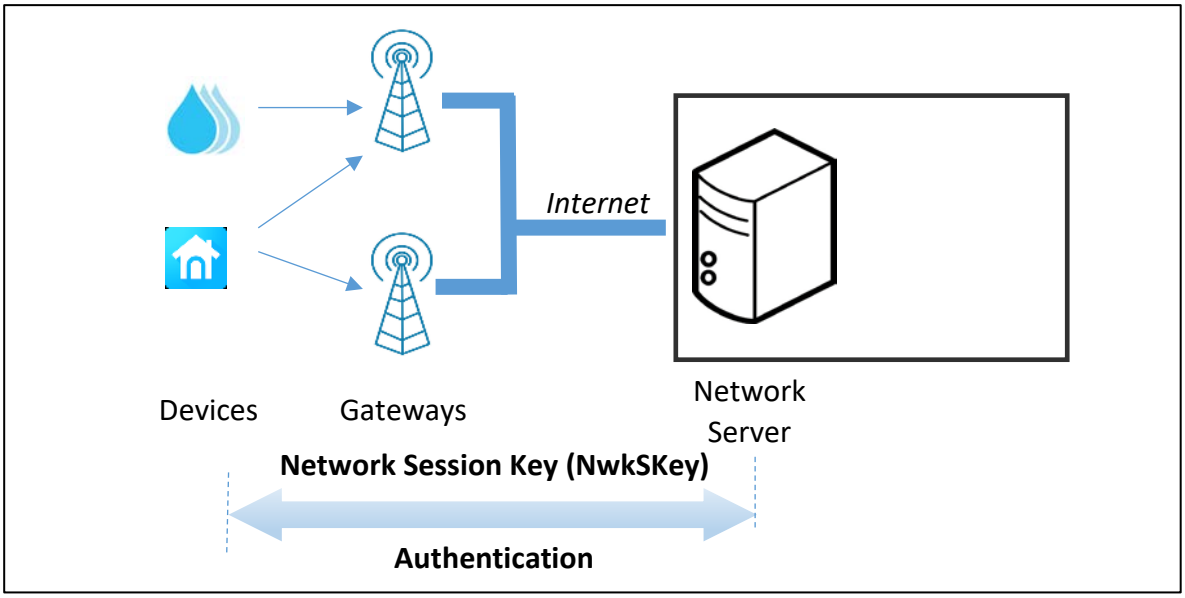


Figure 32: Authentication between the LoRaWAN end-device and the Network Server

- If the authentication process fails, the Network Server drops the LoRaWAN message.
- If the authentication process succeeds, the Network Server transfers the message to the Application Server.

4.2.4 Application Server

The Application Server receives encrypted messages from a Network Server. The encryption and decryption is made thanks a 128-bit AES key called **AppSKey** (**A**pplication **S**ession **K**ey). We will detail this process in chapter 4.2.7

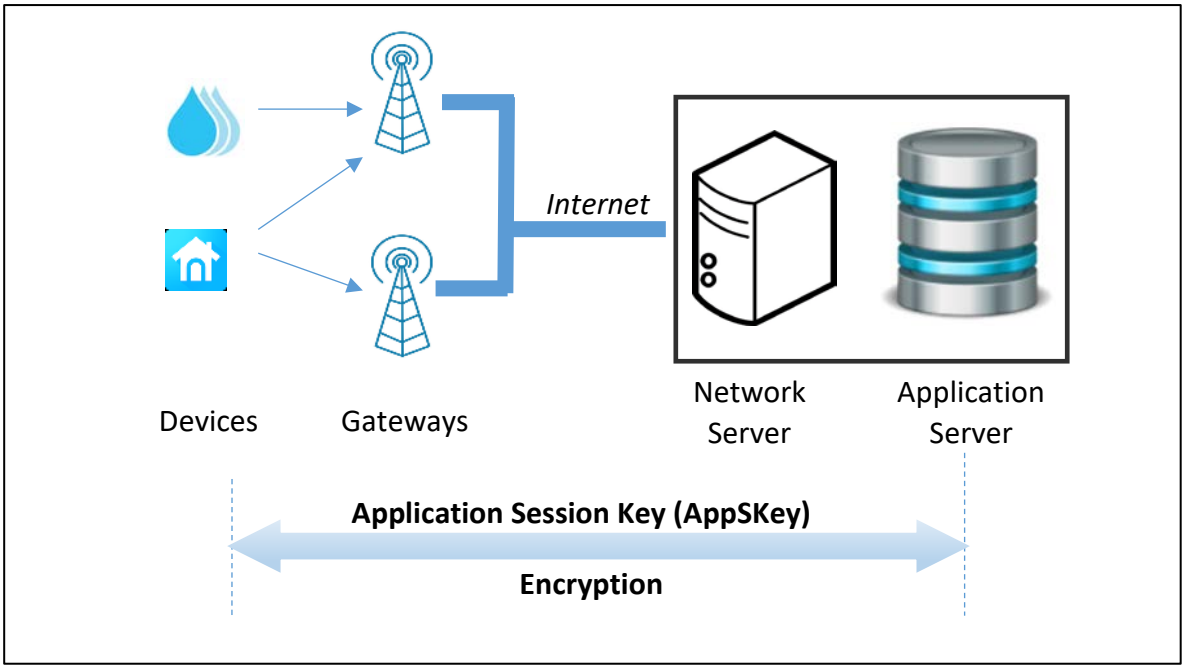


Figure 33: Encryption between the LoRaWAN end-device and the Application Server

4.2.5 The IoT platform

This is where the user Application stands. The three mains services are:

1. A connector with the Application Server to collect the data. Most of the time, this is done either with the HTTP or the MQTT protocol. During development, we sometime use the non-secure versions of these protocols. Obviously, we would use HTTPS and MQTTS during asset deployment.
2. A database to store data.
3. A dashboard accessible by the user via a web page or a mobile App.

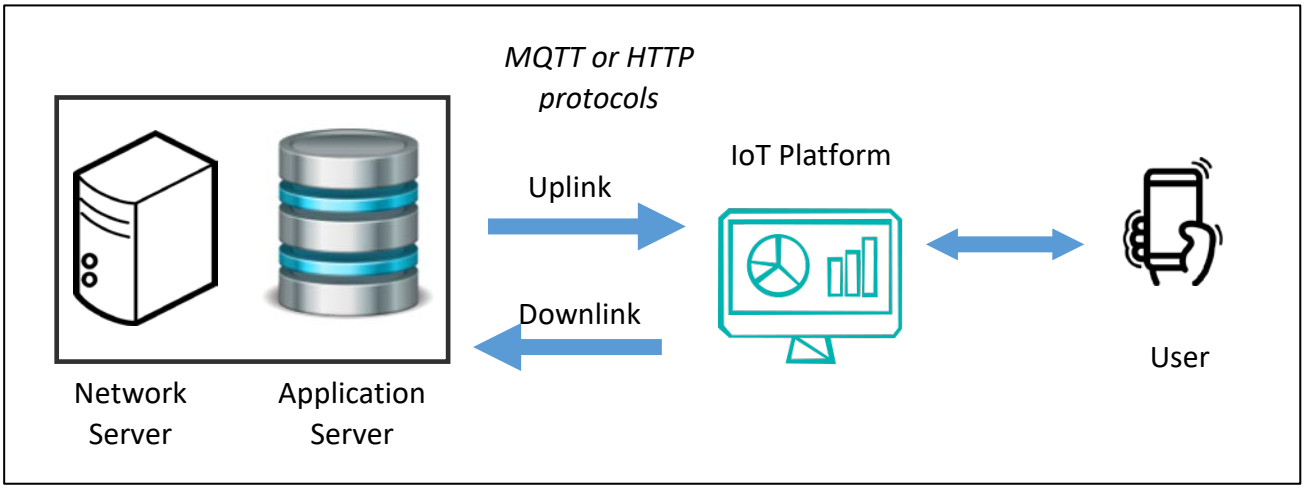


Figure 34: Connection between the Application Server and the IoT platform

In LoRaWAN, we mostly use uplink transfer (data from the end-device to the servers). As we will explain in section 4.3, it is also possible to transfer data to the end-devices using a downlink transfer.

4.2.6 Network Server and Application Server = LoRaWAN server

The LoRaWAN server is the association of the Network Server and the Application Server.

The term "LoRaWAN server" is not defined by the LoRa Alliance. It is just used in this book to combine both notion of authentication and encryption.

- We have a Network Session Key (**NwkSKey**) for authentication between the LoRaWAN end-device and the Network Server.
- We have an Application Session Key (**AppSKey**) for encryption between the LoRaWAN end-device and the Application Server.

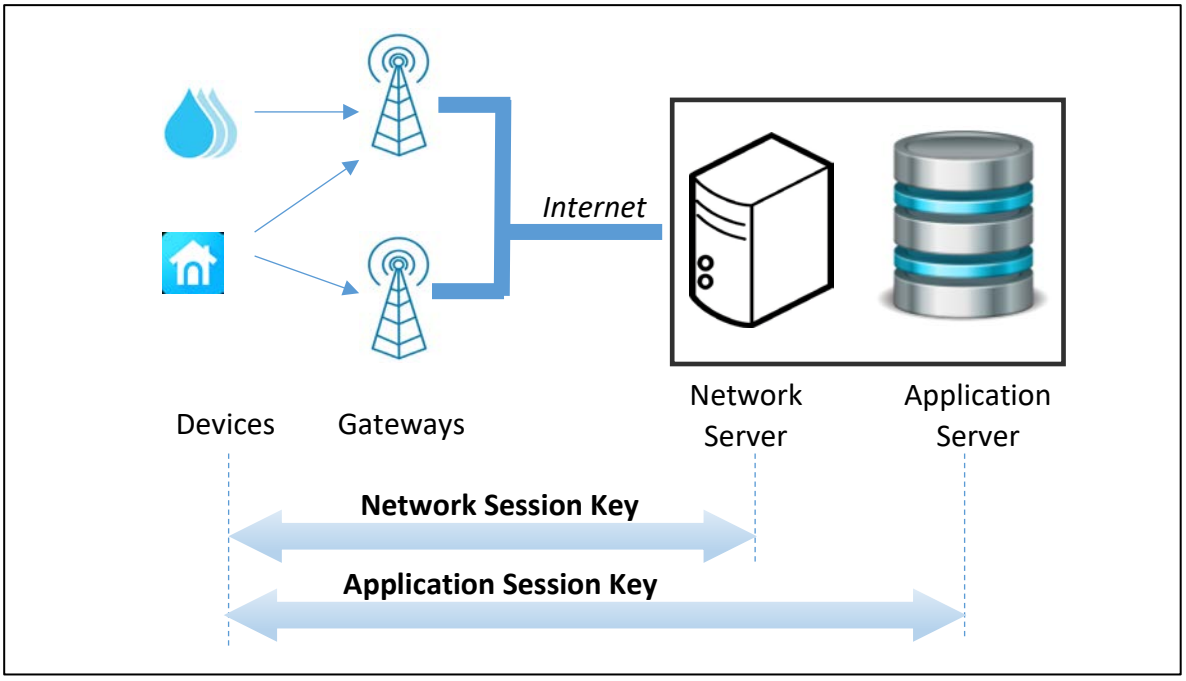


Figure 35: Authentication and encryption process

Unfortunately, the Network Server and the Application Server are often used in the same infrastructure. This is not the initial idea of a LoRaWAN Network because it does not provide end-to-end security.

End-to-end security is the ability to have the user message encrypted on the end-device and decrypted on the user Application side (IoT Platform for example).

Any time you see your data decrypted in your LoRaWAN server, that means that end-to-end security is not enable. The example below show a clear "01 02 03 04 05" payload sent earlier by an end-device. Your security standard might not accept that your LoRaWAN server provider can see your data.

Figure 36: "01 02 03 04 05" clear message in the LoRaWAN server

The solution is to integrate your own Application Server. This is not always easy but once the communication between the Network Server and the Application Server will be normalized (by the LoRa Alliance), the optimized architecture would the following:

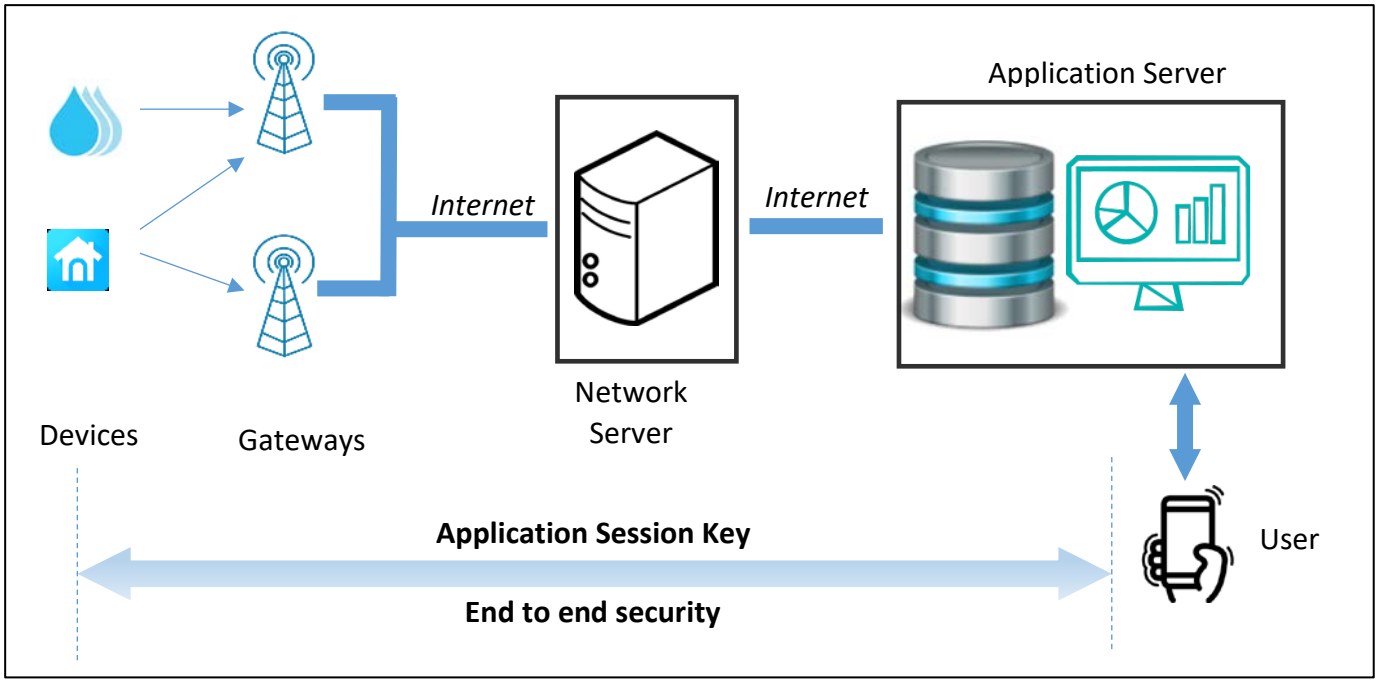


Figure 37: End to end security in a LoRaWAN Network

In that situation, the name "Application Server" (defined by the LoRa Alliance) will make more sense as it will really be the user Application combined with the promised end-to-end security.

4.2.7 Data encryption

The Application Session Key (**AppSKey**) is used to encrypt the user data on the LoRaWAN end-device. The data will be decrypted on the Application Server. This is a symmetric encryption, so AppSKey on the end-device should be the same than the one stored on the Application Server.

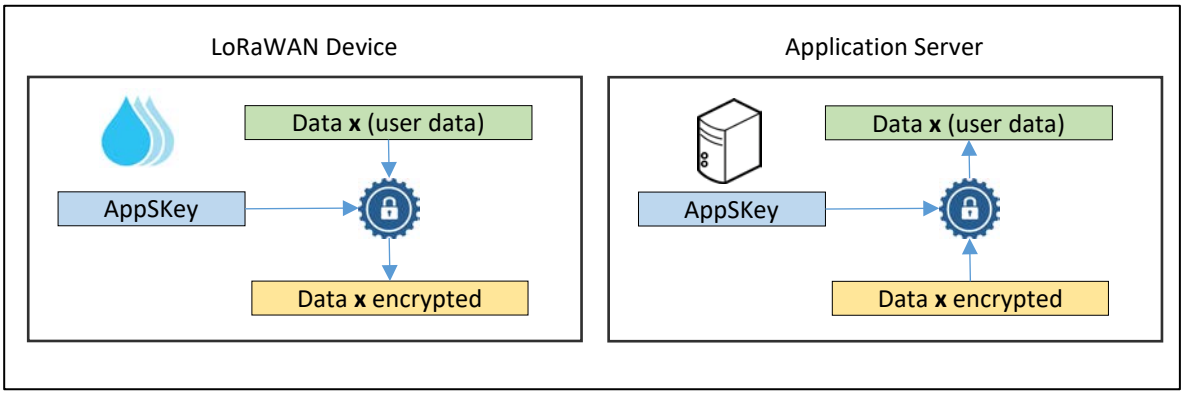


Figure 38: Encryption and decryption process

There is no way for the gateway and the Network Server to understand the real value of the user data. The channel is secured (confidential).

4.2.8 Authentication with the Network Server

The **N**etwork **S**ession **K**ey (**NwkSKey**) is used for authentication between the LoRaWAN end-device and the Network Server. In order to perform this authentication, a **MIC** (**M**essage **I**ntegrity **C**ontrol) field is added to the frame. The MIC depends on the encrypted transmitted data and the NwkSKey. During reception, the same calculation is performed. If NwkSKey is the same in the end-device and

in the Network Server, then the MIC transmitted should be the same than the one generated during reception.

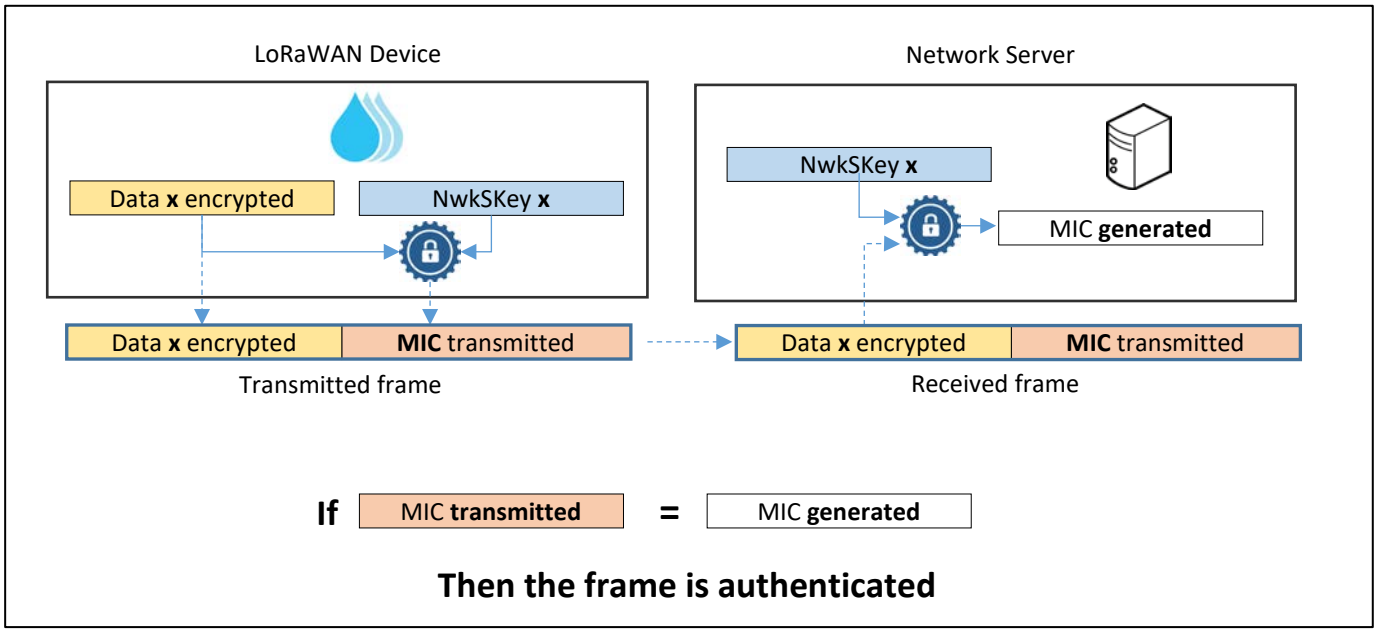


Figure 39: Device authentication by the Network Server

4.2.9 Combining authentication and encryption

We can now represent on the same figure the construction of the LoRaWAN frame with both authentication and encryption.

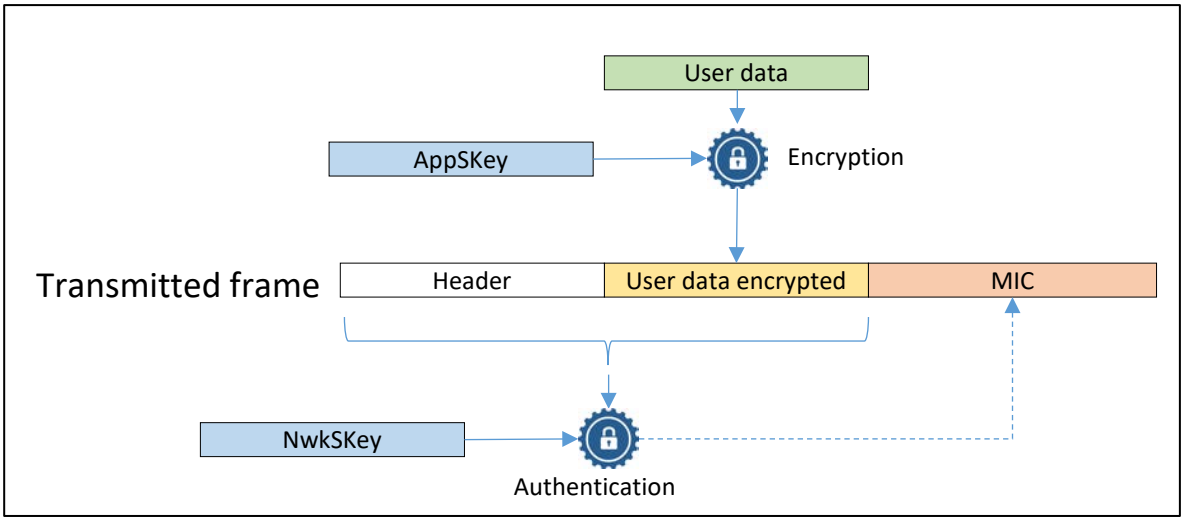


Figure 40: Encryption, then Authentication

On the server side, the decryption process applies only if authentication succeeds.

4.3 LoRaWAN end-device classes

LoRaWAN end-devices are classified in three categories (A, B, C) according to their power consumption and their **downlink** capabilities: the ease with which a user can transmit a frame to the end-device.

4.3.1 Class A (All): Minimal power Application

All LoRaWAN devices are class A. Each end-device can transmit (Uplink) to the gateway without checking the gateway availability. This transmission is followed by two very short reception windows. The Server (via the gateway) can transmit a downlink message during RX1 or RX2 slot, but not both.

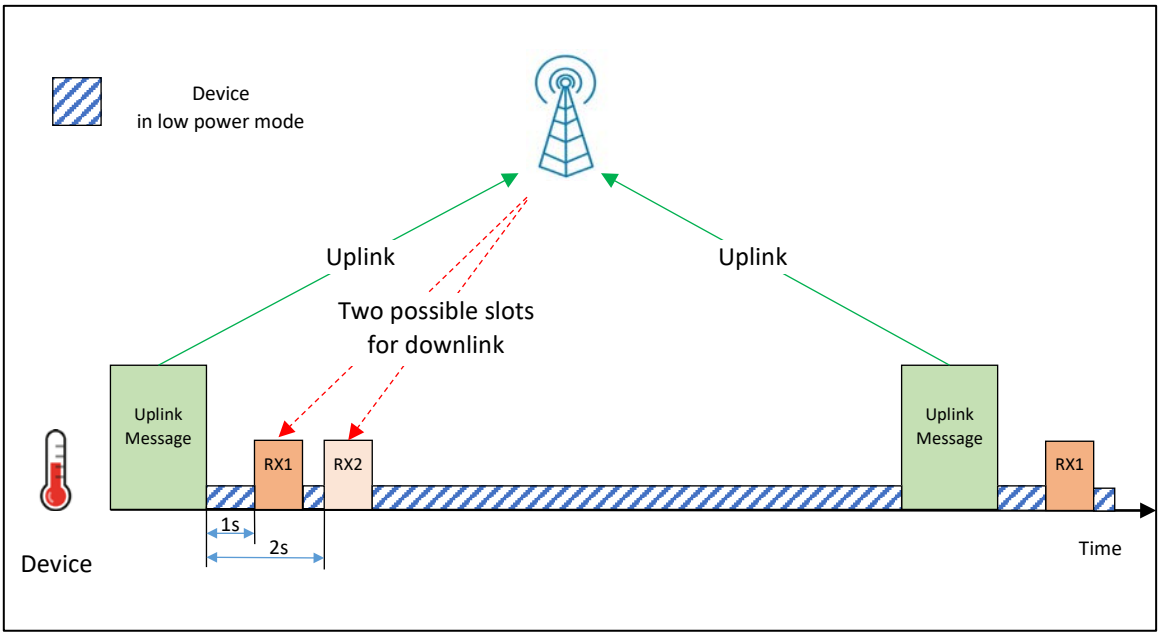


Figure 41: Receive slots for a Class A end-device.

The duration of the windows must be at least the preamble length (12.25 T_{symbol}). When a preamble is detected, the receiver must remain active until the end of the transmission. If the frame received during the first reception window was destined to the LoRaWAN device, then the second window is not opened.

First reception window RX1:

- Slot RX1 is programmed by default at 1 second +/-20 µs after the end of the uplink transmission. This value can be changed according to the Network Server configuration.
- The channel, Spreading Factor and bandwidth are the same as those chosen during the transmission (uplink).

Second reception window RX2:

- Slot RX2 is programmed by default at 2 seconds +/-20 µs after the end of the uplink transmission. This value can be changed according to the Network Server configuration.
- The channel, Spreading Factor and bandwidth are configurable but static.

A class A end-device can't receive if it has not transmitted uplink data. Therefore, you can't easily reach a class A end-device.

All end-devices start and join the network as Class A end-device.

4.3.2 Class B (Beacon): Scheduled receive slot

Class B end-devices behave in the same way as Class A devices, but other reception windows are scheduled at specific times. In order to synchronize the LoRaWAN end-device reception windows, gateways must transmit beacons on a regular basis.

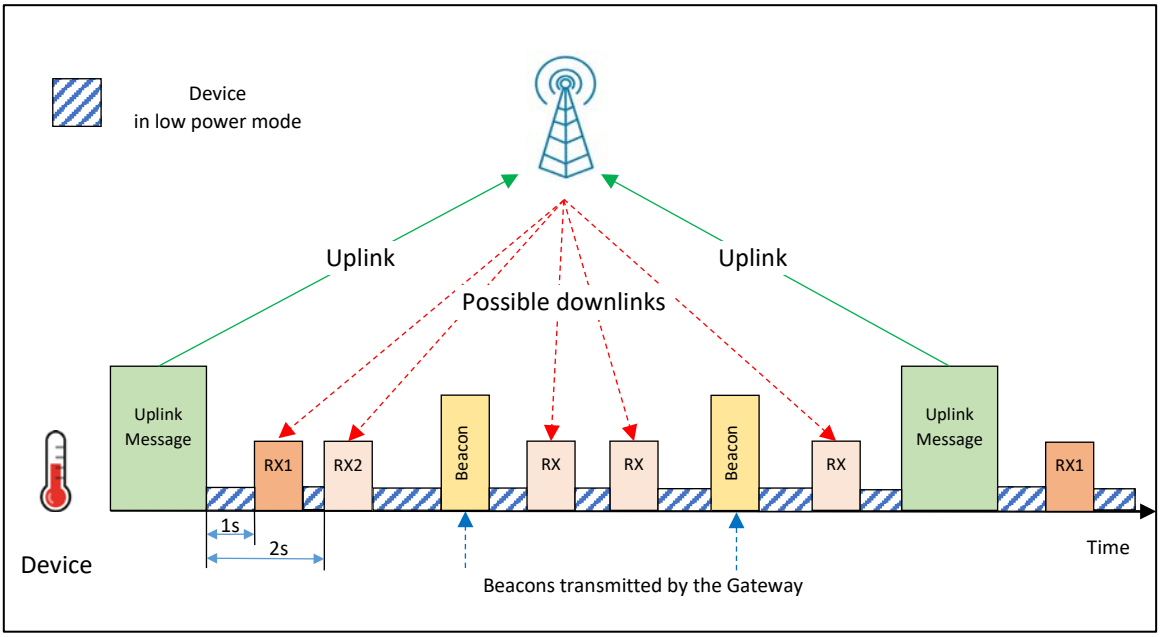


Figure 42: Receive slots for a Class B device

A class B end-device can be reached regularly without necessarily having to transmit. On the other hand, it consumes more power than a class A device.

All end-devices can decide to switch to class B if its firmware supports it. Obviously, the Network Server needs to be aware of it, but that is not managed by LoRaWAN. The complete specification for class B end-device have been released since the 1.0.3 version of the LoRaWAN protocol.

4.3.3 Class C (Continuous): Continuously listening

Class C devices have reception windows that are constantly open between two uplinks. These devices consume much more power.

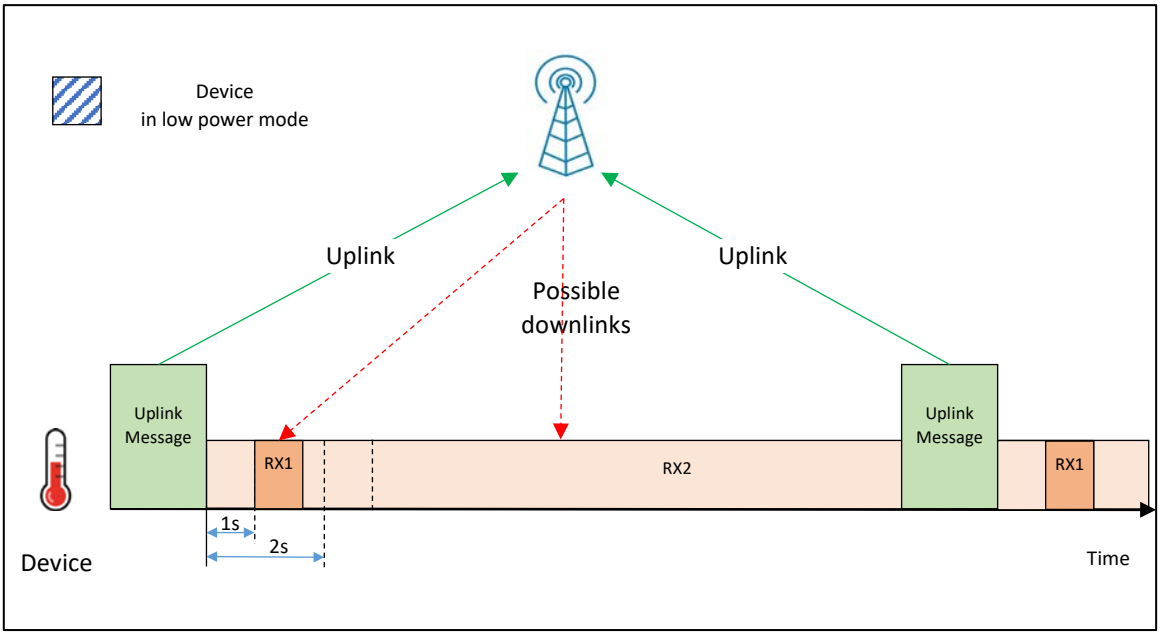


Figure 43: Receive slots for a Class C device

The LoRaWAN end-device is continuously listening between two uplinks messages. All RX slots are set to the same parameters (channel, Spreading Factor and bandwidth) as RX2 except the RX1 windows that still have the same behaviour as in class A and B.

A class C end-device is always reachable. However, this is the class of device that consumes the more energy.

All end-devices can decide to switch to class C. Obviously, the Network Server needs to be aware, but that is not managed by LoRaWAN before the version 1.2.0 of the protocol specification.

4.3.4 Summary of end-device classes

From the previous figures, we can notice that:

- A class B end-device is also a class A device (RX1 and RX2 slot are still present).
- A class C end-device is also a class A device (RX1 and RX2 slot are still present).

Class B and class C are therefore an extension to the class A. We can represent the LoRaWAN end-device classes as follows:

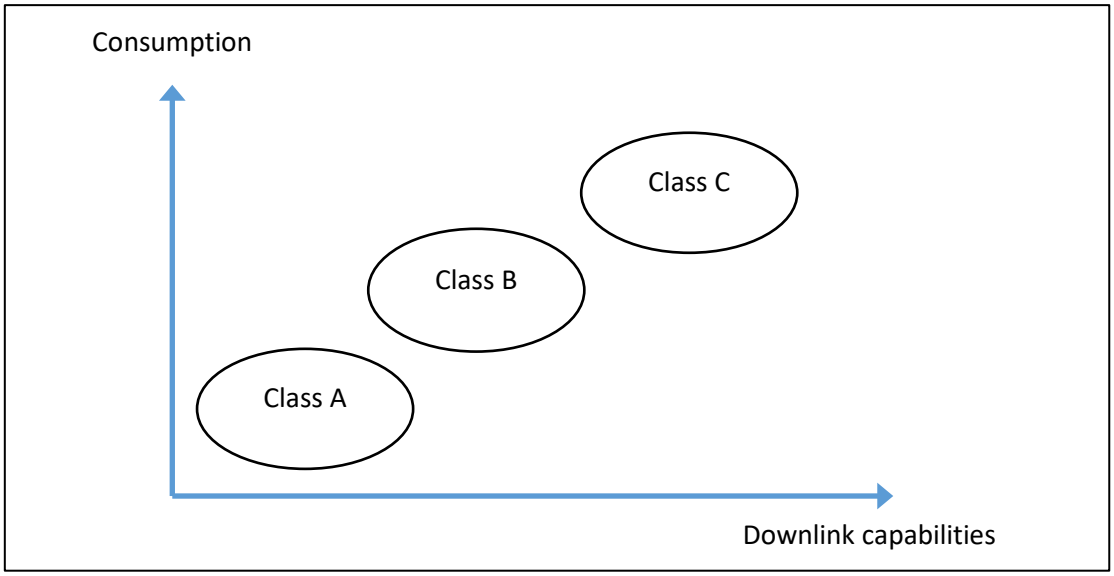


Figure 44: Power consumption and downlink capabilities

4.3.5 Which gateway for downlink?

The common data transfer in LoRaWAN is uplink. But a user might need to send data to its end-device and therefore use the downlink capabilities of LoRaWAN. In that case, one may wonder which gateway will be chosen to transfer the data. Indeed, the location of the LoRaWAN end-device is not necessarily known in advance and obviously, all gateways will not send the message over the entire network.

The gateway used for the Downlink is the one that received the last uplink message. If several gateways received the last uplink message, a selection is made with the RSSI value to ensure the best chance of reaching it.

A downlink message will never reach a LoRaWAN end-device if it has never transmitted before, regardless of its class A, B or C.

4.4 Activation of LoRaWAN end-devices: ABP and OTAA

In LoRaWAN, the three essential elements for communication are the **DevAddr** for the identification of the end-device, as well as two keys: the **NwkSKey** for authentication and the **AppSKey** for encryption. Two methods are possible to provide this information to both the end-device and the LoRaWAN server:

- Activation By Personalization: **ABP**.
- Over The Air Activation: **OTAA**.

4.4.1 ABP: Activation By Personalization

This is the simplest method. It is therefore perhaps the one we tend to use when testing a prototype and setting up a LoRaWAN communication.

- Static **DevAddr**, **NwkSKey** and **AppSKey** are stored in the end-device.

- The same **DevAddr**, **NwkSKey** and **AppSKey** are stored in LoRaWAN server.

In ABP, all the information needed for communication is already known by the end-device, the Network Server and the Application Server.

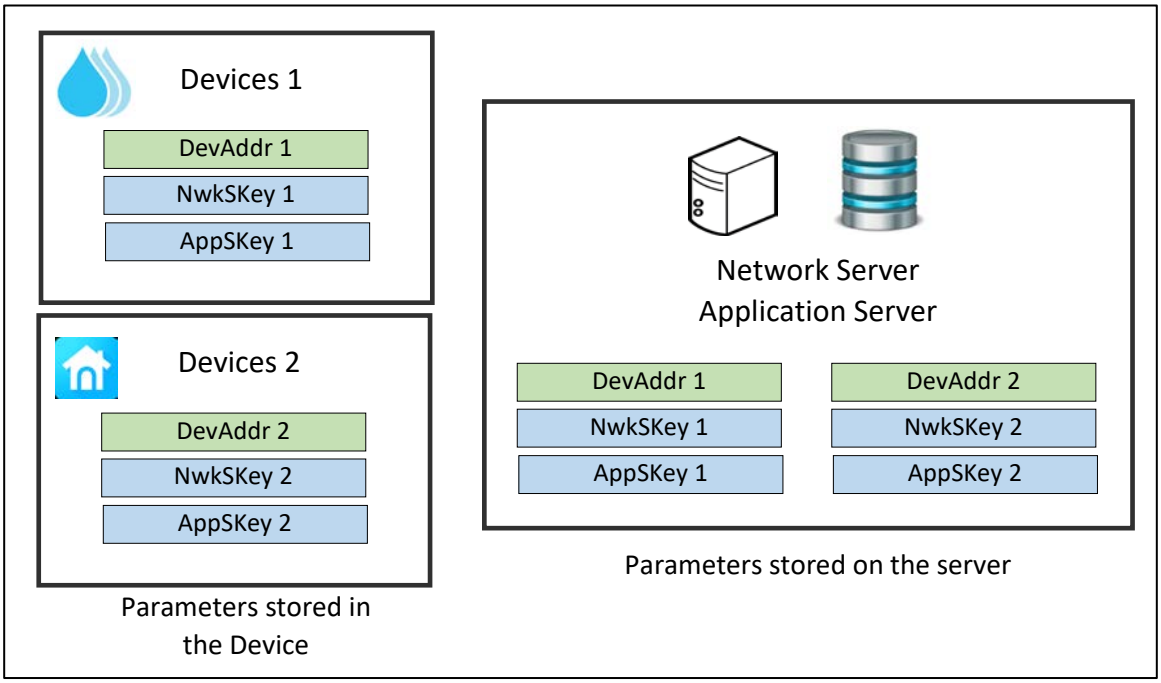


Figure 45: DevAddr, NwkSKey and AppSKey in ABP

As soon as the end-device has started, it can send and receive LoRaWAN messages.

4.4.2 OTAA: Over The Air Activation

With this activation mode, the **DevAddr**, **AppSKey** and **NwkSKey** will be generated during a Join procedure when the LoRaWAN end-device connects to the Network Server. To achieve this Join procedure, the LoRaWAN end-device must be configured with:

- **DevEUI**
- **AppEUI/JoinEUI**
- **AppKey**

The Network Server must know the same **DevEUI**, **AppEUI/JoinEUI**, and **AppKey** and the main purpose of the Join-Request is to retrieve the final configuration with **DevAddr**, **NwkSKey** and **AppSKey** on both sides.

All the items named "EUI" (**E**xtended **U**nique **I**dentifier) are always unique with an 8 bytes size.

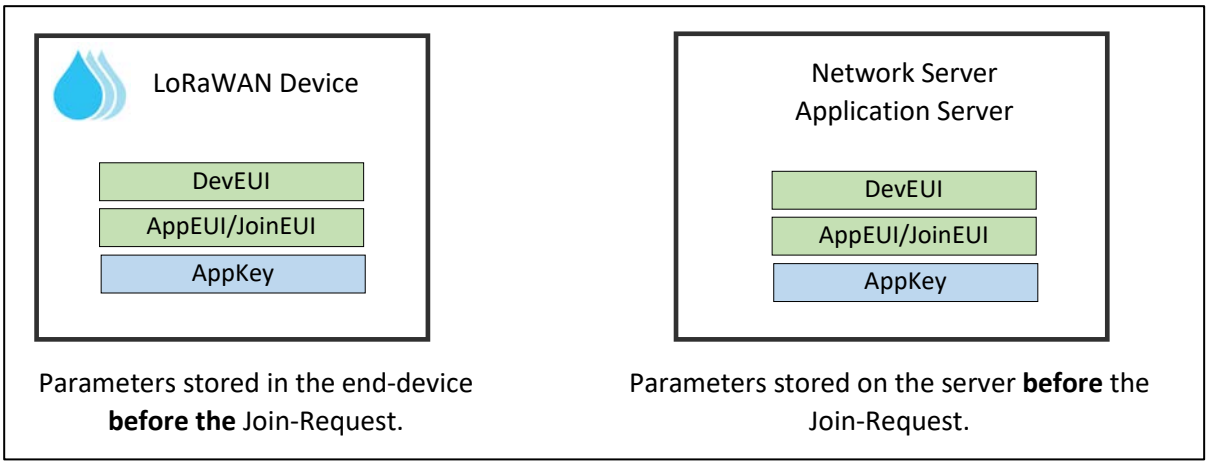


*Figure 46: Parameter stored **before** the Join-Request (OTAA)*

When the Join-Request has taken place, then the generated parameters **DevAddr**, **NwkSKey** and **AppSKey** are saved on both sides.

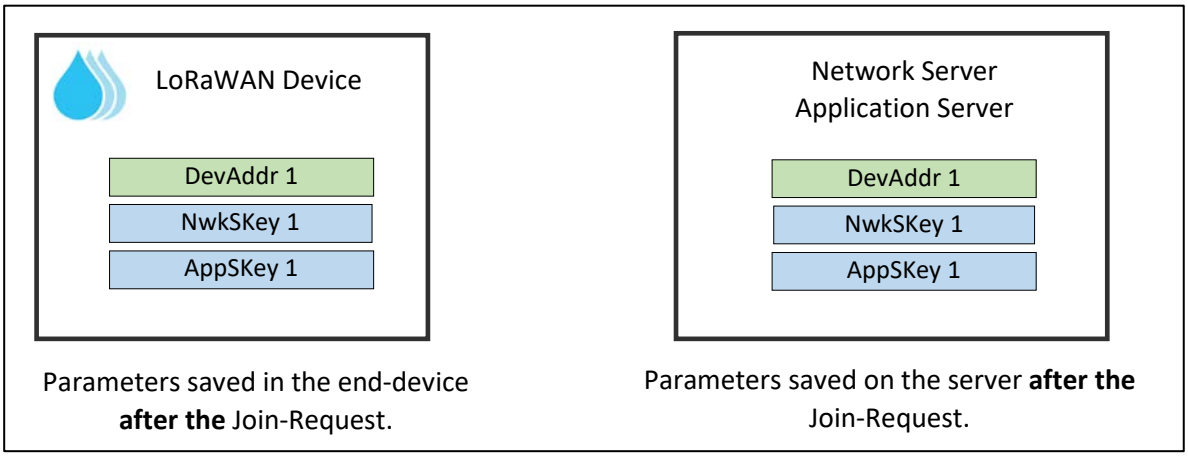


*Figure 47: Configuration **after** the Join-Request (OTAA)*

We will explain later the pros and cons of each activation mode. For the moment, we need to understand the meaning of the initial configuration stored in the end-device and on the server before the Join-Request:

- **DevEUI**: Unique Identifier for the LoRaWAN end-device. This is Equivalent to a MAC address on Ethernet. Some LoRaWAN end-devices already have a fixed DevEUI stored during factory firmware programming and cannot be changed.
- **AppKey**: AES 128 key used to authenticate the Join-Request, to encrypt the Join-Accept and to generate the session keys. This key must be kept secret and never be shared with anyone.
- **AppEUI/JoinEUI**: This parameter has different meaning depending on the LoRaWAN version. In LoRaWAN 1.0.3 and before, it was an application identifier (AppEUI). From LoRaWAN 1.0.4, this parameter has been rename in JoinEUI as it defines a Join Server identifier.

To keep it simple and for educational purpose, we will not use any Join Server at the beginning of this book, so we can simplify this EUI and use "0000000000000000" as the AppEUI/JoinEUI.

As a reminder, we detail once again the purpose of the final configuration after the Join procedure:

- **NwkSKey**: Used for authentication with the Network Server.
- **AppSKey**: Used for data encryption with the Application Server.
- **DevAddr**: 32-bit identifier within a LoRaWAN network.

Figure 48 show the simplified representation of the Join procedure.

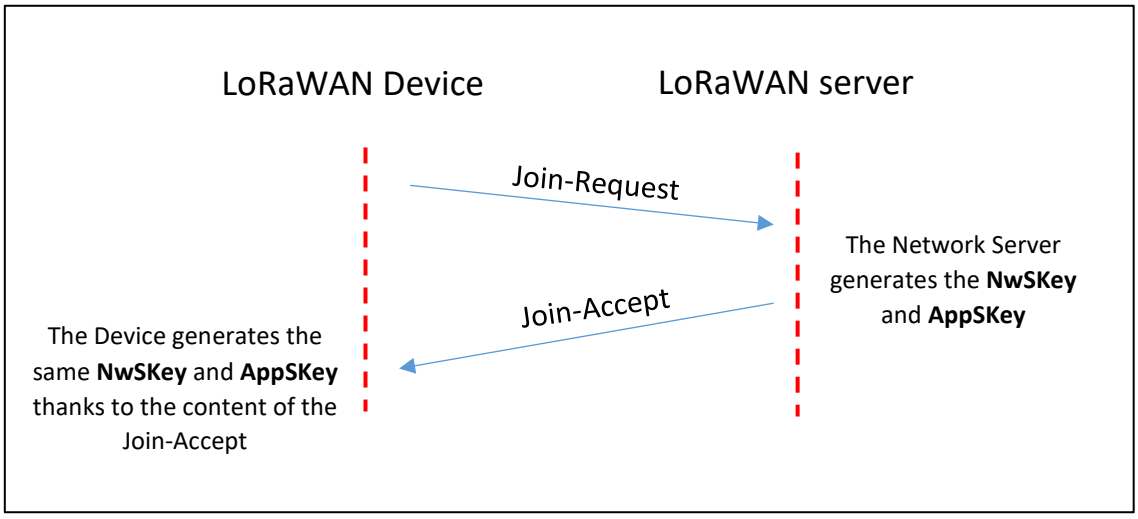


Figure 48: Join-Request - Join-Accept in OTAA

4.4.3 The Join procedure

Figure 48 represents only the simplified Join procedure. In this chapter, we will go further in order to have a better understanding of the process.

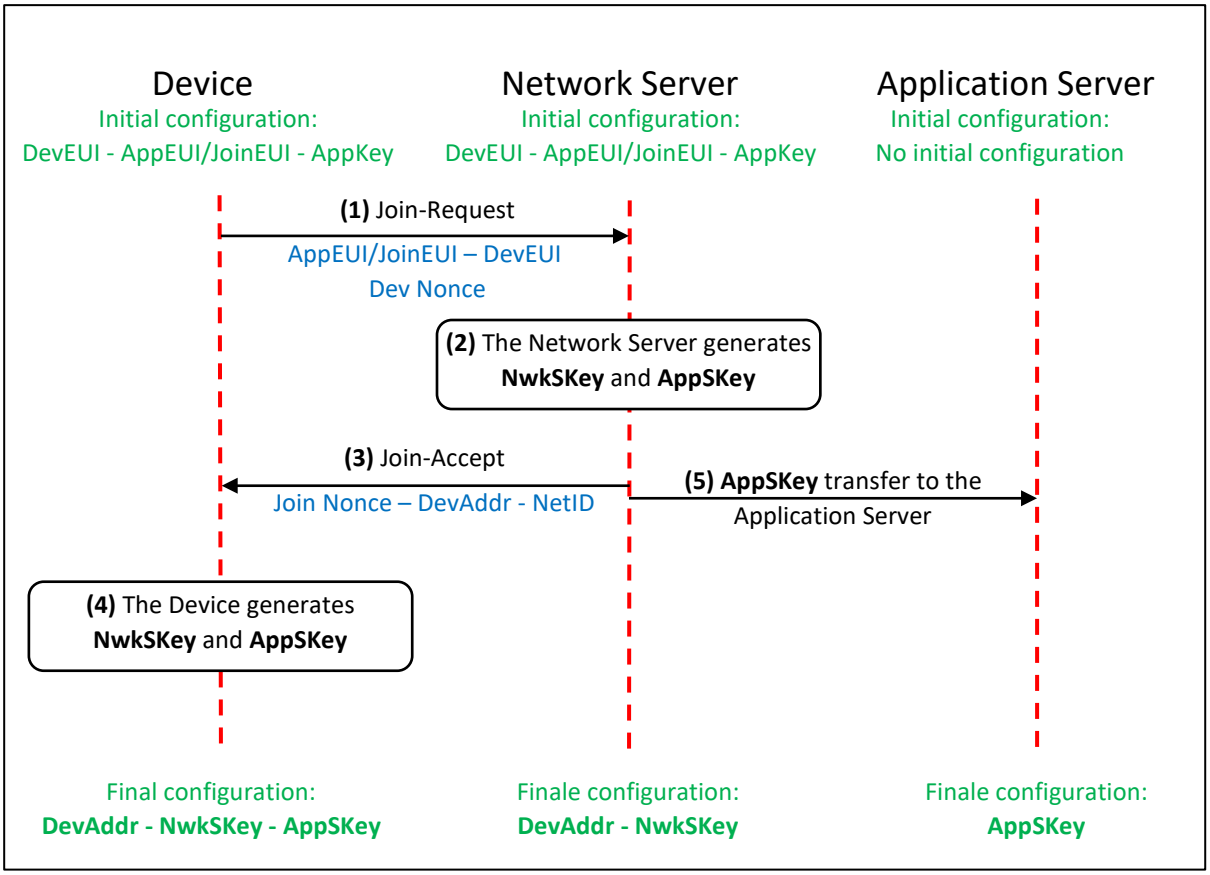


Figure 49: Join-Request and Join-Accept procedure in detail

(1) - The LoRaWAN end-device transmit a MIC (**M**essage **I**ntegrity **C**ode) to authenticate its request so only an end-device registered on the Network Server can trigger a Join-Accept response. The MIC field (4 bytes) is calculated thanks to the AppKey, JoinEUI, DevEUI and DevNonce. The Join Request frame is not encrypted, so you can easily see the AppEUI/JoinEUI and the DevEUI on the activity logs of your gateway or Network Server. DevNonce is just a random number to prevent replay attack.

Size (octets)	8	8	2
Join-Request payload	JoinEUI	DevEUI	DevNonce

Figure 50: Join-Request MAC payload

Figure 51 represents a Join-Request captured from an end-device in Actility LoRaWAN server using the Wireless-Logger tool. We can clearly see the DevEUI, JoinEUI, DevNonce and MIC (no encryption).

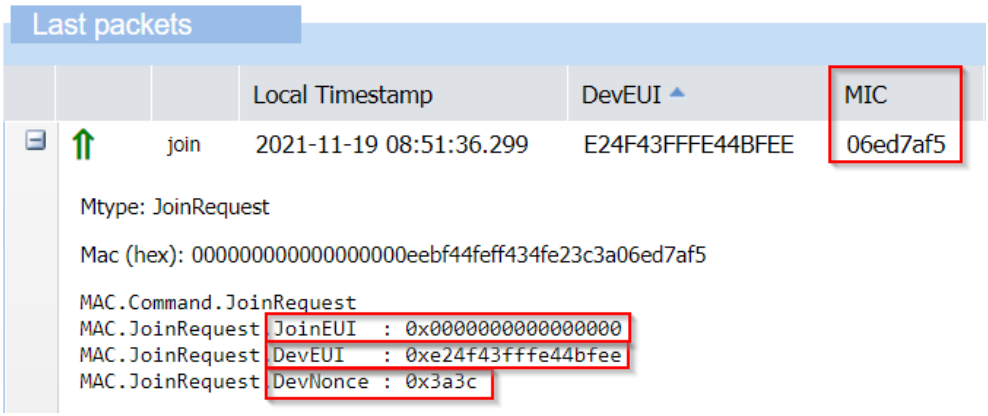


Figure 51: DevEUI, JoinEUI, DevNonce and MIC in a Join Request.

(2) If the end-device is authenticated, then the Network Server generates NwkSKey and AppSKey.

(3) A Join-Accept frame is scheduled 5 seconds (RX1) or 6 seconds (RX2) after the Join-Request. The NwkSKey and AppSkey are not directly transferred.

Size (octets)	3	3	4	1	1	(16) optional
Join-Accept payload	JoinNonce	NetID	DevAddr	DLSettings	RXDelay	CFList

Figure 52: Join-Accept MAC payload

The DevAddr and the NetID are the two first information that the end-device saves. It also receives parameters needed for a proper communication with the Network Server:

- the **D**ownlink **S**ettings (**DLS**ettings)
- the delay for the downlink (RXDelay)
- the channel list that the end-device has to use (CFList)

The last value is the JoinNonce that has to be used to recalculate the NwkSKey and AppSkey on the end-device side.

The Join-Accept is encrypted by the AppKey, so only the end-device can understand its content. Figure 53 represents the Join-Accept with its encrypted content captured 5 seconds after the Join-Request.

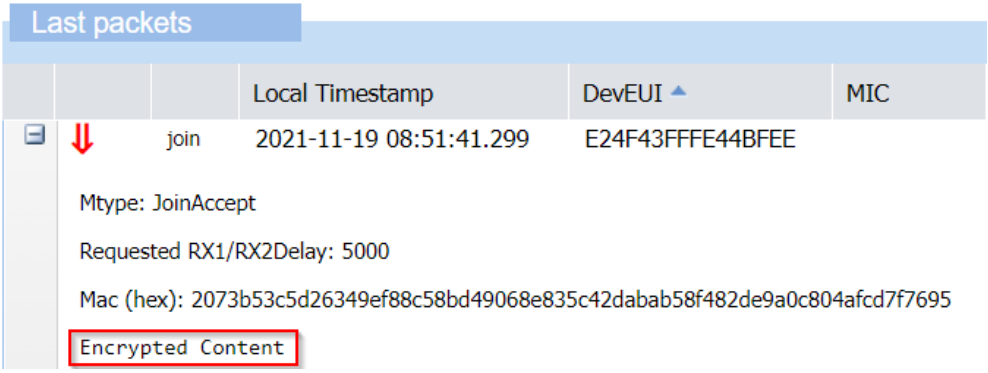


Figure 53: Join-Accept

(4) The end-device generate the NwkSKey and AppSkey using the content of the Join-Accept.

(5) The AppSKey is transferred to the Application Server.

In recent revision of the specification, the Network Server is not in charge of the Join procedure any more. The Join procedure is handled by another server called **Join Server**. So the AppSKey is not known by the Network Server (NS) and therefore the NS doesn't have access to the user data. It is a great improvement in terms of security. We will see this architecture later in this book.

4.5 Pro and cons of ABP and OTAA

We have discussed about the two different methods to activate a device and we now need to explain when it is more appropriate to use one or the other method.

4.5.1 Security

The weak point of each method is the key permanently stored in the LoRaWAN end-device:

- Session keys: **NwSKey** and **AppSKey** in ABP.
- Root Key: **AppKey** in OTAA.

They therefore have to be stored in highly secured memories.

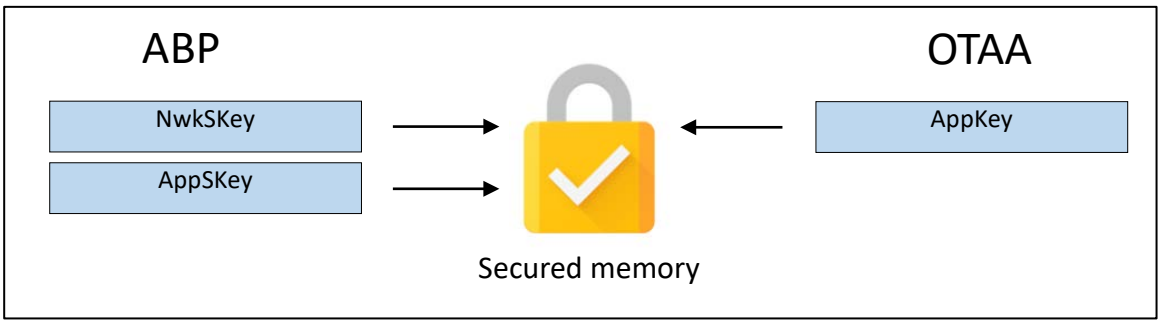


Figure 54: Keys stored in a secured memory

However, since the ABP method keeps the session keys forever, there is a greater chance of having them stolen by brute force attack, especially if the end-device replays the same sequences many times, or resets with the same behaviour: this was allowed until LoRaWAN 1.0.3. Moreover, manipulating session keys in ABP (if one wants to change LoRaWAN network for example) gives more opportunities to expose the keys and thus make them visible.

From a security point of view, the OTAA activation mode should always be preferred.

4.5.2 Network change

A client can decide to change of LoRaWAN server provider. In case of ABP activation mode, he will have to manually transfer all DevAddr and session keys from one Server to another. He cannot even be sure that the old provider have erased all session keys and will therefore be able to continue listening to the content of the packet transmitted. This is another security threat.

In case of OTAA, a Join Server can be used. We will present later the role of this server but for the time being, we can only say that the client will just have to tell the Join Server that another Network Server is used. The root keys (AppKey) can stay in their initial safe place, and only the session keys will be regenerated thanks to a new Join Request.

4.5.3 Protection against replay attack (uplink messages)

The 128-bit AES keys encrypts the data. Despite this encryption, a known attack in Wireless is the REPLAY attack: the hacker records encrypted frames transmitted on the LoRa network and will transmit them again later. Even if the hacker does not understand the content (the frames are encrypted), the data that are transported are well understood by the Application Server. Actions can therefore be performed simply by repeating a previous frame.

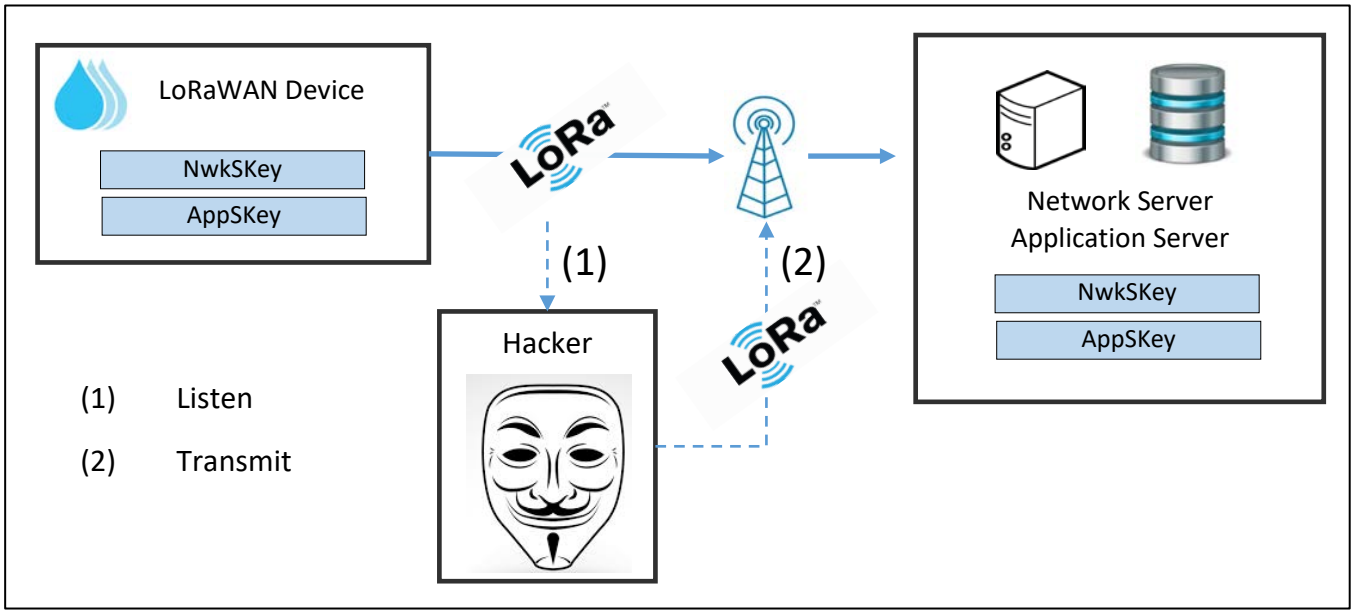


Figure 55: REPLAY attack

To avoid this, the LoRaWAN frame includes a variable field called **Frame Counter**. This number increments itself each time a new frame is transmitted on the end-device. The same kind of counter exists on the server side and it increments itself each time it receives a valid frame.

- The server accepts a frame only if the frame counter is strictly greater than the last valid frame counter received from that device.
- If the hacker retransmits the frame as he has recorded it, the Frame Counter received on the server will be lower (or egal) than the one on the Server side: if so, the server drops the frame silently.

If the hacker decides to modify the Frame Counter field with a random value, the authentication will fail because the calculation of the MIC field (with the Network Session Key) will not be valid anymore.

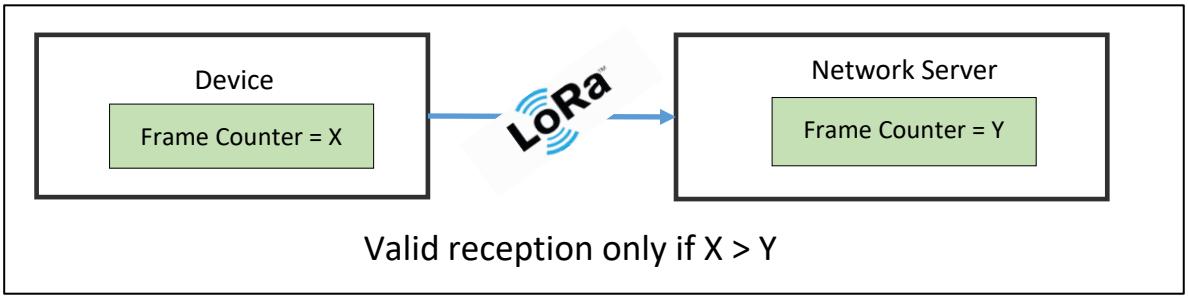


Figure 56: Frame Counter to avoid the Replay attack

The Frame Counter is interesting to avoid a REPLAY attack, but during the development of a LoRaWAN end-device it can be a problem. Indeed, each time we restart the microcontroller, our Frame Counter goes back to zero, while the server Frame Counter keeps incrementing. This can be solved by different ways:

1. Enable the possibility to "Reset Frame Counter" on the end-device. On some LoRaWAN server you have the option to accept any Frame Counter whatever their values. Of course, you have to keep in mind that this is security threat.

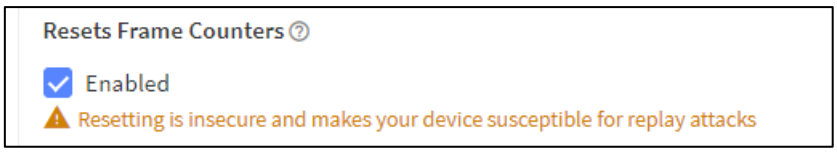


Figure 57: Enabling the "Resets Frame Counters" on TTN LoRaWAN server

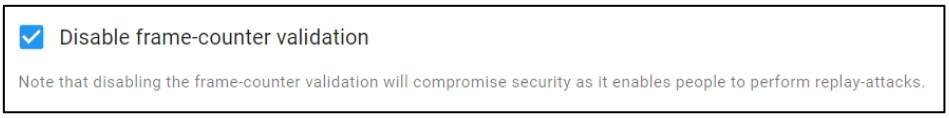


Figure 58: Disabling the "Frame Counter Validation" on Chirpstack LoRaWAN server

2. Use OTAA activation instead of ABP. Indeed, at each OTAA Join, the Frame Counter is reset on the end-device AND on the server side.
3. Keep the value of the Frame Counter in a non volatile memory and retrieve its value when the LoRaWAN end-device restarts.

Keeping the Frame counter in a non-volatile memory is mandatory in recent version of the LoRaWAN specification.

Exercise: Try to play the hacker role by producing a replay attack on your LoRaWAN server.

Solution:

1. Send a message from your LoRaWAN end-device with a simple payload and check its good reception on your Application Server.
2. Check your gateway logs and pick up the PHY Payload. You cannot understand it, but you will replay it. That is what a hacker does.
3. Disable the Frame Counter check on your LoRaWAN server: you open the replay attack vulnerability.
4. Program your end-device in order to send a LoRa Frame (not LoRaWAN!) with the PHY Payload stolen.
5. You should see the payload on your Application Server.

If not, ask for help ;-)

4.5.4 Protection against replay attack (downlink messages)

There is another Frame Counters used for downlink messages. A Frame counter on the server side increments each time the server sends a downlink message to the end-device. An end-device accepts a downlink message only if the Frame Counter received is equal or greater than the one on its side.

Last packets

			Local Timestamp	DevAddr	FPort	FCnt ⇑	NFCnt ⇓	RSSI	SNR	ESP	SF/DR
⊞	⇓		2021-11-27 23:45:05.202	0493C363	1		5				SF7
⊞	⇑	data	2021-11-27 23:45:04.202	0493C363	1	6		0.0	10.75	-0.3508...	SF7
⊞	⇓		2021-11-27 23:44:57.151	0493C363	1		4				SF7
⊞	⇑	data	2021-11-27 23:44:56.151	0493C363	1	5		-1.0	9.5	-1.4618...	SF7
⊞	⇓		2021-11-27 23:44:52.946	0493C363	1		3				SF7
⊞	⇑	data	2021-11-27 23:44:51.946	0493C363	1	4		0.0	9.75	-0.4372...	SF7
⊞	⇓		2021-11-27 23:44:35.259	0493C363	1		2				SF9
⊞	⇑	data	2021-11-27 23:44:34.259	0493C363	1	3		-1.0	11.75	-1.28097	SF9

Figure 59: Frame Counter for uplink and downlink

Figure 59 represents both uplink and downlink Frame Counter captured in Actility LoRaWAN server using the Wireless-Logger tool. We can clearly see that it increases on each packet.

4.5.5 Protection against replay attack (Join-Request)

A hacker can also use a replay attack during the transfer of the Join-Request in OTAA. A counter whose name is "DevNonce" is used on the same purpose. Since version 1.0.4 of the LoRaWAN specification, an end-device in OTAA must save this number in a non-volatile memory, otherwise the Join-Request will not be accepted if the end-device resets.

4.5.6 Communication parameters

When we operate in OTAA, the LoRaWAN end-device sends a Join-Request. A Join-Accept is returned by the Network Server if the end-device has been registered on this server. We saw in chapter 4.4.3 that the Join-Accept includes:

A **DLSettings** field: The **DLSettings** field indicates information on the Spreading Factor and Bandwidth that the device should use to receive on RX1 and RX2 receive windows.

An **RX Delay** field: The **RX Delay** field indicates the time between the end of the transmission (uplink) and the beginning of the reception window (downlink). We called that RX1 delay.

A **CFList** field: **CFList** indicates the list of all available channels for this network in addition to channels 0 to 2 (compulsory channel): 868.1 MHz, 868.3 MHz and 868.5 MHz. The other channels assigned in the Join-Accept frame (channels 3 to 7). This gives a maximum of 8 channels overall.

Size (bytes)	3	3	3	3	3	1
CFList	Freq Ch3	Freq Ch4	Freq Ch5	Freq Ch6	Freq Ch7	CFListType

Figure 60: Channels 3 to 7 defined in the Join-Accept CFlist

These three settings are only present in the Join-Accept frame, so an end-device running in ABP cannot benefit from them.

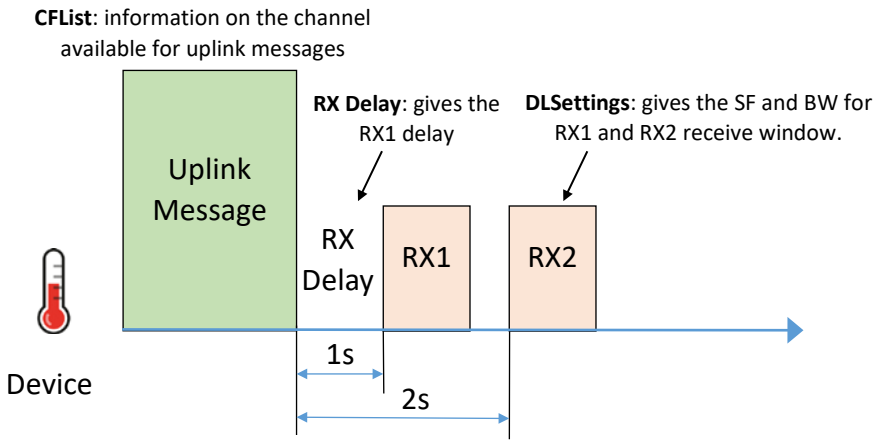


Figure 61: DLSettings, RX Delay and CFList parameters

4.5.7 Summary

We can therefore summarize the two activation methods in the following table.

	ABP	OTAA
Global security	Less secured than OTAA	More secure
Frame Counter management	Backup in non-volatile memory is mandatory. Possibility to disable the counter check and open a security threat.	Supported by OTAA
Network change	Complicated and unsecured	Supported by OTAA with a Join Server
Modification RX Delay DLSettings	Manage by MAC commands	Supported by OTAA
Adding channels	Manage by MAC command	Supported by OTAA

Table 14: Comparison of ABP and OTAA activation methods

4.6 LoRaWAN frame types

A LoRaWAN end-device can send and receive frames:

- Uplink: Frame sent by the end-device
- Downlink: Frame received by the end-device

If a collision occurs, or if the gateway is out of coverage area, a frame may not reach the Network Server. It can be important for an end-device to have a more reliable connection, so there are two types of frames.

- Unconfirmed: The Network Server doesn't send an acknowledgment.
- Confirmed: The Network Server sends an acknowledgement to confirm the reception.

There is exactly the same behaviour for the connection between the Network Server and the end-device when using downlink.

- Unconfirmed: The end-device does not send an acknowledgment.
- Confirmed: The end-device sends an acknowledgement.

We can test these different configurations with the Wireless-Logger tool in Actility LoRaWAN server.

4.6.1 Unconfirmed uplink frame

In that case, there is no way to know whether the frame has reached the Network Server. This is however, the most efficient way to communicate in terms of power consumption. In Figure 62, we can see the Join-Request, the Join-Accept, and every uplink frames (FCnt = 1, 2, 3) without acknowledgment.

Last packets

			Local Timestamp	DevAddr	FPort	FCnt ⇑	NFCnt ⇓	RSSI	SNR	ESP	SF/DR
⊞	⇑	data	2021-11-18 00:36:56.788	0493C363	1	3		-12.0	10.5	-12.370...	SF7
⊞	⇑	data	2021-11-18 00:36:39.018	0493C363	1	2		-10.0	9.25	-10.487...	SF7
⊞	⇑	data	2021-11-18 00:35:16.044	0493C363	1	1		-10.0	8.75	-10.543...	SF7
⊞	⇓	join	2021-11-18 00:34:38.339		None						SF12
⊞	⇑	join	2021-11-18 00:34:32.339		None			-11.0	9.75	-11.437...	SF12

Figure 62: Uplink unconfirmed

If we detail an uplink frame we note that in the MAC header, the MType field indicates that we are dealing with an "UnconfirmedDataUp".

Last packets

			Local Timestamp	DevAddr	FPort	FCnt ⇑	NFCnt ⇓	RSSI	SNR	ESP	SF/DR
⊟	⇑	data	2021-11-18 00:35:16.044	0493C363	1	1		-10.0	8.75	-10.543...	SF7

Mtype: UnconfirmedDataUp

Flags: ADR : 0, ADRAckReq : 0, ACK : 0

Figure 63: Unconfirmed uplink frame

4.6.2 Confirmed uplink frame

In that case, every frame is confirmed with a specific downlink message after each uplink. In Figure 64, we can see the Join-Request, the Join-Accept, and every uplink (FCnt = 1, 2) following by the downlink acknowledgment (NFCnt = 0, 1).

Last packets

			Local Timestamp	DevAddr	FPort	FCnt ⇑	NFCnt ⇓	RSSI	SNR	SF/DR
⊞	⇓		2021-11-17 23:51:32.038	0493C363	1		1			SF7
⊞	⇑	data	2021-11-17 23:51:31.038	0493C363	1	2		-12.0	9.75	SF7
⊞	⇓		2021-11-17 23:50:57.081	0493C363	1		0			SF7
⊞	⇑	data	2021-11-17 23:50:56.081	0493C363	1	1		-11.0	9.25	SF7
⊞	⇓	join	2021-11-17 23:50:12.593		None					SF12
⊞	⇑	join	2021-11-17 23:50:06.593		None			-11.0	10.0	SF12

Figure 64: Confirmed uplink frames and acknowledgments

If we detail the uplink frame (FCnt = 2) we note that in the MAC header, the MType field indicates that we are dealing with a "ConfirmedDataUp".

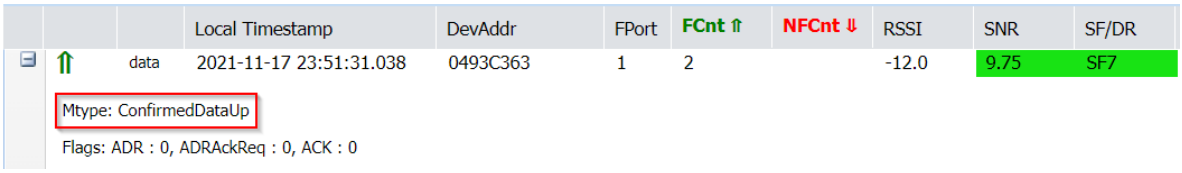


Figure 65: Confirmed uplink frame

We can see that the Network Server sends an ACK in a downlink frame (NFCnt = 1) just after the uplink.

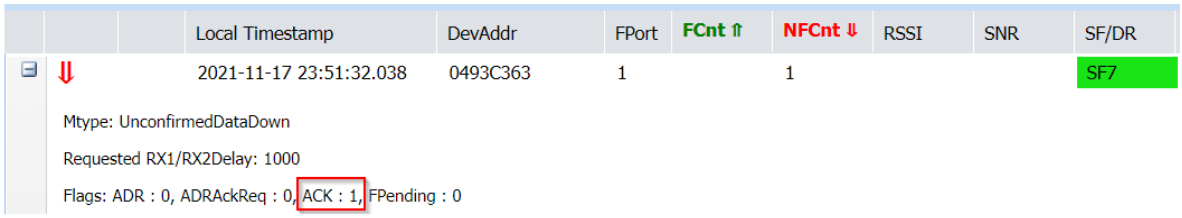


Figure 66: Acknowledgment with the ACK Flag

4.6.3 Unconfirmed downlink frame

In that case, there is no way to know whether the frame has reached the end-device. In Figure 67, we can see the Join-Request, the Join-Accept and one unconfirmed uplink (FCnt = 1). This is at this moment that the server scheduled a new unconfirmed downlink. But in class A, a downlink can only be sent after an uplink. So the Network Server waits to receive a new uplink (FCnt = 2) to send the downlink frame (NFCnt = 0).

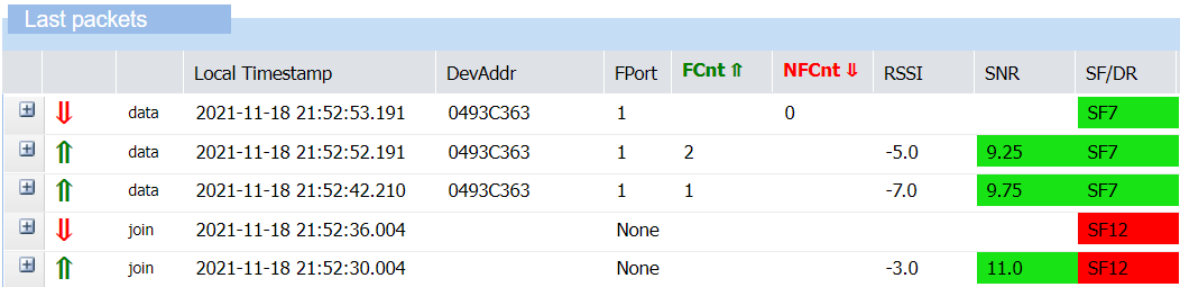


Figure 67: Unconfirmed downlink frame

If we detail the downlink frame (NFCnt = 0), we note that in the MAC header, the MType field indicates that we are dealing with an "UnconfirmedDataDown".

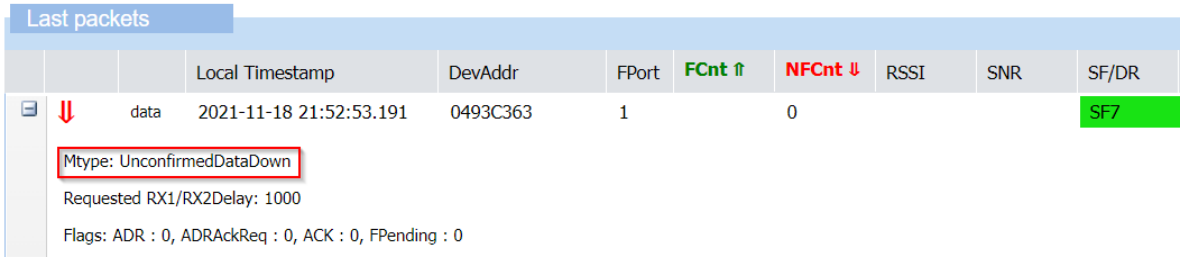
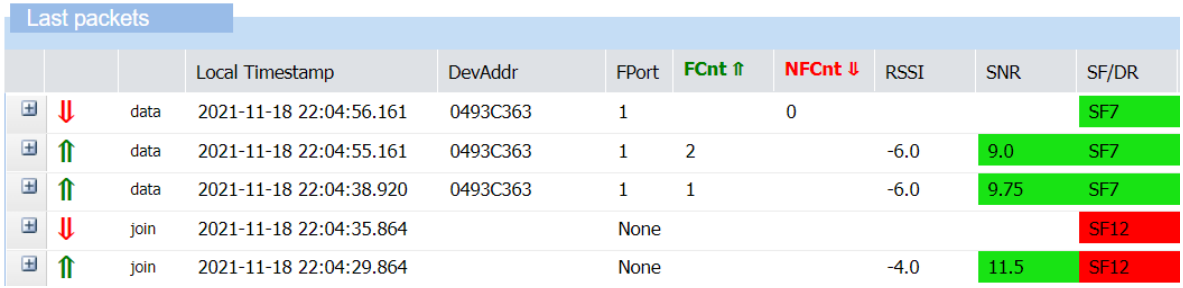


Figure 68: Unconfirmed downlink frame

When the server has several frames to transmit to the Device, it can take a long time because it has to wait for the Device to transmit before (class A behaviour). In order to speed up this process, it is possible to tell the end-device that other frames are waiting on the server by using a specific Flag (Fpending). It will be up to the end-device to decide if it wishes to speed up the transmission of these frames (by transmitting more uplink frames).

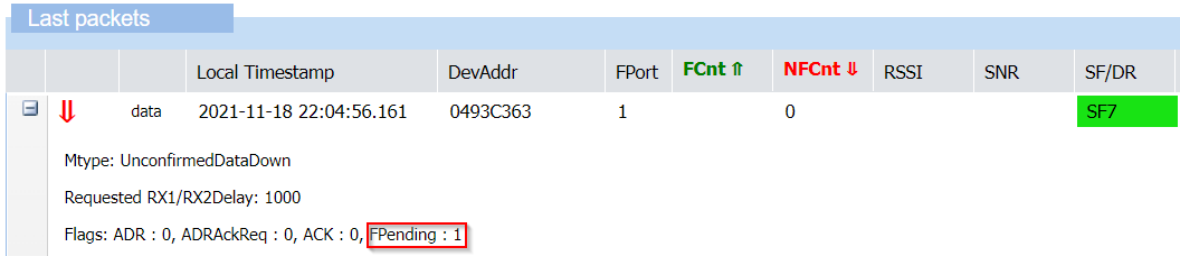
To highlight this behaviour, in the Figure 69, we launch the same scenario as before: Join-Request, Join-Accept, then one uplink unconfirmed (FCnt = 1). Then we schedule several downlinks that are queued on the Network Server. Then, we wait for a new uplink (FCnt = 2) to send the downlink (NFCnt = 0).


Last packets

			Local Timestamp	DevAddr	FPort	FCnt ⇑	NFCnt ⇓	RSSI	SNR	SF/DR
⊞	⇓	data	2021-11-18 22:04:56.161	0493C363	1		0			SF7
⊞	⇑	data	2021-11-18 22:04:55.161	0493C363	1	2		-6.0	9.0	SF7
⊞	⇑	data	2021-11-18 22:04:38.920	0493C363	1	1		-6.0	9.75	SF7
⊞	⇓	join	2021-11-18 22:04:35.864		None					SF12
⊞	⇑	join	2021-11-18 22:04:29.864		None			-4.0	11.5	SF12

Figure 69: Frame pending scenario

Downlink can be sent only one at a time, so when it is sent (NFcnt = 0) there is still the second message queued it the Network Server. We can see the FPending flag set.


Last packets

			Local Timestamp	DevAddr	FPort	FCnt ⇑	NFCnt ⇓	RSSI	SNR	SF/DR
⊟	⇓	data	2021-11-18 22:04:56.161	0493C363	1		0			SF7

Mtype: UnconfirmedDataDown

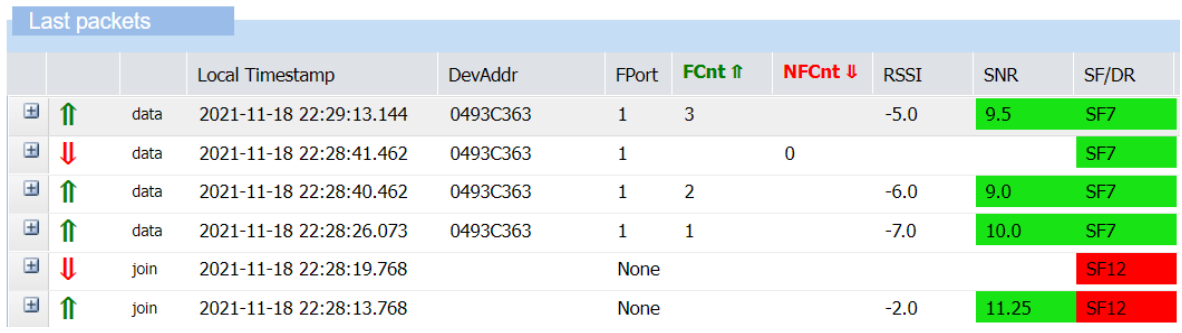
Requested RX1/RX2Delay: 1000

Flags: ADR : 0, ADRAckReq : 0, ACK : 0, FPending : 1

Figure 70: The FPending flag

4.6.4 Confirmed downlink frame

In that case, every downlink frame is confirmed. The acknowledgment is made in the next uplink frame with the ACK bit set. In Figure 71, we can see the Join-Request, the Join-Accept, and one uplink (FCnt = 1). Then a downlink confirmed is scheduled. When the next uplink arrives (FCnt = 2) the downlink confirmed is sent (NFCnt = 0). The next uplink (FCnt = 3) carries the acknowledgment.


Last packets

			Local Timestamp	DevAddr	FPort	FCnt ⇑	NFCnt ⇓	RSSI	SNR	SF/DR
⊞	⇑	data	2021-11-18 22:29:13.144	0493C363	1	3		-5.0	9.5	SF7
⊞	⇓	data	2021-11-18 22:28:41.462	0493C363	1		0			SF7
⊞	⇑	data	2021-11-18 22:28:40.462	0493C363	1	2		-6.0	9.0	SF7
⊞	⇑	data	2021-11-18 22:28:26.073	0493C363	1	1		-7.0	10.0	SF7
⊞	⇓	join	2021-11-18 22:28:19.768		None					SF12
⊞	⇑	join	2021-11-18 22:28:13.768		None			-2.0	11.25	SF12

Figure 71: Confirmed downlink

If we detail the downlink frame (NFCnt = 0), we note that in the MAC header, the MType field indicates that we are dealing with a "ConfirmedDataDown".

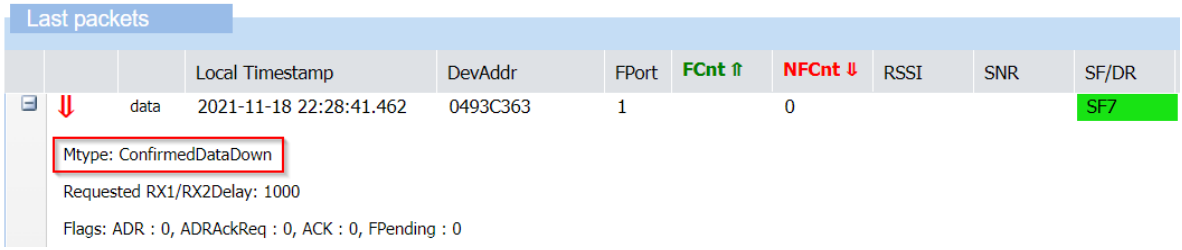


Figure 72: Confirmed downlink frame

We also see that the following uplink frame (FCnt = 3) has its ACK flag set to 1.

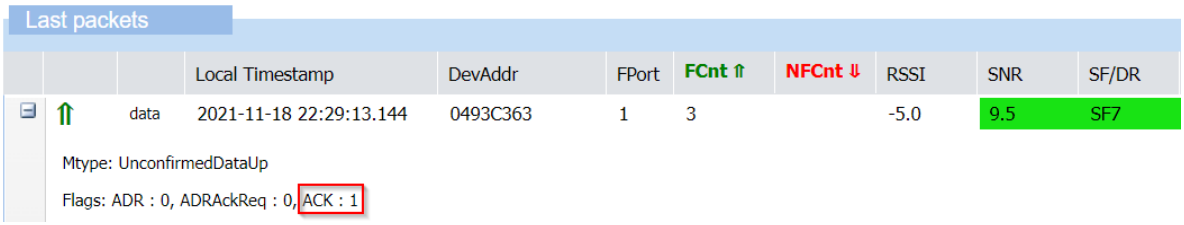


Figure 73: Acknowledgment with the ACK flag

4.6.5 Confirmed uplink and confirmed downlink

We can have confirmed uplink and downlink at the same time. Here is the scenario:

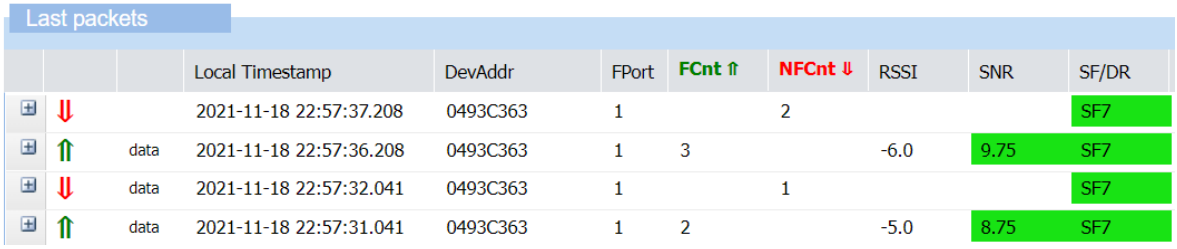


Figure 74: Confirmed uplink and confirmed downlink

- FCnt = 2 ConfirmedDataUp Uplink Data
- NFCnt = 1 ConfirmedDataDown Downlink Data + Flag ACK to confirm FCnt =2
- FCnt = 3 ConfirmedDataUp Uplink Data + Flag ACK to confirm NFCnt =1
- NFCnt = 2 ConfirmedDataDown No Data + Flag ACK to confirm FCnt =3

Even if the LoRaWAN network provides such possibilities, we must be aware that the downlink capabilities of these networks are limited. A LoRaWAN communication should limit the downlink (acknowledgments of messages) as much as possible.

4.7 MAC Commands

For network administration, a set of MAC commands may be exchanged between the Network Server and the end-device. On the end-device and Network Server, theses commands belongs to the LoRaWAN stack and should never reach the user application:

- The Application Server should never receive these commands.

- The user application in the end-device should not be aware of these commands.

Each MAC Command has a specific identifier called CID (**C**ommand **ID**entifier).

CID	Command	Transmitted by		Description
		Device	Network Server	
0x02	**LinkCheckReq**	X		Used by an end-device to validate its connectivity to a network
0x02	**LinkCheckAns**		X	Answers **LinkCheckReq**. Contains the received signal power estimation, which indicates the quality of reception (link margin) to the end-device.
0x03	**LinkADRReq**		X	Requests the end-device to change data rate, TX power, redundancy, or channel mask.
0x03	**LinkADRAns**	X		Acknowledges **LinkADRReq**
0x04	**DutyCycleReq**		X	Sets the maximum aggregated transmit duty cycle of an end-device.
0x04	**DutyCycleAns**	X		Acknowledges **DutyCycleReq**.
0x05	**RXParamSetupReq**		X	Sets the reception slot parameters.
0x05	**RXParamSetupAns**	X		Acknowledges **RXParamSetupReq.**
0x06	**DevStatusReq**		X	Requests the status of the end-device.
0x06	**DevStatusAns**	X		Returns the status of the end-device, namely its battery level and its radio status.
0x07	**NewChannelReq**		X	Creates or modifies the definition of a radio channel.
0x07	**NewChannelAns**	X		Acknowledges **NewChannelReq**.
0x08	**RXTimingSetupReq**		X	Sets the timing of the reception slots.
0x08	**RXTimingSetupAns**	X		Acknowledges **RXTimingSetupReq**.
0x09	**TXParamSetupReq**		X	Used by a Network Server to set the maximum allowed dwell time and MaxEIRP of end-device, based on local regulations.
0x09	**TXParamSetupAns**	X		Acknowledges **TXParamSetupReq**.
0x0A	**DlChannelReq**		X	Modifies the definition of a downlink RX1 radio channel by shifting the downlink frequency from the uplink frequencies (i.e. creating an asymmetric channel).
0x0A	**DlChannelAns**	X		Acknowledges **DlChannelReq**.
0x0D	**DeviceTimeReq**	X		Used by an end-device to request the current GPS time.
0x0D	**DeviceTimeAns**		X	Answers **DeviceTimeReq**

Table 15: LoRaWAN MAC Commands (source LoRaWAN 1.0.4 spec – LoRa Alliance)

4.8 Data Rate, channels and power

4.8.1 Data Rate (DR)

As we discussed earlier, the Spreading Factor (SF) and the bandwidth (BW) are the two parameters that have an impact on the bit rate.

$$Bit\ Rate\ (bit/s) = SF.\frac{Bandwidth}{2^{SF}}$$

The combination of SF and BW is named DR (**D**ata **R**ate). In the EU868 band, it is normalized from DR0 to DR6.

Data Rate	Spreading Factor	Bandwidth
DR 0	SF12	125 KHz
DR 1	SF11	125 KHz
DR 2	SF10	125 KHz
DR 3	SF9	125 KHz
DR 4	SF8	125 KHz
DR 5	SF7	125 KHz
DR 6	SF7	250 KHz

Table 16: Data Rate(DR) according to SF and bandwidth

4.8.2 Adaptive Data Rate (ADR)

Adjusting the SF and the P_T (**P**ower **T**ransmitted) is not straightforward. Even if we find a good configuration, the transmission can be altered by the local environment or the weather forecast. To overcome this difficulty, an automatic adjustment method has been implemented by the LoRaWAN protocol: it is the **A**daptive **D**ata **R**ate (**ADR**). The idea is to let the Network Server computing the best combination of SF / P_T.

To perform this operation, the Network Server checks:

- The RSSI (power received on the gateway): The RSSI is compared to the sensitivity of the receiver. The RSSI must be higher than the sensitivity after applying a margin.
- The SNR of the transmission: The SNR of the transmission must be higher than the minimum SNR accepted by the receiver after applying a margin.
- The packet loss estimation: By checking the Frame Counter, the Network Server can estimated the number of lost packets during the last transmissions.

The Network Server computes its result after averaging RSSI and SNR on several uplink frames. The ADR algorithm is not defined by the LoRaWAN specification. Each Network Server can use its own strategy. Basically, we use the following scheme:

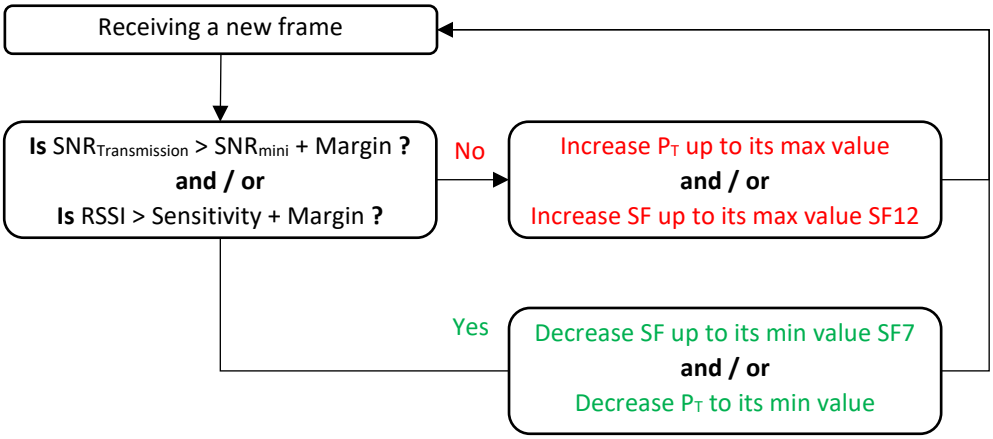


Figure 75: Example of an ADR algorithm

A LoRaWAN Device must enable the ADR mode (ADR Flag) to use this feature.

ADR commands are part of the LoRaWAN MAC Commands (see chapter 4.7).

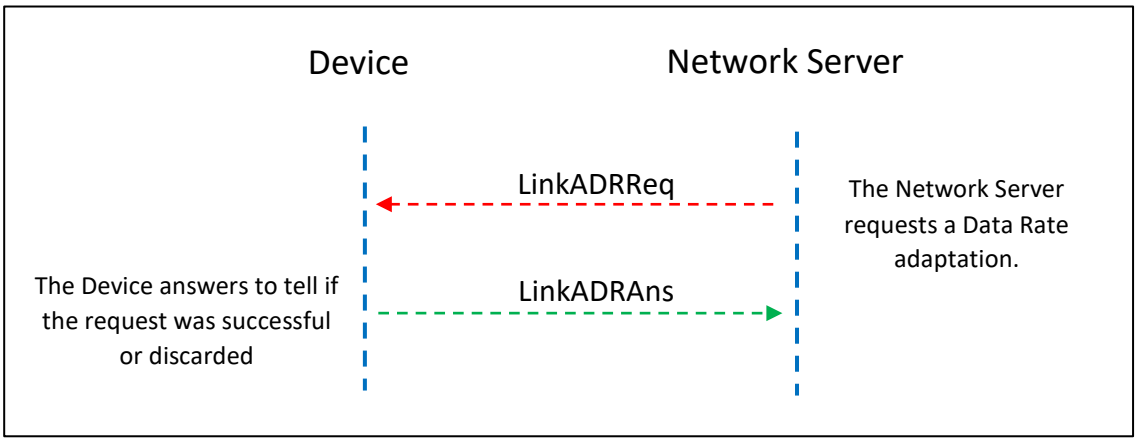


Figure 76: ADR MAC Commands

LinkADRReq MAC Command is not sent in a separate frame. It is piggybacked in a downlink data message.

Piggybacking is a way to take advantage of a data transfer to include a command, acknowledgment or any network administration.

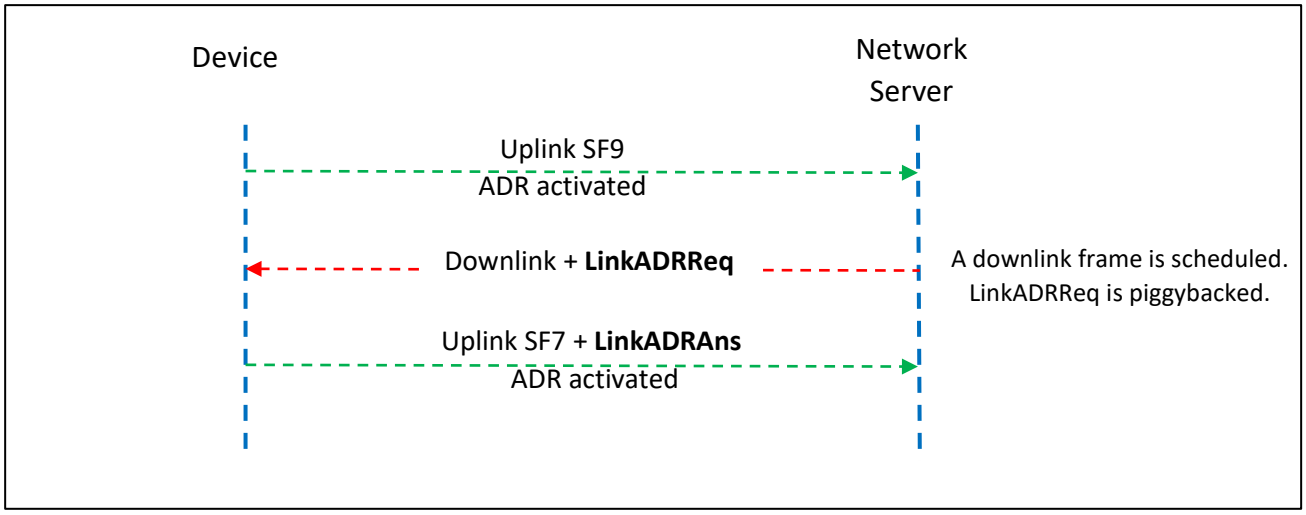


Figure 77: ADR MAC Command

With Wireless-Logger in Actility LoRaWAN server, we have the following scenario: an uplink confirmed is sent with SF9 and ADR enabled (FCnt = 5). The Network Server uses the acknowledgment to send the **LinkADRReq** MAC command (NFCnt = 7). The next uplink sends the **LinkADRAns** MAC command to confirm the optimization process (FCnt = 6).

Last packets

		Local Timestamp	FPort	SF/DR	FCnt ⇑	NFCnt ⇓
⊞	⇑ mac data	2021-11-28 18:46:20.008	1	SF7	6	
⊞	⇓ mac	2021-11-28 18:45:59.743	0	SF9		7
⊞	⇑ data	2021-11-28 18:45:58.743	1	SF9	5	

Figure 78: Optimization of the Spreading Factor

The uplink confirmed has ADR enabled (FCnt = 5).

Last packets

		Local Timestamp	FPort	SF/DR	FCnt ⇑	NFCnt ⇓
⊟	⇑ data	2021-11-28 18:45:58.743	1	SF9	5	
	Mtype: ConfirmedDataUp					
	Flags: ADR : 1, ADRAckReq : 0, ACK : 0					

Figure 79: ADR Flag

The Network Server uses the confirmation to send the **LinkADRReq** MAC command (NFCnt = 7). A new power transmission and Data Rate (DR5 = SF7) is proposed.

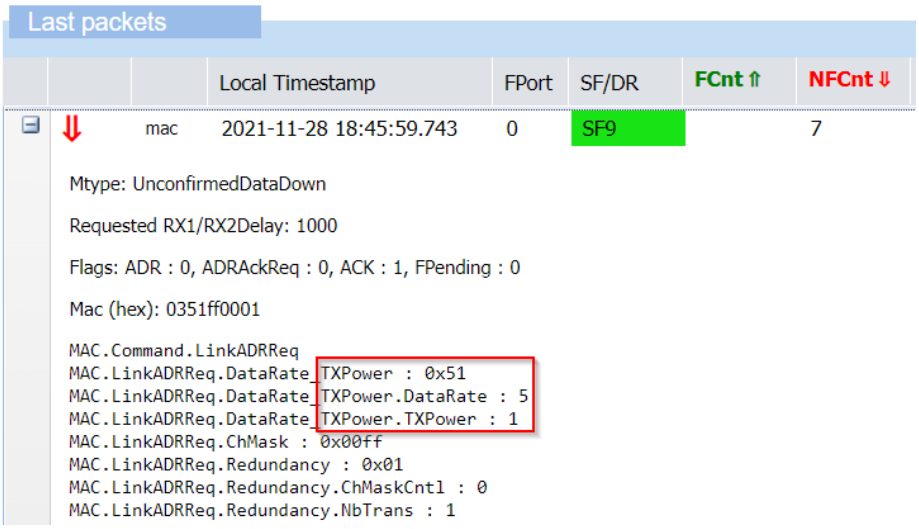


Figure 80: LinkADRReq MAC Command

The next uplink sends the **LinkADRAns** MAC Command to confirm the optimization process (FCnt = 6). The Spreading Factor is now SF7.

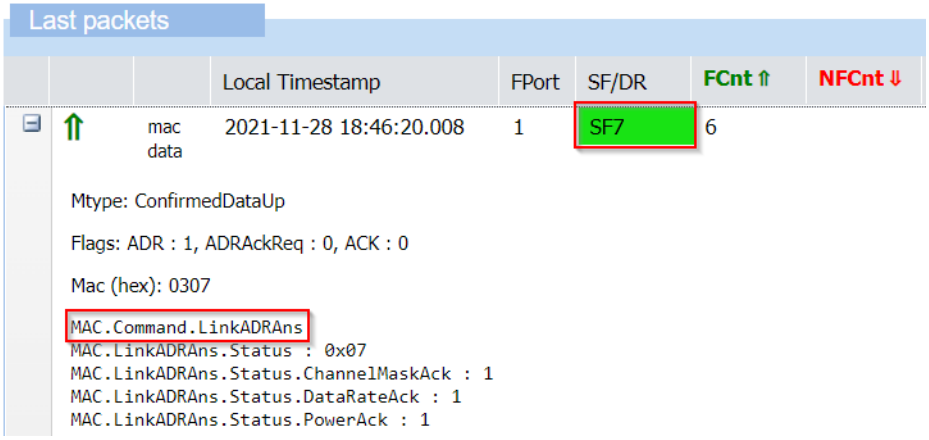


Figure 81: LinkADRAns Command

A problem occurs when there is no downlink frame available to piggyback the **LinkADRReq** command. In that case the optimization process never occurs. To prevent that situation, when the end-device does not hear from the Network Server for a long time, then it will explicitly ask for an optimization. This is done thanks to the **ADRAckReq** bit.

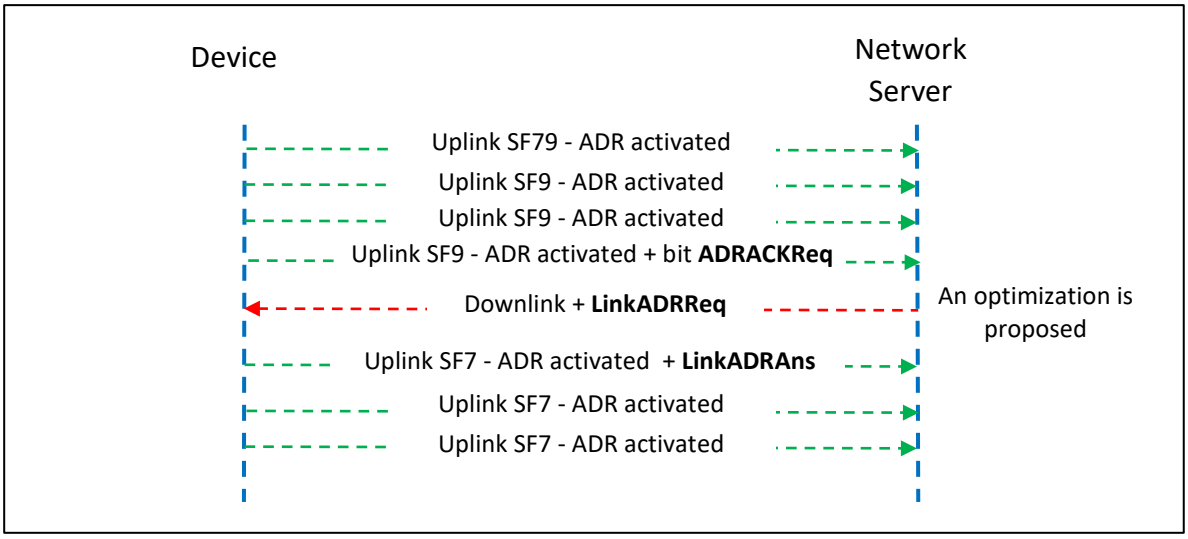


Figure 82: ADRAckReq bit to request Data Rate optimization

In the following scenario, we send only unconfirmed uplink. The uplink frame (FCnt = 77) has the **ADRAckReq** bit raised, and we can see that the Network Server answers. In that case, there was no optimization to be done because we were already with the best SF7 and P_T.

Last packets

			Local Timestamp	FPort	SF/DR	FCnt ⇑	NFCnt ⇓
⊞	⇑	data	2021-11-28 22:40:57.470	1	SF7	80	
⊞	⇑	data	2021-11-28 22:40:47.449	1	SF7	79	
⊞	⇓		2021-11-28 22:40:44.294	1	SF7		2
⊞	⇑	data	2021-11-28 22:40:43.294	1	SF7	77	
⊞	⇑	data	2021-11-28 22:40:33.283	1	SF7	76	

Figure 83: ADRAckReq bit

If the end-device does not obtain a response within a given time, then the communication with the Network Server is considered lost and the LoRaWAN end-device will increment SF / P_T until it can reach it again.

You can check the full demonstration in video available on our website: www.univ-smb.fr/lorawan

The ADR algorithm can also provide a number of retransmission, so each unconfirmed uplink will be sent **NbTrans** times. The end-device gets the information of **NbTrans** in the **LinkADRReq** frame as you can see in Figure 80.

4.8.3 Channels

LoRa uses different frequency bands in different parts of the world. In Europe, the band used is 868 MHz [From 863 MHz to 870 MHz]. Among this band, the LoRaWAN server defines a plan with a number of channels and Data Rate to be used for uplink and downlink. An end-device **must** know at least these three channels: 868.1 MHz, 868.3 MHz and 868.5 MHz from DR0 to DR5. The other channels depend on the Network Server. Table 17 represents the mandatory channels.

	Channels	Data Rate	Direction
Mandatory	868.1 MHz	DR0 to DR5	Uplink / Downlink
	868.3 MHz	DR0 to DR5	Uplink / Downlink
	868.5 MHz	DR0 to DR5	Uplink / Downlink

Table 17: Mandatory LoRaWAN frequency plan

On top of this mandatory frequency plan, the LoRaWAN server propose other channels and DR to the end-device during the Join procedure (OTAA). When using ABP, the end-device must stick to the mandatory frequency plan, or the user must store them manually in the firmware.

	Channels	Data Rate	Direction
Default EU868 frequency plan	867.1 MHz	DR0 to DR5	Uplink / Downlink
	867.3 MHz	DR0 to DR5	Uplink / Downlink
	867.5 MHz	DR0 to DR5	Uplink / Downlink
	867.7 MHz	DR0 to DR5	Uplink / Downlink
	867.9 MHz	DR0 to DR5	Uplink / Downlink
	869.525 MHz	DR3	Downlink

Table 18: Default EU868 frequency plan

	Channels	Data Rate	Direction
Other frequency plan (TTN, Actility…)	867.1 MHz	DR0 to DR5	Uplink / Downlink
	867.3 MHz	DR0 to DR5	Uplink / Downlink
	867.5 MHz	DR0 to DR5	Uplink / Downlink
	867.7 MHz	DR0 to DR5	Uplink / Downlink
	867.9 MHz	DR0 to DR5	Uplink / Downlink
	869.525 MHz	DR0	Downlink

Table 19: Other EU868 frequency plan

It is also necessary for an operator to share as many channels as possible with other operators to optimize the probability of successfully transmission in case of a Roaming (another operator will carry out the reception). The LoRa Alliance has therefore recommended the following channels in case of roaming:

	Canaux	Data Rate	Direction
Roaming Frequency plan	867,1 MHz	DR0 à DR5	Uplink / Downlink
	867,3 MHz	DR0 à DR5	Uplink / Downlink
	867,9 MHz	DR0 à DR5	Uplink/ Downlink

Table 20: Roaming frequency plan

A LoRaWAN gateway must be properly configured with Network Server frequency plan to which it is connected. The end-device configuration is carried out during the Join-Accept.

4.8.4 Power consumed by the end-device

The power consumption of a LoRaWAN end-device is directly related to two parameters of the LoRa transmission:

- The Time on Air: the longer the message, the longer the radio will be powered.
- Power transmitted P_T: The more power used for transmitting, the more power is consumed by the end-device.

Obviously, the first tip is to reduce the power transmission. The direct consequence is that the end-device might not reach gateways anymore. Reducing the power transmission can be done only if we have a margin large enough between the power received by the GW and the gateway sensitivity.

The second tips is to reduce the Time on Air. A simple solution is to reduce the SF (Spreading Factor). Indeed, when the SF is reduced by 1, the Time on Air is divided by two (see chapter 3.1.2). But reducing the SF has another consequence on the gateway side: it drastically reduces its sensitivity and its ability to detect signal among noise.

As an example, Table 20 lists several transmissions with different SF used on a transmitter. On the other hand, we calculate on the receiver for each transmission the best sensitivity we can reach, as well as the worst acceptable SNR.

Transmitter	**Receiver**	
Spreading Factor	**Sensitivity**	**Minimum SNR**
7	-123 dBm	7.5 dB
8	-126 dBm	10 dB
9	-129 dBm	12.5 dB
10	-132 dBm	15 dB
11	-134.5 dBm	17.5 dB
12	-137 dBm	20 dB

Table 21: SF, Sensitivity and SNR

Note: Table 20 is a compilation of data from the SX1261 documentation (SNR) and calculations performed by the LoRa Calculator (Sensitivity).

So reducing the SF and the P_T will both result in a lower transmission range. They have to be adjusted consistently. The choice of SF and P_T is not simple, and we often use an empirical method to determine their values by checking the SNR and RSSI on the gateway. Otherwise, we should use ADR.

5 LoRaWAN networks and LoRaWAN servers

5.1 The different types of networks

We have the choice either to set up the entire network infrastructure, or to rely on an operator. There are three possibilities for the LoRaWAN architecture.

- We can use public operators that have nation-wide operational networks.
- We can build our own private LoRaWAN network.
- We can build a hybrid network by setting up only one part of the infrastructure.

5.1.1 Public operator LoRaWAN networks

Public operators propose their LoRaWAN nation-wide network to connect IoT end-devices. Objenious (a Bouygues subsidiary), Orange; KPN, Proximus are a few examples. They usually have an excellent coverage. When using a public operator, the user just needs to take care of his LoRaWAN end-devices and the user application (IoT Platform). Public operators manage the gateways and the LoRaWAN server.

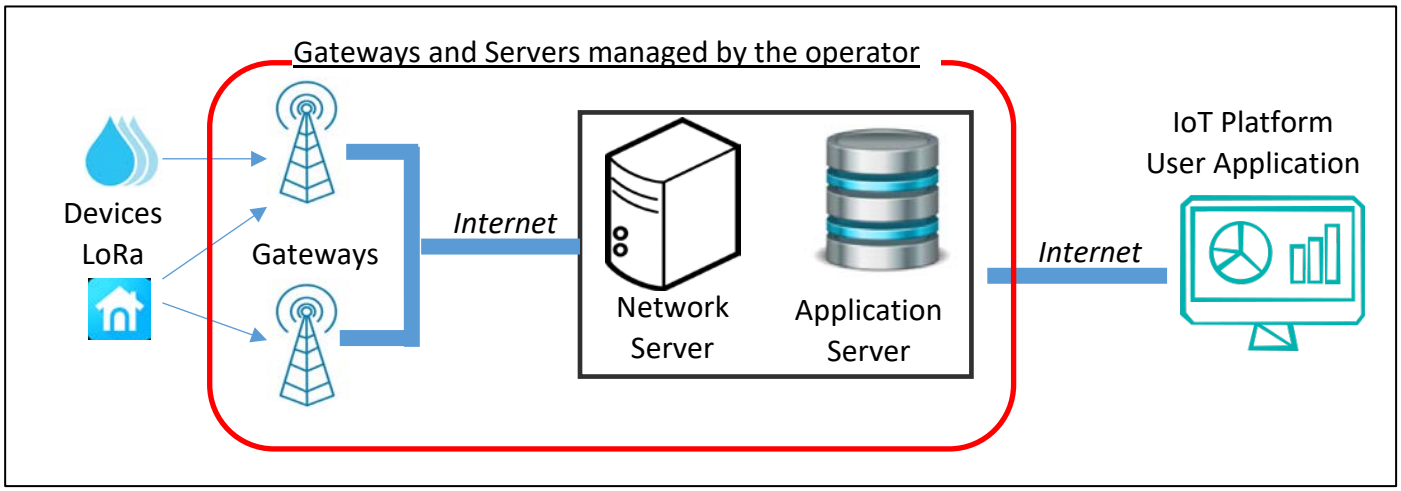


Figure 84: Infrastructure of an public LoRaWAN network

The user subscribes to one or more plans to be able to connect his devices. As an example, here are the subscriptions proposed by Objenious and Orange in 2022 to have access to their LoRaWAN network:

Orange:

- Unlimited uplink (with respect of the duty cycle).
- Price of each uplink message is 5 cts.
- Subscription varies from 1€/month (36 months) to 2€/month (without commitment)

Bouygues Objenious:

- 144 uplink messages per day.
- 6 downlink messages per day.
- Subscription is 20 € / year

Objenious has announced the end of his LoRaWAN service at the end of 2024.

The user journey to connect LoRaWAN end-devices is simple and seamless:

1 Buy a subscription
2 Declare the end-devices on the operator platform ("Live Object" for Orange, "Spot" for Objenious)
3 Activate the end-devices
4 Retrieve the data on the operator platform
5 Redirect your data to an IoT platform.

In 2022, there are 165 LoRaWAN network operators in 171 countries.

5.1.2 Private LoRaWAN networks

Everyone is free to create his own private network. You will need to set up your own gateway and your own server infrastructure to communicate with your LoRaWAN end-devices. You will also have to take care of the administration of the Network Server and Application Server.

In some gateways, an instance of a LoRaWAN server is proposed. It simplifies the overall infrastructure because you have everything in one package (the gateway) but that will considerably limit the potential of your LoRaWAN network. In Figure 85, we can see that the gateway includes the Network Server and Application Server (green solid line). The IoT Platform is sometimes also included is the gateway (dashed green line).

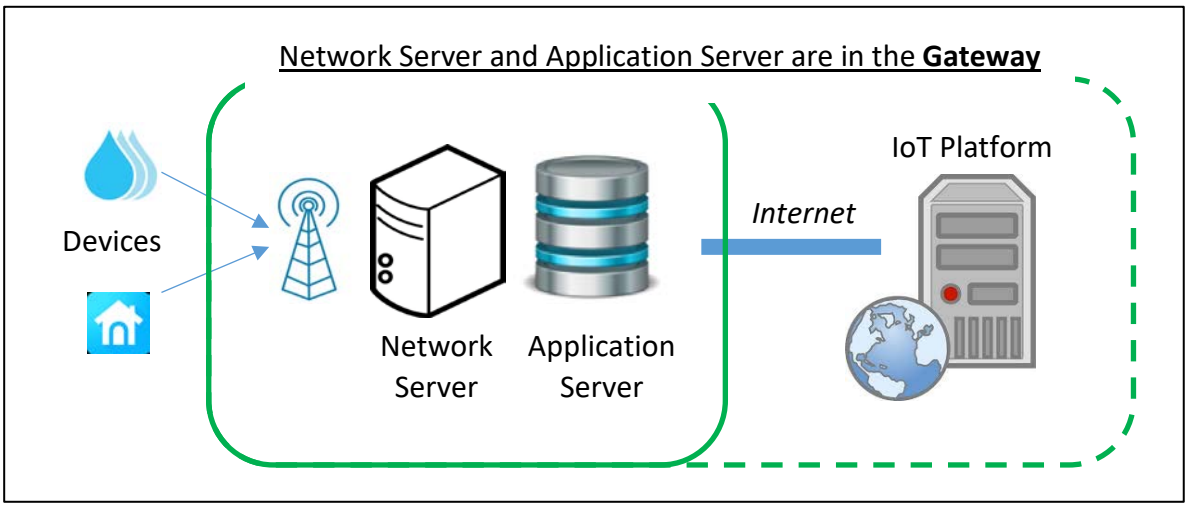


Figure 85: Private Network packaged in a single gateway

Here is an example of a private LoRaWAN Network packaged is a gateway:

- Kerlink: Wanesy SPN (**S**mall **P**rivate **N**etwork)

Unless you develop your own LoRaWAN server (which is a huge work), you will have to buy this software. The cost depends on the features, the service and the number of gateway/end-devices you want to register. Once you have bought the licence, you will be allowed to install the LoRaWAN server on your own infrastructure. This is called an "on premises licence". Here are a few of them:

- Actility: Actility on customer premises
- Kerlink: Wanesy WMC on premises (**W**anesy **M**anagement **C**enter)
- The Things Industries: The Things Stack Enterprise (on premises)
- ResIOT: ResIOT Network Server (on premises)

Another possibility to set up a private network is to use a free and open source LoRaWAN server. The two well-known open source stacks are:

- The Things Network: The Things Stack.
- Chirpstack: ChirpStack open-source LoRaWAN®

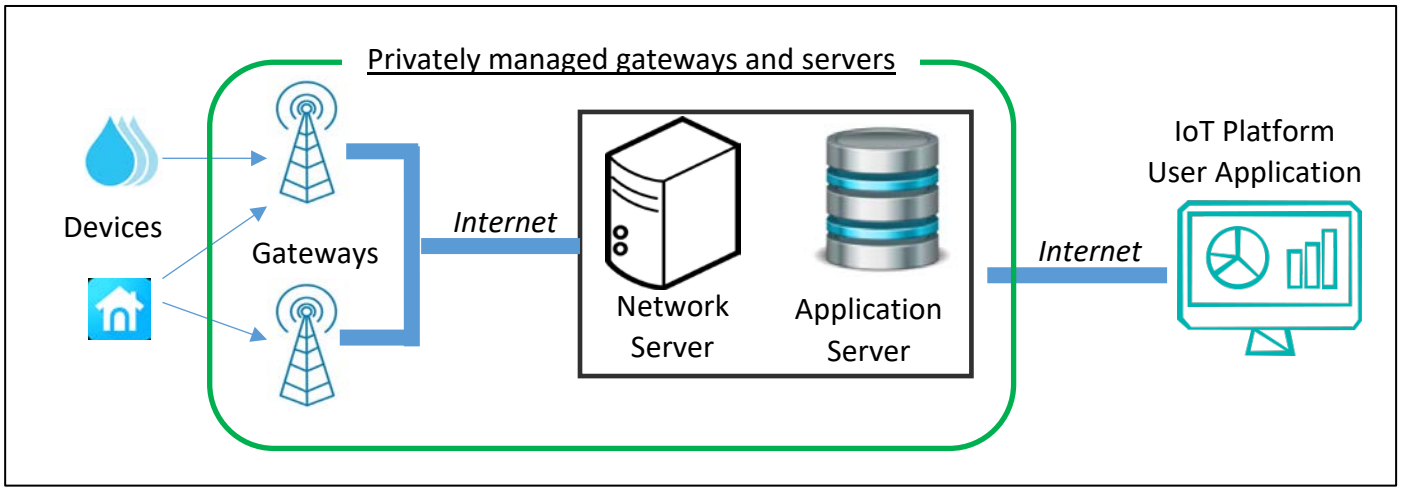


Figure 86: Infrastructure of a private LoRaWAN network

The user journey to connect LoRaWAN end-devices in a LoRaWAN private network requires skills:

1 Buy one or more gateways.
2 Deploy them on site.
3 Buy a LoRaWAN server (or use a free one).
4 Install the LoRaWAN server on your own infrastructure.
5 Register the end-devices on your LoRaWAN server.
6 Activate the end-devices
7 Retrieve the data on your server.
8 Redirect your data to an IoT platform.

5.1.3 Choice of network type: operated or private?

We can summarize the advantages and disadvantages of each of these network types in Table 21.

	Private network	Operated network
Subscription cost	No subscription	Approximately 1.5 € / month per LoRaWAN end-device
Infrastructure costs	Important investment at the beginning (gateways and Servers)	Included in the subscription
Skills required	Requires skills for installation, administration and maintenance.	Everything is managed by the operator
Coverage	Optimized according to needs	Depends on the chosen operator Possibility of roaming between operators
Uplink	Unlimited within the duty-cycle	Limited according to the subscription
Downlink	Unlimited within the duty-cycle	Limited in number or pay as you go

Table 22: Pros and cons of private and public operated network

5.1.4 An alternative, the hybrid LoRaWAN network

In case none of the previous solution is suitable, there is an alternate solution which is an intermediate between public and private network. It has the advantage of managing the LoRa network coverage using its own gateways, while entrusting the LoRaWAN server infrastructure to a service provider in order to limit investments and maintenance.

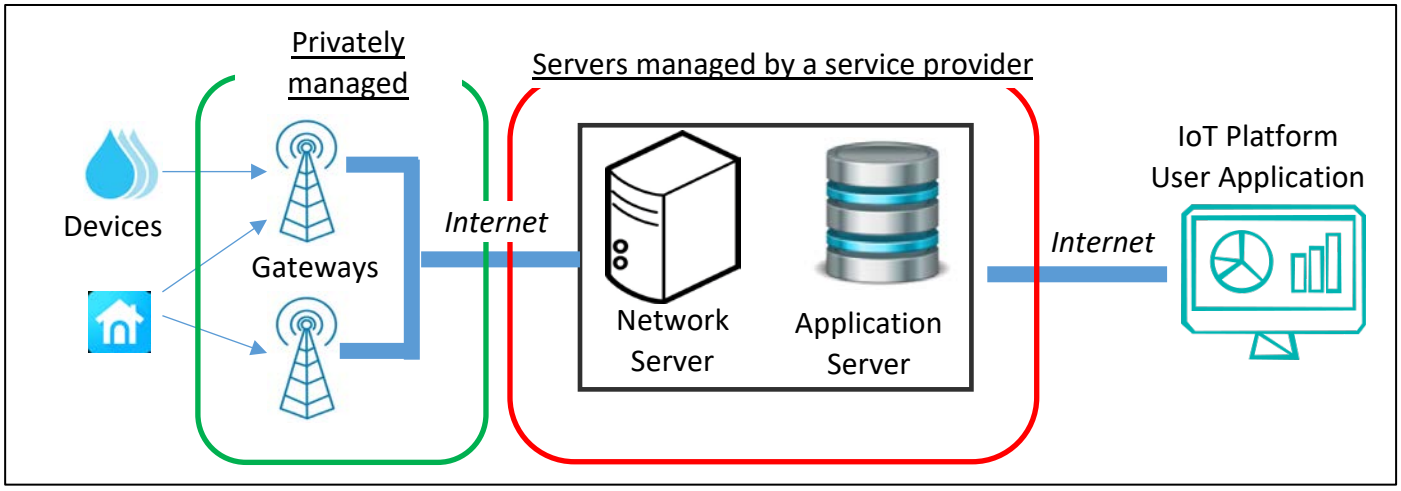


Figure 87: Infrastructure of a hybrid LoRaWAN network

You will have to buy a cloud hosted solution for the LoRaWAN server. Here are a few examples:

- Actility: ThingPark Entreprise
- Kerlink: Wanesy Management as a Service
- The Things Industries: The Things Stack Enterprise (Cloud)
- Loriot: Loriot Network Server (Cloud)
- ResIOT: ResIOT Network Server (Cloud)

On each cloud hosted LoRaWAN server, there is very often a free entry with limited gateways and end-devices.

- Actility: ThingPark Community
- The Things Network: The Things Stack community Edition
- And many others …

The user journey to connect LoRaWAN end-devices in a LoRaWAN hybrid network is the following:

1 Buy one or more gateways.
2 Deploy them on site.
3 Subscribe to a cloud hosted LoRaWAN server.
4 Declare the end-devices on your LoRaWAN server.
5 Activate the end-devices
6 Retrieve the data on your server.
7 Redirect your data to an IoT platform.

In this course, most of the time, we will use a hybrid network. Indeed, we use our own gateways that we connect to a cloud hosted LoRaWAN server (Actility, TTN, Loriot...).

5.1.5 Coverage areas

The coverage areas of the public operated network are updated on the operator's websites. For community network, they use specific application such as TTN Mapper for The Things Stack community Edition. The idea is to associate the LoRaWAN end-device with a GPS in the coverage area of the gateways. For each received frame, TTN Mapper register the end-device coordinate and display it on a map.

5.2 LoRaWAN Network configuration

For this chapter, whatever the chosen solution for your network, the infrastructure must be operational:

- If you have a **public operated network**, there is nothing to do.
- If you have a **hybrid network**, your gateways must be connected to internet and you must have subscribed to a cloud hosted LoRaWAN server.
- If you have a **private network**, your gateways must be connected to internet and you must have set up the LoRaWAN server by yourself. The set up of a LoRaWAN server is described in chapter 9.

The configuration is always the same:

- Step 1: Gateway configuration.
- Step 2: Gateway registration on the LoRaWAN server.
- Step 3: Device registration on the LoRaWAN server.
- Step 4: Device configuration.

5.2.1 Step 1: Gateway configuration

There are many brands of LoRaWAN gateway. Each model is intended for a particular use (Indoor, outdoor, prototyping,...). In all cases, here are the elements we need to configure:

1. The type of **Packet Forwarder**.
2. The **IP address** of the Network Server.
3. The **channel list** and associated **Data Rate**.

The **Packet Forwarder** is one part of the internal software that will specify the protocol used to communicate with the Network Server. There are many Packet Forwarder available. The most famous is probably the "Semtech UDP Packet Forwarder", but that is also probably the less secured and it does not have many functionalities. "Semtech UDP Packet Forwarder" is now deprecated and should not be used anymore for deployment. Some Network Server are compatible with several Packet Forwarder, but other requires a specific one and does not give you the choice.

The gateway receives LoRaWAN data and transfers them to the Network Server. The gateway must know the Network Server IP address as well as the TCP/UDP ports to use. You can find in the appendices of this book, the IP address and ports of some major Network Server.

The gateway receives a LoRa modulation signal. It listens only on a specific list of **channels** and **Data Rate**. This information has to be specified in the gateway.

```
CHANNEL0: 867100000, A, SF7/SF12, BW125KHz     (LORA_MULTI_SF)
CHANNEL1: 867300000, A, SF7/SF12, BW125KHz     (LORA_MULTI_SF)
CHANNEL2: 867500000, A, SF7/SF12, BW125KHz     (LORA_MULTI_SF)
CHANNEL3: 867700000, A, SF7/SF12, BW125KHz     (LORA_MULTI_SF)
CHANNEL4: 867900000, A, SF7/SF12, BW125KHz     (LORA_MULTI_SF)
CHANNEL5: 868100000, B, SF7/SF12, BW125KHz     (LORA_MULTI_SF)
CHANNEL6: 868300000, B, SF7/SF12, BW125KHz     (LORA_MULTI_SF)
CHANNEL7: 868500000, B, SF7/SF12, BW125KHz     (LORA_MULTI_SF)
CHANNEL8: 868300000, B, SF7, BW250KHz          (LORA_STANDARD)
```

Figure 88: List of channel and Data Rate for TTNv3

If the Packet Forwarder is provided by your LoRaWAN server provider, it already contains the IP address and port. There is also a big chance that the information of the channel list and Data Rate would be exchanged when the gateway connects on the Network Server. So the only thing to do in that case, it is to install the Packet Forwarder on the gateway.

For example, if you use Actility as a Network Server than you will need to use **LRR** (**L**ong **R**ange **R**elay Packet Forwarder). When you register your gateway (here a Kerlink iBTS compact), Actility propose to download the gateway image with the LRR already set up (see Figure 89)

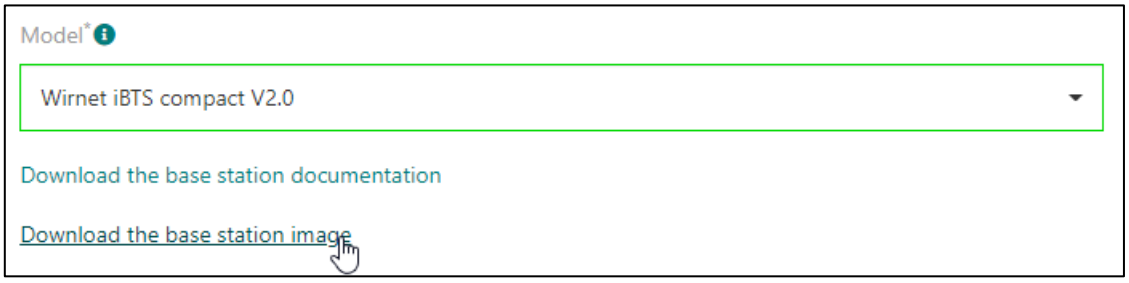


Figure 89: Registration of a gateway in Actility

5.2.2 Step 2: Gateway registration

The purpose of the gateway registration is to allow the gateway to send data to the Network Server. This is a very straightforward task. You need to enter:

- The Name

- The ID
- The region (frequency band)
- The location (optional)

Then the gateway appears on a list when you usually can check the traffic for debugging.

5.2.3 Step 3: Device registration

All LoRaWAN end-device has a DevEUI. This is a unique identifier that is often stored in the firmware and cannot be easily changed. Whatever the activation mode you use (ABP or OTAA), you need the DevEUI. Even if the ABP activation does not really need it, the DevEUI is useful for the Network Server to identify the end-devices registered.

Devices that are part of a specific use case can be grouped in a specific entity usually called application. This allows the network administrator to apply specific settings to a fleet of end-devices. For example:

- Add a script to decode the received payload in JSON format.
- Add a connexion to push the received data to a specific IoT platform.

The registration of an end-device needs several information:

- Name: A human-readable identifier
- LoRaWAN version: From 1.0.0 to the last available version (see chapter 4.1.2)
- Device class: A, B or C.
- Regional parameters version: see chapter 4.1.3
- Frequency plan: the frequency band you are working with (EU868, US915,…)
- Activation mode: ABP or OTAA

If you do not know the exact LoRaWAN version of your end-device and you just try to set up a basic test, you can use safely use the LoRaWAN version 1.0.0.

Then, depending on the chosen activation mode, you will need the following information:

For ABP (**A**ctivation **B**y **P**ersonalization):

- The **DevAddr**: you must generate a Device Address and program it into the end-device.
- The **NwkSKey**: you must generate a Network Session Key and program it into the end-device.
- The **AppSKey**: you must generate an Application Session Key and program it into the end-device.

For OTAA (Over The Air Activation):

- The AppEUI/JoinEUI: It should be provided by the end-device manufacturer. Otherwise you probably don't use a Join Server so you can use 00 00 00 00 00 00 00 00 and program it in your end-device.
- The AppKey: It should be provided by the end-device manufacturer. Otherwise you can generate one and program your end-device with it.

5.2.4 Step 4: Device configuration

Whatever the activation mode used, all information stored in the LoRaWAN server must be the same than the one in the end-device.

- If your end-device has been pre-provisioned, than you have nothing to do.
- If your end-device is programmable and you generated new root/session keys in the LoRaWAN server, than you have to program them.

6 The LoRa / LoRaWAN frame

6.1 LoRaWAN protocol layers

LoRa is a modulation method to transfer data from one point to another. This refers to the physical layer and is called **LoRa PHY**.

The LoRaWAN protocol adds authentication of the end-device, encryption of the data, acknowledgment, network administration…etc. All these protocol properties are added on top of the LoRa protocol, in a layer called **LoRa MAC**.

At last, the **Application** layer is simply the raw user data but there are also other services available in the LoRaWAN specification.

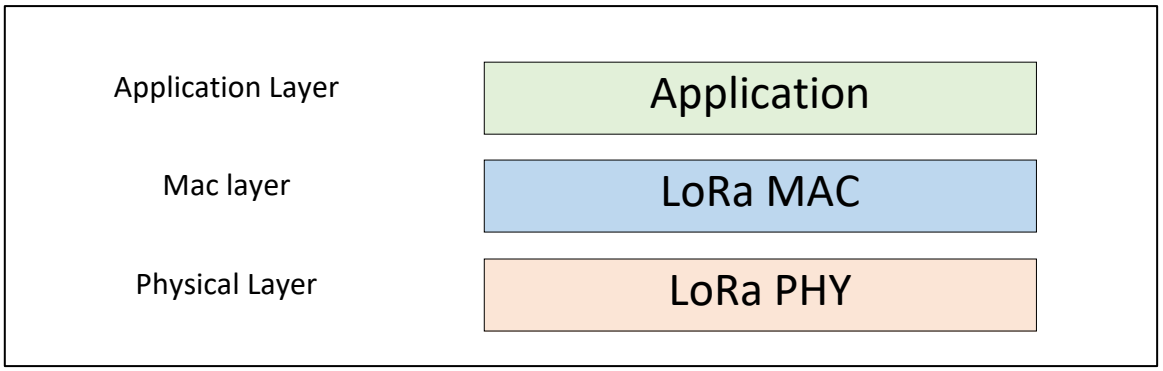


Figure 90: LoRaWAN protocol layers

Each layer adds a functionality. When the frame is sent, the user data is encapsulated in each lower layer. The detail of the whole LoRaWAN frame per layer is described in Figure 91:

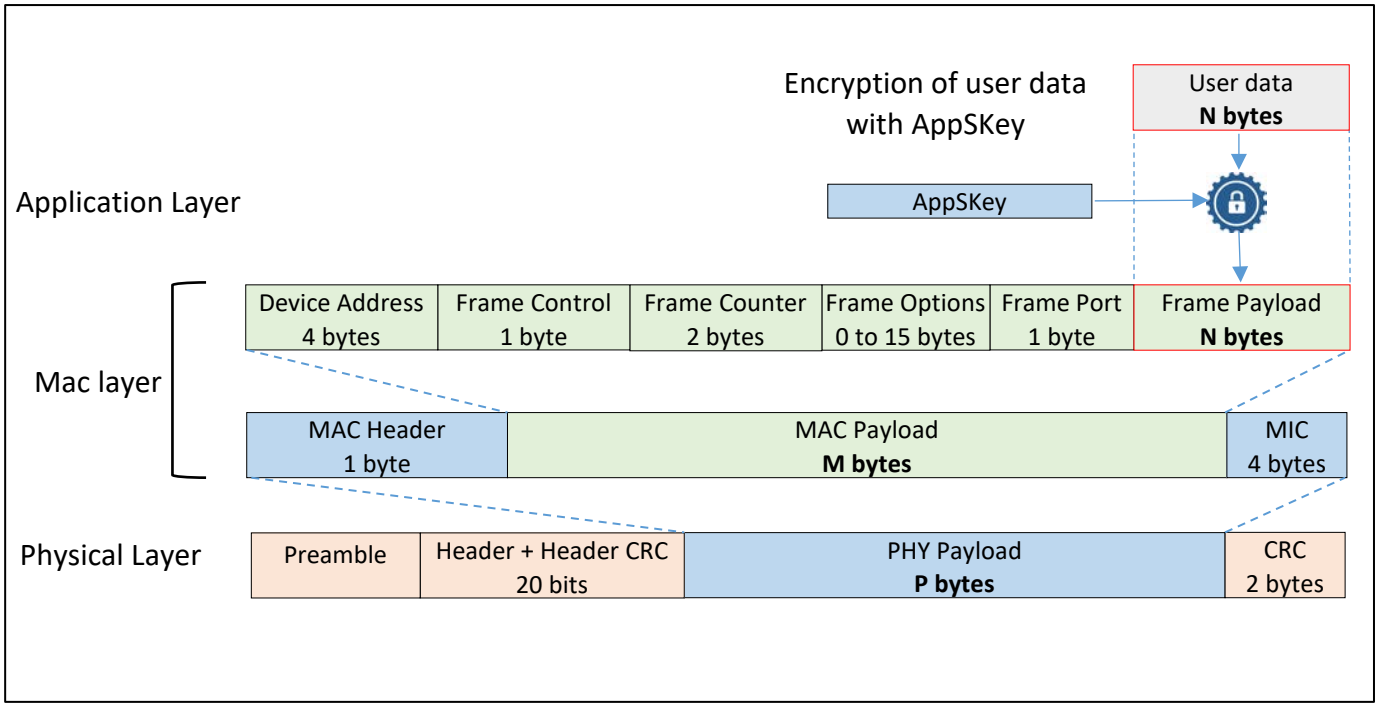


Figure 91: LaRaWAN protocol layer

6.1.1 Application layer

Most of the time, the Application layer is only composed of the user's data. Before encapsulating them in the LoRaWAN frame, they are encrypted with the AppSKey to secure the transaction. Data can be as simple as a single byte from a sensor. The way the user organizes the Frame Payload is completely free as long as it fits the overall maximum size of a LoRaWAN frame.

The LoRa Alliance is working on a scheme to publish data format and corresponding codecs.

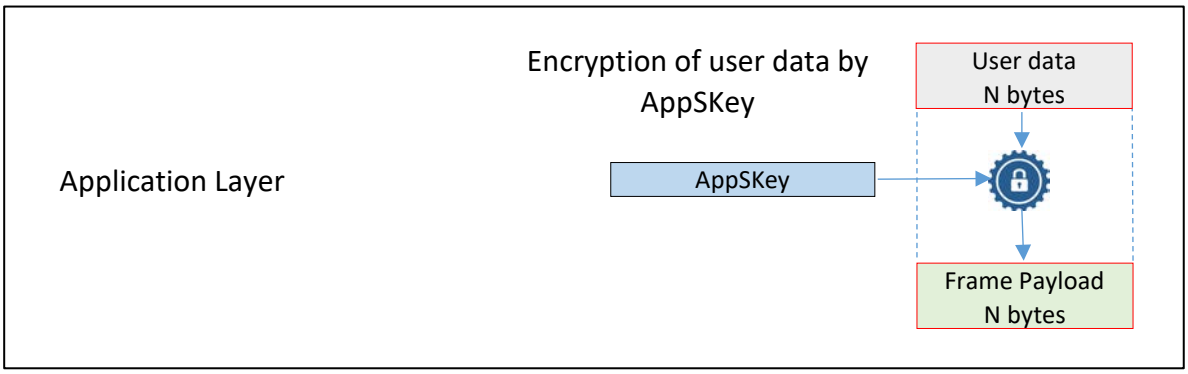


Figure 92: LoRaWAN application layer

The LoRaWAN specification proposes three services on the Application Layer:

Clock Synchronisation: Precise time can be very useful for synchronously performing a measurement. The clock synchronisation application provide a method to synchronize the real time clock of an end-device to the network GPS clock with second accuracy. This application runs on the port 202 by default.

Fragmented Data Block Transport: This application can send a fragmented block of data to one or many end-devices. It runs on the port 201 by default.

Remote multicast Setup: This application can send frames to a group of end-devices. It runs on the port 200 by default.

6.1.2 LoRa MAC layer

The LoRa MAC layer is the heart of the LoRaWAN protocol.

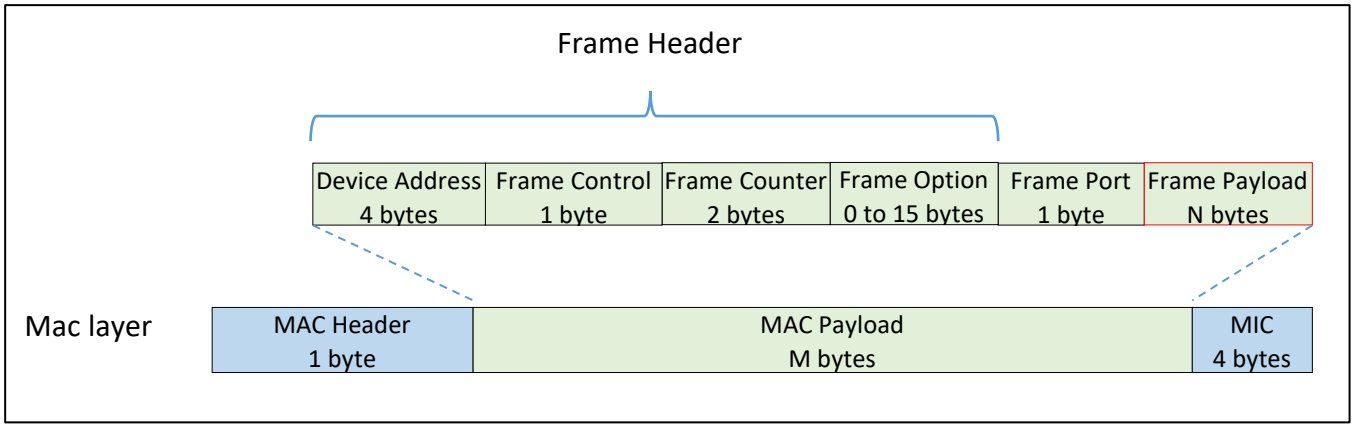


Figure 93: LORA frame LORA MAC layer

- The **MAC Header** field defines the type of message: Join-Request, Join-Accept, data up, data down, confirmed, unconfirmed…
- The **DevAddr** is the Device Address.
- The **Frame control** gives information on the ADR (Adaptive Data Rate), the acknowledgment of messages and also gives the length of the optional "Frame Option".
- The **Frame counter** properties are detailed in chapter 4.5.3.
- The **Frame option** field carries eventual MAC Command
- **Frame Port** is the application port.
- The **Frame Payload** is the encrypted user data.
- **MIC** is the Message Integrity Control that allows the message to be authenticated by the Network Server.

The maximum number of bytes that can be transmitted as a MAC Payload (M bytes) is given in the following table:

Data Rate	Spreading Factor	Bandwidth	Max Frame Payload (Number N)
DR 0	SF12	125 KHz	51 bytes
DR 1	SF11	125 KHz	51 bytes
DR 2	SF10	125 KHz	51 bytes
DR 3	SF9	125 KHz	115 bytes
DR 4	SF8	125 KHz	242 bytes
DR 5	SF7	125 KHz	242 bytes
DR 6	SF7	250 KHz	242 bytes

Table 23: Maximum MAC Payload Size

6.1.3 LoRa PHY layer

Transmitting a LoRaWAN message is simple because there is no need of synchronisation between the end-device and the gateway. Therefore, the PHY layer is very light with only a preamble, an optional header and a CRC.

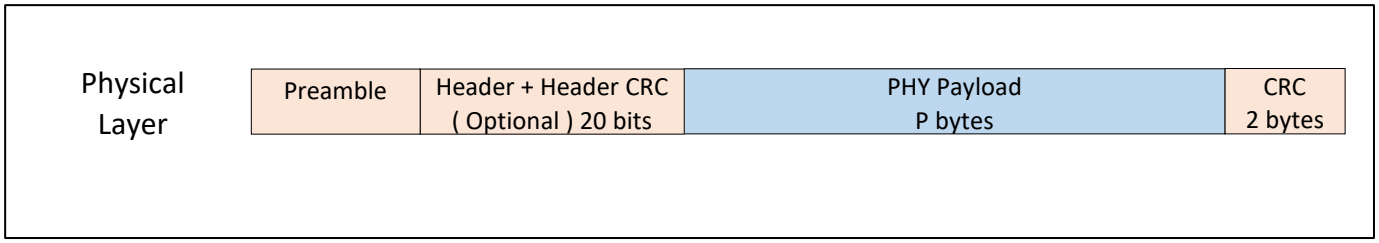


Figure 94: LORA physical layer frame

The **Preamble** is represented by 8 symbols + 4.25 = 12.25 T_{symbol} (see chapter 3.1 for a reminder of the definition of a symbol).

The **optional** header is only present in the default (explicit) transmission mode. It is transmitted with a Coding Rate of 4/8. It indicates the size of the data, the Coding Rate for the rest of the frame. It also specifies if a CRC will be present at the end of the frame.

The **PHY Payload** contains all the information of the LoRa MAC Layer.

The **CRC** is used to detect errors in the LoRa frame.

6.2 Gateways and Network Server communication

The gateway receives on one side a LoRa modulated radio message and transmits on the other side an IP frame to the Network Server.

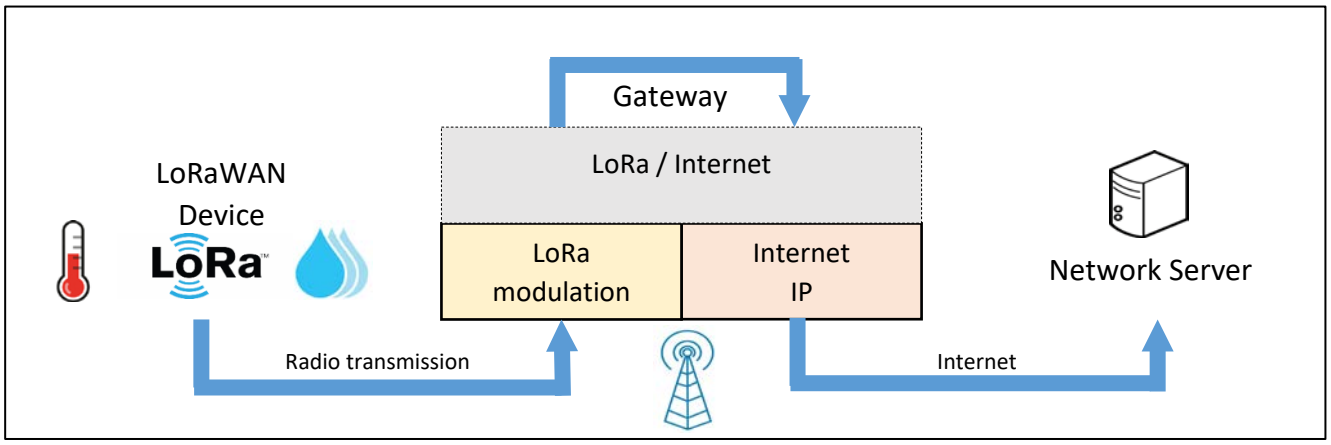


Figure 95: Role of the gateway

Radio interface side: The gateway receives a LoRaWAN frame and extracts the PHY Payload. The 64 base ASCII format is often used (see paragraph 6.3.2). The gateway also extracts all the useful information on the transmission parameters: SF, Bandwidth, RSSI, Time On Air... etc.

On the IP network interface side: The gateway transmits all this information in an IP packet to the Network Server. The transmitted data is in JSON text format (see paragraph 6.3).

6.2.1 The Packet Forwarder

On every gateway, a software called Packet Forwarder is set up to carry out the data transmission to the Network Server.

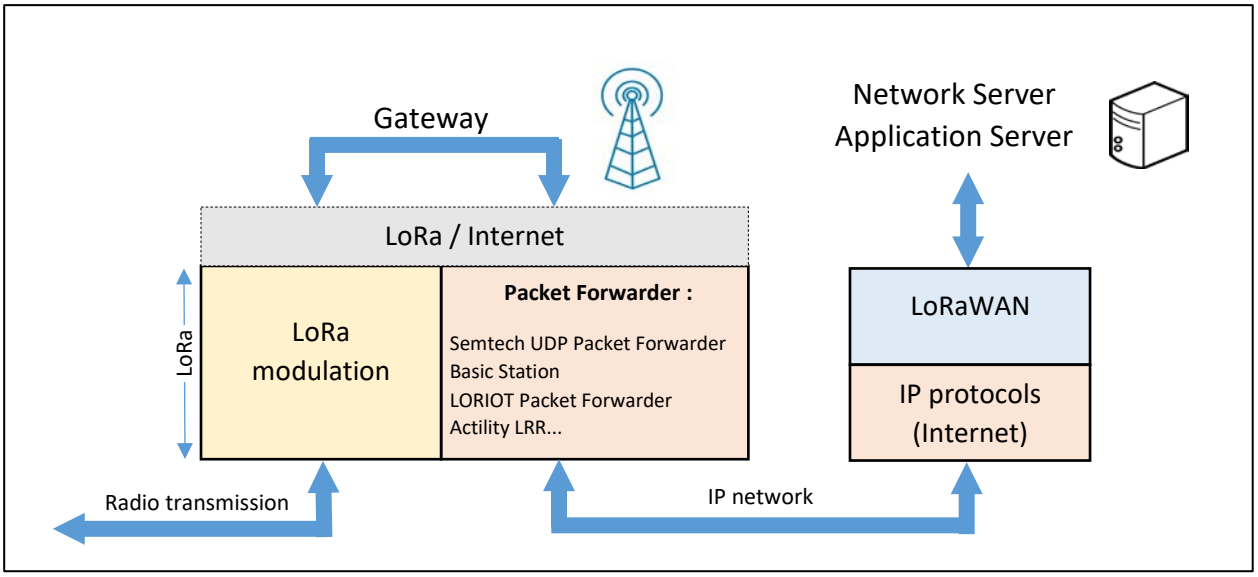


Figure 96: LoRaWAN gateway and Server

Depending on the Packet Forwarder used, the communication protocol between the gateway and the LoRaWAN server is different. First of all, it is mandatory to check if the Packet Forwarder is supported by the Network Server because there are many available and non is imposed by the specification.

- **SENTECH UDP Packet Forwarder**: This was the first widely used Packet Forwarder. It is now deprecated because it has no functionalities other than transmitting uplink and downlink data. It is also a non-secure protocol. This packet forwarder is supported by The Things Stack, but according to the documentation, it will be removed in the future. It is still supported by Chirpstack.
- **Basic Station™ Packet Forwarder**: Basic Station™ is the new Packet Forwarder provided by Semtech. It offers many additional features: TLS, gateway software update, time synchronization, gateway management, etc...
- **Long Range Relay (LRR) Packet Forwarder**: This is the Packet Forwarder used by Actility. It is secure and has many functionalities.

6.2.1 Presentation of UDP Packet Forwarder (Semtech)

Despite all the shortcomings of **Semtech UDP Packet Forwarder**, its use for educational purposes simplifies the analysis of the traffic. The documentation of this protocol is available on GitHub: https://github.com/Lora-net/packet_forwarder . In this folder, a file named PROTOCOL.TXT explains that this protocol works on top of UDP.

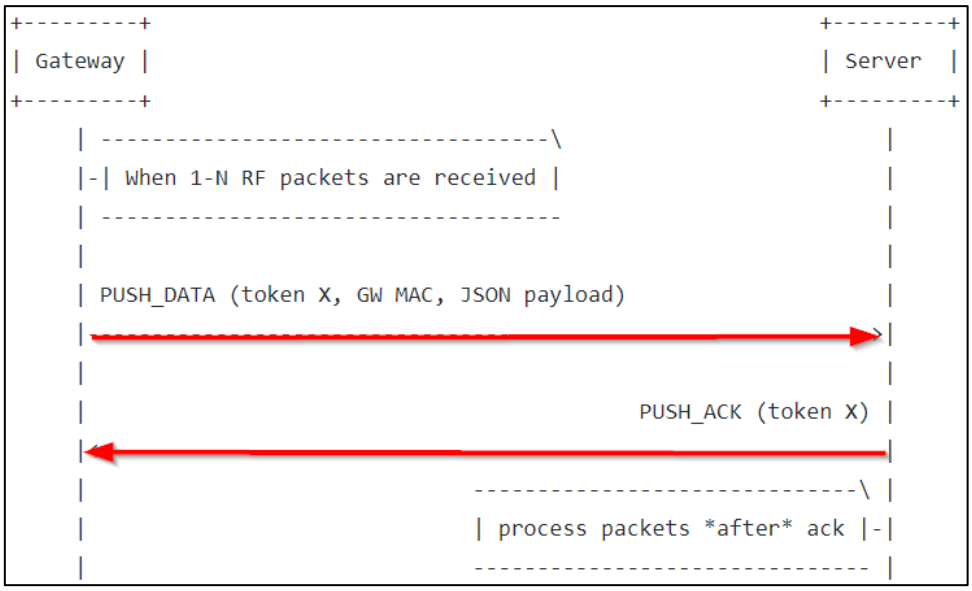


Figure 97: Uplink protocol (PUSH_DATA of the Packet Forwarder)

Packets are sent in UDP to the Network Server in a frame called **PUSH_DATA**. This frame is acknowledged by the Network Server in a frame called **PUSH_ACK**.

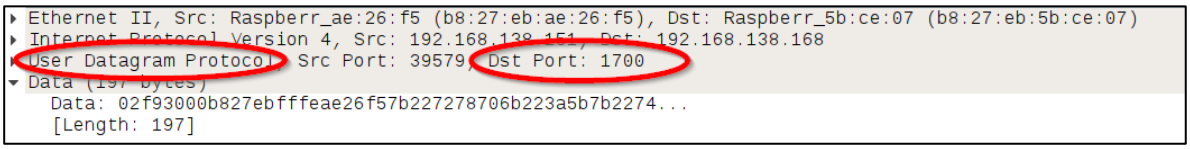


Figure 98: PUSH_DATA frame in Wireshark

This Packet Forwarder was configured to use the UDP port 1700. The content of the data (Data field) is detailed in the following table:

Field	Byte	Function
[1]	0	protocol version = 0x02
[2]	1-2	random token
[3]	3	PUSH_DATA identifier = 0x00
[4]	4-11	Gateway unique identifier (MAC address)
[5]	12-end	JSON object, starting with {, ending with }

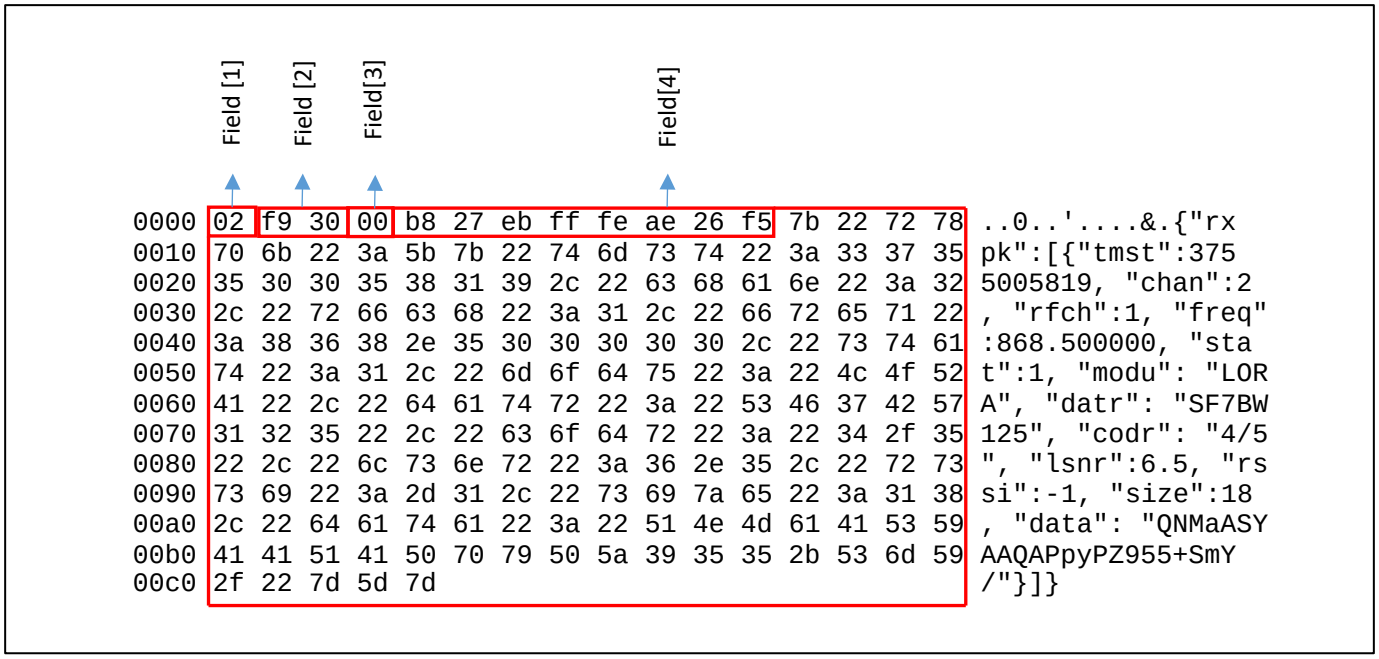


Figure 99: Semtech UDP Packet Forwarder analysis

The JSON object of the transmission is the following:

```
{
      "rxpk":[{   "tmst":3755005819,
                  "chan":2,
                  "rfch":1,
                  "freq":868.500000,
                  "stat":1,
                  "modu": "LORA",
                  "datr": "SF7BW125",
                  "codr": "4/5",
                  "lsnr":6.5,
                  "rssi":-1,
                  "size":18,
                  "data": "QNMaASYAAQAPpyPZ955+SmY/"
            }]
}
```

The "data" field corresponds to the PHY Payload. In the same way we capture the Acknowledgement Frame:

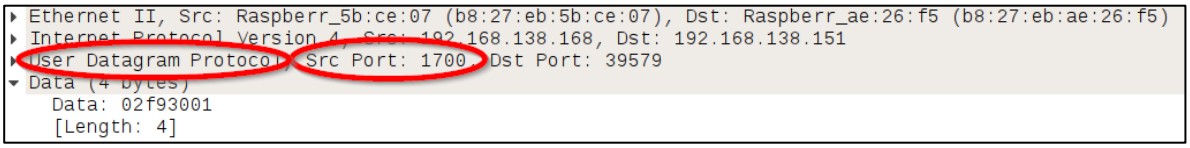


Figure 100: PUSH_ACK frame in Wireshark

```
Fields | Byte | Function
-------|-------------------------------------------------------
  [1]  |  0   | Protocol version = 0x02
  [2]  | 1-2  | Same token as the PUSH_DATA to acknowledge
  [3]  |  3   | PUSH_ACK identifier = 0x01
```

We find all these fields in the Wireshark frame.

6.3 IP frame analysis

6.3.1 The JSON format

Application data are formatted in JSON. The JSON format is a text format composed of a succession of name/value pairs. In Figure 101, "gw_id" is a name and "eui-b427ebfffeae26f5" is the associated value. In this example, the value is a string. The objects are delimited by a pair of braces { }. The Table 23 represents the different JSON format value type.

Value type	Example
String	"coding_rate": "4/5"
Number	"spreading_factor": 12
Object	"lora": **{** "spreading_factor": 12, "air_time": 2465792000 **}**
Boolean	"service" : true

Table 24: Value type in JSON format

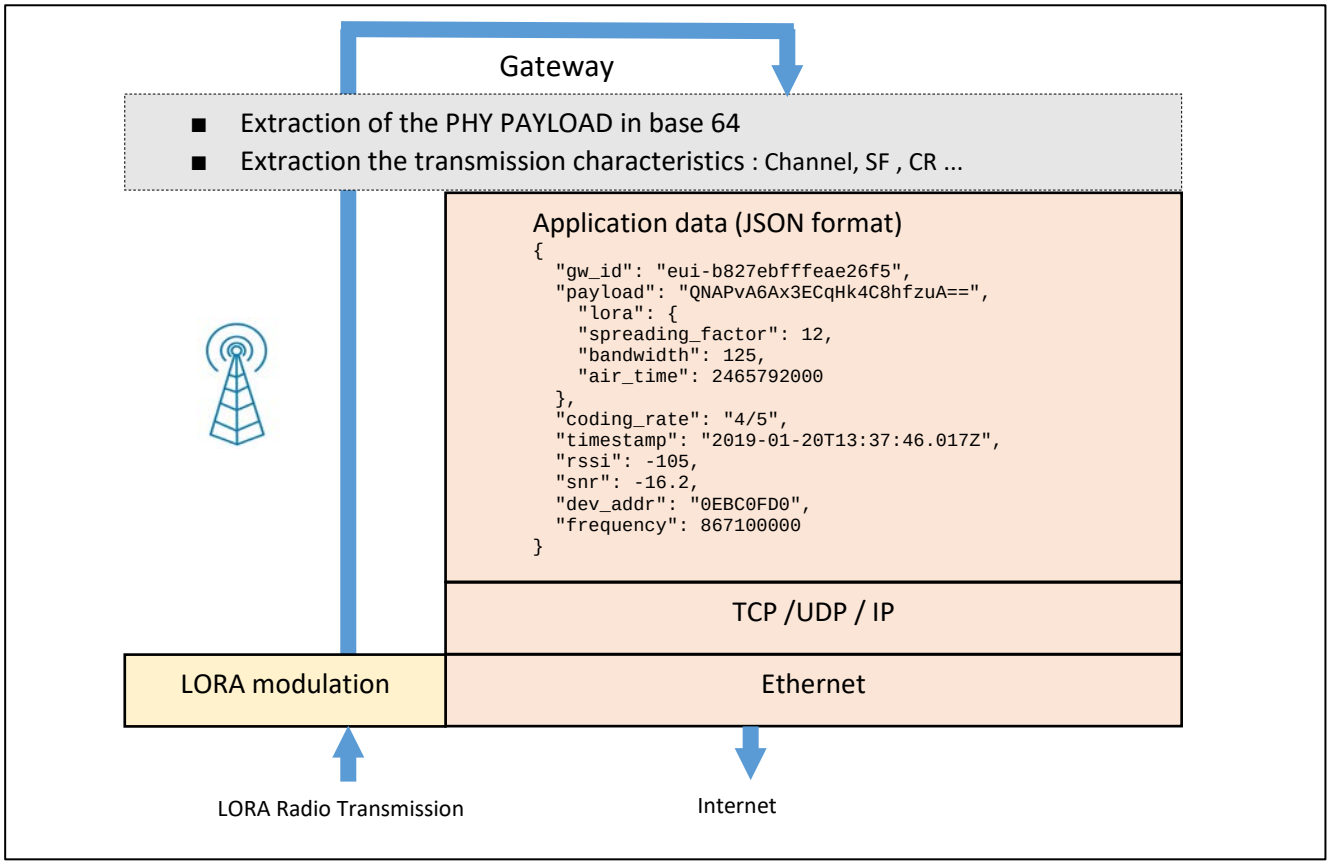


Figure 101: LORAWAN gateway

6.3.2 The 64 base

The gateway extract the PHY Payload from the LoRa modulation. How can we represent this binary data?

Binary (base 2): The simplest way is to represent every binary element (0 and 1). This method is simple, but it has a very big drawback: the representation is complex to read because of the number of bit representing the message. If we want to represent a 50 bytes LoRa frame, we would have to write 400 bits '0' or '1'.

Hexadecimal (base 16): We make groups of 4 bits, which makes 16 possible combinations. The 16 characters used are 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F. This method has the advantage of being 4 times more compact than the binary representation. Can we do better?

Base 64: The bits are grouped by 6, which makes 64 possible combinations. The 64 characters used are those of the Table 24:

Index	Binary	Char	Index	Binary	Char	Index	Binary	Char	Index	Binary	Char
0	000000	A	16	010000	Q	32	100000	g	48	110000	w
1	000001	B	17	010001	R	33	100001	h	49	110001	x
2	000010	C	18	010010	S	34	100010	i	50	110010	y
3	000011	D	19	010011	T	35	100011	j	51	110011	z
4	000100	E	20	010100	U	36	100100	k	52	110100	0
5	000101	F	21	010101	V	37	100101	l	53	110101	1
6	000110	G	22	010110	W	38	100110	m	54	110110	2
7	000111	H	23	010111	X	39	100111	n	55	110111	3
8	001000	I	24	011000	Y	40	101000	o	56	111000	4
9	001001	J	25	011001	Z	41	101001	p	57	111001	5
10	001010	K	26	011010	a	42	101010	q	58	111010	6
11	001011	L	27	011011	b	43	101011	r	59	111011	7
12	001100	M	28	011100	c	44	101100	s	60	111100	8
13	001101	N	29	011101	d	45	101101	t	61	111101	9
14	001110	O	30	011110	e	46	101110	u	62	111110	+
15	001111	P	31	011111	f	47	101111	v	63	111111	/
Padding		=									

Table 25: Base 64 coding characters (source Wikipedia)

This method has the advantage of being 6 times more compact than the binary representation, using only ASCII printable characters. Can we do better?

Base 256 (ASCII): The bits are grouped by 8, which makes 256 possible combinations. The 256 characters used are those of the ASCII table that you can easily find on the web. This method has the advantage of being 8 times more compact than the binary representation. But this representation has a huge disadvantage: many characters of this representation are non-printable (line feed, space, EOF, ...) and therefore are not visible. This representation is therefore useful for text encoding, but unusable if you want to represent binary data.

The best compromise is the base 64. This method is often used to represent the payload of the LoRa frame.

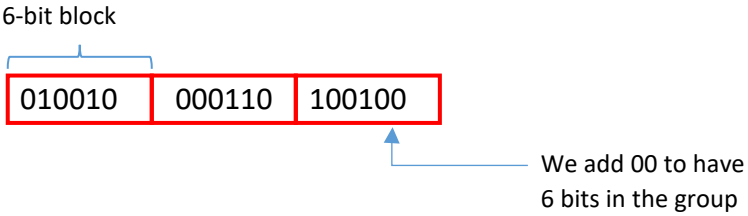
6.3.3 Example of Base 64 encoding

The explanation of the representation method in base 64 is provided through an example: we want to encode the hexadecimal value 0x4869.

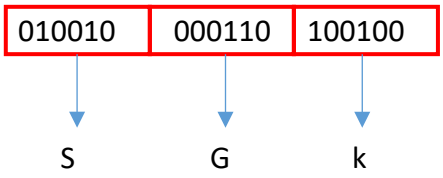
1. The hexadecimal data is written in binary

0x4869 = 0100 1000 0110 1001

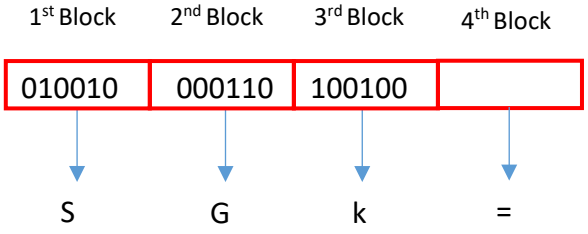
2. The binary elements are grouped in blocks of 6 bits. The number of 6-bit blocks must be a multiple of 4 (minimum 4 blocks). If bits are missing to form a 6-bit group, zeros are added.



3. If there are missing blocks to make a minimum of 4 blocks, special characters will be added.
4. Each group of 6 bits is translated using Table 24.



5. If a block of 6 bits is missing (they must be a multiple of 4), one or more are added (character " = ")



Result: The encoding of 0x4869 in base 64 is "**SGk=**".

We want to encode the ASCII code "AA" in base 64. Find the procedure and show that the base 64 result is "QUE=".

6.3.4 Uplink frame: LoRaWAN end-device to Network Server

When the Network Server receives an IP frames from the gateway, several information can be easily extracted: DevAddr, SF, Bandwidth, etc... but the user data are of course encrypted (with the **AppSKey**). Without knowing the **AppSKey**, it is not possible to understand the content of the message received.

Let's assume that the IP frame received by the Network Server is as follows:


```
{
  "gw_id": "eui-b827ebfffeae26f6",
  "payload": "QNMaASYABwAP1obuUHQ=",
  "f_cnt": 7,
  "lora": {
    "spreading_factor": 7,
    "bandwidth": 125,
    "air_time": 46336000
  },
  "coding_rate": "4/5",
  "timestamp": "2019-03-05T14:00:42.448Z",
  "rssi": -82,
  "snr": 9,
  "dev_addr": "26011AD3",
  "frequency": 867300000
}
```


Than the Network Server will displays the following information:

time	frequency	mod.	CR	data rate	airtime (ms)	cnt		
▲ 15:00:42	867.3	lora	4/5	SF 7 BW 125	46.3	7	dev addr: 26 01 1A D3	payload size: 14 bytes

Figure 102: Frame received on the Network Server side

We can find the following values provided by the gateway:

- Timestamp
- Channel : 867.3 MHz
- Modulation : Lora
- Coding Rate : 4/5
- Data Rate: SF 7 / 125 kHz (DR5)
- Time on Air : 46,3 ms

If we want more information, we can dive into the PHY Payload. Of course, there is an encrypted part (**Frame Payload**), but the headers are in clear (see paragraph 6.1). The PHY Payload of our example is "QNMaASYABwAP1obuUHQ=" (Base 64) or "40D31A01260007000FD686EE5074" (hexadecimal). It has a size of 14 bytes.

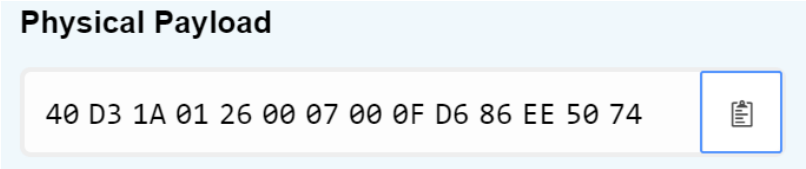


Figure 103: PHY Payload

With the hexadecimal format of the PHY Payload, we can find every field of the frame by using the Figure 91. We have the following result:

```
            PHYPayload = 40D31A01260007000FD686EE5074

PHYPayload = MAC Header[1 byte] | MACPayload[..] | MIC[4 bytes]
        MAC Header                =    40 (Unconfirmed data up)
        MACPayload                =    D31A01260007000FD6
        Message Integrity Code    =    86EE5074

MACPayload = Frame Header | Frame Port | Frame Payload )
        Frame Header              =    D31A0126000700
        FPort                     =    0F
        FramePayload              =    D6

Frame Header = DevAddr[4] | FCtrl[1] | FCnt[2] | FOpts[0..15]
        DevAddr                   =    26011AD3 (Big Endian)
        FCtrl (Frame Control)     =    00 (No ACK, No ADR)
        FCnt (Frame Counter)      =    0007 (Big Endian)
        FOpts (Frame Option)      =
```

You can check the result by yourself using the online packet decoder for 1.0.x LoRaWAN protocol.

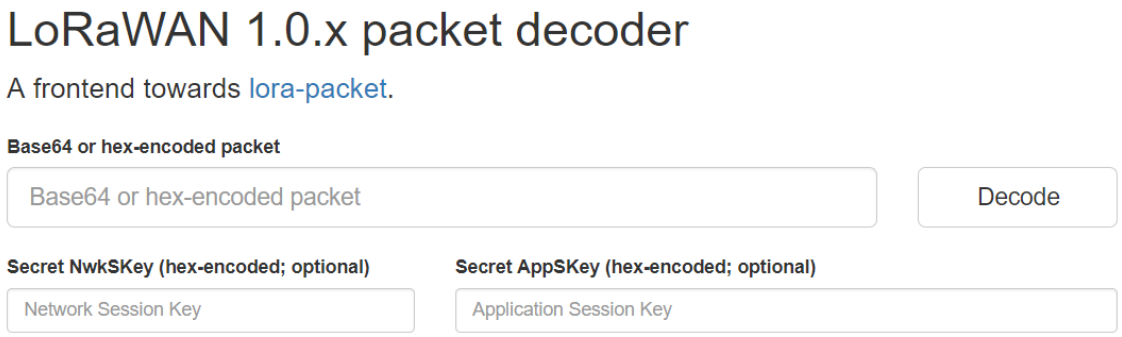


Figure 104: LoRaWAN packet decoder

6.3.5 Uplink: Network Server to Application Server

We will use the same PHY Payload as the Figure 103. For information, the NwkSKey and AppSKey used during this transmission are:

- NwkSKey: E3D90AFBC36AD479552EFEA2CDA937B9
- AppSKey : F0BC25E9E554B9646F208E1A8E3C7B24

The Network Server has decoded the whole frame and check the MIC value. If the MIC is correct (authentication of the frame by the **NwkSKey**) then the Network Server will pass the content of the encrypted message (Frame Payload) to the Application Server. According to the decoding result of the previous chapter, the Frame Payload is:

FramePayload	=	D6

D6 is the encrypted content. When it is decrypted with **the AppSKey**, we find **01**. You can verify all this information with the online packet decoder.

The Application Server will receive the encrypted data only if the LoRaWAN end-device has been registered. On our Application Server, we can check that the received value is really **01**.

Figure 105: Frame received in the Application Server

7 Exporting data from the LoRaWAN server

7.1 The services provided by the IoT Platform

We have seen so far how to send data from an end-device to the LoRaWAN server. This data needs to be transferred on the user server, stored in a database, presented in different forms (tables, graphs...), and finally made available through a web service that the user can query. This is the role of an IoT Platform.

This chapter is independent from the LoRa and LoRaWAN protocol we have studied so far, and the following explanations can be easily transposed to any other protocols related to the Internet of Things.

On the Figure 106, we have on the left side the communication between the LoRaWAN end-device and the LoRaWAN servers (NS and AS). On the right side, we have the communication between the LoRaWAN server and the IoT Platform. The IoT platform will be the link with the user and will have to perform the following actions:

- Receive data from the LoRaWAN server (uplink).
- Transmit data to the LoRaWAN server (downlink).

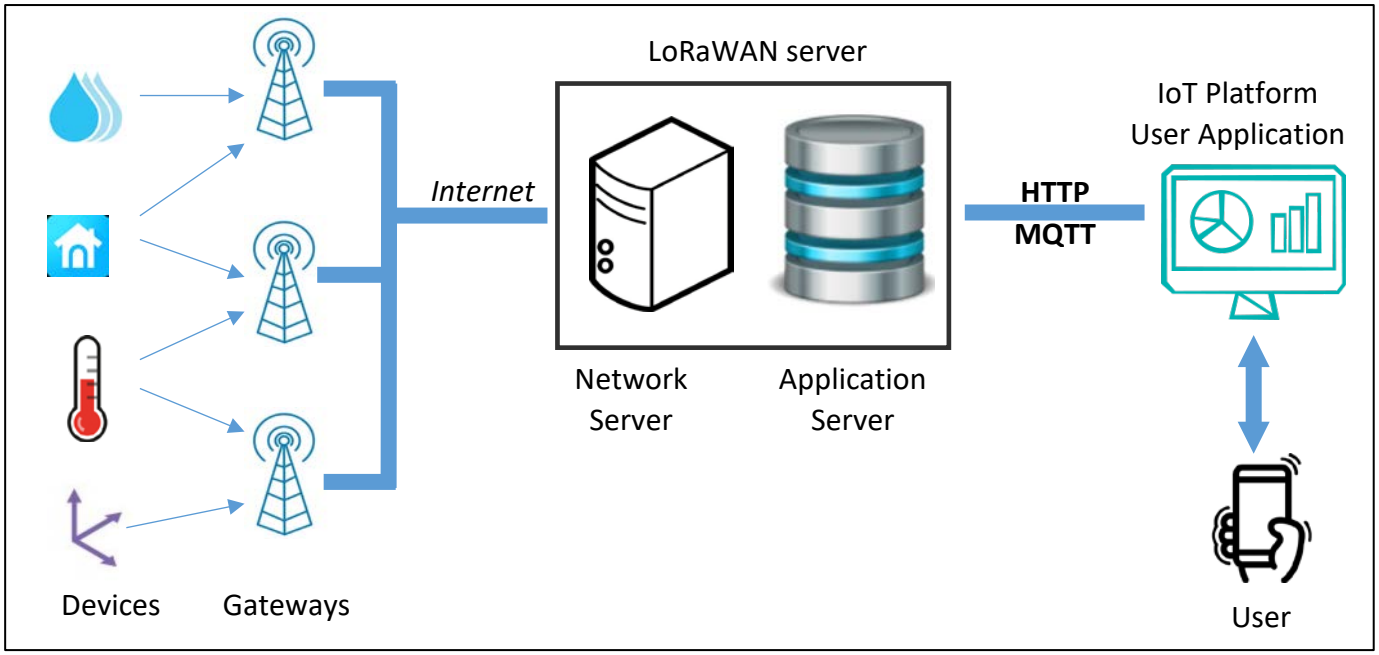


Figure 106: Overall structure of a LORAWAN network

Note that if your Network Server provides a more secure environment with end to end security, you would have the following architecture (Figure 107). In that case, the IoT Platform is the Application Server.

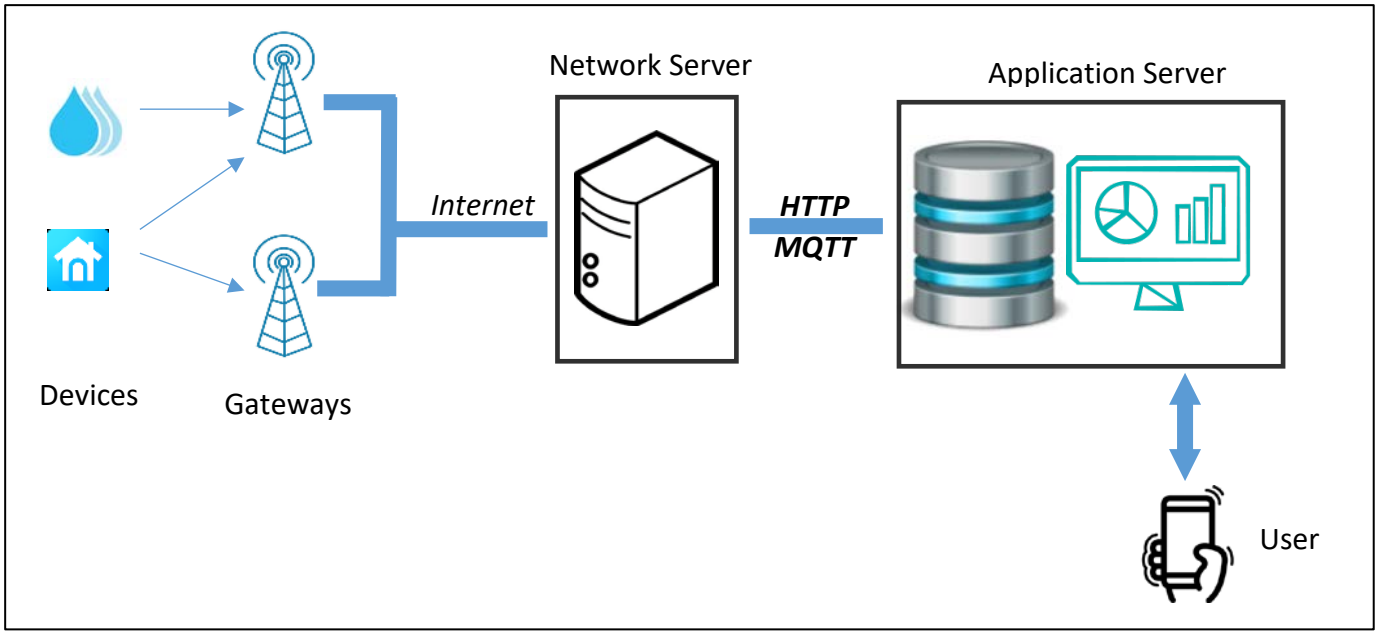


Figure 107: Overall LoRaWAN architecture with end-to-end security

The dialogue between the LoRaWAN server and the IoT Platform can be done using different protocols that we will study in the next chapters.

In the uplink direction, our IoT Platform will have the following roles:

1. **Data import**
2. Data storage (backup)
3. Data format into useful form (graphics, tables...)
4. Data display for the user (web interface)

In the Downlink direction, our IoT Platform will have the following role:

1. User interface display (Button, text field, ...)
2. User request processing
3. Storage of the query (backup)
4. **Data transmission to the LoRaWAN server**

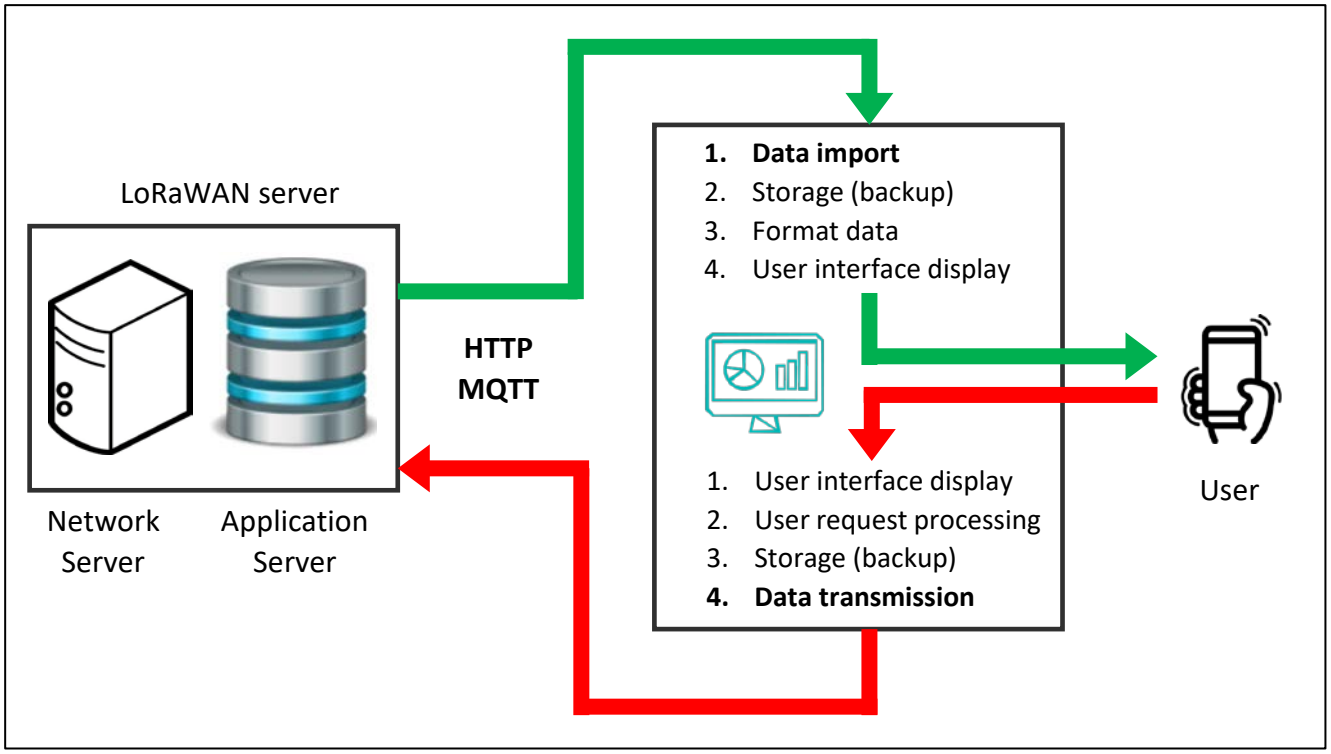


Figure 108: IoT Platform services

In this chapter, we will deal only with the two **bold** items in the previous list, i.e. **"Data Import"** (uplink) and **"Data transmission"** (downlink).

We will see two methods to communicate: **HTTP** and **MQTT** protocols.

7.2 Exporting data with HTTP GET protocol

7.2.1 Presentation of the Client - Server principle

Like most protocols, HTTP involves the transfer of information between a client and a server. The client and server are two remote entities that wish to communicate with each other. The client makes requests and the server answers. The client and the server can be of any type: mail (SMTP), files (FTP)… Here we use HTTP client and HTTP server.

Figure 109 shows a client, a server and the two types of message sent: requests and replies. It is very important to know who will be the client and who will be the server. Indeed, we will have to assign a role (client or server) to our LoRaWAN server and to our IoT Platform.

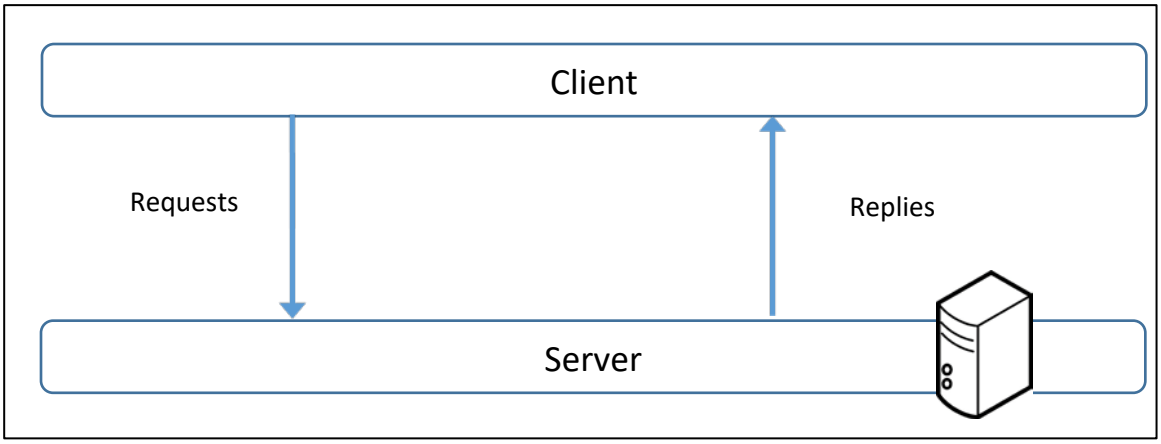


Figure 109: "Client – Server" and "Requests – Replies"

7.2.2 Client and server designation

The dialogue takes place between the LoRaWAN server and the User Application (IoT Platform). The Figure 110 represents these two entities and the exchange. At the top of the figure, we have the LoRaWAN server (TTN, Actility, Loriot, Chirpstack, etc…), and at the bottom, we have the User IoT Platform. We now need to understand "who is requesting?" and "who is serving?"

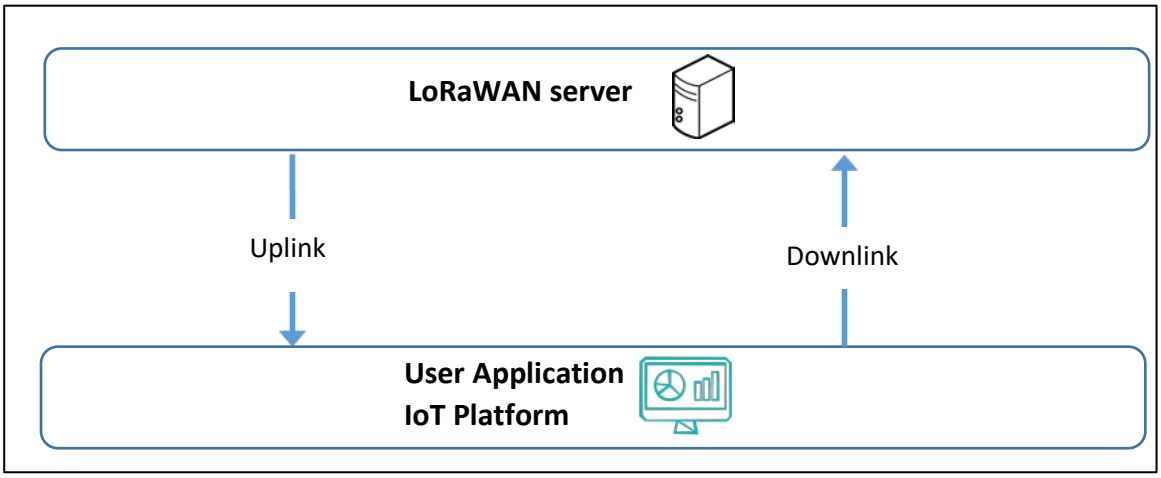


Figure 110: Communication between the LoRaWAN server and the IoT Platform

We will now study the two situations represented in Figure 111:

- Uplink: from the LoRaWAN server to the IoT Platform (User Application).
- Downlink: from the IoT Platform (User Application) to the LoRaWAN server.

Let's start with the uplink situation that is the most common. From the User Application, we want to retrieve the data which are on the LoRaWAN server. The first idea is to make a request **(1)** from our User Application in order to get the data by making a HTTP GET request. When you make an HTTP GET request to a Web server, it returns the content of the HTML page it contains. Here, The LoRaWAN server, will return the LoRaWAN data **(2)** in JSON format. In this case:

- The User Application is the HTTP client.
- The LoRaWAN server is the HTTP Server.

Now, let us talk about downlink. From the LoRaWAN server, we want to retrieve the data which are on the User Application. The idea is to make a request **(3)** from the LoRaWAN server, and the User Application will reply **(4)**. In this case:

- The LoRaWAN server is the HTTP client.
- The User Application is the HTTP server.

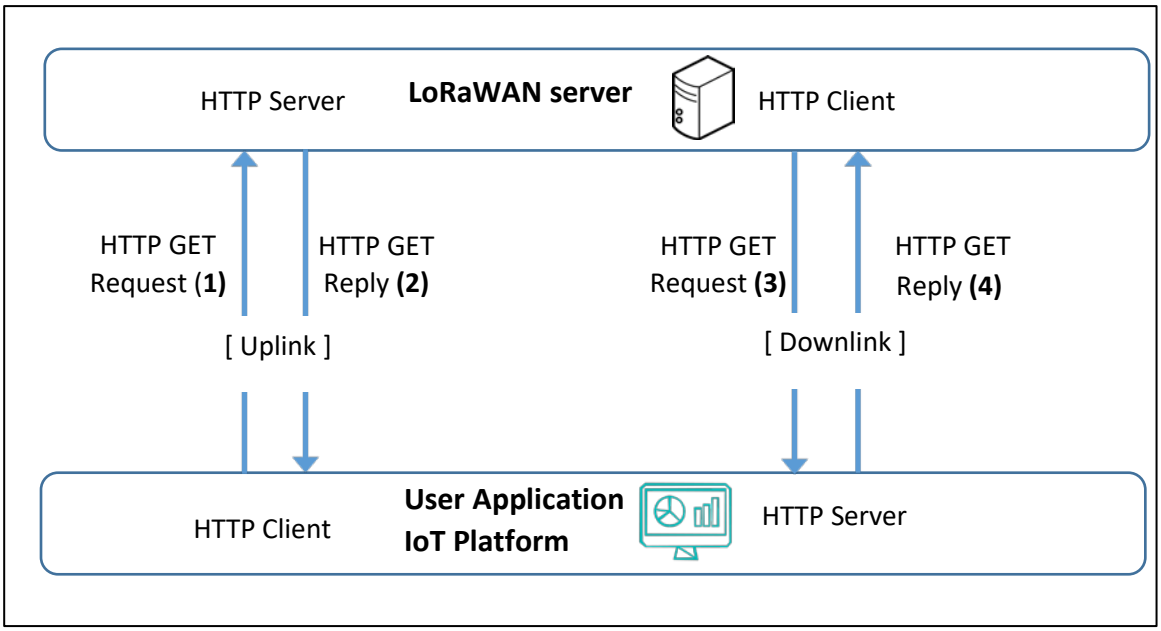


Figure 111: Uplink and downlink (HTTP GET)

From the client, to carry out a HTTP GET Request, we need:

- The URL format corresponding to the request
- An API Key token to have the right to make a request on the HTTP server.

7.2.3 Setting up HTTP GET service (uplink)

We are going to set up a HTTP GET server. In the Figure 111, this represents the frames **(1)** and **(2)**. The HTTP server is on the LoRaWAN server and the HTTP client on the User Application.

In the example, we use TTN community (v3.15.1) LoRaWAN server. The HTTP GET service is called "Storage Integration".

We start by setting up the HTTP server on TTN. To do this, we need to go to our TTN console. **TTN > Applications > Application name > Integration > Storage Integration > Activate Storage Integration.** The server is then available as well as a temporary data backup.

To see the APIs available to retrieve data, check the documentation on "Storage Integration API".

We are going to use the software called POSTMAN as the HTTP client which allows to generate all kinds of HTTP requests. We will just have to refer to the documentation to choose the right formats.

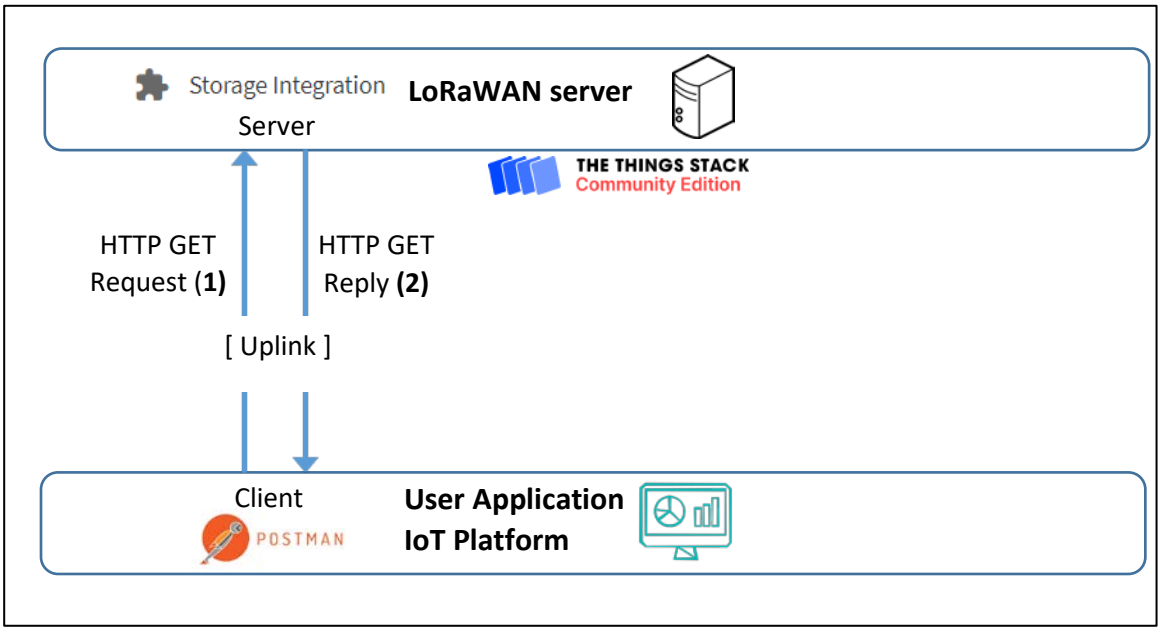


Figure 112: Importing data with HTTP GET request

In POSTMAN, we will now perform a HTTP GET request. We follow the documentation and try to retrieve all uplink messages from an application.

Create an API Key: **TTN > Applications > your Application > API Key > Add API Key** > Grant all current and future rights, then copy the key.

- **POSTMAM > Import > Raw Text**

```
curl -G
"https://eu1.cloud.thethings.network/api/v3/as/applications/app1/p
ackages/storage/uplink_message" \
    -H "Authorization: Bearer $API_KEY" \
    -H "Accept: text/event-stream"
```

- Change "**app1**" by your application name
- Change **$API_KEY** by your API Key
- Import the request and send it.

You should have a reply (Status 200: OK) with a JSON text message containing your uplink frame.

7.2.4 Information on the HTTP GET method

This method is interesting for its simplicity since we only have to send HTTP GET requests whenever we need the uplink message received on the LoRaWAN server.

The first drawback is that we only worked on the uplink stream. There is no possibility to send downlink message to the LoRaWAN server. No LoRaWAN server has the right part of Figure 111 implemented **(3)** and **(4)**.

The second drawback is that for the uplink stream, we spend our time requesting data that potentially does not exist. Indeed, we query data without knowing if they have been received. If a sensor is non-regularly emitting values, then we will have to make periodic requests with a high chance of getting empty responses.

For the Downlink stream, if we could have installed the client on the LoRaWAN server, the problem would be the same. We would spend our time asking for commands whereas there is a good chance that the user has not sent any.

The solution is to reorganize the client and server roles to optimize the way we transfer data. This will be possible thanks to HTTP POST requests.

7.3 Exporting data with HTTP POST protocol

For the uplink stream, we assign the roles in a different way by imagining that the User Application will not ask the LoRaWAN server for the information, but that the LoRaWAN server will rather provide it itself. The LoRaWAN server will therefore transmit to the User Application a request called HTTP POST **(1)** to "post" the data. The response **(2)** is a simple acknowledge and contains no data. In this case:

- The LoRaWAN server is the HTTP client.
- The User Application is the HTTP server.

For the downlink stream, the user who wants to transmit data to the LoRaWAN server must provide a HTTP POST **(3)** request and the server will answer an acknowledgement. In this case:

- The User Application is the HTTP client.
- The LoRaWAN server is the HTTP server.

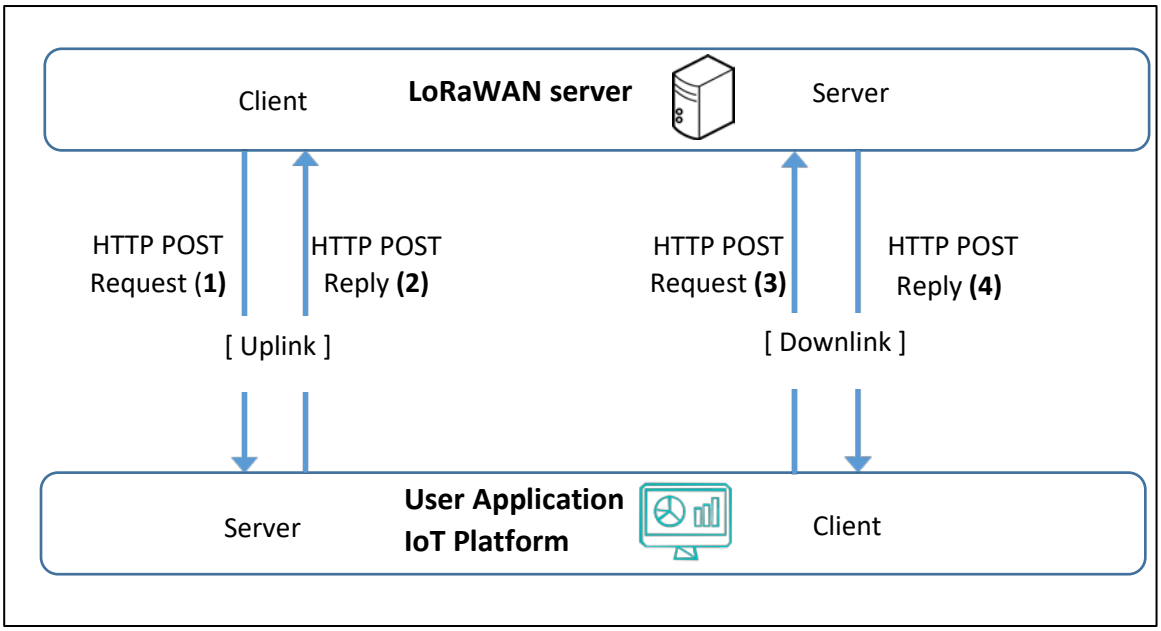


Figure 113: Uplink and downlink - HTTP POST

7.3.1 Setting up HTTP POST service (uplink)

In this example, we use LORIOT (v7.0.14) LoRaWAN server. The HTTP POST service is called "HTTP Push Output".

We want to receive the data present from the LoRaWAN servers. On Figure 113, this represents the uplink exchanges frames **(1)** and **(2)**.

The process for other LoRaWAN server is similar. You can find on our website the method for other Server (Actility, TTN, Chirpstack, LoRa Cloud…).

We need to set up a HTTP POST client on LORIOT's server and a HTTP POST server on our User Application.

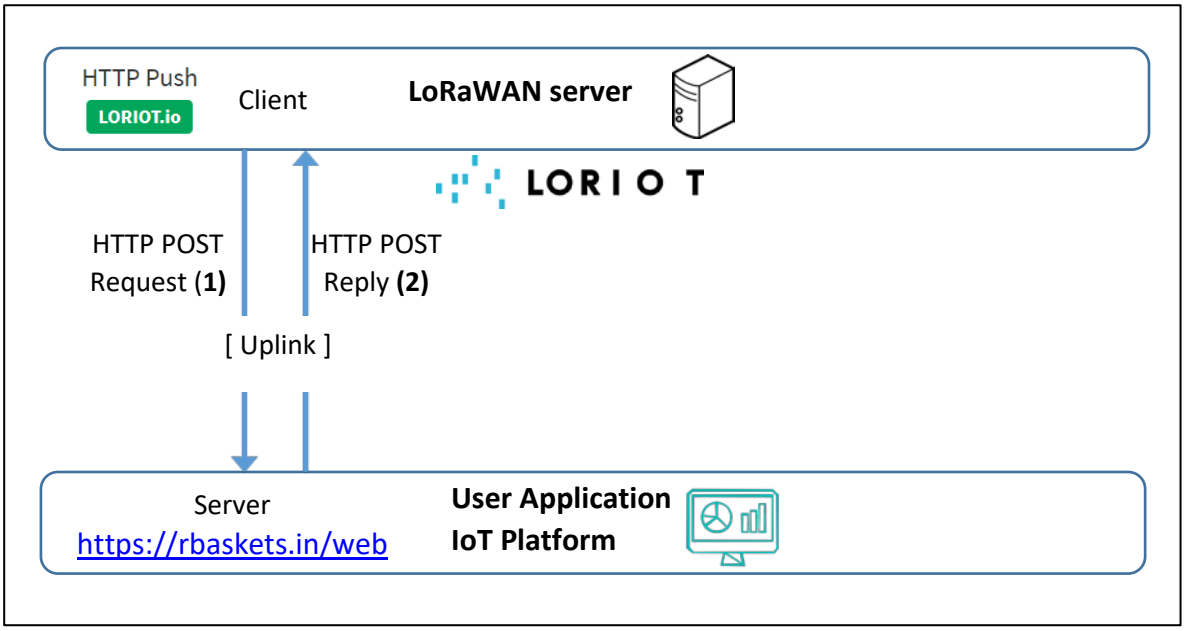


Figure 114: Uplink HTTP POST with LORIOT

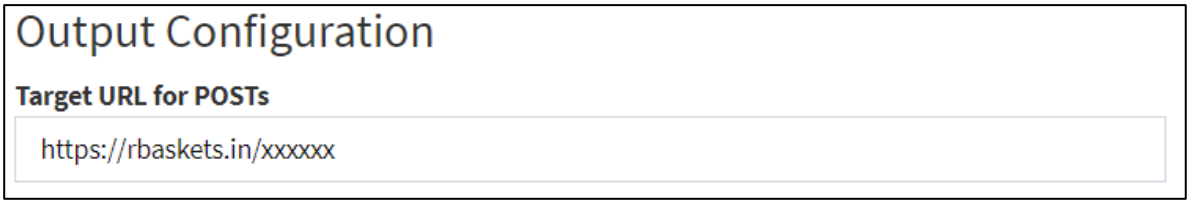
We start by setting up the server. We will use a HTTP server available on the web, handling HTTP POST requests [https://rbaskets.in/] or [https://beeceptor.com/] for example.

- Go to https://rbaskets.in/ and create a new Basket (Endpoint server).
- Keep the token if you want to come back to the same server later and click on "Open Basket".
- The address to which you should send your requests is then specified and that is the one we will use to configure the client.

The HTTP POST server is ready and you are waiting for data. Your basket is empty but that is where your will receive the data from LORIOT.

Now, we need to set up the HTTP POST client on the LORIOT LoRaWAN server.

- **LORIOT > Applications > Your Application > Output > Add New Output > HTTP Push**, and enter your HTTP POST server address under "Target URL for POSTs".



The HTTP POST client is set up. You can send data with your end-device and receive them in JSON format on your server.

7.3.1 Setting up HTTP service (downlink)

We will send user data to the LoRaWAN server. In Figure 113, this represents the downlink exchanges frames **(3)** and **(4)**. We need to set up a HTTP client on our User Application, and a HTTP server on LORIOT.

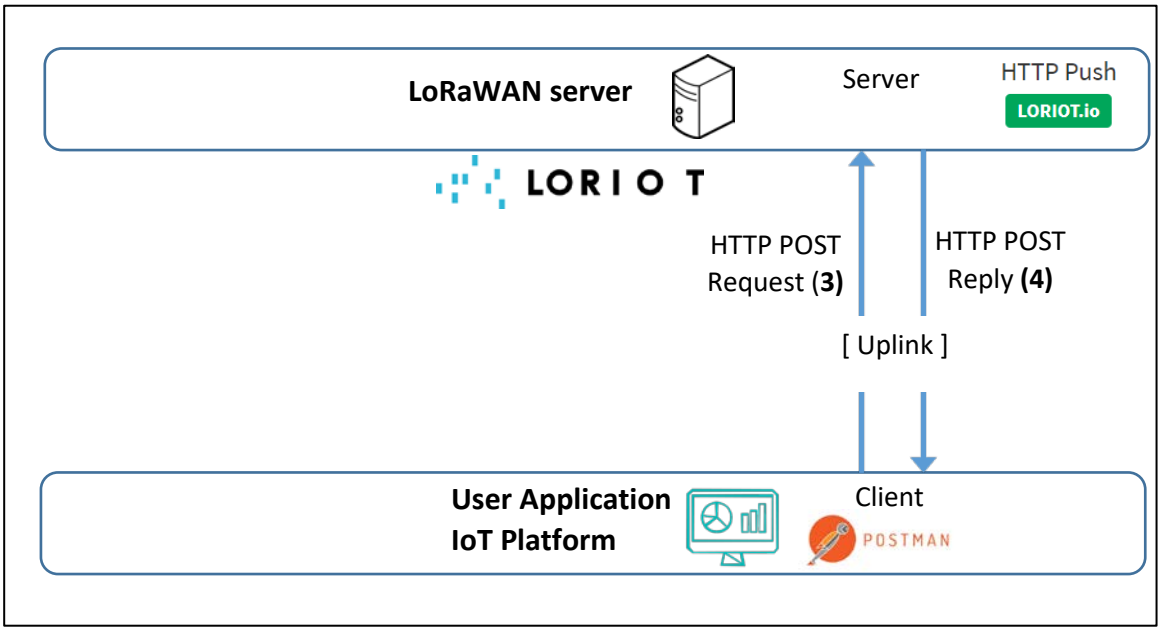


Figure 115: Downlink HTTP POST with LORIOT

Setting up the server is very simple since it already exists in all LoRaWAN server (LORIOT, Actility, TTN, …). It is already running so there is nothing to do.

For the HTTP POST client, we will use POSTMAN to generate the request. You will have to check on your server documentation the HTTP POST request format.

Once again we will use LORIOT but the process for other LoRaWAN server is similar. You can find on our website the method for other Server (Actility, TTN, Chirpstack, LoRa Cloud…).

For LORIOT:

POSTMAM > Import > Raw

```
curl --location --request POST 'https://eu1.loriot.io/1/rest' \
--header 'Content-Type: application/json' \
--header 'Authorization: Bearer <API TOKEN>' \
--header 'Content-Type: text/plain' \
--data-raw '{
"cmd": "tx",
"EUI": "XXXXXXXXXXXXXXXX",
"port": XX,
"confirmed": false,
"data": "ABCDEF",
"appid": "XXXXXXXX"
}'
```

With the following modifications:

- ***<API TOKEN>*** : Application> Access Token > Generate authentication token
- **"EUI" :** DevEUI of the end-device
- **"appid"** : LORIOT Application ID (4 bytes)
- **"confirmed":** true or false
- **"data":** Hexadecimal data to send

7.4 Presentation of the MQTT protocol

7.4.1 MQTT Protocol Overview

MQTT is a lightweight protocol for the Internet of Things. Rather than the classic client/server architecture that works with requests / replies, MQTT is based on a publisher / subscriber model. The difference is important, because it avoids having to request data that you have no idea when it will arrive. A data will be directly transmitted to the subscriber as soon as it has been received in the broker (central server). In order to receive the data that belongs to a Topic, a Subscriber must first subscribe (as the name suggests) to that Topic.

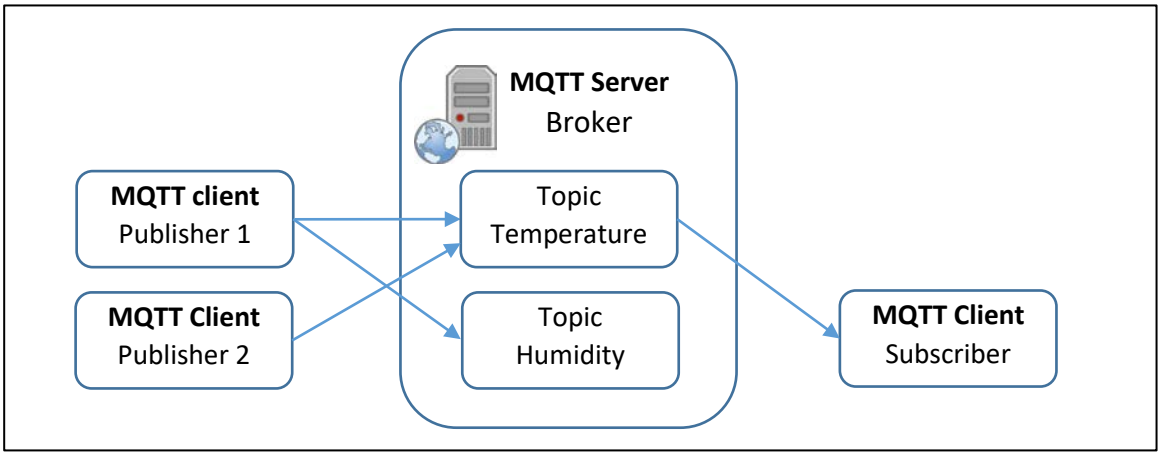


Figure 116: Publisher / Subscriber model of the MQTT protocol

MQTT is a protocol based on TCP represented by the following stack:

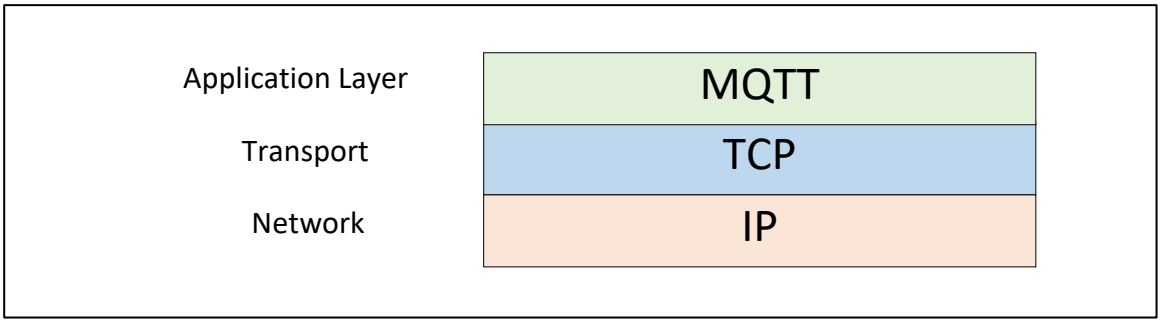


Figure 117: Protocols used for communication with MQTT

This can be verified by a frame capture on Wireshark.

```
> Ethernet II, Src: Raspberr_f3:03:41 (b8:27:eb:f3:03:41), Dst: Dell_7d:b5:7e (10:65:30:7d:b5:7e)
> Internet Protocol Version 4, Src: 192.168.0.200, Dst: 192.168.0.11
> Transmission Control Protocol, Src Port: 1883, Dst Port: 62454, Seq: 15, Ack: 5, Len: 4
> MQ Telemetry Transport Protocol, Publish Release
```

Figure 118: Capturing an MQTT frame with Wireshark

Note that the TCP port used for the MQTT protocol (not encrypted) is 1883.

- **Publishers and Subscribers do not need to know each other.**
- **Publishers and Subscribers do not have to run at the same time.**

7.4.2 Connection to the MQTT broker

We will focus on the connection options managing the Quality of Service (QoS). To connect, a MQTT client sends two important information to the broker:

A keepAlive number: This is the longest period during which the publisher or subscriber client can remain silent. After that, they will be considered disconnected.

A Boolean value "cleanSession": When the client and the broker are shortly disconnected (beyond the announced keepAlive), we can wonder what will happen when the client connects again.

- cleanSession = True. The connexion is non-persistent. The non-transmitted messages are lost regardless of the QoS (Quality of Service) level.
- cleanSession = False. The connection is persistent. The non-transmitted messages will eventually be retransmitted, depending on the QoS level. See Chapter 7.4.4**.**

7.4.3 Quality of Service during a unique connection

When the client connects to the broker, it is possible to choose the QoS level. We are talking here about the case of a unique connection, between the moment the client connects and the moment when:

- The client closes explicitly its connection.
- It has not shown any sign of life during the ''keepAlive'' time.

In that case, we have the following Quality of Service:

QoS 0 ''At most once'': QoS level 0 is "no acknowledgement". The publisher sends each message only once to the broker. The broker sends each message only once to the subscribers. This mechanism does **not guarantee** the correct reception of MQTT messages.

QoS 1 ''At least once'': QoS level 1 is ''with acknowledgement''. The publisher sends each message to the broker and waits for its confirmation. In the same way, the broker sends each message to its subscribers and waits for their confirmation. This mechanism **guarantees** the reception of at least one MQTT message.

However, if the acknowledgements do not arrive in time, or if they are lost, the re-transmission of the original message may result in a duplicate message. The last QoS level prevent this.

QoS 2 ''Exactly once'': QoS level 2 is ''guaranteed once''. The publisher sends a message to the broker and waits for confirmation. The publisher then gives the order to broadcast the message and waits for confirmation. This mechanism **ensures** that no matter how many times a message is retransmitted, it will **only** be delivered once.

The figure below shows the frames transmitted for each QoS level.

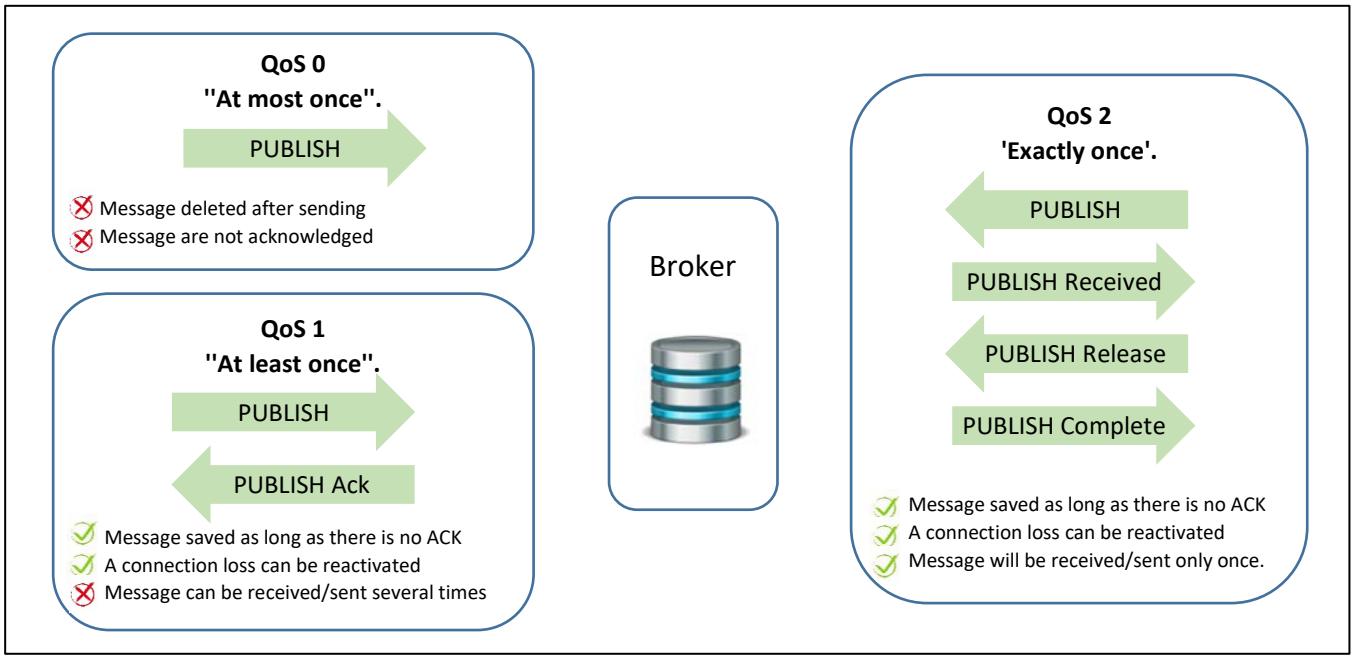


Figure 119: Quality of Service in MQTT protocol

Figure 120 shows the three QoS frames captured in Wireshark.

Source	Destination	Protocol	Length	Info
192.168.0.200	192.168.0.11	MQTT	66	Publish Message [test]

Source	Destination	Protocol	Length	Info
192.168.0.200	192.168.0.11	MQTT	68	Publish Message (id=4) [test]
192.168.0.11	192.168.0.200	MQTT	58	Publish Ack (id=4)

Source	Destination	Protocol	Length	Info
192.168.0.200	192.168.0.11	MQTT	68	Publish Message (id=5) [test]
192.168.0.11	192.168.0.200	MQTT	58	Publish Received (id=5)
192.168.0.200	192.168.0.11	MQTT	60	Publish Release (id=5)
192.168.0.11	192.168.0.200	MQTT	58	Publish Complete (id=5)

Figure 120: Frame capture with QoS = 0, then QoS = 1, then QoS = 2

Of course, a better QoS increases the network load:

- One frame for QoS 0.
- Two frames for QoS 1.
- Four frames for QoS 2.

7.4.4 Quality of Service after a reconnection

What happens to the messages published on the broker when one or more subscribers are temporarily unreachable? It is possible to save messages that have been published on the broker in order to retransmit them the next time you connect. This possibility of saving messages must be activated when the client connects to the broker (cleanSession = 0). The connection will then be persistent.

Table 25 summarizes the effect of the cleanSession flag and QoS level when there is a reconnection of the client.

Clean Session Flag	Subscriber QoS	Publisher QoS	Behaviour
True (= 1)	0 / 1 / 2	0 / 1 / 2	Lost messages
False (= 0)	0	0 / 1 /2	Lost messages
False (= 0)	0 /1 / 2	0	Lost messages
False (= 0)	1 / 2	1 / 2	All messages are retransmitted

Table 26: QoS value and the cleanSession flag

7.4.5 MQTT Protocol Topics

A topic is a hierarchy of strings separated with the slash "/" character as a separator. Figure 121 and Table 26 give an example of a topic hierarchy.

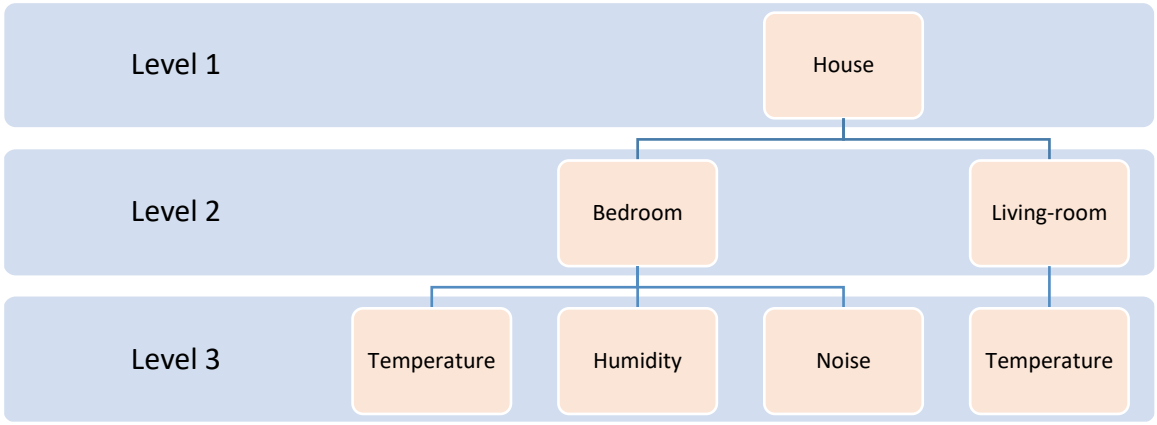


Figure 121: Example of MQTT topic hierarchy

Topic Name	Topic detail
House/Bedroom/Temperature	The **Temperature** of the **Bedroom** in the **House.**
House/ Bedroom /Noise	The **Noise** of the **Bedroom** in the **House.**
House/Living Room/Temperature	The **Temperature** of the **Living Room** in the **House.**

Table 27: Example of topics

A client can subscribe (or unsubscribe) to several branches of the hierarchy by using wildcards that cover several topics. Two wildcards characters exist:

- The plus sign "+" replaces any string on the same level.

- The hash sign "#"replaces any string on all subsequent levels. It must be placed at the end.

Topic Name	Topic detail
Home/+/Temperature	The **Temperature** of all **Rooms** in the **House**.
House/#	All **measurements** of all **Rooms** in the **House**.

Table 28: Example of topics using wildcards

7.4.6 Setting up an MQTT broker

To understand how the MQTT protocol works, we set up a simple MQTT infrastructure with one client publisher, a broker and one client subscriber. There are lots of broker and MQTT client available. For our test, we use:

- The Mosquitto **broker**.
- The MQTT Box (for Windows) client **subscriber**.
- The MQTT Box (for Windows) client **publisher**.

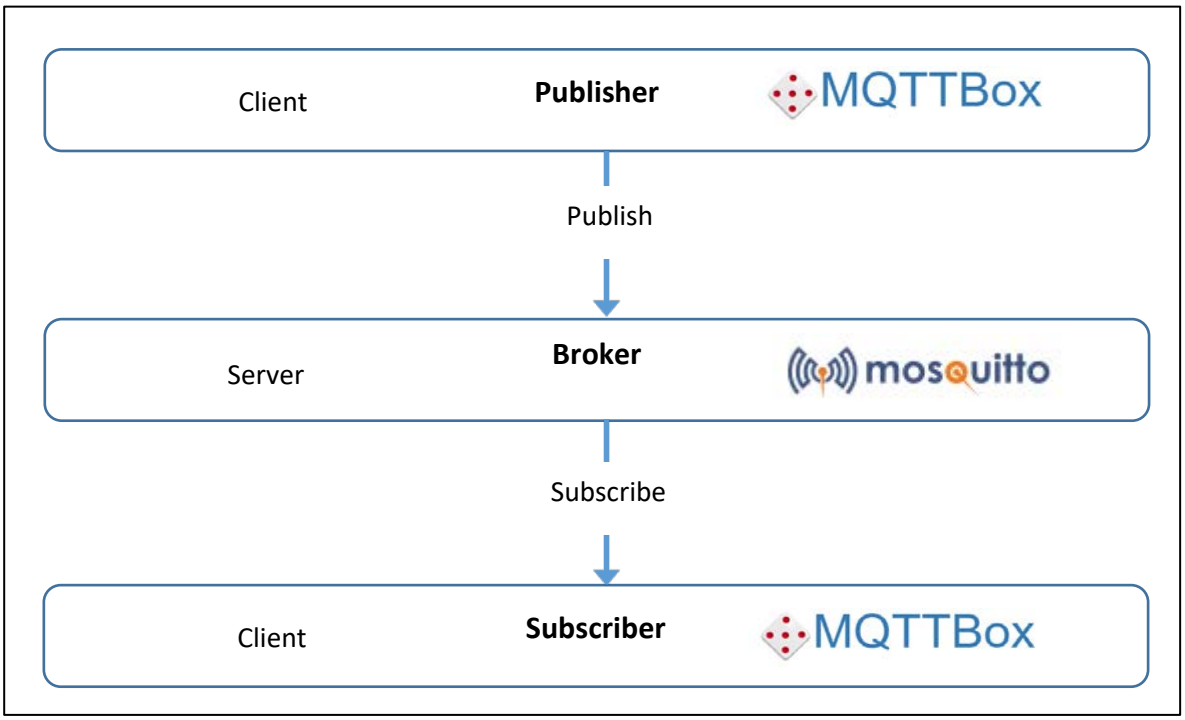


Figure 122: Test infrastructure of the MQTT protocol

The MQTT broker is common to everyone. We can either set it up on a local machine or use a public test MQTT broker. In our case, we will use the public Mosquitto MQTT broker.

7.4.7 Setting up a publisher and a subscriber MQTT

We use the MQTTBox MQTT client.

- Launch the MQTTBox software.
- **MQTTBox > Create MQTT Client**.

A MQTT client needs to know the address of the Broker:

- **Protocol**: mqtt / tcp
- **Host** : test.mosquitto.org

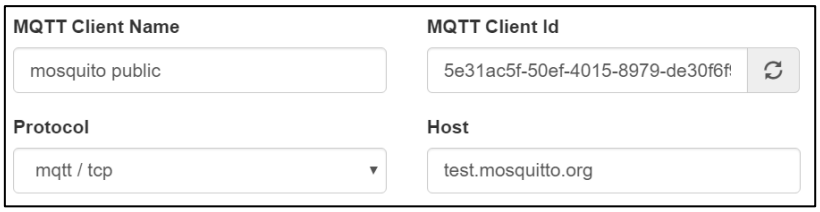


Figure 123: Configuring an MQTT Client in MQTT Box

You can now subscribe on a topic of your choice and publish on the same topic. The subscriber should receive the data. The Figure 124 show a publisher (on the left) sending data "test" on the topic "lorawan". The subscriber (on the right) receives the data "test" as it has subscribe to the topic "lorawan".

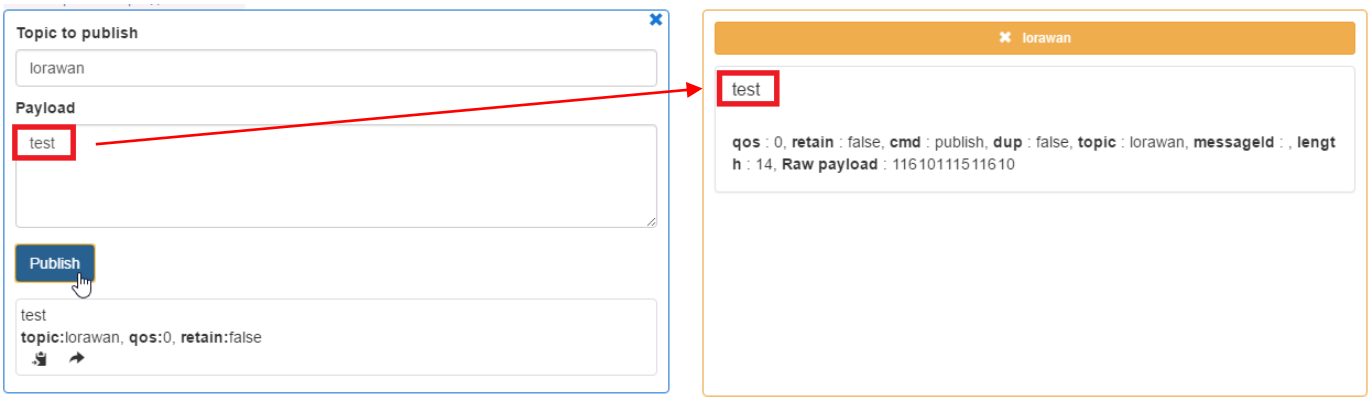


Figure 124: Test of the Mosquitto broker with MQTTBox client

7.5 Exporting data with MQTT protocol

There are several possibilities to use the MQTT protocol with our LoRaWAN server. It depends who is the broker and who will be the publisher/subscriber. We will see both cases but some are easier to set up.

7.5.1 LoRaWAN server as a MQTT broker

The first network architecture is simple and is represented on the Figure 125.

- The LoRaWAN server is the MQTT Broker.
- You need to subscribe **(1)** to the proper topic to receive.
- You need to publish **(2)** on the proper topic to send data.

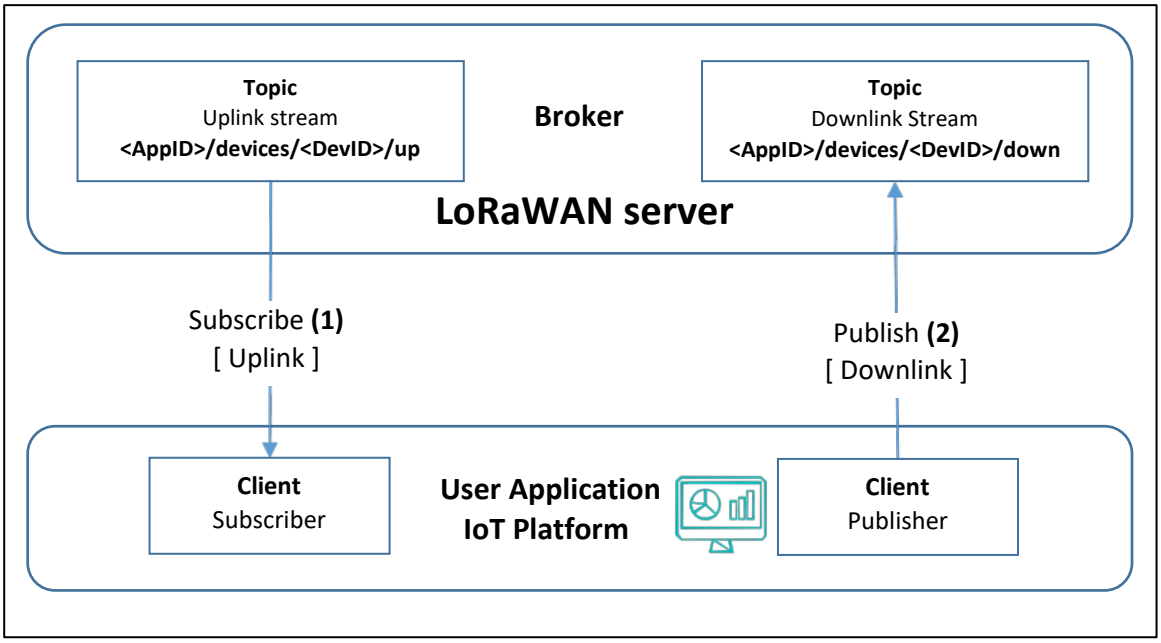


Figure 125: MQTT connection with LoRaWAN server as a Broker

The configuration parameters that your MQTT client need to know are in the documentation of your server LoRaWAN:

- The URL of the MQTT Broker exposed on your LoRaWAN server.
- The Username
- The Password
- The Topic to subscribe when you want to receive data.
- The Topic to publish when you want to send data.

You can find on our website www.univ-smb.fr/lorawan, the configuration details for many LoRaWAN server that can act as a MQTT Broker.

7.5.2 LoRaWAN server as a MQTT client

This second network architecture is represented below in Figure 126.

- An MQTT Broker is placed between the LoRaWAN server and the User Application.
- The LoRaWAN server publishes **(1)** to the appropriate topic to send data to the broker.
- You must subscribe **(2)** to the appropriate topic on the broker to receive data.
- You must publish **(3)** to the appropriate topic on the broker to send data.
- The LoRaWAN server subscribes **(4)** to the appropriate topic to receive data from the broker.

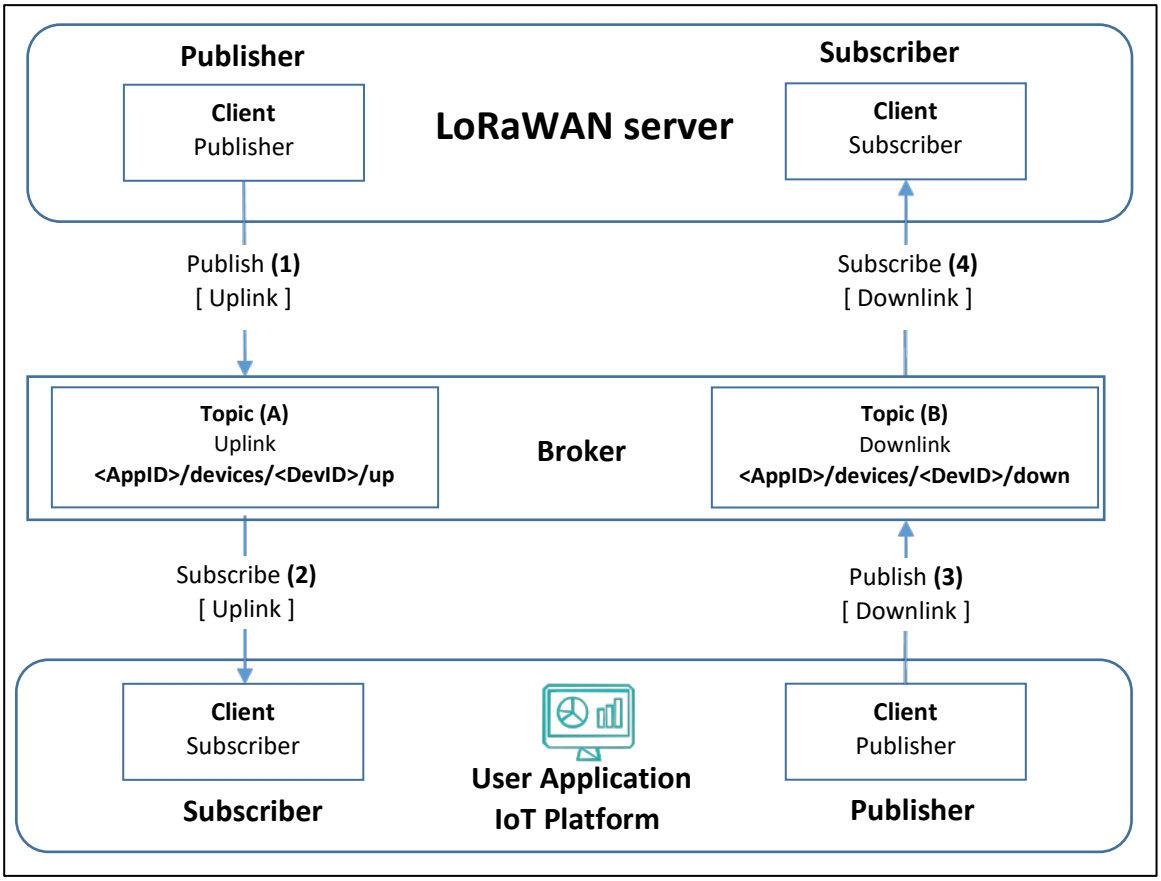


Figure 126: MQTT connection with LoRaWAN server as a MQTT client.

The configuration parameters for the "LoRaWAN server" and "User Application" MQTT client depend on where you set up your broker, but you always need the following information:

- The URL of the MQTT Broker
- The Username to connect to your Broker.
- The Password to connect to your Broker.
- The Topic you choose for the reception of your data [**(A)** in Figure 126].
- The Topic you need to publish to send your data [**(B)** in Figure 126].

In this example, we use Actility (v3.6.0) LoRaWAN server, the public "HiveMQ" broker and MQTTBox client.

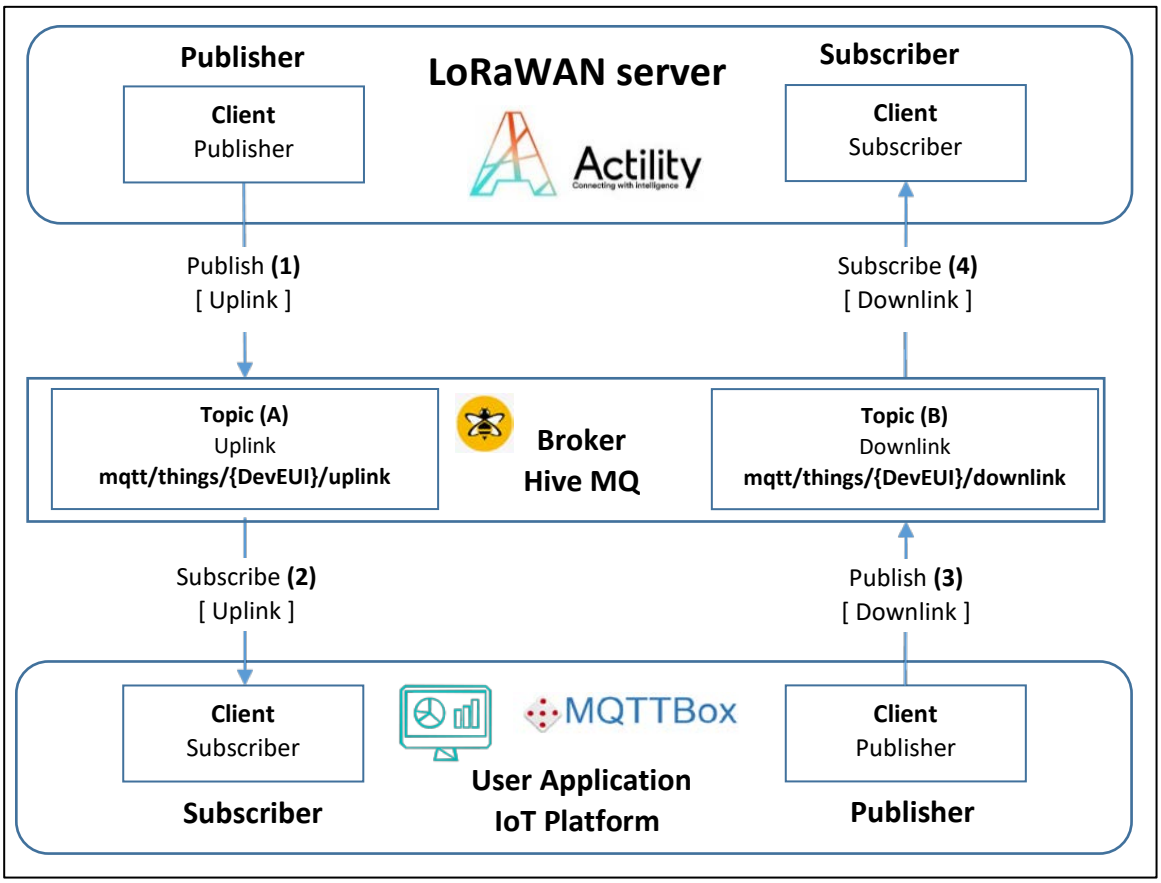


Figure 127: Test of the LoRaWAN server as an MQTT client

In Actility, we first need to create and configure the MQTT client: **Actility > Connections > TPX > MQTT** than you enter the following parameters:

- Hostname: **broker.hivemq.com:1883**
- MQTT Username: test
- MQTT Password: test
- Uplink topic pattern: **mqtt/things/{DevEUI}/uplink**
- Downlink topic pattern: **mqtt/things/{DevEUI}/downlink**

You need to change {DevEUI} by your end-device EUI.

- We now assign this new connection to your end-device: **Actility > Device > List > Your Device > Connections > The MQTT Connection you have just created**.

We now create the MQTT client.

- With MQTT Box, create a client with the same parameters used above.
- Create un subscriber on the topic "**mqtt/things/{DevEUI}/uplink**". You should receive uplink data.

- Create un publisher on the topic "**mqtt/things/{DevEUI}/downlink**". Then enter the following message to send a downlink frame:

```
{
  "DevEUI_downlink": {
        "Time": "2021-12-12T15:38:46.882+02:00",
        "DevEUI": "Your-Device-EUI",
        "FPort": "1",
        "payload_hex": "9e1c48525120002200 20e3831071"
  }
}
```

You should receive the downlink message on your LoRaWAN end-device.

7.6 Using an IoT Platform

An IoT platform is not specific to the LoRaWAN protocol. On the contrary, it combines many protocols and heterogeneous network in one place in order to help companies to manage their assets and build an end-user application.

There are hundreds of IoT Platform and they all have specific benefits and target different market. The connexions between the LoRaWAN server and the IoT Platform is often easy and most of the time, they support HTTP POST and MQTT. Some IoT Platform have partnership with LoRaWAN server in order to ease the connection. You can see in the figure below a few of the direct connections available in ACTILITY.

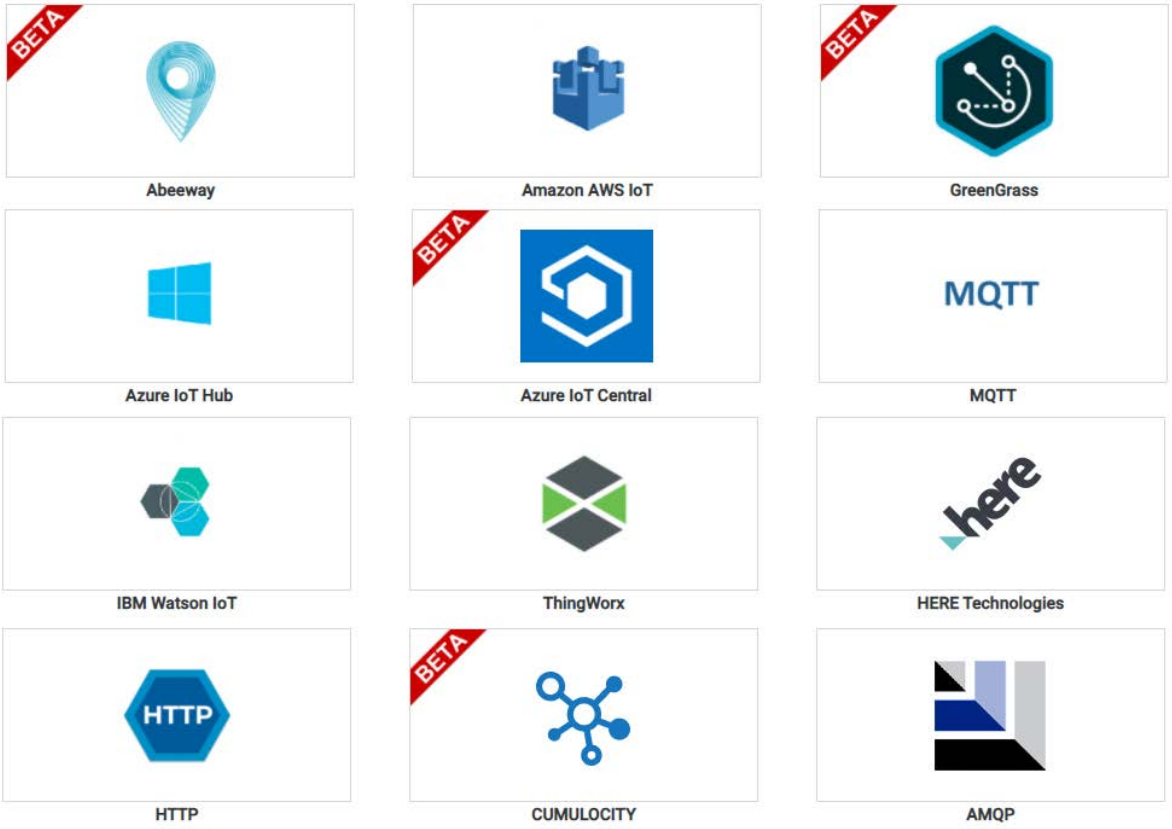


Figure 128: Connections from ACTILITY to some IoT Platform

Here is a few examples of IoT Platform:

- Vertical M2M –[CommonSense IoT Platform]: Independent Software Vendor specialized in Industrial & B2B IoT solutions with 13+ year-experience in the IoT market .
- IoThink – [KHEIRON IoT suite]: KHEIRON is flexible and customized IoT solutions to system integrators, device makers, machine builders and network operators.

IoT Platform are usually available as a cloud base service or on premises.

We are going to try the Vertical M2M (CommonSense IoT Platform) and connect it to our LoRaWAN server. CommonSense IoT Platform, enables territories, utilities, telecom operators and industrial customers, to deploy large-scale IoT projects by solving the 3 critical issues:

- dealing with heterogeneity of devices and IoT technologies: LoRaWAN but also many others such as Sigfox, NB-IoT, LTE-M, cellular 2G-3G-4G-5G…
- securely managing the fleet of heterogeneous end-devices: advanced supervision, alerting, commands, reporting and management features.
- easily connecting all IoT data and end-devices to customer's business applications: this is done by a lowcode/nocode solution (called IoT APP STUDIO module) to design and build full customizable IoT applications for vertical markets such as smartcity, smartwater or smartbuilding.

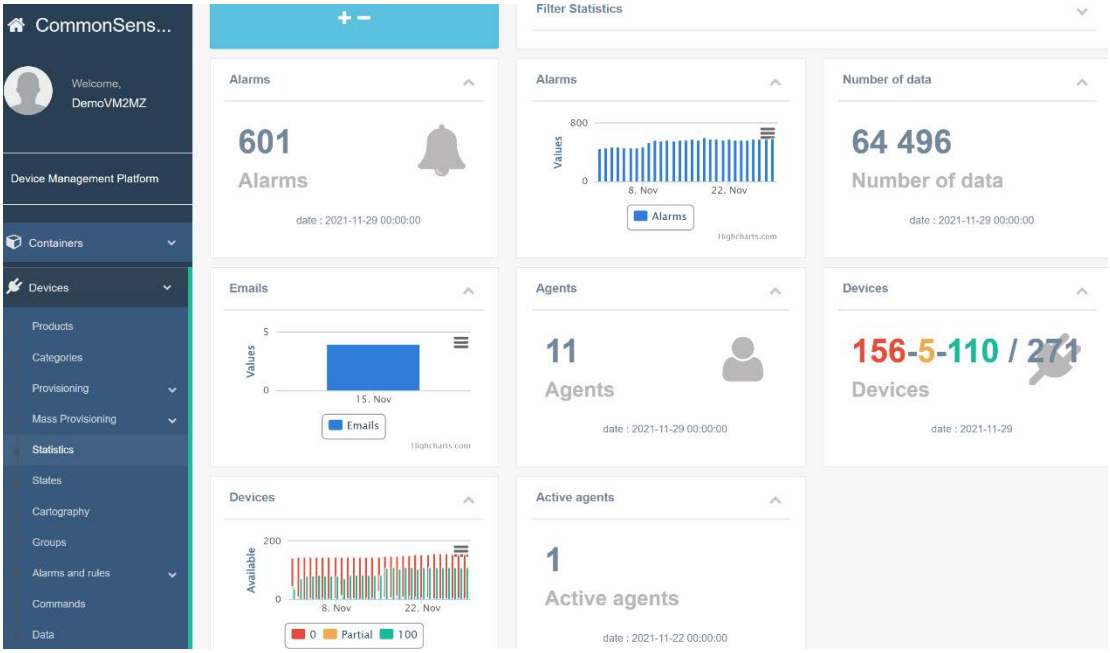


Figure 129: CommonSense IoT Platform

8 Designing your own LoRaWAN end-device

In this chapter, we are going to list all the LoRaWAN end-device architectures. Of course, each of them has its own advantages and disadvantages so we will study and summarize them in the last section. A LoRaWAN end-device needs:

1. A user firmware.

The user firmware is dealing with the sensor measurement process, calculation, wake-up and low-power transition, user interface (push button, Led…) and any other things that answer the client need. This has nothing to do with the LoRaWAN protocol.

2. A LoRaWAN protocol stack.

This is where all the LoRaWAN protocol executes the sequential routine of the MAC Layer to make a data transmission. If the end-device is LoRaWAN certified, it should stick to the LoRaWAN specification to ensure a proper and secure transaction. The protocol stack is region dependent.

3. A LoRa radio interface.

A transceiver modulate the RF signal following the LoRa modulation standard LoRa PHY. The transceiver can cover multiple regions around the world but the antenna design is region specific.

8.1 LoRaWAN stacks available

If we use an end-device that already integrates a LoRaWAN stack, then we don't have to worry about integrating it into our component. On the other hand, for any other designs, we must integrate it ourselves. Here is a short description of the LoRaWAN stacks available:

1. The most famous stack is developed by Semtech. The stack is called LoRa MAC-Node™ and is available for all microcontrollers. It is very well maintained and follows the LoRaWAN specification.
2. The other well known stack is LMIC (**L**oRa **M**ac **I**n **C**). This is the preferred stack when using Arduino boards.

There are other stacks available. For example, ST proposes its own stack, that is based on the **LoRa MAC-Node™** from Semtech, but which has been improved to better match the specificities of STM32 microcontrollers. Arm MBED also proposes a LoRaWAN stack, but it's not open source.

8.2 Microcontroller + Transceiver architecture

8.2.1 Presentation of this architecture

In this type of architecture, one microcontroller manages both the LoRaWAN stack and the user application firmware. This requires to have a LoRaWAN stack available.

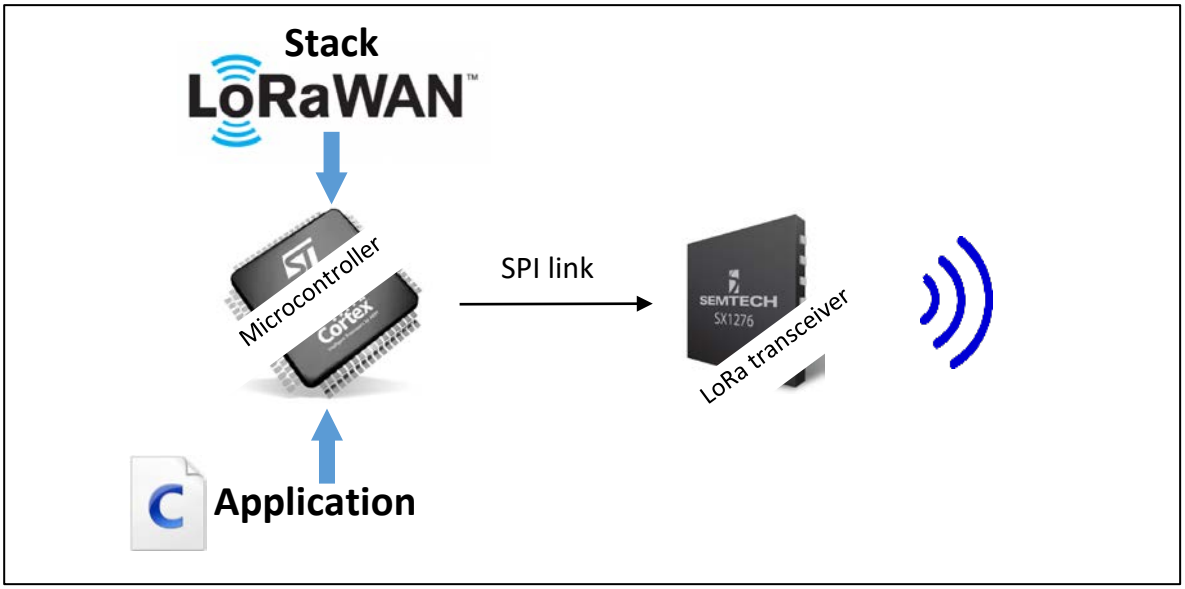


Figure 130: LoRaWAN end-device with microcontroller + transceiver

The SX1262 is an example of a LoRa transceiver from Semtech. It manages the physical part of the protocol: modulation, preamble detection.... The complete management of the LoRaWAN protocol is carried out by a software stack implemented inside the microcontroller. The microcontroller drives the transceivers with a SPI bus.

This choice is quite successful and optimized in terms of power consumption, because there is only one microcontroller that manages everything. On the other hand, it is a more complex solution as it is necessary to dive into the stack. Indeed, even if the user application and the stack are well separated in the code, it is quite a challenge to be certain that one does not interfere the other as they run on the same MCU at the same time. You always have to be very careful not to take resources away from one another.

8.2.2 Example of a development board

We can find many development boards for Semtech transceivers on their website. Here is one propose by ST. The P-NUCLEO-LRWAN1 development board associates a STM32L073 microcontroller (Cortex M0+) with a SX1272 transceiver.

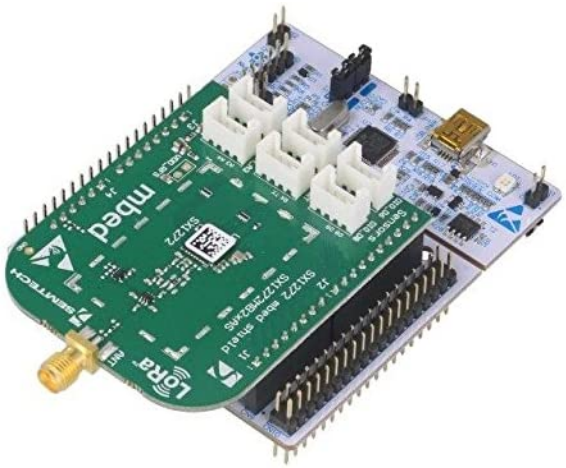


Figure 131: P-NUCLEO-LRWAN1 package

8.2.3 Implementation solution

After the prototyping period, you can move on to the realization of your own PCB. If you want to do the soldering yourself, you will need a reasonably equipped workshop. An interesting solution is to use breakout boards, which already contains the transceiver a few useful component inside.

Figure 132: Example of a transceiver module (NiceRF)

8.3 Standalone LoRaWAN module architecture

8.3.1 Presentation of this architecture

In this type of architecture, we use a stand-alone module that includes both:

- A microcontroller (which has the application firmware and the LoRaWAN stack).
- A transceiver.

There are several components in the module. However, the transceiver is not integrated in the microcontroller as we will see in section 8.4. However, from the programmer's point of view it is almost the same.

This solution has the advantage of simplifying the hardware part of the previous solution. On the other hand, we still have the proximity of the application firmware and the LoRaWAN stack to manage. The module has a number of peripherals available (ADC, I2C, UART, GPIO...) to carry out the sensor connection for the user application.

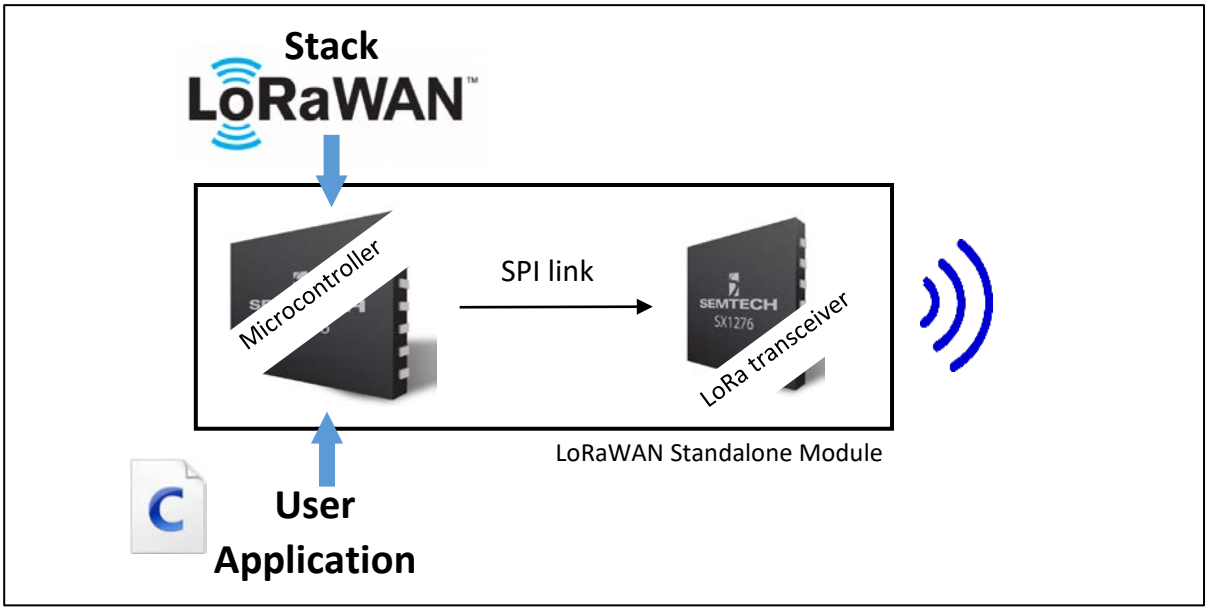


Figure 133: LoRaWAN Standalone Module

8.3.2 Example of a module

The Murata CMWX1ZZABZ can work in standalone mode. It will not need any other components to work but the antenna with impedance matching circuit.

Figure 134: Module integrating a LoRaWAN stack and a LoRa transceiver

The Figure 135 shows the diagram of the Murata module.

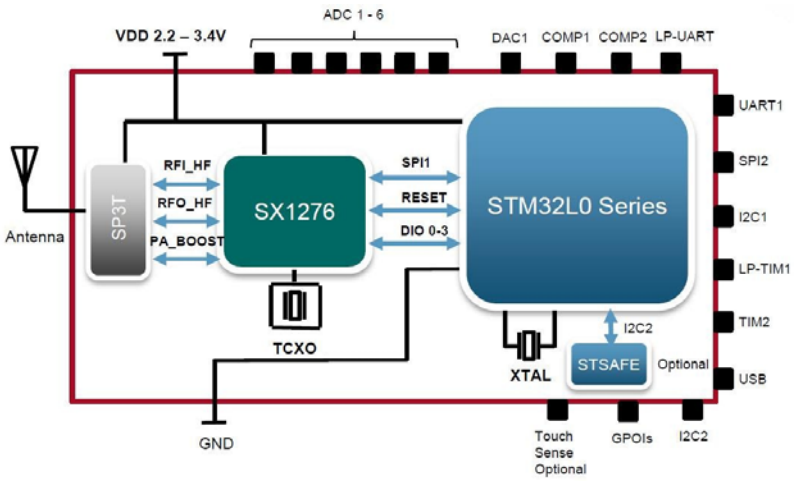


Figure 135: Inside the Murata ABZ module

Other module may integrate a SMA connector on the circuit to ease the antenna interface.

8.3.3 Example of a development board

ST has a development board for the CMWX1ZZABZ component. It is the Discovery B-L072Z-LRWAN1 board.

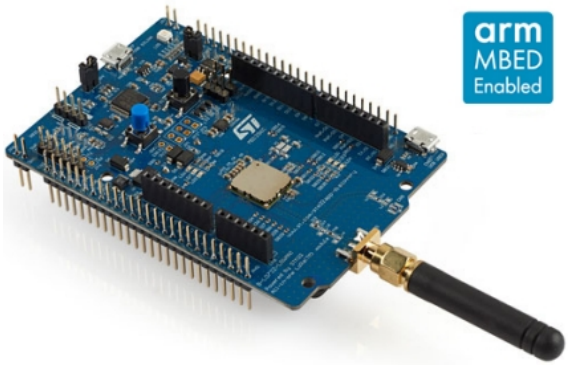


Figure 136: ST B-L072Z-LRWAN1 Discovery Board

8.4 Microcontroller + LoRaWAN module architecture

8.4.1 Presentation of this architecture

In this type of architecture, the microcontroller only manages the user application firmware. An external module, driven by a serial link, manages the whole LoRaWAN protocol. The module is exactly the same type we saw in the previous chapter 8.3.

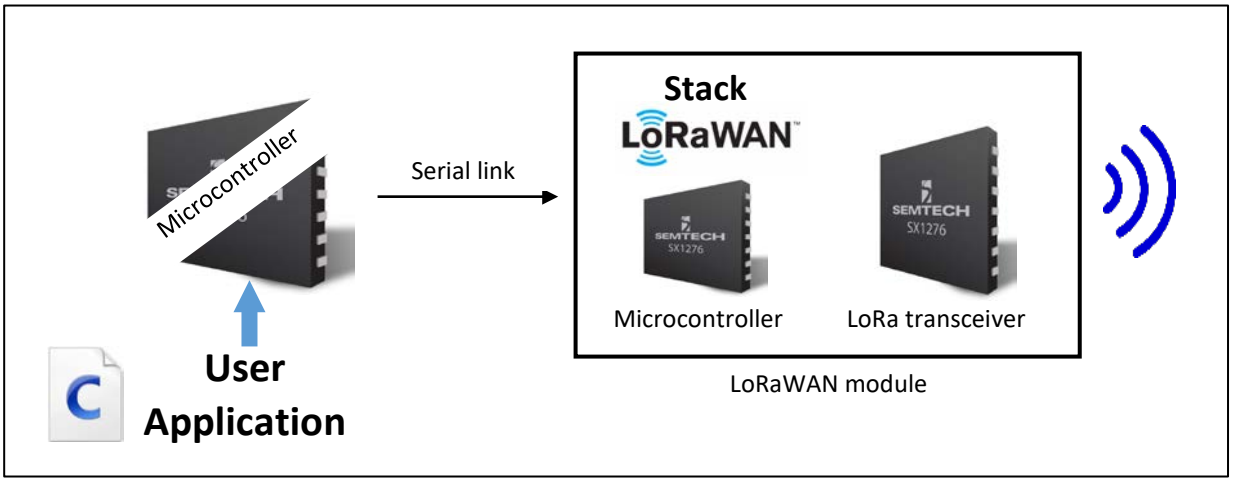


Figure 137: LoRaWAN end-device with microcontroller and LoRaWAN module

We can choose any microcontroller (even 8 bits) as the LoRaWAN stack executes on the module and therefore does not take any resources on the user application MCU.

8.4.2 Example of LoRaWAN module

Here are two commonly used modules:

- RAK Wireless : RAK3172

- Murata: CMWX1ZZABZ: This is the same module as the one seen in paragraph 8.2, but the firmware respond to a set of AT commands sent from a master in serial link.

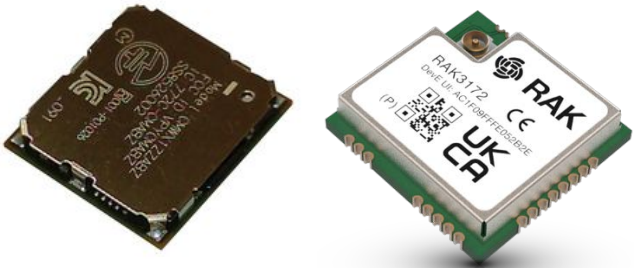


Figure 138: Module integrating LoRaWAN stack and transceiver

There are libraries available to use the AT command.

This choice has the considerable advantage of its simplicity. We do not have to manage anything of the LoRaWAN protocol because everything is done in the module. By sending simple AT command, the LoRaWAN end-device designer has only to focus on the user application.

The counterpart is obviously the fact that the overall system has two microcontrollers: one for the user firmware and one for the LoRaWAN stack management in the module. This will influence the price of the whole system, and of course, its consumption.

8.4.3 Example of development board

Here is one example of development board.

Figure 139: Arduino MKRWAN 1310: ATMEL 32 bits microcontroller + CMWX1ZZABZ Module

8.5 Wireless LoRaWAN Microcontroller Architecture

8.5.1 Presentation of this architecture

STMicroelectronics developed the first microcontroller with LoRa transceiver integrated: STM32WL. It comes with two flavours: single core (STM32WLEx) or dual core (STM32WL5x). In this architecture, the LoRaWAN stack and the user firmware are in the same microcontroller but can be separated if we use the dual core solution. It is important to say that this component is built on the same silicon die.

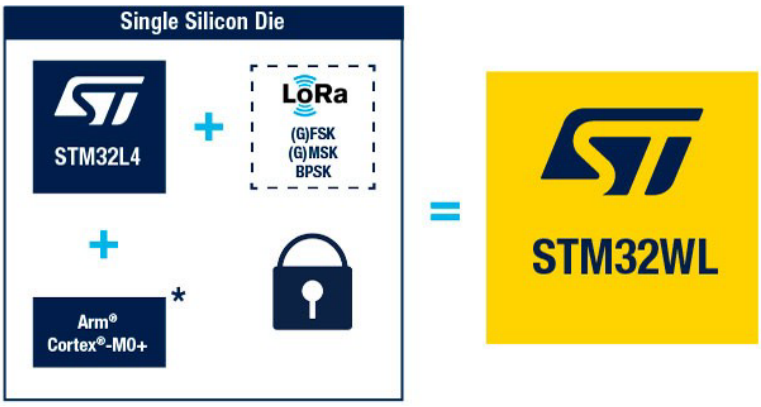


Figure 140: STM32WL dual-core microcontroller

This is the most interesting solution in terms of cost, power consumption and footprint.

8.5.2 Example of development board

STMicroelectronics propose its own Nucleo board for this microcontroller.

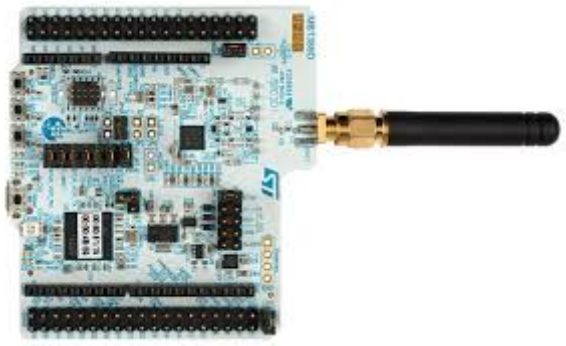


Figure 141: STM32WL Nucleo board

The STM32WL is also available in modules to ease the prototyping. We can quote the RAK3172 from RAK Wireless or the LoRa-E5 from Seed Studio.

Figure 142: RAK3172 (left) and LoRa-E5 (right)

8.6 Summary of architectures

A comparative summary of the solutions is given in Table 28. The characteristics provided are not quantified.

	Microcontroller + Transceiver	LoRa Standalone Module	Microcontroller + LoRa Module	Wireless Microcontroller
Footprint	Medium	Low	High	Very low
Cost	High	Medium	Very high	Low
Code complexity	High	High	Low	High

Table 29: Advantages and disadvantages of the different architectures

9 Setting up your own LoRaWAN server

The LoRaWAN network we have been working on so far was a hybrid network (see chapter 5.1.4): we were using our own gateway but the LoRaWAN servers did not belong to us. We will now install our own LoRaWAN server to create a complete private network (see chapter 5.1.2).

Demonstration of the overall installation process are available in our website www.univ-smb.fr/lorawan for ChirpStack and The Things Stack.

It is also possible to set up other non-open source private LoRaWAN server on premises.

9.1 Preliminary information

9.1.1 Order your own gateway

Switching to a private network is only possible if you have your own gateways. You need to reconfigure them in order to point to the LoRaWAN servers that you have set up.

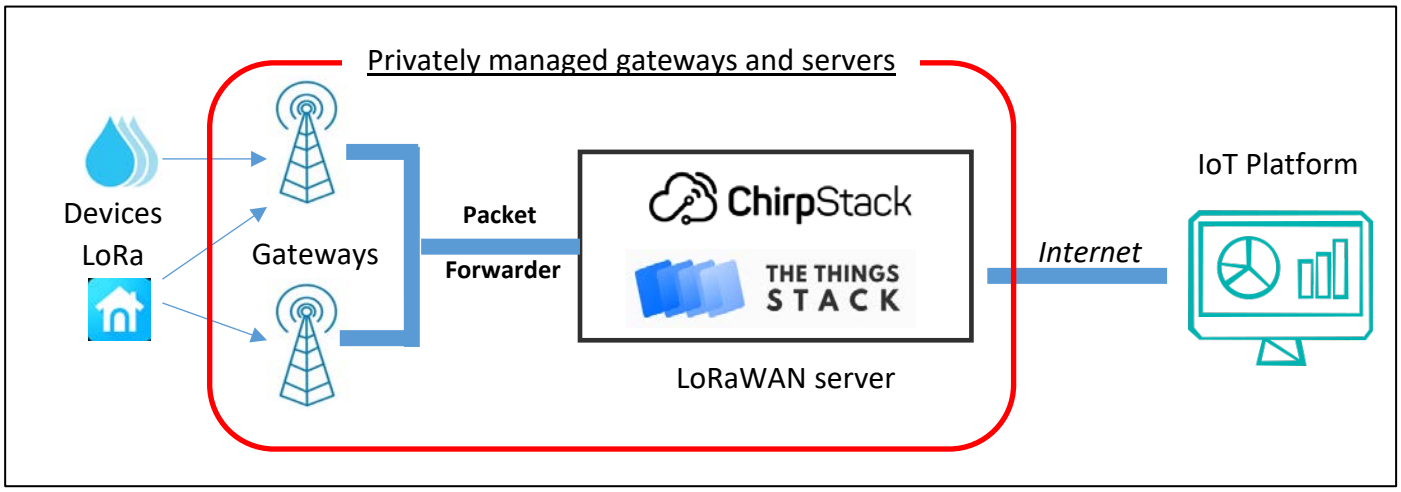


Figure 143: Infrastructure of a private LoRaWAN network

9.1.2 Check the Packet Forwarder

Some LoRaWAN server impose to use certain Packet Forwarder. For ChirpStack and The Things Stack, Basic™ Station is the preferred choice for deployment. For development, Semtech UDP Packet Forwarder is still available.

9.1.3 Gateway and Server LoRaWAN location

The gateway needs to exchange data with your LoRaWAN server. There are many possibilities:

- The gateway and the LoRaWAN server are in the public Internet. In that case, they both have a public IP address and they can easily connect together.

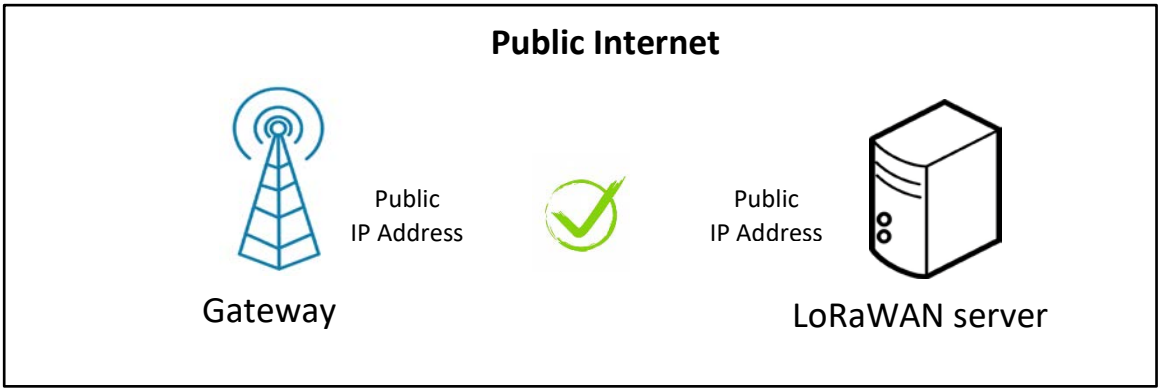


Figure 144: Gateway and LoRaWAN server in public Internet

- The gateway and the LoRaWAN server are in the same private network. In that case, they both have a common network address and they can easily connect together.

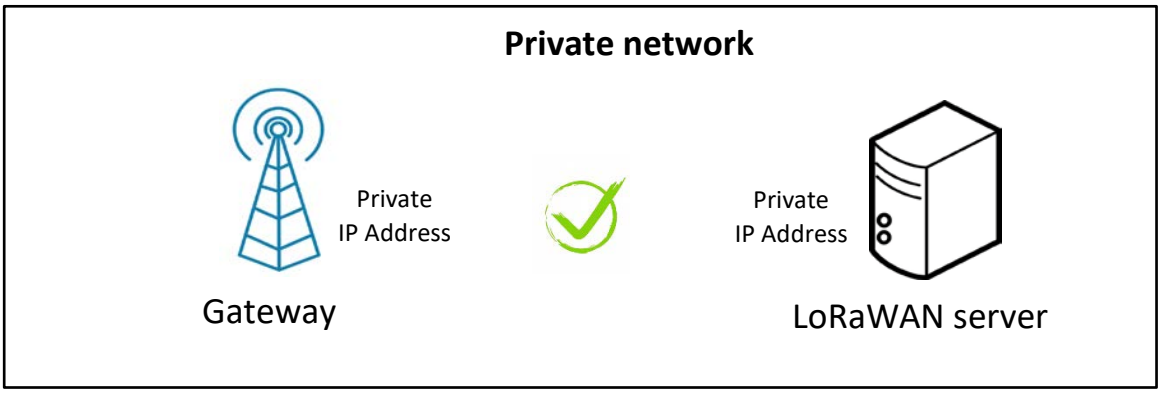


Figure 145: Gateway and LoRaWAN server are in the same private network

- The gateway is in a private network and the LoRaWAN server is on the public Internet. In that case the ports translation process of the router will allow the gateway to connect to the Server. Once the translation is active on the router, the LoRaWAN server can also send data to the gateway. That works fine.

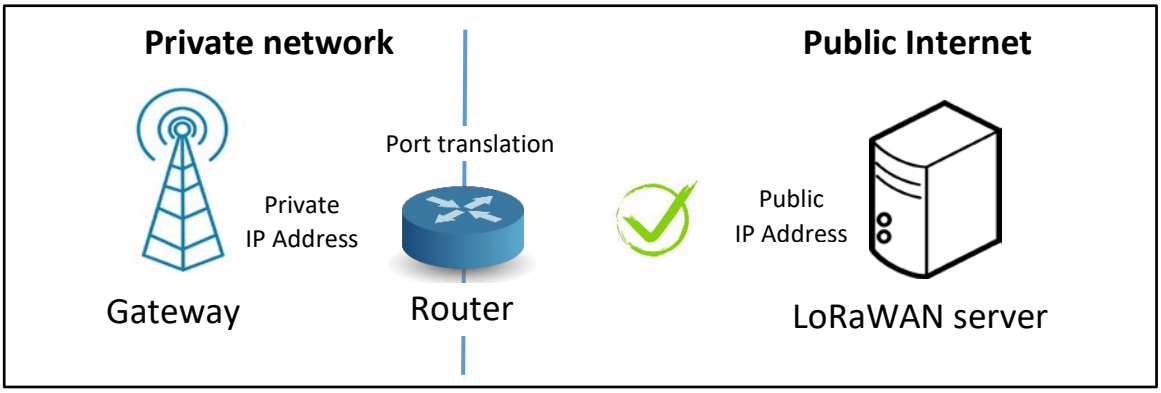


Figure 146: Gateway in private network and LoRaWAN server on public Internet

- The gateway is on the public Internet and the LoRaWAN server is in the private network. In that case, the gateway will not be able to connect to the Server if no static port translation are manually configured in the router. The same situation occurs if the gateway and the LoRaWAN server are in two different private network.

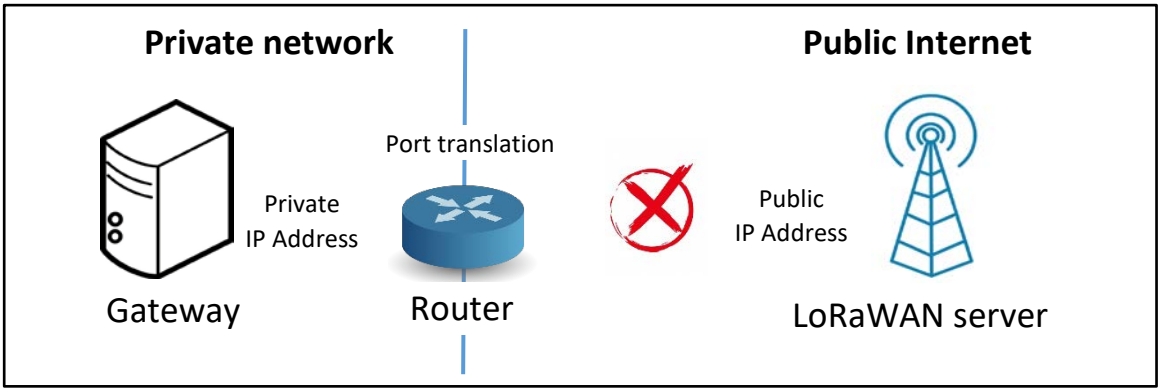


Figure 147: Gateway on Public Internet and LoRaWAN server in a private network

9.1.4 LoRaWAN server host

To install our LoRaWAN server, we need a host connected to the internet and accessible from the gateway. This can be:

- On the gateway itself.
- Your own local PC (Windows, Linux, MacOS).
- An external computer (Raspberry PI…).
- A virtual machine (Virtual Box, VMware...).
- A server from a provider (OVH, AWS...).

The diversity of architectures makes the installation quite challenging, so we need to standardize the process. Docker and Docker Compose are developed for this purpose. They isolate the services by using containers, so they become:

- Independent of the OS Host (Linux, MacOS, Windows).
- Independent of the Hardware Host (ARM, X86 architecture).
- Independent of the other services installed on the machine.

Docker allows to perform installations in an extremely simple way, without having to manage dependencies and libraries specific to each operating system (Windows, Linux, MacOS) and processors used (ARM, X86). It will act as an abstraction layer that will isolate the installed service from the rest of the system.

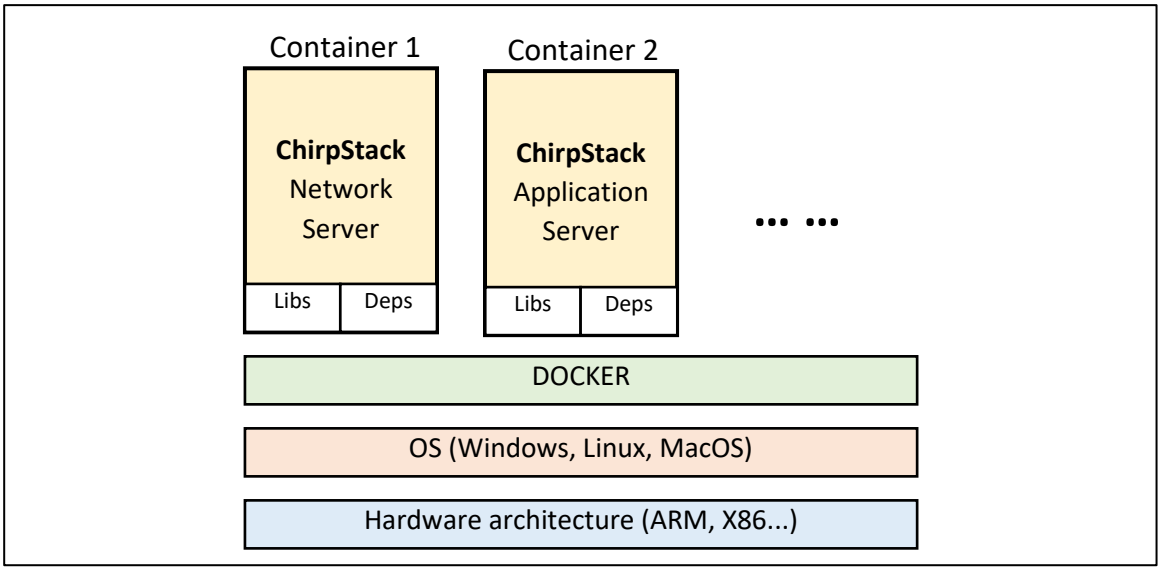


Figure 148: Using Docker and Docker Compose to install a LoRaWAN server

The installation of Docker and Docker Compose is platform dependant and is done as follow:

- Installing **Docker:** https://docs.docker.com/engine/install/
- Install **Docker Compose:** https://docs.docker.com/compose/install/

9.2 ChirpStack LoRaWAN server

9.2.1 The ChirpStack project

ChirpStack is an open-source project with the following simplified architecture.

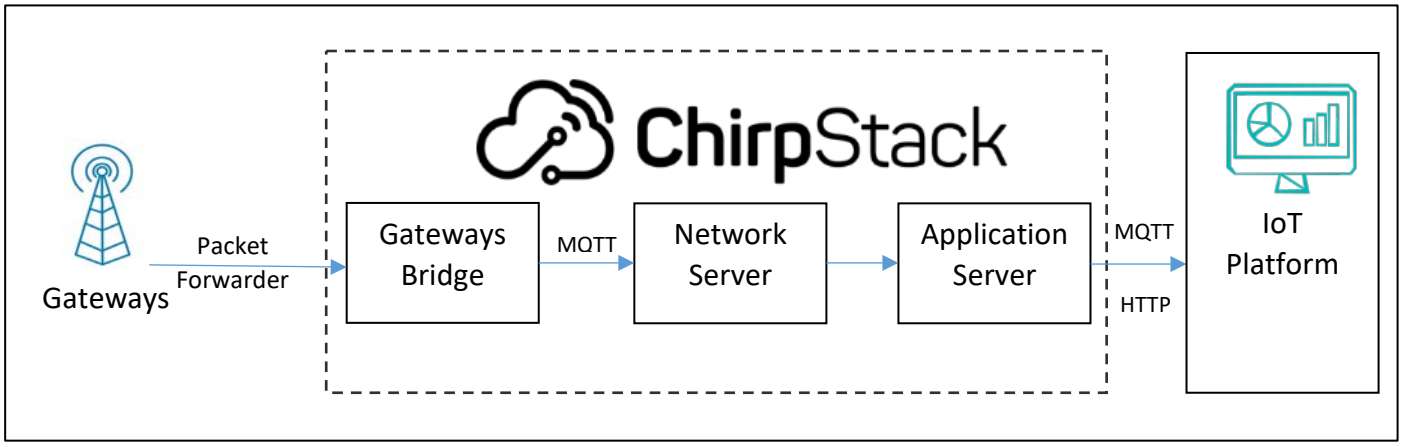


Figure 149: ChirpStack architecture

ChirpStack has the two usual services: **Network Server** and **Application Server**. There is also another service called **Gateway Bridge**. Thanks to the Gateway Bridge, ChirpStack is able to communicate with several Packet Forwarder without impacting the Network Server side. Gateway Bridge converts the different Packet Forwarder protocol formats into a common MQTT protocol used by the ChirpStack Network Server.

ChirpStack Gateway Bridge can interface with the **Semtech UDP Packet Forwarder** or **Basic Station™**.

9.2.2 Setting up ChirpStack with Docker

You can use Docker if you install ChirpStack on a Raspberry PI, on a cloud Server or on your local PC with any OS distribution and processor. There is nothing special other than following the ChirpStack documentation: [ChirpStack documentation.]

The demonstration of ChirpStack installation using Docker is available here on video: [www.univ-smb.fr/lorawan/videos].

Figure 150 shows the services (Docker container) generated with Docker Desktop on Windows.

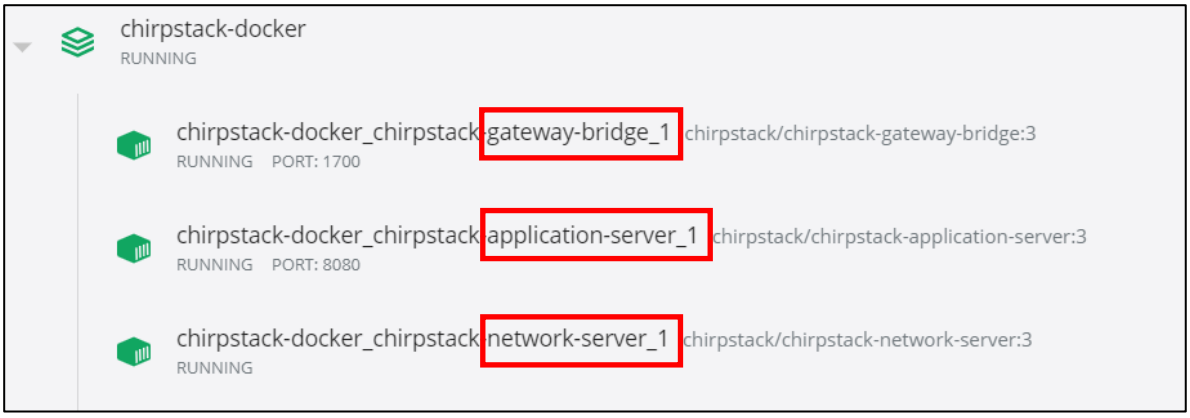


Figure 150: ChirpStack Docker containers (Docker Windows Desktop)

Figure 151 shows the services (Docker container) generated with Docker on a Linux distribution.

```
NAMES                                          PORTS
chirpstack_chirpstack-application-server_1     0.0.0.0:8080->8080/tcp
chirpstack_chirpstack-network-server_1
chirpstack_chirpstack-gateway-bridge_1         0.0.0.0:1700->1700/udp
```

Figure 151: ChirpStack Docker containers (Docker on a Linux Distribution)

We can notice that **Gateway Bridge** service listens on port 1700/udp because we use Semtech UDP packet forwarder. This is the port where our gateway must send its data. The Application Server listens on the 8080/tcp port. This is the web interface access to configure the LoRaWAN server.

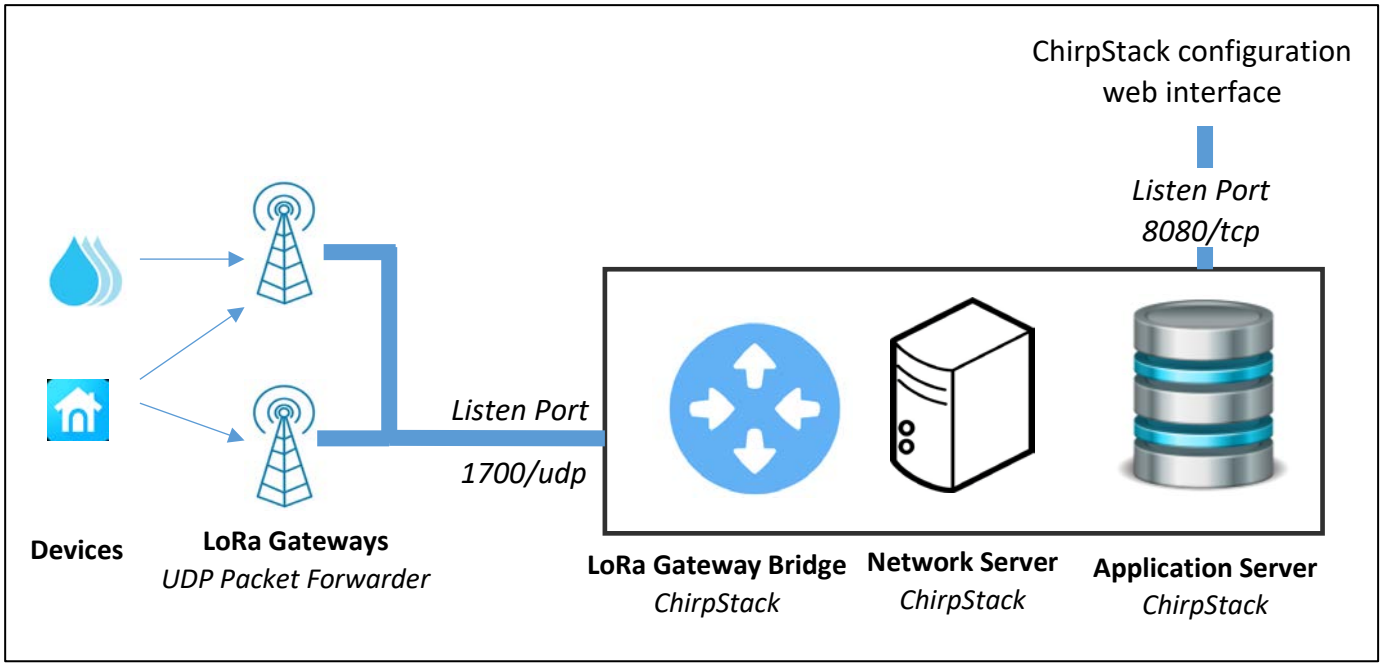


Figure 152: Overall architecture after installing ChirpStack

9.2.3 Setting up ChirpStack on a Raspberry Pi gateway

The Raspberry Pi is often used to build a gateway and it is possible to install a build-in LoRaWAN server. Gateway and LoRaWAN server are therefore in the same package. ChirpStack provides an image called **chirpstack-gateway-os-full**. This image is ready to run and includes

- An open-source Linux based embedded OS which can run various LoRaWAN gateways (Semtech SX1301 LoRa CoreCell, IMST, RAK, RisingHF…)
- A setup of Gateway Bridge, Network Server and Application Server.

This is a one-box-solution to collect and expose IoT data quickly and easily.

9.3 Configuring ChirpStack

We configure ChirpStack via a graphical web interface accessible via port 8080/tcp: http://@IP-ChirpStack:8080

The default Usernames and Password are:

- Username: admin
- Password: admin

The demonstration of ChirpStack configuration is available in video on our website www.univ-smb.fr/lorawan.

ChirpStack Application Server is able to connect to one or multiple ChirpStack Network Server instances. Global admin users are able to add new Network Servers during setup.

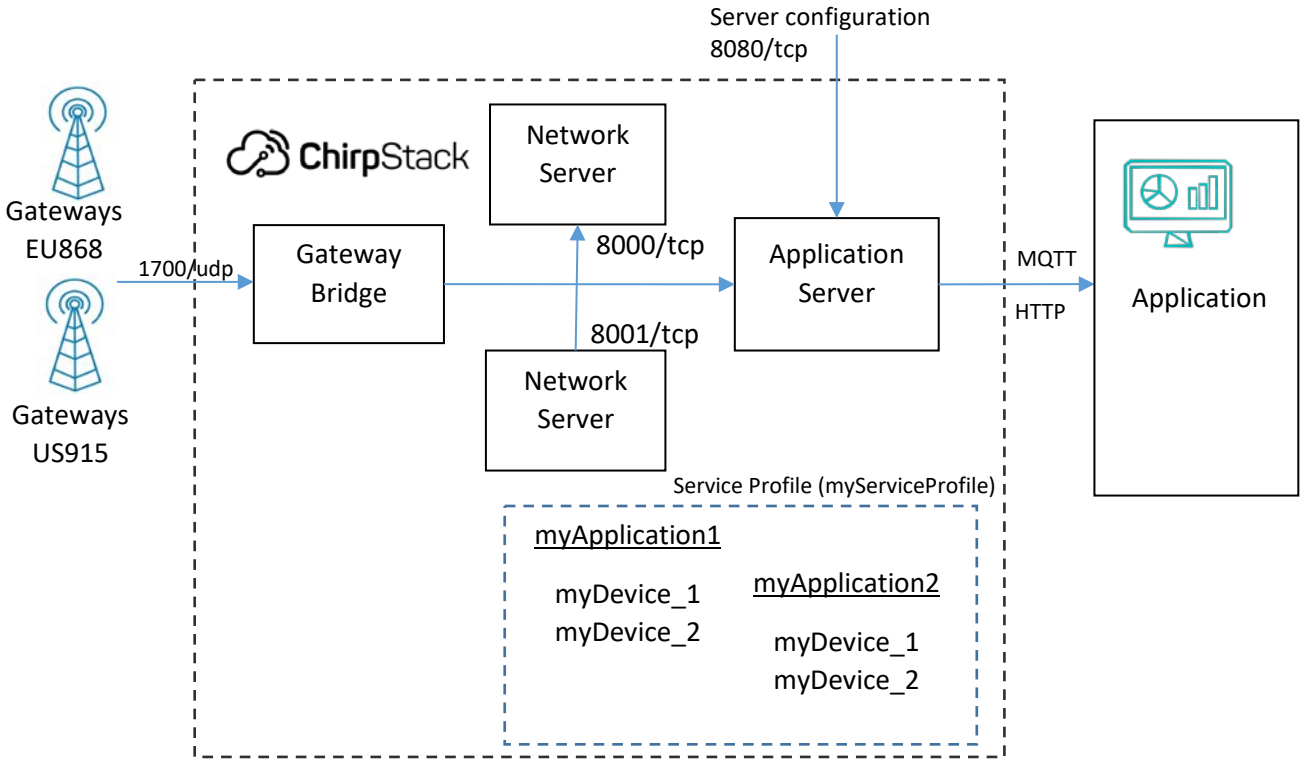


Figure 153: ChirpStack with multiple Network Server

In the **Application > myApplication > Devices > myDevice > LORAWAN FRAMES** tab, we can see the LORAWAN frames.

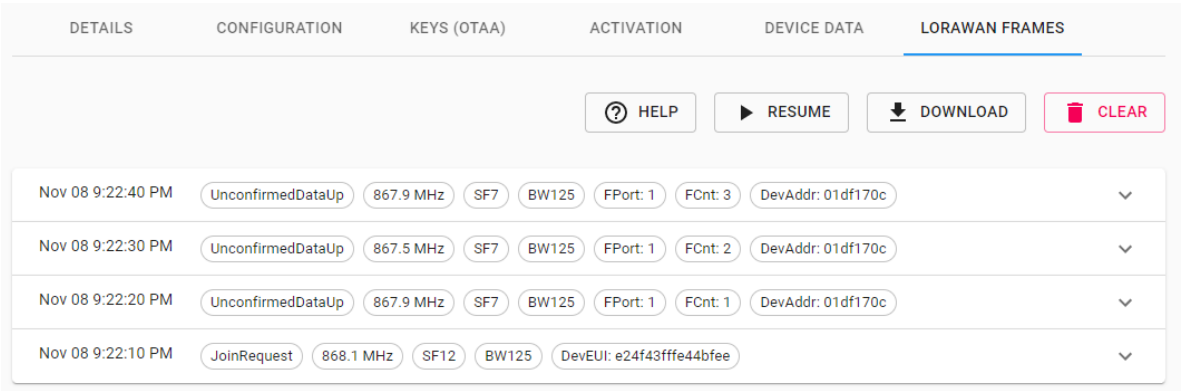


Figure 154: LoRaWAN Frames received in ChirpStack

9.3.1 Exporting user data

You can find in our website:

- The HTTP POST client request and URL format for downlink.
- The MQTT topic to subscribe for uplink.
- The MQTT topic to publish for downlink.

The demonstration of user data exportation using HTTP POST and MQTT for both uplink and downlink is available here on video: www.univ-smb.fr/lorawan.

10 Setting up your own User Application – IoT Platform

10.1 Choices overview

In Chapter 7, we explained how to export data from our LoRaWAN server (uplink), and to provide data to the LoRaWAN server (downlink). We will now look at the overall User Application and present several complete functional architectures. Our user Application needs to import, store, process, and display this data on a web page, this is what an IoT platform does.

Table 29 lists the different technological choices available for prototyping this User Application. This list is far from being exhaustive, but it lets us understand the different functionalities we need to build.

What do we want?	How can we do it?	Possible technological choices			
A Web page available for the user	*A Web server* *A User interface*	Node-RED	Grafana chronograf	kibana	NGINX APACHE SOFTWARE FOUNDATION
Graphics, tables, gauges, tables…	*A monitoring solution* *Libraries for Dashboard*				plotly Dash HIGHCHARTS
Data backup	*A database*		influxdb	elasticsearch	SQLite MySQL Prometheus
Data importation	*A HTTP Endpoint* *MQTT Subscriber or Publisher*		telegraf	logstash	python mosquitto **HTTP / MQTT**

Table 30: Technological choices to build an IoT Platform

10.2 Building an IoT Platform from scratch

A very common choice to build an IoT Platform from scratch is to use the open source **TICK** stack from Influxdata [www.influxdata.com]

- **T**elegraf: Data import
- **I**nfluxdB: Storage (backup)
- **C**hronograf: Format data and dashboard display
- [**K**apacitor: We don't use this service in this demonstration]

Another famous service for creating dashboard is Grafana. It has to same purpose than Chronograf. We will therefore include Grafana in our test. Once again, we use Docker and Docker Compose to set up the services for our User Application. Here is the list of containers:

- Telegraf
- InfluxDB (connexion via port 8680/tcp).
- Grafana (connexion via port 3000/tcp).
- Chronograf (connexion via port 8888/tcp).

When all containers are active, they are able to communicate each other using known IP addresses within the Docker infrastructure. These IP addresses can be resolved by the container name (influxdb, telegraf, ...).

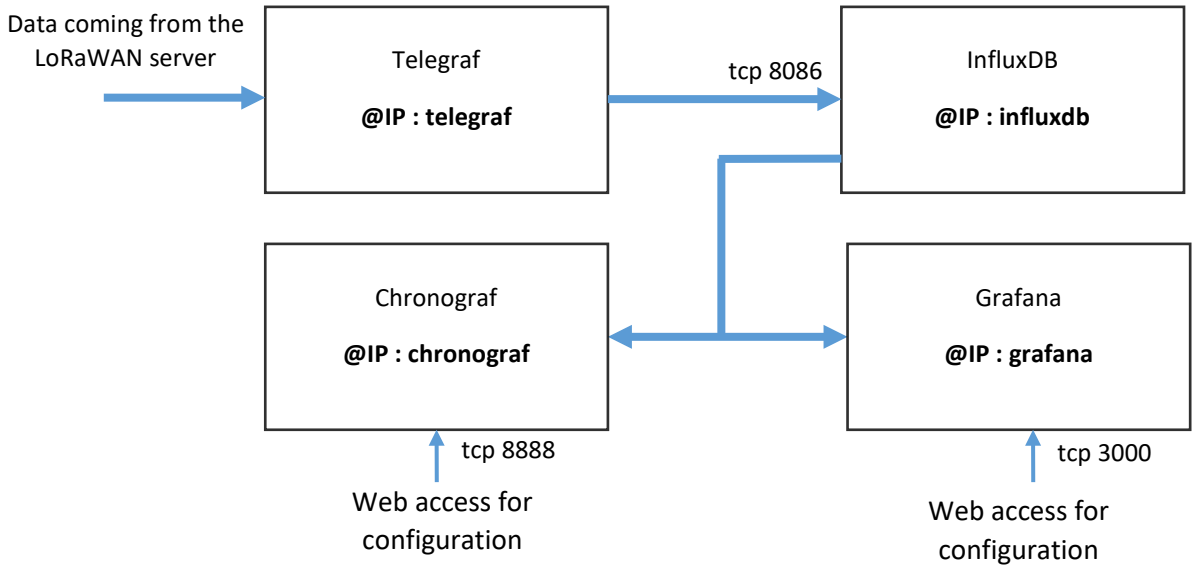


Figure 155: Docker containers for our User Application

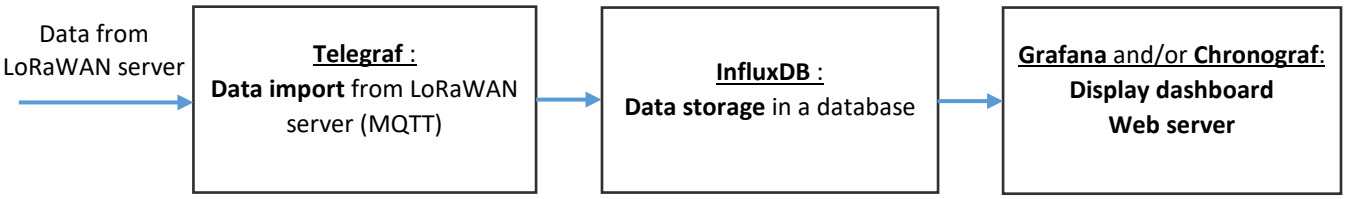


Figure 156: Telegraf – InfluxDB – Chronograf - Grafana

All this set up is available on Github on the following link:
https://github.com/SylvainMontagny/dashboard-lorawan

The demonstration to set up the entire IoT Platform is available here on video: www.univ-smb.fr/lorawan.

11 Advanced LoRaWAN

We now have a global vision of the LoRaWAN protocol. This should be enough for a simple application. However, some topics have been voluntarily hidden in order to make things easier to understand. This chapter is intended to go into more details and to learn new features.

11.1 The Join Server

11.1.1 The role of the Join Server

We know that the LoRaWAN 1.0.x protocol requires two keys to work:

- The NwSKey for authentication.
- The AppSKey for encryption.

The ABP activation mode provides these keys directly to the end-device. This simplifies the process but this is not the most secure way to provision the end-device. The recommended activation mode is therefore OTAA so that a new set of NwSKey and AppSKey is generated at each Join-Request. But who exactly manages this Join-Request?

Until now, we considered that this activation phase was handled by the Network Server. In reality, it is a specific entity called the Join Server that handles it. The Join Server is directly connected to both the Network Server and the Application Server and has a unique 8-byte identifier (EUI) called the JoinEUI that specify which Join Server will be used to process the activation phase.

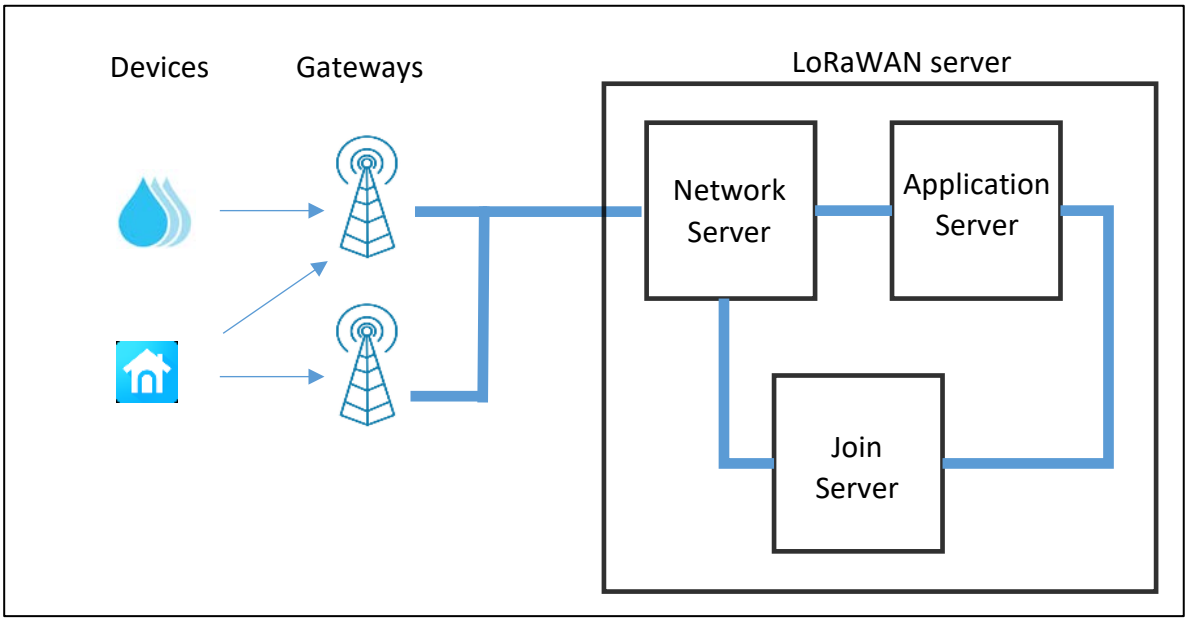


Figure 157: The Join Server

These three entities are often at the same location, but they can be on separated instances. For example, in TTN, you can choose for each end-device where is the associated Application Server and Join Server.

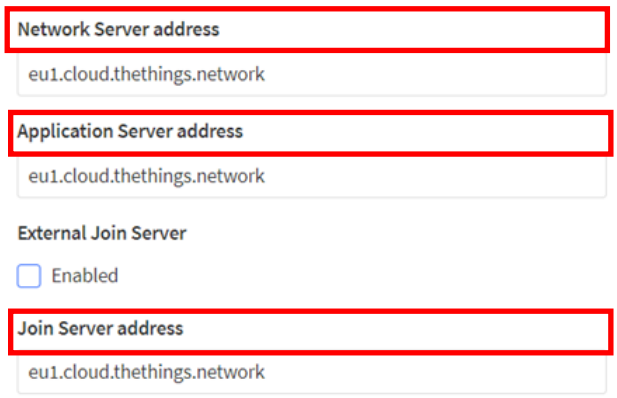


Figure 158: Location of the Network Server, Application Server and Join Server on TTN

Since the LoRaWAN protocol version 1.0.4, the JoinEUI replaces the AppEUI. You have therefore different field to fill in whether you chose a LoRaWAN version or another. On the first line of the Figure 159, we use the LoRaWAN version 1.0.3, hence the AppEUI field. On the second line, we use the LoRaWAN version 1.0.4, hence the JoinEUI field.

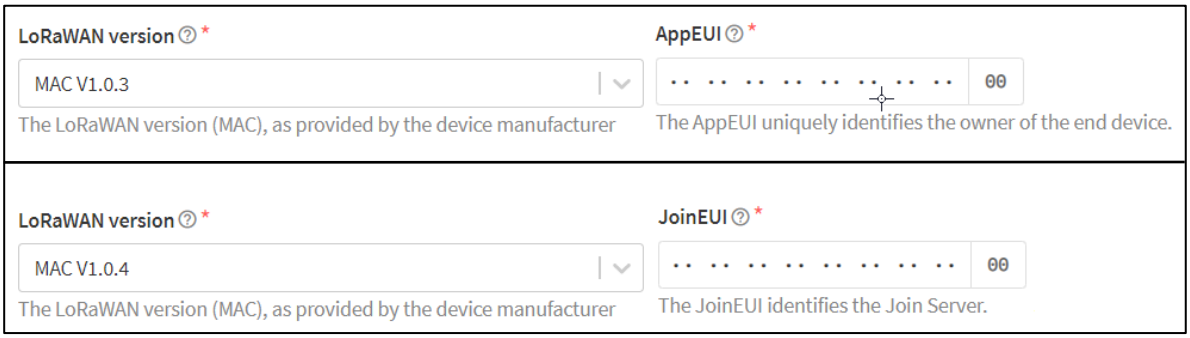


Figure 159 : AppEUI or JoinEUI choice depending on the protocol version

You must know the LoRaWAN version of your end-device as you need to specify it during the end-device registration.

Your Network Server, Application Server and Join Server can be from different operators. We can imagine having an Actility Network Server, The Things Industries Join Server, and Common Sense (IoT Platform).

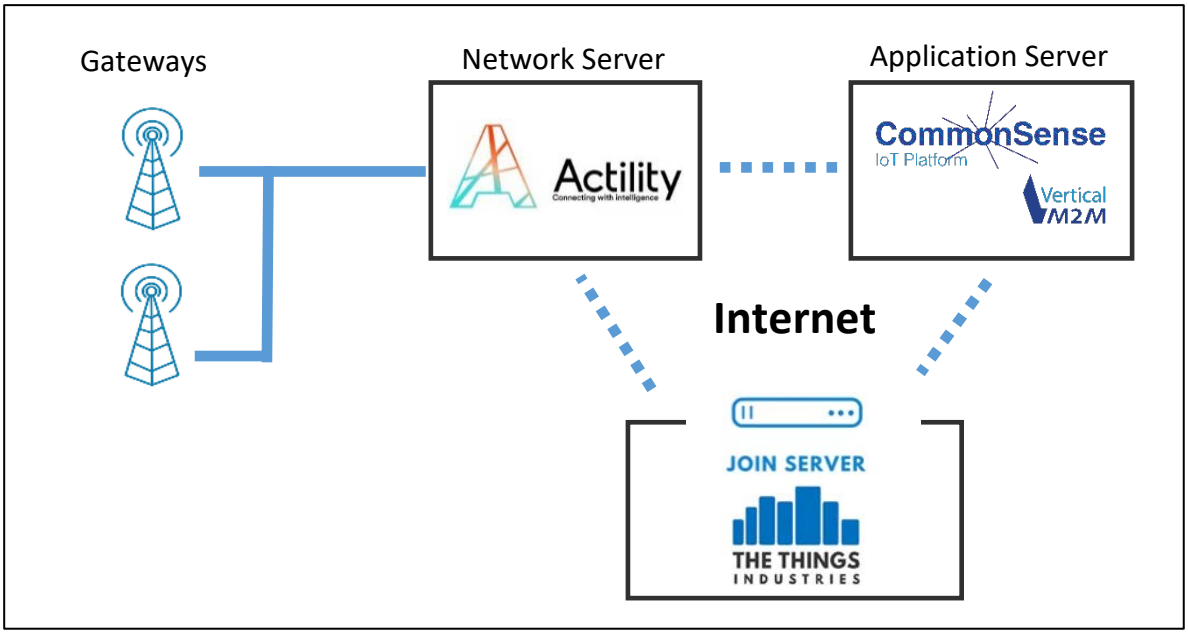


Figure 160: Network Server – Join Server – Application Server

11.1.2 The activation procedure in OTAA

In a previous chapter, the Figure 49 presented the Join procedure. The Join-Request frame is composed of a JoinEUI, DevEUI and a DevNonce as shown in Figure 161.

Size (octets)	8	8	2
Join-Request payload	JoinEUI	DevEUI	DevNonce

Figure 161: Join-Request frame

The DevNonce is an important value for the Join-Request because it protects from replay attacks exactly like we saw for the Frame Counter in chapter 4.5.3: A DevNonce can't be used twice. The maximum number of Join-Request is therefore 65536, which should never happen in a normal operation because a LoRaWAN end-device rarely generates Join-Request. The LoRaWAN specification proposes a different management of the DevNonce depending on the LoRaWAN standard we are working with:

- **Until LoRaWAN 1.0.3**, the Join Server accepts the Join-Request only if the DevNonce has never been used. Up to LoRaWAN 1.0.3, the LoRaWAN end-device randomly chose a DevNonce among the 65536 available. If the end-device has not saved the values used previously for the DevNonce, it is not very serious because at the next Join-Request it will generate a new random value with many chances to find a value that has never been used.
- **From LoRaWAN 1.0.4**, the Join Server accepts the Join-Request only if the value used is higher than the one used previously. Since LoRaWAN 1.0.4, the end-device uses an incrementing counter for the DevNonce. If the end-device reboot, it must keep the last value of the DevNonce in a non-volatile memory. Otherwise, the Join-Request will be discarded.

When the Join-Request is accepted, the Join Server returns a Join-Accept with the information shown in Figure 162. Among these are the DevAddr, network configuration information, as well as everything the end-device needs to generate the NwkSKey and AppSKey on its side.

Size (octets)	3	3	4	1	1	(16) optional
Join-Accept payload	JoinNonce	NetID	DevAddr	DLSettings	RXDelay	CFList

Figure 162: Format of a Join-Accept frame

The Join Server's purpose is to:

- Securely store the root keys (AppKey).
- Respond to the Join-Request issued by the authorized end-devices.
- Store and securely transmit the generated NwSKey to the Network Server.
- Store and securely transmit the generated AppSKey to the Application Server.

The Join Server contains the following information for each End end-device under its control:

- DevEUI
- AppKey
- Network Server Identifier
- Application Server identifier
- LoRaWAN version of the end-device

One may wonder what is the point of such a structure since the management of keys is simply deported to another server. To answer this question, we need to understand that the Join Server is not supposed to be an internal service connected to a single Network Server and Application Server. Figure 163 represents the multiple possible connections between them.

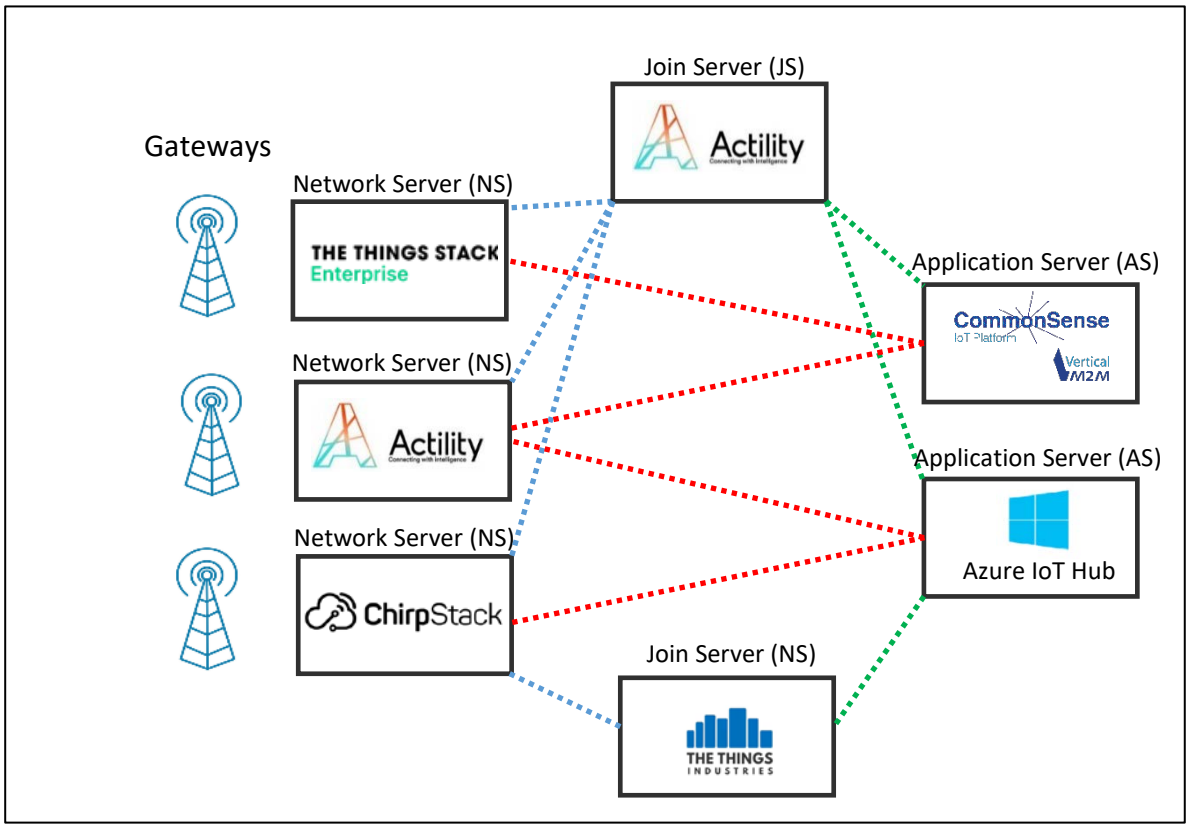


Figure 163: Network Server, Join Server, and Application Server connections

- A Network Server (NS) can be connected to many Join Server (JS). For example, ChirpStack can connect to Actility Join Server and TTI Join Server. Actility Network Server knows which JS to request thank to the JoinEUI sent by the end-device during the Join procedure. A NS can also connect to many AS.
- A JS can connect many NS for the exchange of the NwSKey and many AS to exchange the AppSKey.

Such a structure has many benefits from the security point of view and interoperability of devices within different networks.

11.1.3 Device key provisioning

When the OTAA activation process runs, we assume that the root Keys are already stored in the end-device on one hand and in the Join Server on the other hand. On that purpose, the end-device maker, the end-user and the Join Server must exchanges information before selling the end-device as shown in Figure 164.

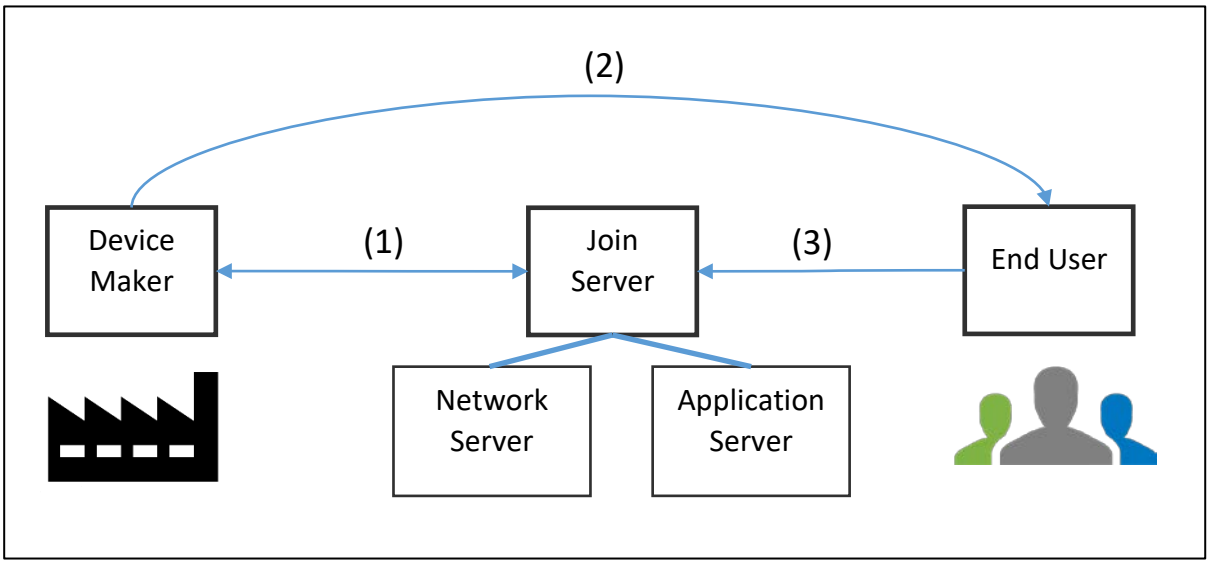


Figure 164: The end-device maker, the end-user and the Join Server

(1) During industrialization, the end-device maker will provision the end-devices with the keys (AppKey in LoRaWAN 1.0.x) as well as the JoinEUI and DevEUI. The DevEUI and AppKey must also be stored on the Join Server. When they shares keys, they must agree on a secure way to ensure that the keys are not exposed. These keys must then be stored very securely on both entities.

(2) Then the end-device is sold to the end-user.

(3) Finally, the end-user must prove ownership of the end-device by requesting it on the Join Server. This action allow authorize the Join Server to creates the Session keys (NwSKey and AppSKey) whenever it receives a Join Request.

11.1.4 Securing the keys

The security of the LoRaWAN protocol depends on the way the root keys are shared and stored (AppKey in LoRaWAN 1.0.x).

The challenges are as follow:

- **How do we provision the same root key (AppKey) in the end-device and in the Join Server without exposing them?** This operation is complicated because the companies making the

LoRaWAN end-device is not necessarily someone you can trust, so you can't necessarily provide them the list of keys to be integrated in your components without precaution.

- **How to react in case of intrusion?** Physical access to the memories storing the keys must be as limited as possible. An intrusion detection must lead to the self-destruction of the keys.
- **How to prevent giving any hint to find keys?** Calculations performed during encryption should not leave any hints, such as voltage variations or current draws that could give clues to the values used.

To secure the storage of these keys and to perform all encryption operations, specific hardware modules are required on both sides (end-device and Join Server).

- On the end-device we use **SE** (**S**ecure **E**lement). These are very small components close to the microcontroller.

For example, the ATECC608B-TNGACT component has all these characteristics, and is already provisioned, i.e. it already contains the root keys for the LoRaWAN server ACTILITY.

- On the Join Server side we use **HSM** (**H**ardware **S**ecurity **M**odule). They have the same characteristics but with more power and functionalities.

11.1.5 Device claim

When the end-device is manufactured, it is provisioned with its keys. These same keys are stored on a Join Server, but it is currently not authorized to respond to the end-device's Join-Request since no end user has proved its ownership through the "end-device Claiming procedure". The proof of ownership can be done through a QR Code provided by the vendor, or through the end-device information (DevEUI) associated with a code (token) found on the end-device (see Figure 165).

The process of associating an owner and verifying permission to use a network is called onboarding. These devices may be offboarded when a user no longer desires to be responsible for the device.

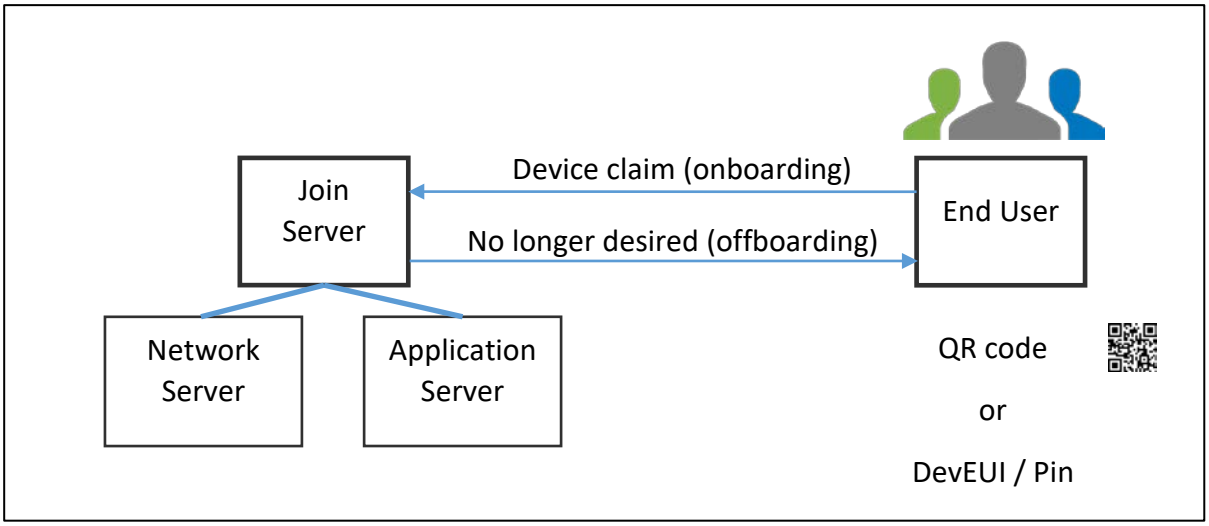


Figure 165: Device claim

If the claim is successful, the Join Server authorize Join-Request for this corresponding end-device. Session keys will then be distributed to the Network Server (NwkSKey) and Application Server (AppSKey).

If a new claim procedure is performed, for example by a new owner, then the end-device will have to perform a new Join-Request. Because claiming an end-device only transfers the ownership of the end-device but does not transfer the session keys.

Document versions

Version 1.0 : December 2021

- Initial version.

Version 1.1 : February 2022

- Many minor changes and corrections
- Add Trademark® Registered ® attribution to Semtech
- Frequency plan correction. Add the Roaming frequency plan recommended by the LoRa Alliance.
- Remove appendices. The documentation is now on http://lorawan.master-stic.fr/

Trademark