

---

# LoRa et LoRaWAN

pour

## l'Internet des Objets



Version du cours : 7

Septembre 2020

Sylvain MONTAGNY

[sylvain.montagny@univ-smb.fr](mailto:sylvain.montagny@univ-smb.fr)

Université Savoie Mont Blanc

# Présentation du document

Ce cours sur l'internet des Objets et les protocoles LoRa / LoRaWAN est le support d'une formation proposée par l'Université de Savoie Mont Blanc [ <https://scem-eset.univ-smb.fr> ]. Tout le cours est disponible en vidéo sur UDEMY en cliquant sur le lien <https://cutt.ly/lorawanDiscount> :



Cette version PDF du cours est mise à disposition pour la communauté et peut être diffusé en l'état. Toutes remarques / modifications / améliorations / corrections peuvent être proposées par email : [sylvain.montagny@univ-smb.fr](mailto:sylvain.montagny@univ-smb.fr). L'auteur vous remercie par avance pour votre contribution.

Cette formation est également proposée avec :

- Un envoi de matériel pour la réalisation des démonstrations (Gateways, devices...)
- Un support personnalisé d'une journée pour vous guider dans la réalisation des démonstrations

Pour toutes informations, vous pouvez envoyer un mail à [sylvain.montagny@univ-smb.fr](mailto:sylvain.montagny@univ-smb.fr)

## Ressources

Certaines ressources nécessaires à la réalisation des manipulations sont disponibles sous GitHub à l'adresse <https://github.com/SylvainMontagny/lora-lorawan>.

## Sigles et Acronymes

LoRa :	<b>L</b> ong <b>R</b> ange
LoRaWAN :	<b>L</b> ong <b>R</b> ang <b>W</b> ide <b>A</b> rea <b>N</b> etwork
PWAN :	<b>L</b> ow <b>P</b> ower <b>W</b> ide <b>A</b> rea <b>N</b> etwork.
TTN :	<b>T</b> he <b>T</b> hings <b>N</b> etwork
MQTT :	<b>M</b> essage <b>Q</b> ueuing <b>T</b> elemetry <b>T</b> ransport
QoS :	<b>Q</b> uality <b>o</b> f <b>S</b> ervice
HTTP :	<b>H</b> yper <b>T</b> ext <b>T</b> ransfer <b>P</b> rotocol
IoT :	<b>I</b> nternet <b>O</b> f <b>T</b> hings
LTE-M :	<b>L</b> ong <b>T</b> erm <b>E</b> volution <b>C</b> at <b>M</b> 1
NB-IoT :	<b>N</b> arrow <b>B</b> and <b>I</b> nternet <b>O</b> f <b>T</b> hings
FDM :	<b>F</b> requency <b>D</b> ivision <b>M</b> ultiplexing
TDM :	<b>T</b> ime <b>D</b> ivision <b>M</b> ultiplexing
CDMA :	<b>C</b> ode <b>D</b> ivision <b>M</b> ultiple <b>A</b> ccess
RSSI :	<b>R</b> eceived <b>S</b> ignal <b>SI</b> ndication.
SNR :	<b>S</b> ignal <b>O</b> ver <b>N</b> oise <b>R</b> atio
SF :	<b>S</b> pread <b>i</b> ng <b>F</b> actor
CR :	<b>C</b> oding <b>R</b> ate
CHIRP :	<b>C</b> ompressed <b>H</b> igh <b>I</b> ntensity <b>R</b> adar <b>P</b> ulse
JSON :	<b>J</b> ava <b>S</b> cript <b>O</b> bject <b>N</b> otation
SDR :	<b>S</b> oftware <b>D</b> igital <b>R</b> adio
TOA :	<b>T</b> ime <b>O</b> ne <b>A</b> ir
BDD :	<b>B</b> ase de <b>D</b> onnées
ADR :	<b>A</b> daptive <b>D</b> ata <b>R</b> ate
BW :	<b>B</b> and <b>W</b> idth

# Sommaire

<b>1 LES SYSTEMES EMBARQUES ET L'IOT .....</b>	<b>8</b>
1.1 L'INTERNET DES OBJETS ( INTERNET OF THINGS / IOT ) .....	8
1.1.1 <i>Les systèmes embarqués dans l'IoT .....</i>	8
1.1.2 <i>Différents protocoles dans l'IoT .....</i>	8
1.1.3 <i>Bandes de fréquence utilisées .....</i>	9
1.2 MODES DE PARTAGE DU SUPPORT.....	10
1.3 NOTION DE CODE D'ETALEMENT .....	11
1.3.1 <i>Transmissions uniques .....</i>	11
1.3.2 <i>Transmissions simultanées.....</i>	12
1.3.3 <i>Cas du protocole LoRa.....</i>	13
<b>2 TRANSMISSION RADIO ET PROPAGATION .....</b>	<b>14</b>
2.1 LES UNITES ET DEFINITIONS.....	14
2.2 DISTANCE DE TRANSMISSION EN LORA.....	16
2.3 ETUDE DE LA DOCUMENTATION D'UN COMPOSANT LoRA.....	16
<b>3 LA MODULATION LORA (COUCHE PHYSIQUE) .....</b>	<b>18</b>
3.1 LA MODULATION LoRA .....	18
3.1.1 <i>La forme du symbole (Chirp) .....</i>	18
3.1.2 <i>Durée d'émission d'un symbole et débit .....</i>	20
3.2 CODING RATE .....	22
3.3 UTILISATION DU LoRA CALCULATOR .....	22
3.4 TRAME COMPLETE TRANSMISE EN LoRA .....	23
3.5 TRAME COMPLETE TRANSMISE EN LoRAWAN .....	24
3.6 DUTY-CYCLE EN LoRAWAN .....	25
3.7 CONSOMMATION ENERGETIQUE .....	26
3.8 MISE EN ŒUVRE : LoRA EN POINT A POINT.....	26
3.8.1 <i>Utilisation de l'IDE Arduino .....</i>	27
3.8.2 <i>Validation du fonctionnement .....</i>	28
3.8.3 <i>Mise en valeur du Spreading Factor.....</i>	28
<b>4 LE PROTOCOLE LORAWAN .....</b>	<b>29</b>
4.1 DIFFERENCES ENTRE LoRA ET LoRAWAN.....	29
4.2 STRUCTURE D'UN RESEAU LoRAWAN .....	29
4.2.1 <i>Les Devices LoRa .....</i>	29
4.2.2 <i>Les Gateways LoRa.....</i>	30
4.2.3 <i>Le Network Server.....</i>	30
4.2.4 <i>Application Server .....</i>	31
4.2.5 <i>Application / Web Serveur .....</i>	32
4.2.6 <i>Authentification avec le Network Server .....</i>	33
4.2.7 <i>Chiffrement des données vers l'Application Server .....</i>	33
4.2.8 <i>Combinaison de l'authentification et du chiffrement .....</i>	34
4.3 CLASSES DES DEVICES LoRAWAN .....	34
4.3.1 <i>Classe A (All) : Minimal power Application .....</i>	34
4.3.2 <i>Classe B (Beacon) : Scheduled Receive Slot.....</i>	35
4.3.3 <i>Classe C (Continuous) : Continuously Listening .....</i>	36
4.3.4 <i>Résumé sur les classes de Device LoRa .....</i>	37
4.3.5 <i>Quelle Gateway pour les flux Downlink ?.....</i>	37
4.4 ACTIVATION DES DEVICES LoRA : ABP OU OTAA .....	37
4.4.1 <i>ABP : Activation By Personalization .....</i>	38

4.4.2	<i>OTAA : Over The Air Activation :</i>	38
4.5	CHOIX DE LA TECHNIQUE D'ACTIVATION : ABP OU OTAA ?	40
4.5.1	<i>Sécurité</i>	40
4.5.2	<i>Protection contre l'attaque par Replay</i>	41
4.5.3	<i>L'attribution du devAddr</i>	42
4.5.4	<i>RX Delay and CFList</i>	43
4.5.5	<i>Résumé</i>	43
4.6	CANAUX ET DATA RATE (DR) ET PUISSANCE	44
4.6.1	<i>Les DR (Data Rate)</i>	44
4.6.2	<i>Les canaux</i>	44
4.6.3	<i>La puissance d'émission</i>	45
4.6.4	<i>L'ADR (Adaptive Data Rate)</i>	45
<b>5</b>	<b>LES RESEAUX ET SERVEURS LORAWAN</b>	<b>49</b>
5.1	LES DIFFERENTS TYPES DE RESEAUX	49
5.1.1	<i>Les réseaux LoRaWAN opérés</i>	49
5.1.2	<i>Les réseaux LoRaWAN privés</i>	50
5.1.3	<i>Choix du réseau : opéré ou privé ?</i>	51
5.1.4	<i>Une alternative, le réseau LoRaWAN dédié (hybride)</i>	52
5.1.5	<i>Les zones de couvertures</i>	52
5.2	EXEMPLE DE SERVEUR : THE THINGS NETWORK (TTN)	52
5.2.1	<i>Présentation de TTN</i>	52
5.2.2	<i>Configuration de la Gateway</i>	53
5.2.3	<i>Enregistrement des Gateways, Applications et Devices</i>	53
5.2.4	<i>Configuration des Devices LoRa</i>	54
5.2.5	<i>Simulation d'Uplink dans l'Application Server</i>	55
5.2.6	<i>Downlink : De l'Application Server au Device LoRa</i>	55
5.3	MISE EN APPLICATION	56
<b>6</b>	<b>LA TRAME LORA / LORAWAN</b>	<b>57</b>
6.1	LES COUCHES DU PROTOCOLE LORAWAN	57
6.1.1	<i>Détail de la couche Application</i>	58
6.1.2	<i>Détail de la couche LoRa MAC</i>	58
6.1.3	<i>Détail de la couche physique : Modulation LoRa</i>	59
6.2	LA GATEWAY : DU LORA A LA TRAME IP	60
6.3	ANALYSE DES TRAMES IP	61
6.3.1	<i>Le format JSON</i>	61
6.3.2	<i>Utilisation de la base 64</i>	62
6.3.3	<i>Intérêt et inconvénient de la base 64</i>	63
6.3.4	<i>Uplink : Du Device LoRa au Network Server</i>	63
6.3.5	<i>Uplink : Du Network Server à l'Application Server</i>	65
<b>7</b>	<b>LA RECUPERATION DES DONNEES</b>	<b>67</b>
7.1	LES SERVICES RENDUS PAR NOTRE APPLICATION	67
7.2	RECUPERATION DES DONNEES AVEC LE PROTOCOLE HTTP (GET)	68
7.2.1	<i>Présentation du principe Client - Serveur</i>	68
7.2.2	<i>Désignation du client et du serveur</i>	69
7.2.3	<i>Installation des services HTTP pour le flux Uplink</i>	70
7.2.4	<i>Remarques sur la méthode HTTP GET</i>	73
7.3	RECUPERATION DES DONNEES AVEC LE PROTOCOLE HTTP (POST)	73
7.3.1	<i>Installation des services HTTP pour le flux Uplink</i>	74
7.3.1	<i>Installation des services HTTP pour le flux Downlink</i>	76
7.4	RECUPERATION DES DONNEES AVEC LE PROTOCOLE MQTT	77

7.4.1	<i>Présentation du protocole MQTT</i> .....	77
7.4.2	<i>Connexion au Broker MQTT</i> .....	79
7.4.3	<i>Qualité de Service au cours d'une même connexion</i> .....	79
7.4.4	<i>Qualité de Service après une reconnexion</i> .....	80
7.4.5	<i>Les Topics du protocole MQTT</i> .....	81
7.4.6	<i>Mise en place d'un Broker MQTT</i> .....	81
7.4.7	<i>Mise en place d'un Publisher et d'un Subscriber MQTT</i> .....	82
7.4.8	<i>Récupérer des données depuis l'Application Server avec MQTT</i> .....	83
7.4.9	<i>Envoyer des données depuis notre Application avec MQTT</i> .....	85
<b>8</b>	<b>LA CREATION DE NOTRE PROPRE DEVICE LORA</b> .....	<b>87</b>
8.1	ARCHITECTURE MICROCONTROLEUR + TRANSCEIVER.....	87
8.1.1	<i>Présentation de l'architecture</i> .....	87
8.1.2	<i>Exemple de carte de développement</i> .....	88
8.1.3	<i>Solution d'implémentation</i> .....	88
8.2	ARCHITECTURE MODULE LORAWAN STANDALONE.....	88
8.2.1	<i>Exemple de module</i> .....	89
8.2.2	<i>Exemple de carte de développement</i> .....	89
8.3	ARCHITECTURE MICROCONTROLEUR + MODULE LORAWAN .....	90
8.3.1	<i>Exemple de module LoRaWAN</i> .....	90
8.3.1	<i>Exemple de carte de développement</i> .....	91
8.4	ARCHITECTURE MICROCONTROLEUR WIRELESS LORAWAN .....	92
8.5	RESUME DES ARCHITECTURES.....	92
8.6	LES STACKS LORA A DISPOSITION .....	92
<b>9</b>	<b>LA CREATION DE NOTRE PROPRE NETWORK ET APPLICATION SERVER</b> .....	<b>93</b>
9.1	LES STACKS DISPONIBLES .....	93
9.2	PRÉSENTATION DE CHIRPSTACK.....	93
9.2.1	<i>Le projet ChirpStack</i> .....	93
9.2.2	<i>LoRa Gateway Bridge</i> .....	94
9.2.3	<i>LoRa Server (Network Server)</i> .....	94
9.2.4	<i>LoRa App Server (Application Server)</i> .....	94
9.2.5	<i>Application</i> .....	95
9.3	INSTALLATION DE LORASERVER .....	95
9.3.1	<i>Mise en place de l'environnement</i> .....	95
9.3.2	<i>Installation sur la Raspberry PI</i> .....	96
9.4	CONFIGURATION DE LORA SERVER POUR L'UPLINK .....	97
9.4.1	<i>Enregistrement d'une nouvelle Organisation</i> .....	97
9.4.2	<i>Enregistrement d'une instance du Network Server</i> .....	97
9.4.3	<i>Enregistrement d'une Application (Sur l'Application Server)</i> .....	98
9.4.4	<i>Enregistrement des Devices LoRa</i> .....	98
9.4.5	<i>Visualisation des trames reçues</i> .....	100
9.4.1	<i>UDP "Packet Forwarder"</i> .....	101
9.5	CONFIGURATION DE LORASERVER POUR L'INTEGRATION D'APPLICATION.....	103
9.5.1	<i>Récupérer des données sur notre Application avec HTTP POST</i> .....	103
9.5.2	<i>Récupérer des données sur notre Application avec MQTT</i> .....	103
<b>10</b>	<b>LA CREATION DE NOTRE PROPRE APPLICATION</b> .....	<b>105</b>
10.1	MISE EN PLACE DE L'ENVIRONNEMENT DE TRAVAIL.....	107
10.1.1	<i>Installation de docker et docker-compose</i> .....	107
10.1.2	<i>Cas d'étude</i> .....	108
10.2	UTILISATION DE NODE-RED .....	108
10.2.1	<i>Présentation</i> .....	108

10.2.2	<i>Récupération des données avec le node TTN</i> .....	109
10.2.3	<i>Modification du format du payload</i> .....	110
10.2.4	<i>Création d'un Dashboard</i> .....	111
10.2.5	<i>Stockage des informations dans une Base de données</i> .....	112
10.3	UTILISATION TELEGRAF, INFLUDB ET GRAFANA.....	113
10.3.1	<i>Telegraf : Récupération des données</i> .....	114
10.3.2	<i>InfluxDB : Sauvegarde dans une BDD</i> .....	114
10.3.3	<i>Grafana : Système de monitoring</i> .....	114
10.4	WORKSHOP LoRA .....	115
<b>11</b>	<b>VERSIONS DU DOCUMENT</b> .....	<b>116</b>

# 1 Les systèmes embarqués et l'IoT

## 1.1 L'Internet des Objets ( Internet of Things / IoT )

### 1.1.1 Les systèmes embarqués dans l'IoT

D'une façon générale, les systèmes électroniques peuvent être caractérisés par leur consommation, leur puissance de calcul, leur taille et leur prix. Dans le cas spécifique des systèmes embarqués utilisés dans l'IoT, nous pouvons affecter le poids suivant à chacune des caractéristiques :

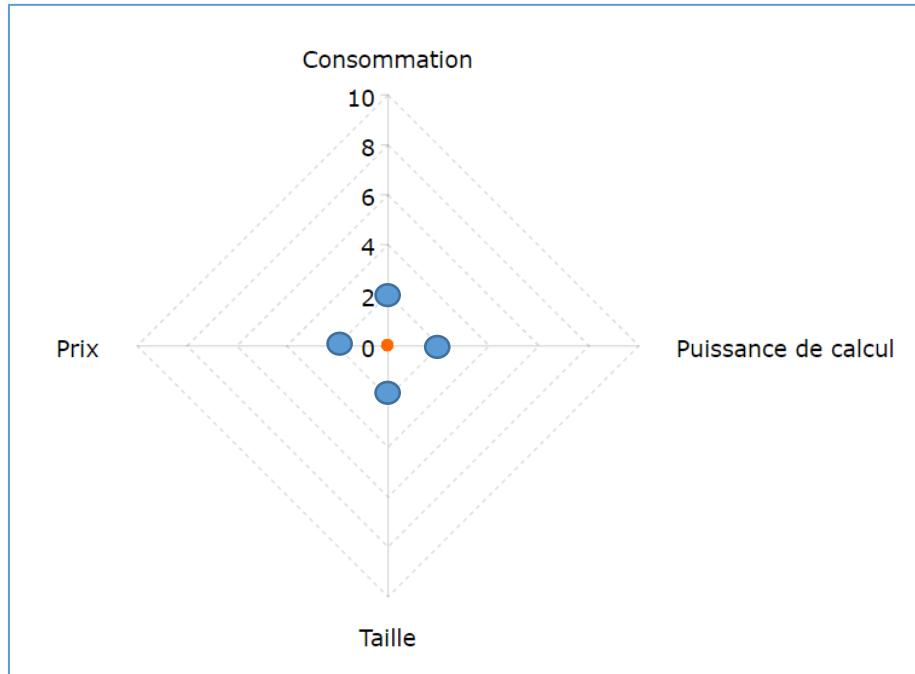


Figure 1 : Consommation, Puissance, Taille et Prix des objets connectés

Comparés aux autres systèmes électroniques, les systèmes embarqués utilisés dans l'IoT possèdent donc :

- Une faible consommation
- Une faible puissance de calcul
- Une petite taille
- Un prix faible

### 1.1.2 Différents protocoles dans l'IoT



💡 Citez les différents protocoles que vous connaissez dans le mode de l'IoT (Internet Of Things) et reportez-les dans le graphique ci-dessous en fonction de leur bande passante et de leur portée.

- |   |   |
|---|---|
| <ul style="list-style-type: none"><li>■ WIFI (2.4Ghz / 5Ghz)</li><li>■ Bluetooth (2.4Ghz),</li><li>■ 2G, 3G, 4G, 5G</li></ul> | <ul style="list-style-type: none"><li>■ Zigbee (2.4 Ghz)</li><li>■ Sigfox / LORA</li><li>■ LTE-M / NB-IoT (Opérateur)</li></ul> |
|---|---|

- NFC

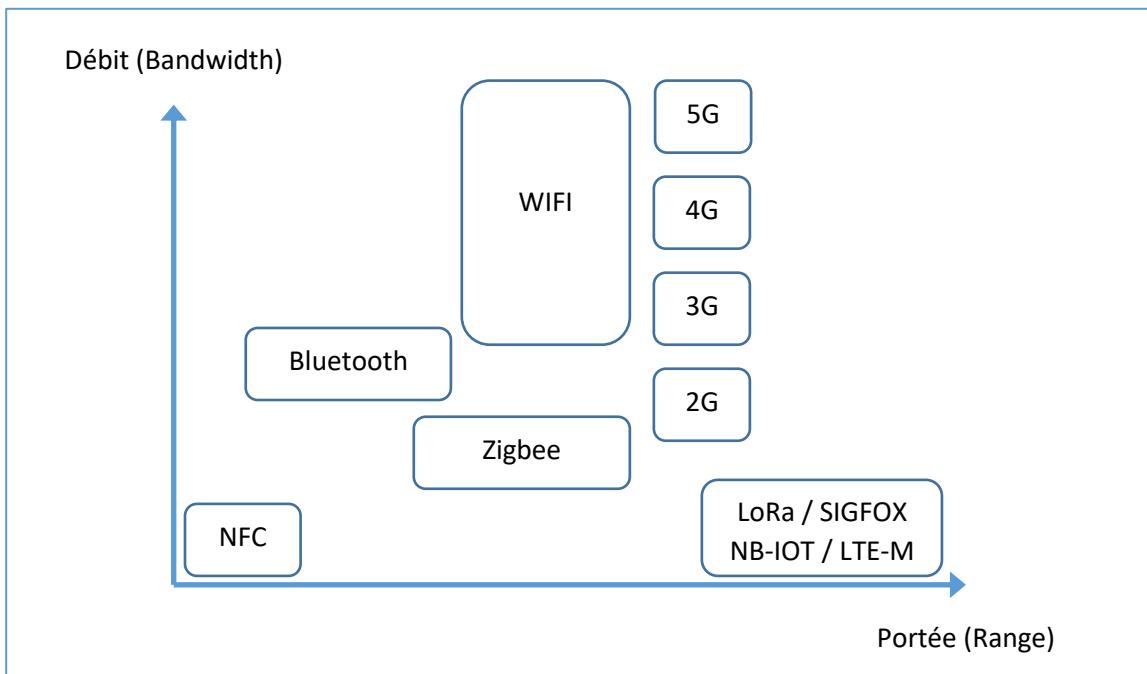


Figure 2 : Protocoles utilisés dans l'IoT en fonction du débit et de la portée



⚠ Dans l'IoT, les protocoles utilisés possèdent une grande portée, et des débits faibles

L'ensemble de ces réseaux sont dénommés LPWAN : **Low Power Wide Area Network**.

### 1.1.3 Bandes de fréquence utilisées

En Europe, certaines bandes de fréquences sont libres d'utilisation. Cela signifie :

- Pas de demande d'autorisation
- Gratuité d'utilisation

Le Tableau 1 présente quelques-unes de ces bandes libres.

Fréquences	Quelques utilisations
13.56 Mhz	RFID, NFC
433 MHz	Talkie-Walkie, télécommande, LoRa
868 MHz	Sigfox, LoRa
2.4 Ghz	WiFi, Bluetooth, Zigbee
5 Ghz	WiFi

Tableau 1 : Bandes de fréquences libres et utilisations

On remarque qu'en Europe, le LoRa peut utiliser la bande des 433 Mhz ou des 868 Mhz.

## 1.2 Modes de partage du support

Quel que soit le protocole utilisé, le support de transfert de l'information est l'air, car tous les protocoles de l'IoT sont Wireless. Le support doit être partagé entre tous les utilisateurs de telle façon que chacun des dispositifs Wireless ne perturbe pas les autres. Pour cela une bande de fréquence est allouée. Par exemple pour la radio FM la bande de fréquence va de 87,5 Mhz à 108 Mhz.

Dans leur bande de fréquence les dispositifs peuvent se partager le support de différentes manières :

- **FDM (Frequency Division Multiplexing)** : Les Devices utilisent des canaux fréquentiels pour séparer leurs transmissions. **Le LoRa utilise ce mode de partage**, c'est-à-dire que la bande libre des 868 Mhz est découpée en plusieurs canaux où chaque Device peut dialoguer.

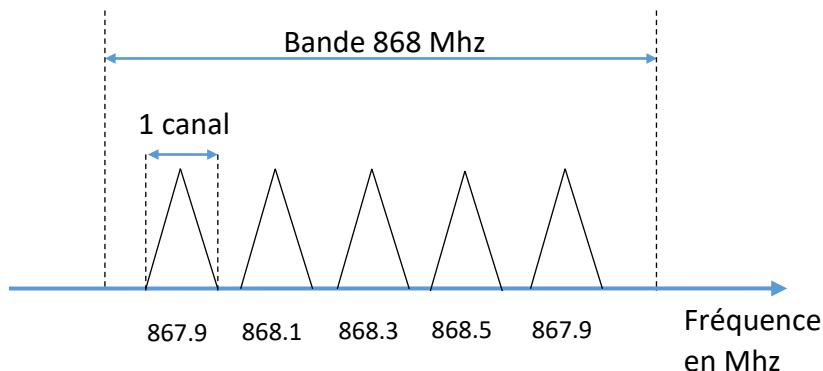


Figure 3 : Utilisation du FDMA en LoRa

- **TDM (Time Division Multiplexing)**. Dans ce mode de transmission, les Devices transmettent par intermittence afin de laisser libre le canal à tour de rôle. **Le LoRa utilise ce mode de partage**. En revanche les Devices ne sont pas synchronisés, donc des collisions peuvent survenir.
- **CDMA (Code Division Multiple Access)** : Dans ce mode de transmission, les Devices transmettent en même temps, sur le même canal. La conséquence de ce type de transmission est appelée "étalement du spectre". **Le LoRa utile un mode de transmission dont les propriétés sont assez similaires au CDMA**.

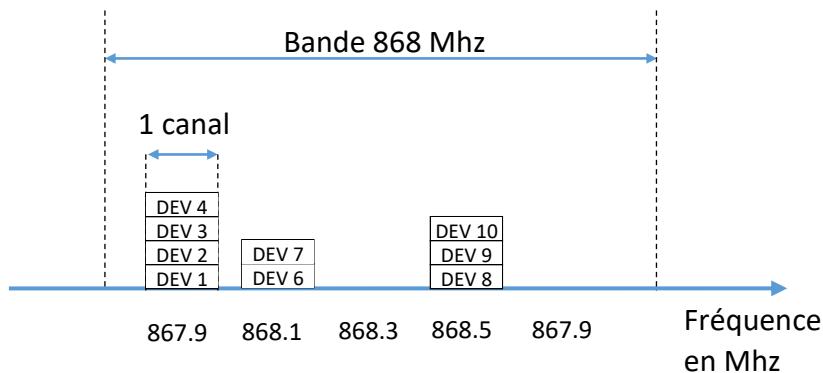


Figure 4 : Utilisation de l'étalement de spectre en LoRa

Les Devices LoRa ont le choix entre plusieurs canaux pour émettre. Sur **un canal choisi**, ils peuvent transmettre à plusieurs, **en même temps**. Le protocole LoRa utilise une modulation très similaire à la méthode de partage CDMA (pour autant, on ne pourra pas dire que le LoRa utilise le CDMA). Afin de comprendre la pertinence de ce mode de partage du support, nous allons valider le fonctionnement du mode CDMA dans le prochain paragraphe. Au chapitre 3, nous expliquerons dans les détails la modulation LoRa.

**TODO : Schéma pour chaque mode de partage du support.**

### 1.3 Notion de code d'étalement

Nous allons voir au travers d'un exercice comment il est possible d'utiliser **tout le spectre, en permanence**, tout en étant capable d'effectuer plusieurs transactions simultanées entre différents émetteurs/récepteurs. Cette méthode est souvent appelée « étalement de spectre » car comme son nom l'indique, elle a pour conséquence d'étaler le spectre du signal transmis.

La méthode consiste à utiliser des codes qui ont des propriétés mathématiques adaptées à notre objectif : Transmettre en même temps sur la même bande de fréquence. La matrice ci-dessous donne par exemple 4 codes d'étalement (1 par ligne) :

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

Tableau 2 : Matrice de Hadamard (mathématicien) d'ordre 4

Les propriétés de ces codes (**1 1 1 1 ; 1 -1 1 -1 ; 1 1 -1 -1 ; 1 -1 -1 1**) ne sont pas expliquées ici. Nous nous conterons de vérifier leur bon fonctionnement.

#### 1.3.1 Transmissions uniques

Chaque tableau ci-dessous représente une transmission qui est indépendante des autres : elles ne se déroulent pas en même temps. On vérifie qu'avec la mise en œuvre des codes d'étalement, chaque transmission arrive bien à destination avec le message qui avait été transmis. Le premier tableau est déjà rempli en guise d'exemple. La méthode est la suivante :

A l'émission :

- Chaque bit du message est multiplié par un code d'étalement (une ligne de la matrice)
- Le résultat de la multiplication est transmis

A la réception :

- Chaque symbole reçu est multiplié par le même code d'étalement
- Le message reçu est égal à la somme des symboles, divisé par le nombre de symbole

1	<b>Message User 1</b>	<b>1</b>	<b>0</b>	<b>1</b>	<b>0</b>
2	Utilisation code orthogonal User 1	1 -1 1 -1	1 -1 1 -1	1 -1 1 -1	1 -1 1 -1
3	Symboles transmis User 1 = (1) x (2)	1 -1 1 -1	0 0 0 0	1 -1 1 -1	0 0 0 0
... transmission ...					
4	Utilisation code orthogonal User 1	1 -1 1 -1	1 -1 1 -1	1 -1 1 -1	1 -1 1 -1
5	Décodage = (3) x (4)	1 1 1 1	0 0 0 0	1 1 1 1	0 0 0 0
6	<b>Message reçu (<math>\sum</math> (5) / nbr_bits)</b>	<b>1</b>	<b>0</b>	<b>1</b>	<b>0</b>

➔ Réaliser la transmission du message du User 2 et du User 3 dans les tableaux suivants :

1'	<b>Message User 2</b>	<b>0</b>	<b>1</b>	<b>0</b>	<b>1</b>
2'	Utilisation code orthogonal User 2	1 1 -1 -1	1 1 -1 -1	1 1 -1 -1	1 1 -1 -1
3'	Symboles transmis User 2 = (1') x (2')	0 0 0 0	1 1 -1 -1	0 0 0 0	1 1 -1 -1
... transmission ...					
4'	Utilisation code orthogonal User 2	1 1 -1 -1	1 1 -1 -1	1 1 -1 -1	1 1 -1 -1
5'	Décodage = (3') x (4')	0 0 0 0	1 1 1 1	0 0 0 0	1 1 1 1
6'	<b>Message reçu (<math>\sum</math> (5') / nbr_bits)</b>	<b>0</b>	<b>1</b>	<b>0</b>	<b>1</b>

1''	<b>Message User 3</b>	<b>1</b>	<b>1</b>	<b>0</b>	<b>0</b>
2''	Utilisation code orthogonal User 3	1 -1 -1 1	1 -1 -1 1	1 -1 -1 1	1 -1 -1 1
3''	Symboles transmis User 3 = (1'') x (2'')	1 -1 -1 1	1 -1 -1 1	0 0 0 0	0 0 0 0
... transmission ...					
4''	Utilisation code orthogonal User3	1 -1 -1 1	1 -1 -1 1	1 -1 -1 1	1 -1 -1 1
5''	Décodage = (3'') x (4'')	1 1 1 1	1 1 1 1	0 0 0 0	0 0 0 0
6''	<b>Message reçu (<math>\sum</math> (5'') / nbr_bits)</b>	<b>1</b>	<b>1</b>	<b>0</b>	<b>0</b>

### 1.3.2 Transmissions simultanées

Les transmissions se déroulent maintenant simultanément : les messages des User 1, User 2 et User 3 sont envoyés en même temps sur la même bande de fréquence. La première colonne du User 1 est déjà remplie en guise d'exemple. La méthode est la suivante :

Dans l'air :

- On additionne les symboles transmis par tous les User (1, 2, 3), on additionne donc les lignes suivantes :

3	Symboles transmis User 1	1 -1 1 -1	0 0 0 0	1 -1 1 -1	0 0 0 0
3'	Symboles transmis User 2	0 0 0 0	1 1 -1 -1	0 0 0 0	1 1 -1 -1
3''	Symboles transmis User 3	1 -1 -1 1	1 -1 -1 1	0 0 0 0	0 0 0 0

Pour la réception :

- Chaque symbole reçu est multiplié par le code d'étalement du User
  - Le message reçu est égal à la somme des symboles, divisé par le nombre de symbole
- ➔ Valider le fonctionnement pour la réception des messages

1'''	<b><math>\Sigma</math> des symboles transmis (3 + 3' + 3'')</b>	2 -2 0 0	2 0 -2 0	1 -1 1 1 -1	1 1 1 -1 -1
2'''	Utilisation code orthogonal User 1	1 -1 1 -1	1 -1 1 -1	1 -1 1 -1	1 -1 1 -1
3'''	Décodage (1''') x (2'''')	2 2 0 0	2 0 -2 0	1 1 1 1	1 -1 -1 1
4'''	<b>Message reçu User 1 (<math>\Sigma</math> (3''')/ nbr_bits)</b>	<b>1</b>	<b>0</b>	<b>1</b>	<b>0</b>
2'''	Utilisation code orthogonal User 2	1 1 -1 -1	1 1 -1 -1	1 1 -1 -1	1 1 -1 -1
3'''	Décodage (1''') x (2'''')	2 -2 0 0	2 0 2 0	1 -1 -1 1	1 1 1 1
4'''	<b>Message reçu User 2 (<math>\Sigma</math> (3''')/ nbr_bits)</b>	<b>0</b>	<b>1</b>	<b>0</b>	<b>1</b>
2'''	Utilisation code orthogonal User 3	1 -1 -1 1	1 -1 -1 1	1 -1 -1 1	1 -1 -1 1
3'''	Décodage (1''') x (2'''')	2 2 0 0	2 0 2 0	1 1 -1 -1	1 -1 1 -1
4'''	<b>Message reçu User 3 (<math>\Sigma</math> (3''')/ nbr_bits)</b>	<b>1</b>	<b>1</b>	<b>0</b>	<b>0</b>

### 1.3.3 Cas du protocole LoRa

Le LoRa utilise une méthode d'étalement de spectre qui n'est pas exactement celle que nous venons d'étudier. La finalité est cependant la même : **Pouvoir transmettre en même temps, sur le même canal.** Le protocole LoRa utilise sept « codes d'étalement » appelés Spreading Factor [ SF6, SF7, SF8, SF9, SF10, SF11 et SF12 ] qui lui permet d'avoir sept transmissions simultanées sur un même canal.

## 2 Transmission radio et propagation

### 2.1 Les unités et définitions

- **dB** : Rapport entre deux puissances : Une atténuation est représentée par un nombre négatif (-). Un gain est représenté par un nombre positif (+).

Rapport de puissances en dB	Rapport de puissance
+ 10 dB	Multiplication par 10
+ 3 dB	Multiplication par 2
0 dB	Egalité
-3 dB	Division par 2
- 10 dB	Division par 10

Tableau 3 : Comparaison entre les gains en dB et en proportion

- **dBi** : Gain théorique d'une antenne
- **dBm** : Puissance ramenée à 1mW : 0 dBm correspond à 1 mW.



En reprenant la définition des dB remplir le tableau suivant :

Puissance en dBm	Puissance en mW
+ 10 dBm	10 mW
+ 3 dBm	2 mW
0 dBm	1 mW
- 3 dBm	0,5 mW
- 10 dBm	0,1 mW

Tableau 4 : Comparaison entre les puissances en dB et en mW



Le Talkie-Walkie a une puissance d'émission de 2W. Quelle est la puissance d'émission exprimées en dBm ?

Talkie-Walkie => 33dBm.

Le schéma globale d'une transmission radiofréquence peut être représenté par la Figure 5 suivante.

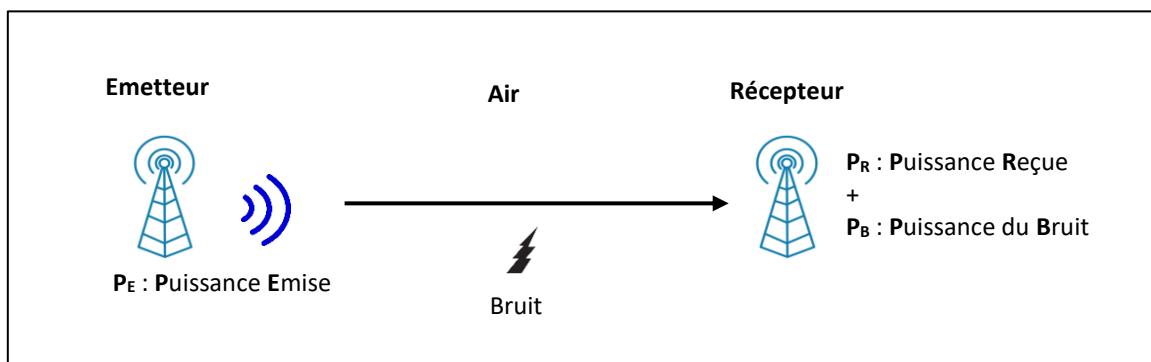


Figure 5 : Schéma d'une transmission radiofréquence

L'émetteur transmet un signal ( $P_E$ ). Le récepteur récupère une fraction de ce signal ( $P_R$ ) à cause des pertes, ainsi que du bruit  $P_B$  qui vient se rajouter.

- On appelle **RSSI** (Received Signal Strength Indication), la puissance  $P_R$  du signal reçu.
- On appelle **Sensibilité**, la puissance  $P_R$  minimale (ou RSSI minimal) qui doit être présent sur le récepteur. Si le RSSI reçu est en dessous de la sensibilité, alors le signal est indétectable par le récepteur.
- On appelle **SNR** (Signal over Noise Ratio), le rapport entre la puissance reçue ( $P_R$ ) et la puissance du bruit ( $P_B$ ).

Toutes ces valeurs (RSSI, Sensibilité, SNR, ...) sont données en décibel. Un signal pourra être reçu convenablement si les deux conditions suivantes sont remplies :

- 1 . Le RSSI est supérieur à la sensibilité du récepteur
- 2 . Le SNR de passe pas en dessous d'un certain seuil qui rendrait le signal impossible à démoduler.

💡 **Un émetteur transmet à une puissance de 13dBm en utilisant une antenne dont le gain est de 2dBi. Les pertes dans l'air sont de 60 dB. L'antenne réceptrice qui possède un gain de 2dBi est reliée à un récepteur dont la sensibilité est de -80 dBm. Le signal pourra-t-il être reçu ?**

Réponse :  $(13 + 2 - 60 + 2) = -43 \text{ dBm} > -80 \text{ dBm}$       Donc ça passe.

Voici par exemple le résultat que me retourne notre Gateway lorsqu'elle reçoit un message LoRa :

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

Comment puis-je améliorer mon bilan de transmission? La première idée serait d'émettre plus fort (augmenter  $P_E$ ). Ceci est possible dans une certaine mesure, car les puissances d'émission sont limitées. En LoRa, la puissance d'émission maximum sur la bande 868 Mhz est de 14 dBm (25 mW). La seconde possibilité est d'améliorer la sensibilité du récepteur. Les concepteurs de modules LoRa s'efforcent de l'améliorer jusqu'aux limites technologiques actuelles. Au final, ce qui compte, c'est surtout la différence entre la puissance  $P_E$  et la sensibilité du récepteur. C'est ce qu'on appelle le **Link Budget**. Dans l'exemple précédent, le budget que nous avons à disposition est de 93dB.

💡 **En LoRa, nous avons un Link Budget d'environ 157 dB**  
💡 **En LTE (4G), nous avons un Link Budget d'environ 130 dB.**

On peut utiliser le **Link budget calculator** : <https://bit.ly/2NTW8zn>

## 2.2 Distance de transmission en LoRa

La puissance émise ( $P_E$ ) sera atténuée dans l'air selon la formule suivante :

$$\text{Atténuation} = 10 \cdot \log(\text{distance}^2 \cdot \text{fréquence}^2 \cdot 1755)$$

- Atténuation : en **dB**
- Distance : en **km**
- Fréquence : en **Mhz**

On peut donc en déduire la distance maximale :

$$\text{distance} = \sqrt{\frac{10^{\frac{\text{Atténuation}}{10}}}{1755 \cdot \text{fréquence}^2}}$$

Le link Budget étant l'atténuation maximale que peut supporter une transmission, nous pouvons en déduire la distance en remplaçant l'atténuation par le link budget :

$$\text{Soit } \text{distance} = \sqrt{\frac{10^{\frac{\text{Link Budget}}{10}}}{1755 \cdot \text{fréquence}^2}}$$

- Le transceiver LoRa SX1272 (Link Budget de 157 dB), cela nous donne une distance théorique de 1940 km
- Le transceiver LoRa SX1276 (Link Budget de 168 dB), cela nous donne une distance théorique de 6907 km

En avril 2020, le record du monde de distance en transmission LoRa a été battu. Il est de 832 km pour une puissance de 25 mW / 14 dBm (puissance maximale autorisée en Europe).

## 2.3 Etude de la documentation d'un composant LoRa

Voici la documentation du Transceiver LoRa Sx1272 que nous utiliserons :

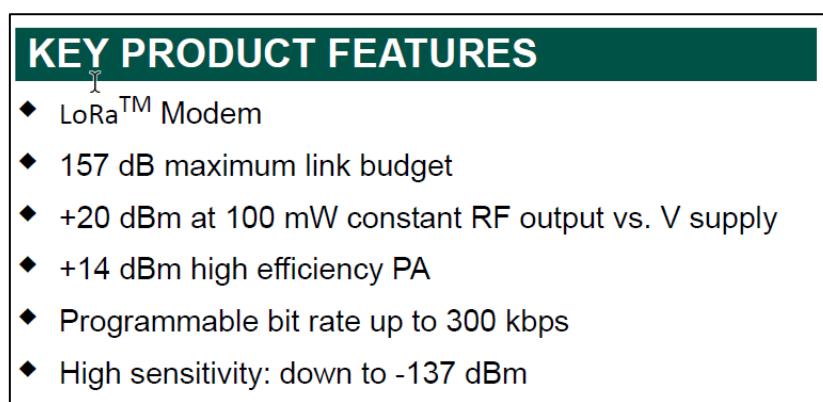


Figure 6 : Caractéristiques principales du composant radiofréquence utilisé

- ➔ Reprenez la définition du Link Budget et retrouver les 157 dB annoncés dans cette documentation.

En LoRa, plus le code d'étalement est grand, plus on est capable d'émettre dans un milieu bruité. La figure ci-dessous présente les rapports signal sur bruit avec lesquels nous serons capables de réaliser une transmission, en fonction des Spreading Factor utilisés.

<b>SpreadingFactor (RegModemConfig2)</b>	<b>Spreading Factor (Chips / symbol)</b>	<b>LoRa Demodulator SNR</b>
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

*Figure 7 : Influence du Spreading Factor sur le SNR acceptable*

On remarque que pour un SF8, nous sommes capables d'émettre avec un SNR de -10 dB : On pourra transmettre malgré le fait que le bruit est 10 fois supérieur au signal.

On remarque que pour un SF12, nous sommes capables d'émettre avec un SNR de -20 dB : On pourra transmettre malgré le fait que le bruit est 100 fois supérieur au signal !

Cependant, on remarque aussi que l'utilisation d'un Spreading Factor plus élevé, augmente considérablement le nombre de symbole émis (2<sup>ème</sup> colonne du tableau). Comme nous le verrons plus tard, cela impactera évidemment le temps de transmission.

### 3 La modulation LoRa (couche physique)

#### 3.1 La modulation LoRa

Comme nous l'avons expliqué plus tôt, la modulation LoRa utilise l'étalement de spectre pour transmettre ces informations. Mais au lieu d'utiliser des codes d'étalement (CDMA), elle utilise une méthode appelée Chirp Spread Spectrum. La finalité est toujours la même : avoir plusieurs transmissions dans le même canal. La conséquence sur le spectre est aussi la même : cela provoque un étalement du spectre.

##### 3.1.1 La forme du symbole (Chirp)

Le signal émis par la modulation LoRa est un symbole dont la forme de base est représentée ci-dessous. Son nom (Chirp) vient du fait que ce symbole est utilisé dans la technologie Radar (**Chirp** : Compressed High Intensity Radar Pulse)

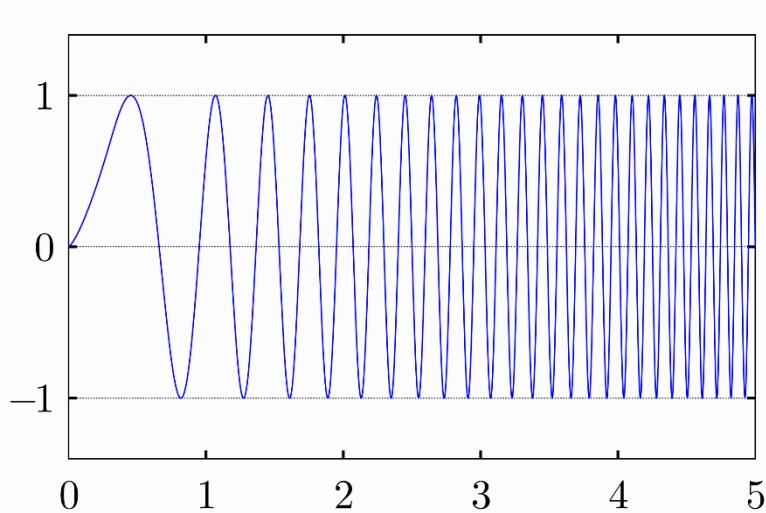


Figure 8 : Symbole de la modulation LoRa (source Wikipédia)

La fréquence de départ est la fréquence centrale du canal moins la Bande Passante divisée par deux.  
La fréquence de fin est la fréquence centrale plus la Bande Passante divisée par deux :

- La fréquence centrale est appelée le canal
- La bande passante est la largeur de bande occupée autour du canal

 On considère une émission sur la fréquence centrale 868 Mhz avec une Bande Passante de 125 kHz. Donner la fréquence de début et la fréquence de fin du sweep.

- Fréquence de début : 867 999 937,5 Hz
- Fréquence de fin : 868 000 062,5 Hz

Pour faciliter la représentation de ce symbole, on utilise plutôt un graphique Temps/Fréquence de la forme suivante :

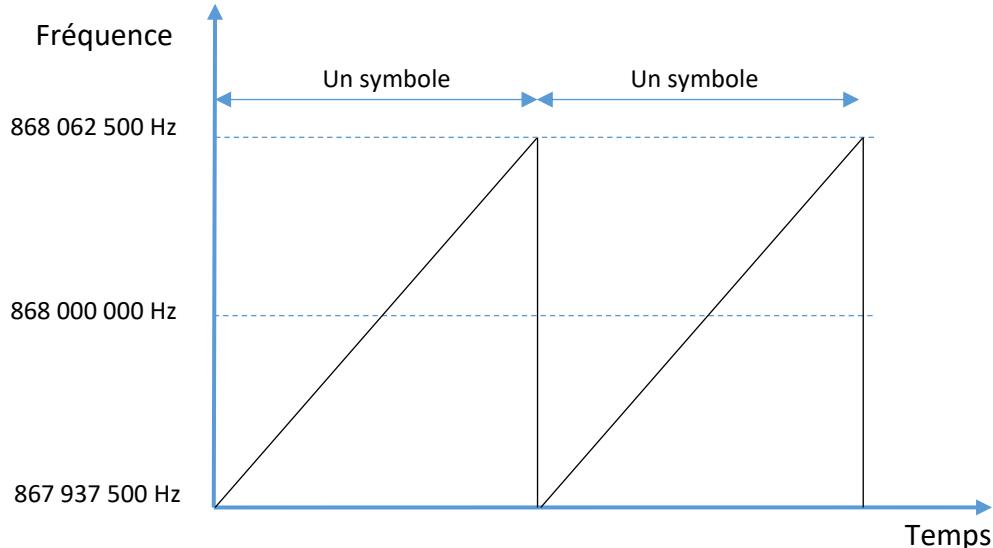


Figure 9 : Forme du symbole de base (CHIRP)

En LoRa, chaque symbole représente un certain nombre de bits transmis. La règle est la suivante :

**Nombre de bits transmis dans un symbole = Spreading Factor**

Par exemple, si la transmission utilise un Spreading Factor de 10 (SF10), alors un symbole représente 10 bits.

C'est-à-dire qu'à l'émission, les bits sont regroupés par paquet de **SF** bits, puis chaque paquet est représenté par un symbole particulier parmi  $2^{SF}$  formes de symboles possibles.

Sur la figure suivante, voici un exemple théorique d'une modulation en SF2 à 868 Mhz, sur une bande passante de 125 kHz. Chaque symbole représente donc 2 bits.

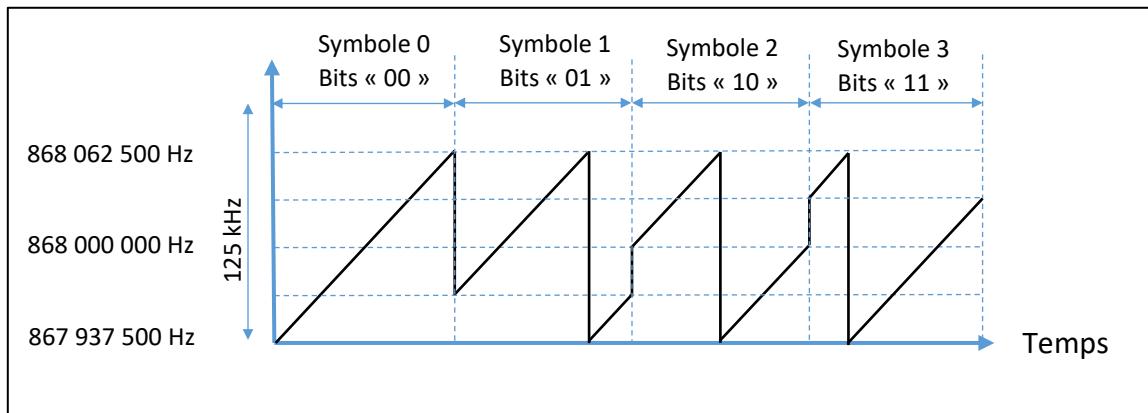


Figure 10 : Symboles émis en Modulation LoRa (Cas théorique en SF2)

Exemple :

- On considère la suite binaire suivante : 0 1 0 1 1 1 0 0 0 1 1 0 0 1 0 0 0 1 1 0 1
- Le Spreading Factor utilisé est SF10

Nous regroupons donc les bits par paquet de 10. Chaque paquet de 10 bits sera représenté par un symbole (sweep) particulier. Il y a 1024 symboles différents pour coder les 1024 combinaisons binaires possibles ( $2^{10}$ ).

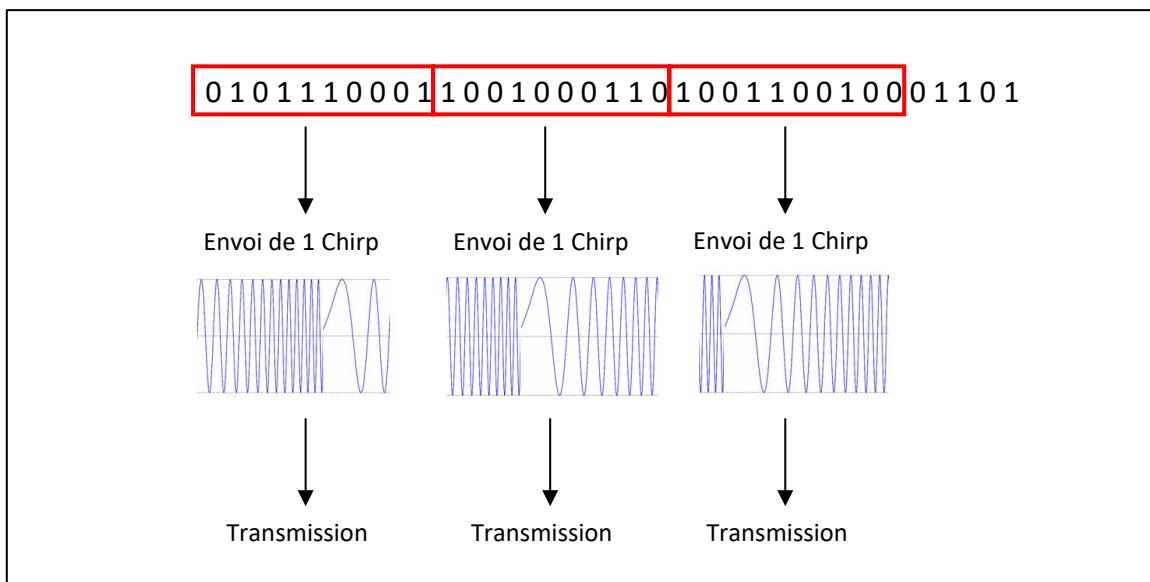


Figure 11 : Emission des Chirp en LoRa

Avec un outils d'analyse spectrale actif pendant l'émission LoRa d'un Device, on peut afficher les successions de symboles qui sont envoyés. Cet oscillogramme a été réalisé à l'aide d'un SDR (Software Digital Radio).

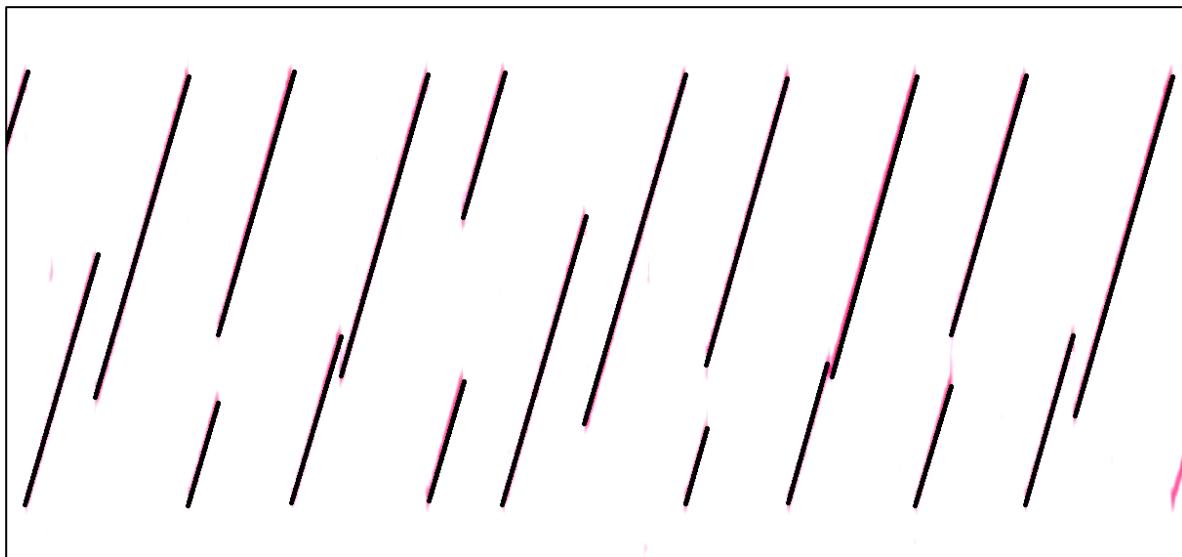


Figure 12 : Visualisation des Chirps LoRa réellement émis pendant une transmission

### 3.1.2 Durée d'émission d'un symbole et débit

En LoRa, la durée d'émission de chaque symbole (Chirp) dépend du Spreading Factor utilisé. Plus le SF est grand et plus le temps d'émission sera long. Pour une même bande passante, le temps d'émission d'un symbole en SF8 est deux fois plus long que le temps d'émission d'un symbole en SF7. Ainsi de suite jusqu'à SF12.

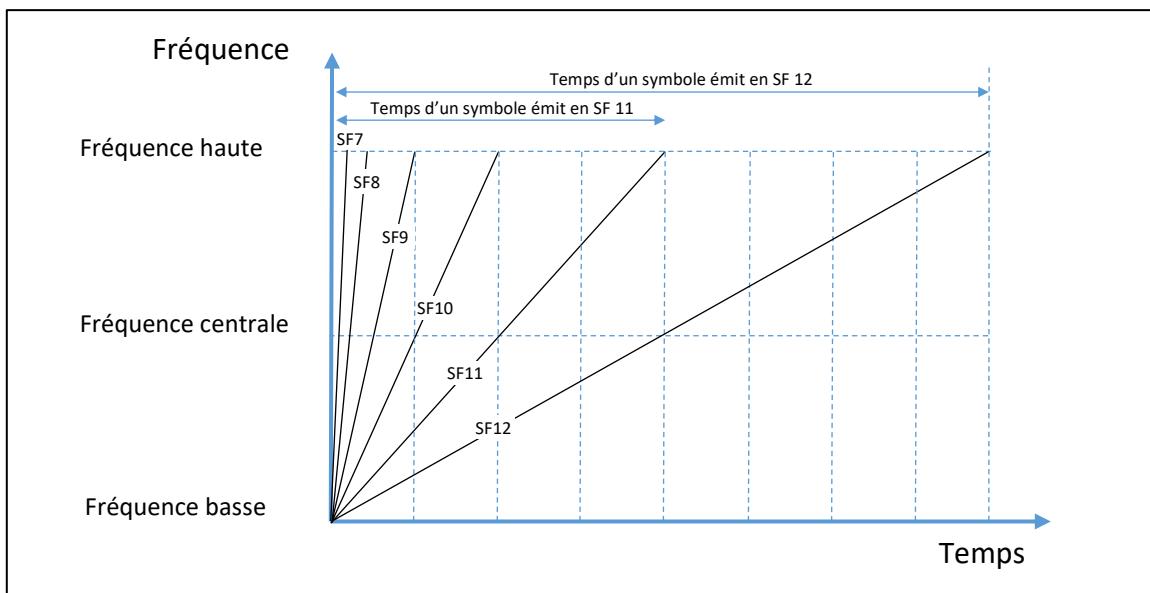


Figure 13 : Temps d'émission d'un symbole (Chirp) en fonction du SF

En revanche, le temps d'émission d'un symbole ( $T_{symbole}$ ) est inversement proportionnel à la bande passante :  $T_{symbole} = \frac{2^{SF}}{Bandwidth}$ . A titre d'exemple, le Tableau 5 représente le temps d'émission d'un symbole pour une bande passante de 125 KHz.

Spreading Factor	Temps d'émission d'un symbole
SF7	1,024 ms
SF8	2,048 ms
SF9	4,096 ms
SF10	8,192 ms
SF11	16,384 ms
SF12	32,768 ms

Tableau 5 : Temps d'émission d'un symbole pour BW125

Le débit des symboles est donc de  $\frac{1}{T_{symbole}} = F_{symbole} = \frac{Bandwidth}{2^{SF}}$ . En toute logique, plus la bande passante est élevée, plus le débit des symboles sera élevé.

Comme chaque symbole comprend SF bits, on retrouve alors le débit binaire :

$$D_b = SF \cdot \frac{Bandwidth}{2^{SF}}$$

- ➲ Plus le Spreading Factor sera élevé, plus le débit binaire sera faible.
- ➲ Plus la Bande Passante sera élevée, plus le débit binaire sera élevé.

On considère les deux cas suivants. Donner le débit binaire correspondant.

- **Cas 1 :** Pour SF7, 125 kHz > Débit = 6.836 bps
- **Cas 2 :** Pour SF12, 125 kHz > Débit = 366 bps

### 3.2 Coding Rate

Le Coding Rate est un ratio qui augmentera le nombre de bits à transmettre afin de réaliser de la détection / correction d'erreur. Dans le cas d'un CR = 4 / 8, il y aura 8 bits transmis réellement à chaque fois que nous souhaitons transmettre 4 bits. Dans cet exemple, cela provoque une transmission d'un nombre de bits multiplié par 2.

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

Figure 14 : Influence du Coding Rate sur le nombre de bits ajoutés

On reprend les deux cas précédents avec un CR de 4 / 5. Donner le débit binaire correspondant.

- Cas 1 : Pour SF7, 125 kHz et CR4/5 > Débit = 6.836 kbps / 1.25 = 5469 bps
- Cas 2 : Pour SF12, 125 kHz et CR4/5 > Débit = 366 bps / 1.25 = 293 bps

- ➔ La documentation d'un transceiver LoRa donne les débits en fonction du Spreading Factor, la Bande Passante et le Coding Rate. Vérifier la cohérence du résultat avec votre calcul précédent.

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

Figure 15 : Débit en fonction des paramètres de la transmission LoRa

### 3.3 Utilisation du LoRa Calculator

Le logiciel « LoRa Calculator » est un petit exécutable fourni par Semtech permettant de simuler une transmission LoRa en fonction des caractéristiques saisies : Canal, SF, CR, etc... Il est téléchargeable à l'adresse suivante : <https://bit.ly/2TyloAh>

Un simulateur en ligne équivalent est aussi disponible à l'adresse :

<https://www.loratools.nl/#/airtime>

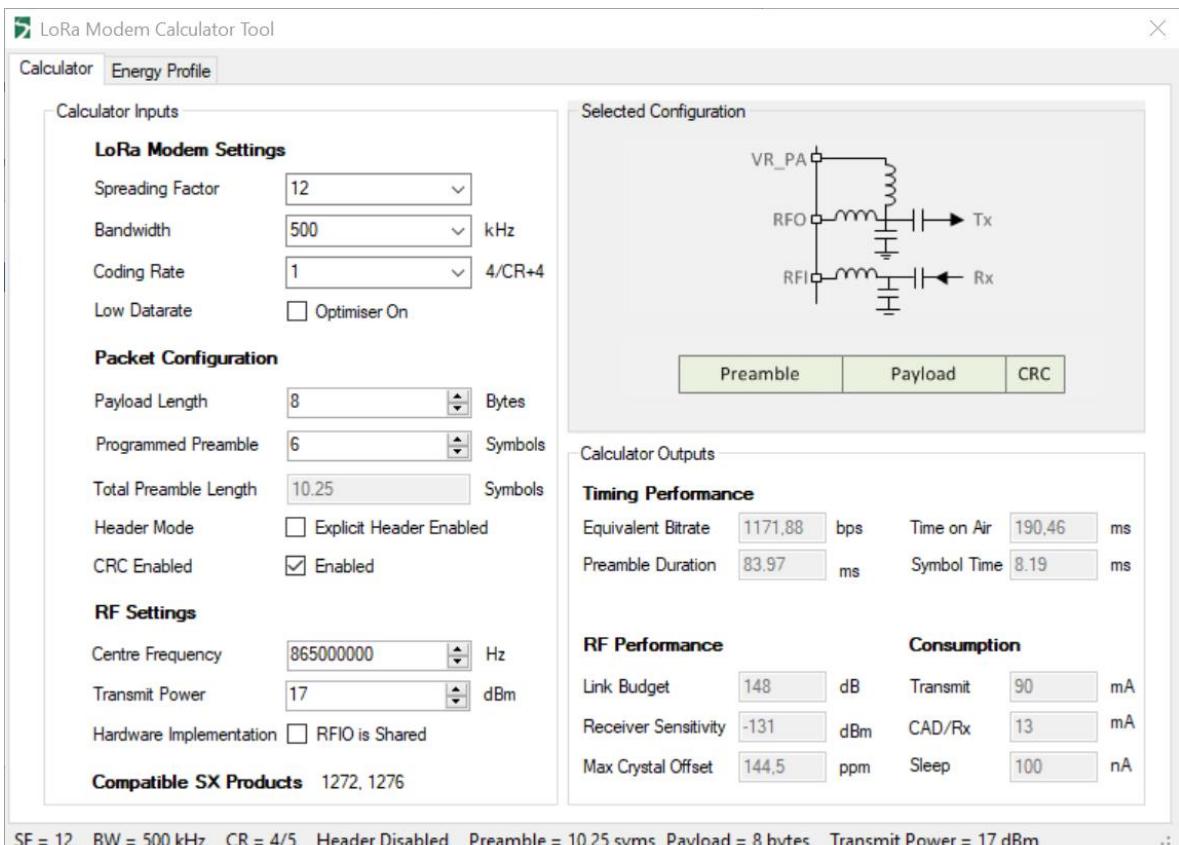


Figure 16 : Le logiciel LoRa Calculator

 En reprenant l'exemple précédent (SF7, Bande Passante 125 kHz, CR 4/5), vérifier les calculs du « Equivalent Bitrate ».

Le document de référence précisant le nombre de symbole par trame est le *LoRa Modem Design Guide : AN1200.13*

### 3.4 Trame complète transmise en LoRa

Nous nous sommes focalisés pour l'instant uniquement sur le débit instantané lors de l'envoi de données. En réalité, il n'y a pas seulement les données qui sont envoyées lors d'une transmission LoRa. Afin d'encadrer les données, il faut transmettre en plus :

- Un préambule permettant au récepteur de se synchroniser
- Des entêtes
- Des champs de CRC (vérification de l'intégrité de la trame)

Les données du protocole LoRa sont appelées **PHY Payload** (données de la couche physique) et la trame complète est représentée par la Figure 17.

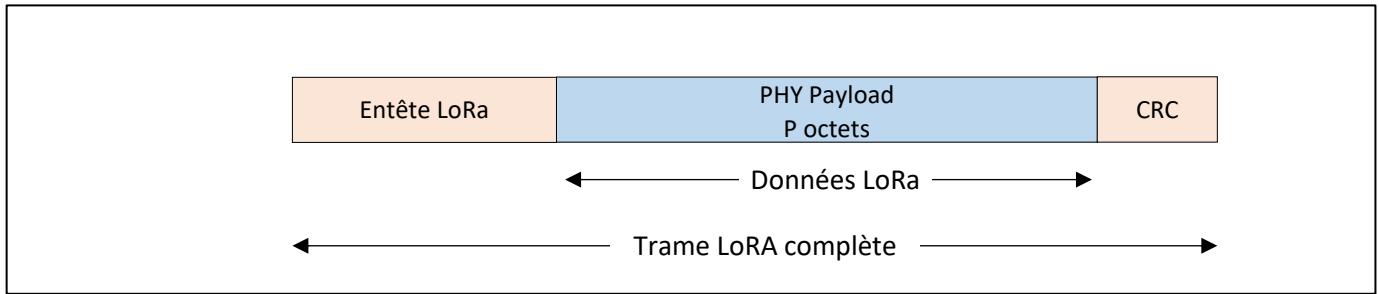


Figure 17 : Trame LoRa

Nous verrons plus tard au paragraphe 6.1.3, la représentation détaillée d'une trame LoRa. Dans l'immédiat, nous allons utiliser le LoRa Calculator pour estimer le temps d'émission de la trame. Ce temps d'émission est appelé **Time On Air** (ou Airtime). La simulation pour SF7, BW125, CR4/5 et un octet de Payload est donnée Figure 18.

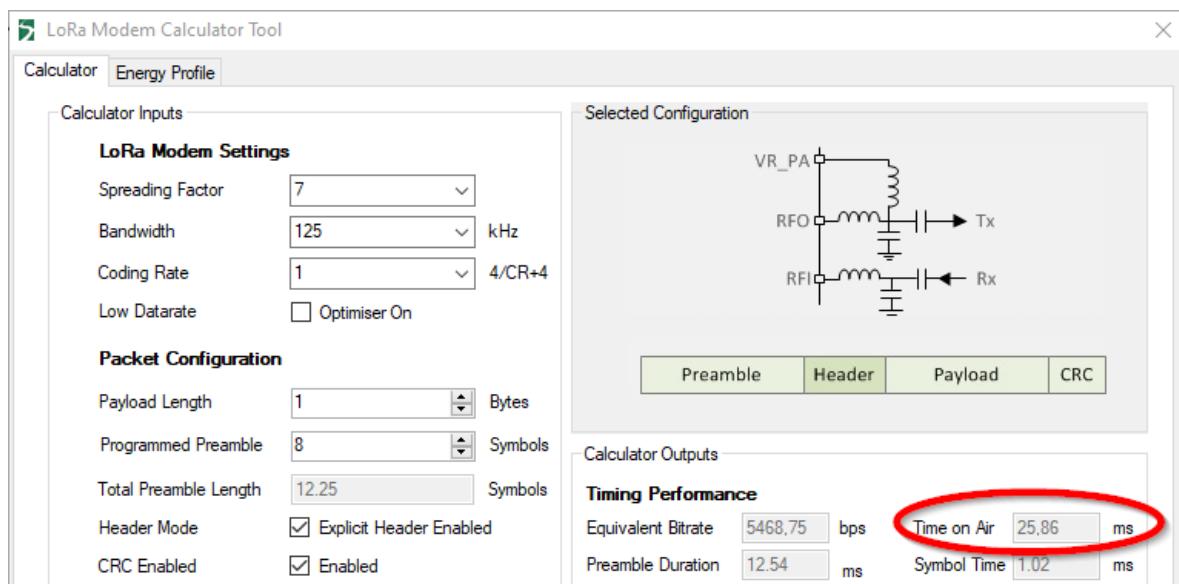


Figure 18 : Simulation du Time On Air en LoRa

- L'envoi d'un octet (Payload Length) en SF7 donne un Time On Air de **25,85 ms**
- L'envoi d'un octet (Payload Length) en SF12 donne un Time On Air de **827,39 ms**
- ➔ En déduire le débit utile de cette transmission
  - **Cas 1 :** Pour SF7, 125 kHz et CR4/5 >  $\text{Débit}_{\text{utile\_LoRa}} = 8 / 25,85 \text{ ms} = 309,3 \text{ bps}$
  - **Cas 2 :** Pour SF12, 125 kHz et CR4/5 >  $\text{Débit}_{\text{utile\_LoRa}} = 8 / 827,39 \text{ ms} = 9,6 \text{ bps}$

### 3.5 Trame complète transmise en LoRaWAN

Le protocoles LoRaWAN a besoin de fournir des informations supplémentaires dans la trame transmise. Nous verrons au paragraphe 4.1 les différences entre le protocole LoRa et LoRaWAN, nous verrons aussi au paragraphe 6.1.1, le détail de la trame LoRaWAN. Dans l'immédiat, on peut simplement préciser que le LoRaWAN apporte un service supplémentaire au protocole LoRa.

La Figure 19 représente un trame LoRaWAN simplifiée. On retrouve l'entête du protocole LoRa, et un entête LoRaWAN a été ajouté. Cet entête LoRaWAN doit être transmis et augmente donc le temps de transmission alors que le nombre de bits utiles est le même. Cela réduit donc le débit utile de la transmission.

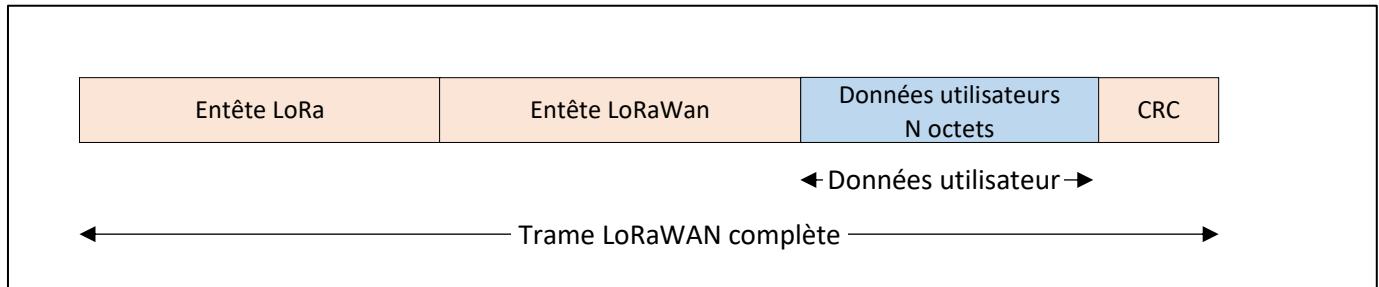


Figure 19 : Trame LoRaWAN

Nous pourrions simuler ce débit utile de la même façon que nous l'avions fait avec le LoRa calculator. Mais cette fois ci, je vous propose de le vérifier par l'intermédiaire d'une trame réelle envoyée depuis un Device LoRa à une Gateway. Dans cette application, nous transmettons un seul octet (une température). Un des rôle d'une Gateway est de nous fournir les informations sur la transmission. Nous recueillons les valeurs suivantes sur la Gateway pour un envoi en SF7, puis pour SF12. Nous nous intéressons à la colonne airtime (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
<hr/>							
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 20 : Time On Air pour un octet transmis en SF7 et SF12

On remarque que :

- L'envoi d'un octet en LoRaWAN en SF7 donne un Time On Air de **46,3 ms**
  - L'envoi d'un octet en LoRaWAN en SF12 donne un Time On Air de **1155.1 ms**
  - Le payload indiqué (14 octets) est très supérieur à 1 octet, ce qui montre qu'un entête supplémentaire a été ajouté.
- ➔ En déduire le débit utile de cette transmission
- **Cas 1 :** Pour SF7, 125 kHz et CR4/5 >  $\text{Débit}_{\text{utile\_LoRaWAN}} = 8 / 46.3 \text{ ms} = 172.7 \text{ bps}$
  - **Cas 2 :** Pour SF12, 125 kHz et CR4/5 >  $\text{Débit}_{\text{utile\_LoRaWAN}} = 8 / 1155.1 \text{ ms} = 6,9 \text{ bps}$

### 3.6 Duty-cycle en LoRaWAN

La norme LoRaWAN impose qu'un Device LoRa ne transmette pas plus de 1% du temps. Cela est dénommé le Duty Cycle. Par exemple un Duty Cycle de 1% signifie que si j'émetts pendant 1 (temps sans unité), je ne dois plus émettre pendant 99, quel que soit l'unité de temps utilisée.

Exemple : A la Figure 20, dans le cas du SF7, le Time On Air est de 46,3 ms. Le Device LoRa ne doit donc plus émettre pendant  $99 \times 46,3\text{ms} = 4,58$  secondes.

- ➔ En reprenant les exemples précédents, quels sont les débits moyens si on prend en compte le « Time On Air » et le duty cycle de 1% de la norme LoRaWAN?
- **Cas 1 :** Pour SF7, 125 kHz et CR4/5 >  $\text{Débit}_{\text{final}} = 172.7 \text{ bps} / 100 = 1,73 \text{ bps}$
- **Cas 2 :** Pour SF12, 125 kHz et CR4/5 >  $\text{Débit}_{\text{final}} = 6,9 \text{ bps} / 100 = 0,07 \text{ bps}$

### 3.7 Consommation énergétique

Le consommation d'un système LoRa dépend de plusieurs paramètres :

- La quantité de données à émettre (Payload)
- Le Spreading Factor
- Les éventuelles collisions à l'émission (et donc retransmission)
- La demande d'acquittement des trames émises
- Le duty-cycle
- La puissance d'émission du transceiver
- La puissance consommée en veille entre 2 transmissions

Le site <https://bit.ly/3f6FZDD> permet d'avoir une première approximation de la consommation et donc de l'autonomie d'un device LoRa.

- ➔ Trouver l'autonomie de device LoRa pour les caractéristiques suivantes :
- SF7, CR4/5, BW125
- Un relevé d'une température (1 octet) toutes les heures
- La batterie est constituée d'une pile [ AA ]

### 3.8 Mise en œuvre : LoRa en point à point

Ce test sera réalisé avec des modules The Things UNO. Il s'agit d'une carte arduino (Leonardo) associée à un module LoRa RN2483 (Microchip). Nous allons réaliser la manipulation suivante :

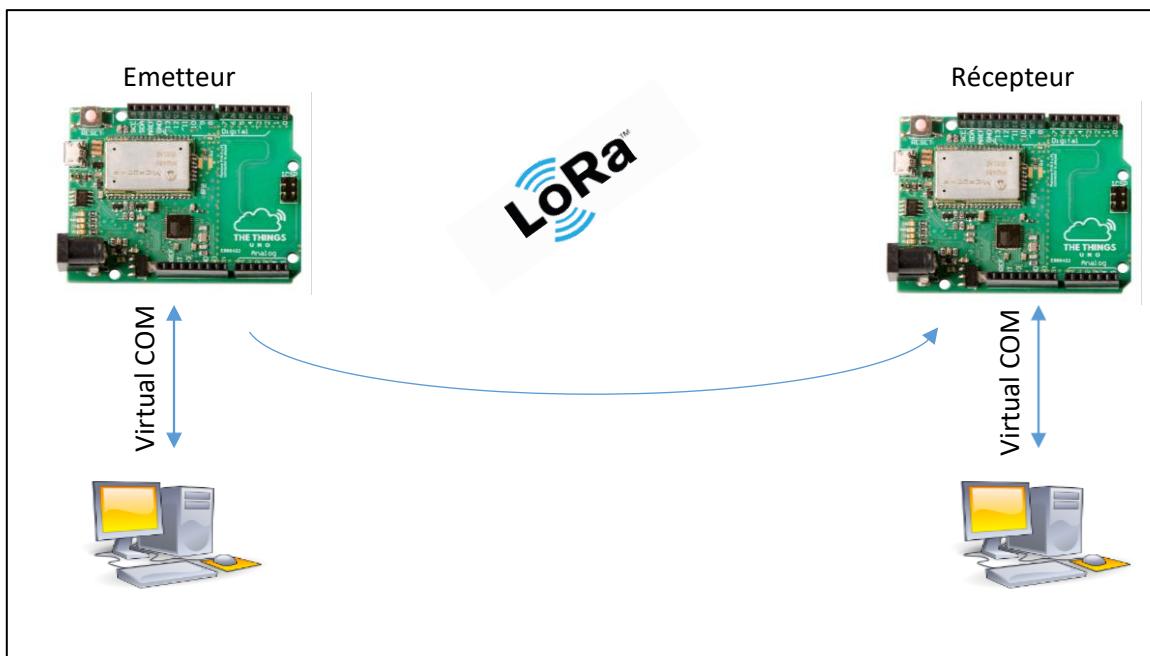


Figure 21 : Communication LoRa en Point à Point

### 3.8.1 Utilisation de l'IDE Arduino

L'environnement Arduino est très simple. Nous allons réaliser un programme pour montrer son fonctionnement.

- ➔ La connexion de la carte est effectuée à l'aide d'un port USB qui émule une liaison série RS232. Il faut donc choisir le bon port : **Arduino IDE > Outils > Port**.
- ➔ Pour créer un nouveau programme (sketch) : **Arduino IDE > Fichier > Nouveau**

Il y a deux parties dans un sketch :

- Une fonction `setup()` qui sera exécutée qu'une seule fois au démarrage
- Une fonction `loop()` qui sera exécutée en boucle
- ➔ Ecrire le code suivant :

```

void setup() {
  Serial.begin(9600);
}

void loop() {
  Serial.println("hello word");

}

```

- ➔ Compiler et Téléverser : **Croquis > Téléverser**
- ➔ Voir la sortie sur la liaison série : **Outils > Moniteur Série > Choisir la vitesse de transmission, ici 9600 bauds (en bas à droite)**.

### 3.8.2 Validation du fonctionnement

Deux binômes joueront le rôle d'émetteur (LoraTX.ino) , tous les autres binômes seront en récepteur (LoraRX.ino).

- ➔ Récupérer les programmes dans Moodle dans le dossier « **TheThingsUNO-Point2Point** ».
- ➔ Répartissez les Emetteurs / Récepteurs et récupérer le programme LoraTX.ino ou LoraRX.ino en fonction de votre rôle. Les 2 binômes en émission devront modifier le code du sketch Arduino pour transmettre des données spécifiques afin de les différencier. [ loraSerial.println("radio tx XX"); ]. XX étant la donnée en hexa qui sera transmise en LoRa.
- ➔ Validez la réception des données émises par les deux binômes qui transmettent des données. Ces données reçues sont écrite à **115200 bauds** sur le moniteur série de l'Arduino.

### 3.8.3 Mise en valeur du Spreading Factor

On s'aperçoit dans la manipulation précédente que les Devices LoRa reçoivent les données des deux émetteurs. Nous allons conserver le même canal d'émission (869,1 Mhz), mais un des deux émetteurs va modifier son « Spreading Factor ». La salle sera donc séparée en deux groupes.

- 1<sup>er</sup> groupe : Un émetteur et au minimum un récepteur utiliseront un SF7
  - 2<sup>ème</sup> groupe : Un émetteur et au minimum un récepteur utiliseront un SF8
- 
- ➔ Validez la réception des données émises seulement pour le Spreading Factor que vous utilisez.

## 4 Le protocole LoRaWAN

### 4.1 Différences entre LoRa et LoRaWAN

Le protocole LoRa est le type de modulation utilisé entre deux Devices LoRa ou entre un Device et une Gateway. Lorsque nous parlons de l'ensemble de la chaîne de communication alors nous parlons de protocole LoRaWAN.

- Protocole LoRa : Type de modulation (Chirp Spread Spectrum) permettant d'envoyer des données entre un émetteur et un récepteur.
- Protocole LoRaWAN : Architecture du réseau (Device, Gateway, Serveurs) et format de trame spécifique permettant à un Device LoRaWAN de transmettre des données à un serveur LoRaWAN.

### 4.2 Structure d'un réseau LoRaWAN

Nous retrouvons d'un côté le Device LoRa qui transmet une donnée. Cette donnée est réceptionnée à l'autre extrémité du réseau par un Utilisateur. La structure globale du réseau LoRaWAN peut être représentée par la figure suivante :

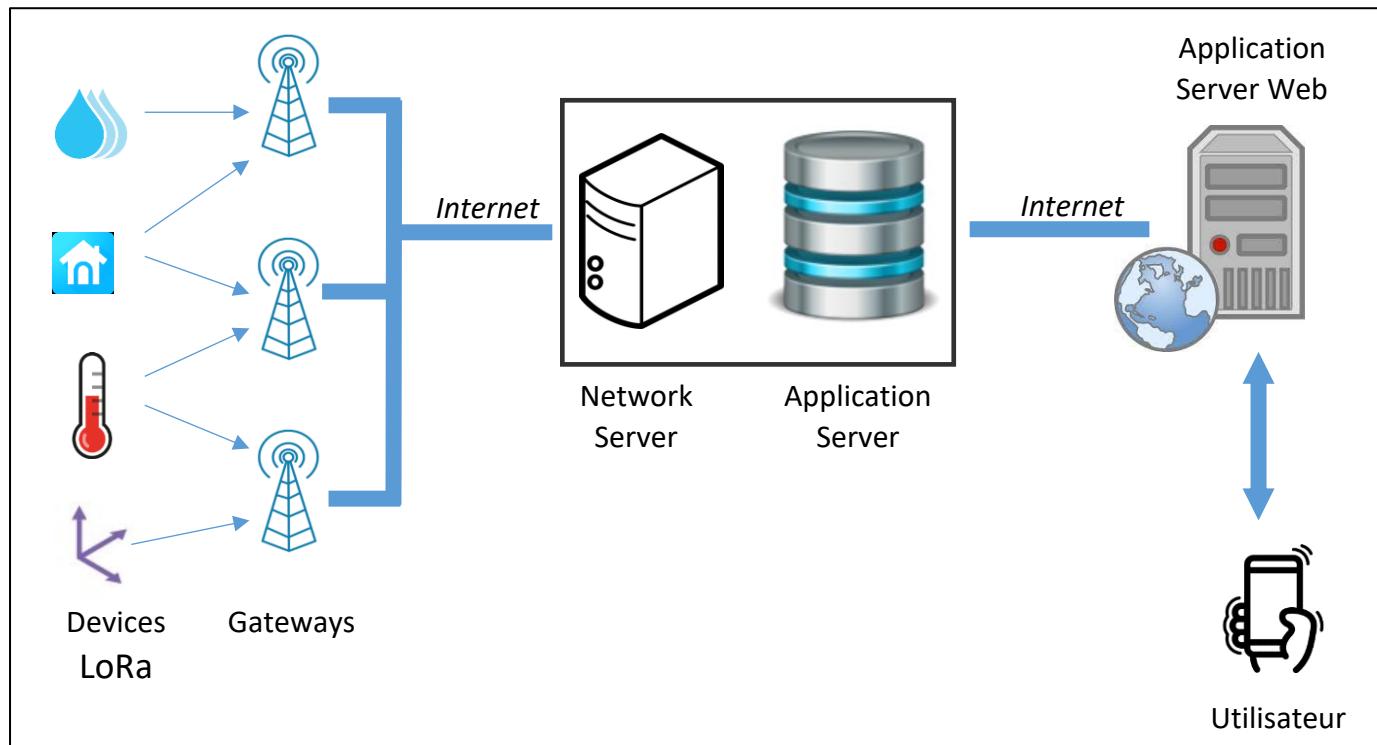


Figure 22 : Structure globale d'un réseau LORAWAN

#### 4.2.1 Les Devices LoRa

Les Devices LoRa sont des systèmes électroniques appartenant au monde de l'IoT : Faible consommation, faible taille, faible puissance et faible coût.

Ils possèdent une radio LoRa permettant de joindre les Gateways. Les Gateways ne sont pas adressées spécifiquement : toutes celles présentes dans la zone de couverture reçoivent les messages et les traitent.

#### 4.2.2 Les Gateways LoRa

Elles écoutent sur tous les canaux, et sur tous les Spreading Factor. Lorsqu'une trame LoRa est reçue, elle transmet son contenu sur internet à destination du Network Server qui a été configuré dans la Gateway au préalable.

Elle joue donc le rôle de passerelle entre une modulation LoRa, et une communication IP.

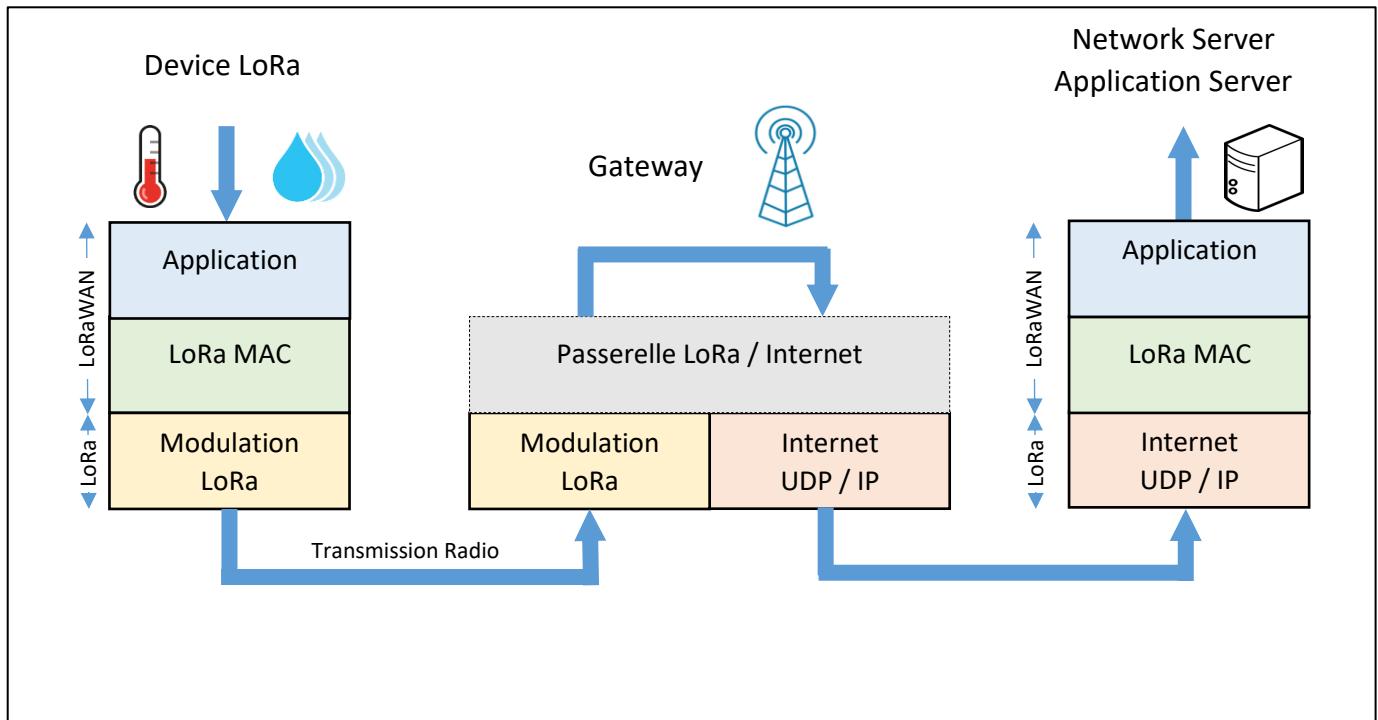


Figure 23 : Le rôle de la Gateway LoRa

Chaque Gateway LoRa possède un identifiant unique (EUI sur 64 bits). Cet identifiant est utile pour enregistrer et activer une Gateway sur un Network Server.

#### 4.2.3 Le Network Server

Le Network Server reçoit les messages transmis par les Gateways et supprime les doublons (plusieurs Gateway peuvent avoir reçu le même message). Les informations transmises au Network Server depuis les Devices LoRa sont authentifiées grâce à une clé AES 128 bits appelée **Network Session Key : NwkSKey**. Nous parlons bien ici d'authentification, et non pas de chiffrement comme nous le verrons au paragraphe 4.2.6.

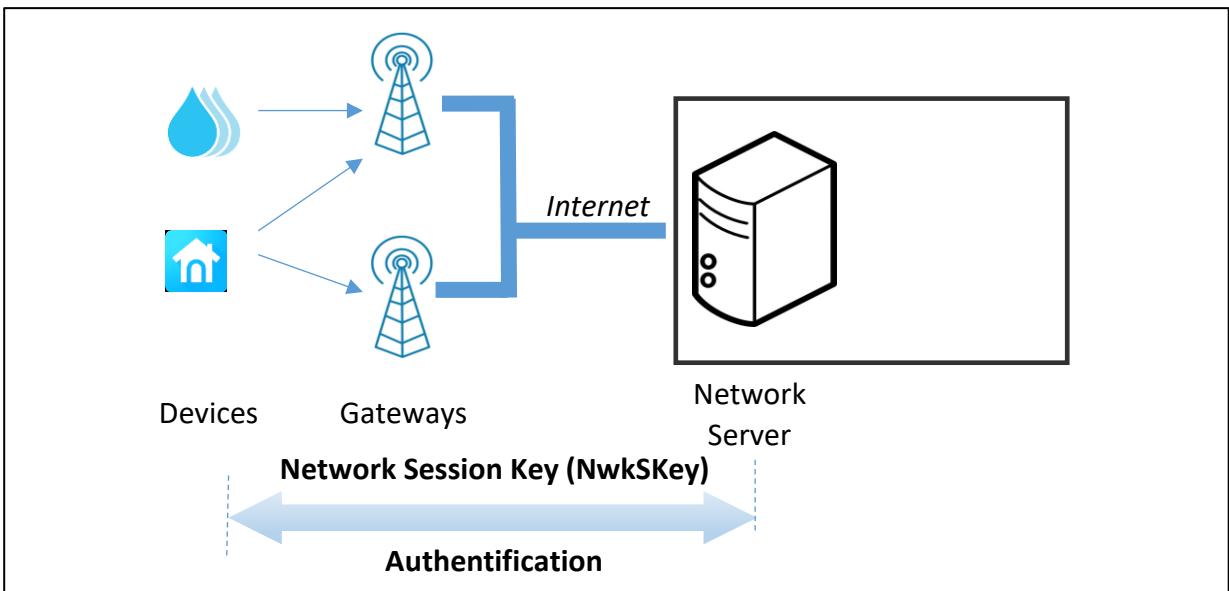


Figure 24 : Authentification entre le Device LoRa et le Network Server

#### 4.2.4 Application Server

Il est souvent sur le même support physique que le Network Server. Il permet de dissocier les applications les unes des autres. Chaque application enregistre des Devices LoRa qui auront le droit de stocker leurs données (Frame Payload). Les messages transmis à l'application server sont chiffrés

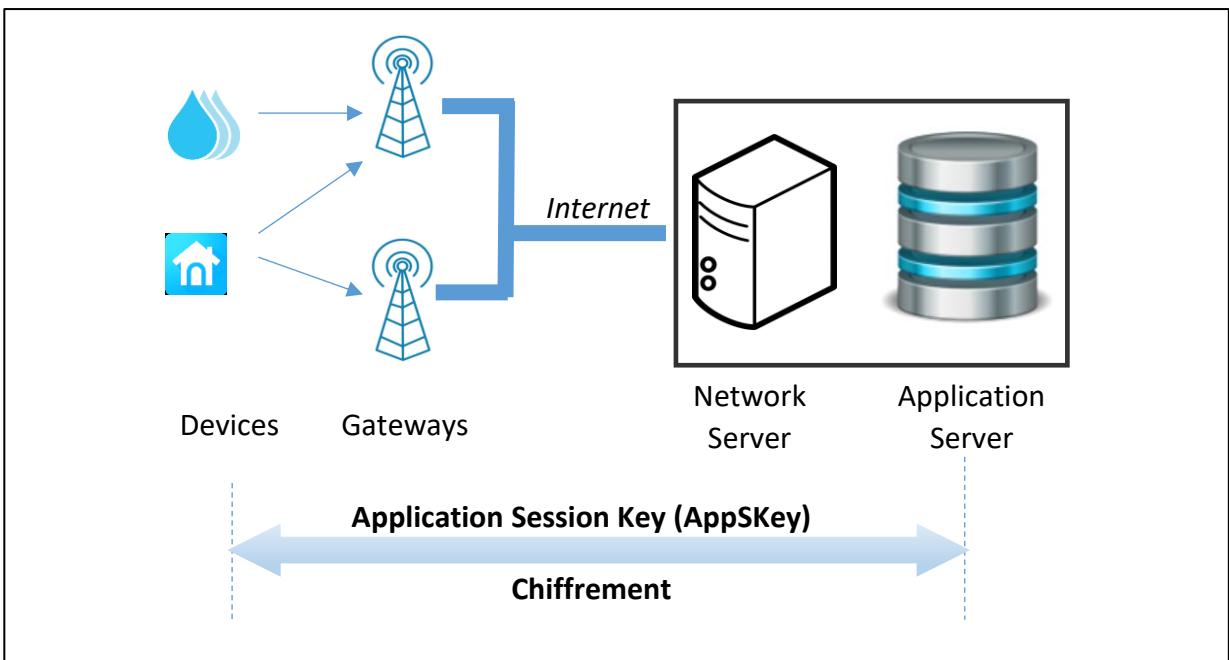


Figure 25: Chiffrement entre le Device LORA et l'Application Server

grâce à une clé AES 128 bits appelée Application Session Key : **AppSKey**. Contrairement au **NwkSKey** (**Network Session Key**), il s'agit bien ici d'un chiffrement comme nous le verrons au paragraphe 4.2.7.

Le schéma suivant résume l'utilisation :

- De la **Network Session Key (NwkSKey)** pour l'authentification entre le Device LoRa et le Network Server

- De l'Application Session Key (**AppSKey**) pour le chiffrement entre le Device LoRa et l'Application Server.

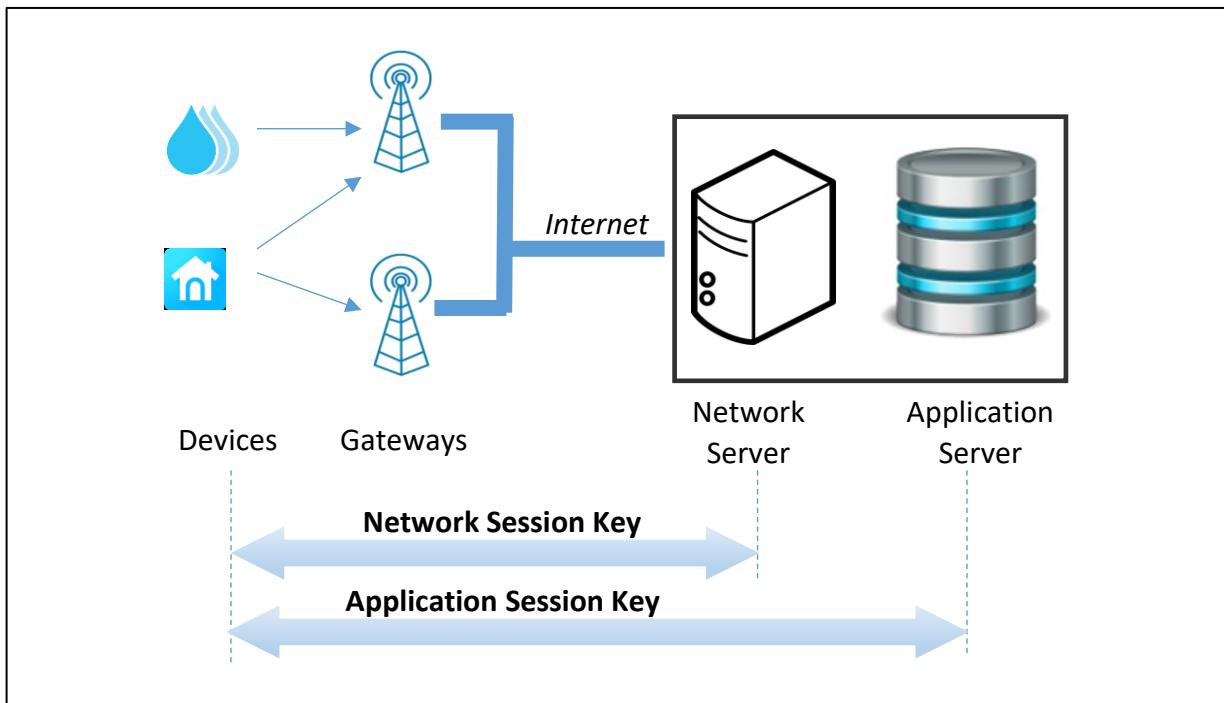


Figure 26 : Authentification et chiffrement en LoRaWan

#### 4.2.5 Application / Web Serveur

Cette application doit d'une part récupérer les données de l'Application Serveur. Dans le cadre de ce cours, cela sera fait de deux façons :

- Avec le protocole HTTP
- Avec le protocole MQTT

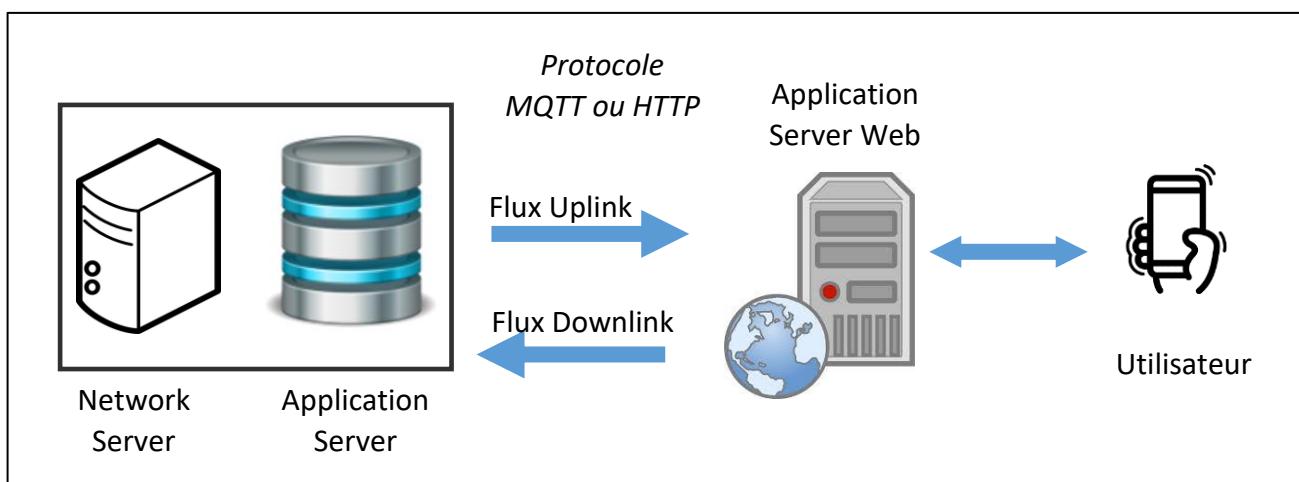


Figure 27 : Application / Web Server

 **Le flux habituel dans l'IoT est le flux Uplink (flux montant), c'est-à-dire l'émission de données des objets vers le serveur. Comme nous l'expliquerons au paragraphe 4.3, en**

**LoRaWAN il est aussi possible de transférer des données aux Device LoRa par un flux Downlink (flux descendant).**

D'autre part, l'application mettra à disposition les données aux utilisateurs sous forme d'un serveur web par exemple.

#### 4.2.6 Authentification avec le Network Server

La Network Session Key (**NwkSKey**) sert à l'authentification entre le Device LoRa et le Network Server. Afin de réaliser cette authentification entre le Device et le Network Server, un champ **MIC** (**Message Integrity Control**) est rajouté à la trame. Il est calculé en fonction des données transmises et du **NwkSkey**. A la réception, le même calcul est effectué. Si les clés NwkSkey sont équivalentes dans le Device et dans le Network Server alors les deux MIC calculés doivent correspondre.

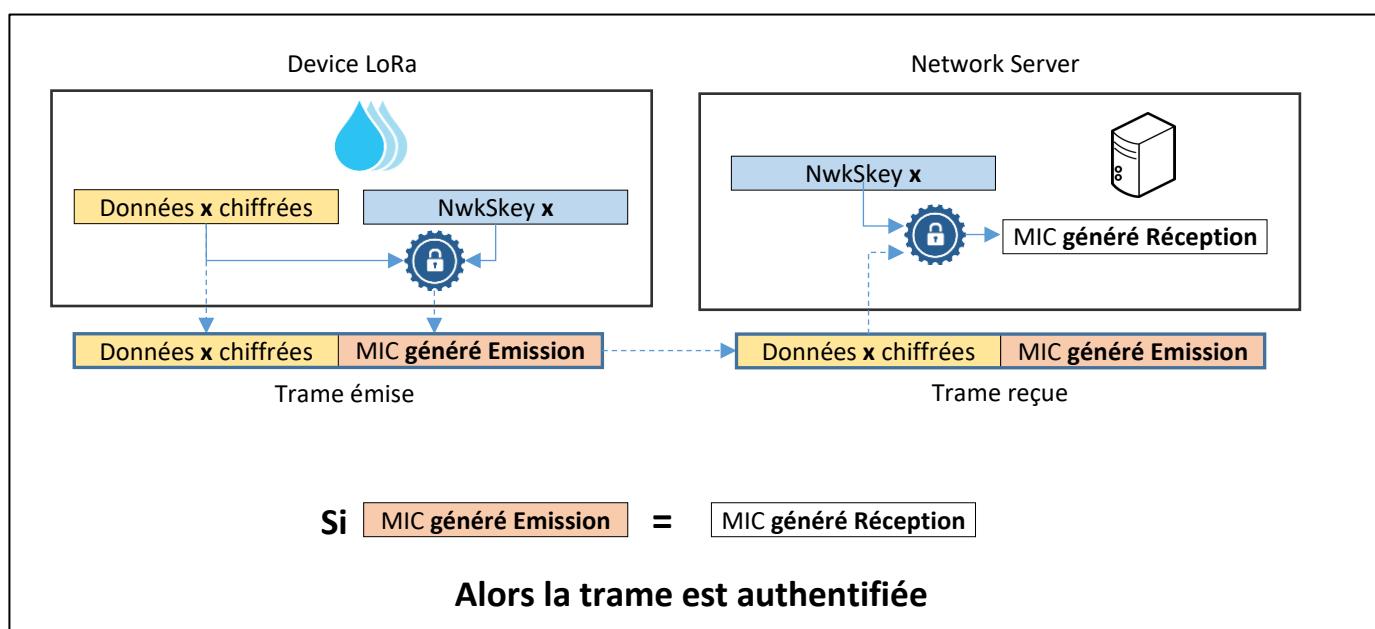


Figure 28 : Authentification d'un Device LoRa par le Network Server

Dans le schéma de la Figure 28, on parle de "Données x chiffrées". En effet, ces données ont au préalable été chiffrées par **l'AppSKey** avant de passer par le processus d'authentification décrit ici.

#### 4.2.7 Chiffrement des données vers l'Application Server

L'**Application Session Key (AppSKey)** sert pour le chiffrement entre le Device LoRa et l'Application Server. Les données chiffrées seront alors décodées à la réception sur l'Application Server s'il possède la même clé. Si on reprend le schéma précédent, les « données x » sont chiffrées / déchiffrées par le processus suivant :

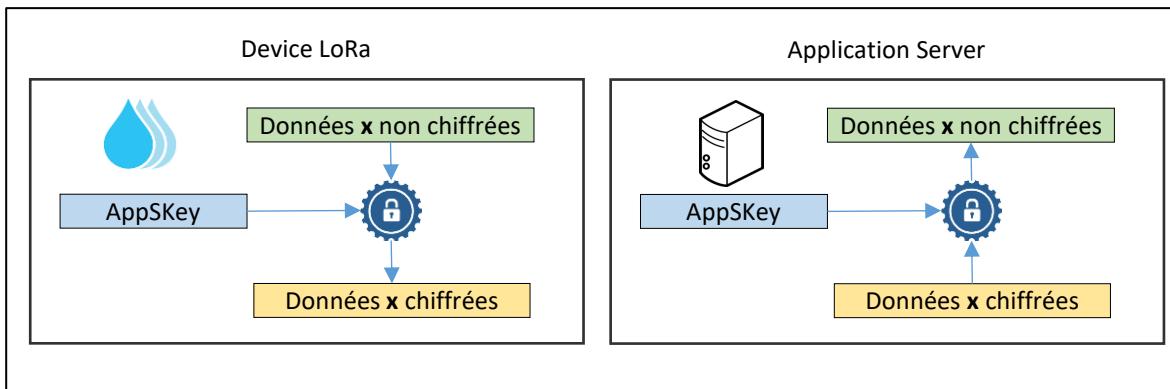


Figure 29 : Chiffrement des données par l'Application Session Key

#### 4.2.8 Combinaison de l'authentification et du chiffrement

On peut maintenant représenter sur le même schéma la construction de la trame LoRaWAN avec d'une part l'authentification et d'autre part le chiffrement.

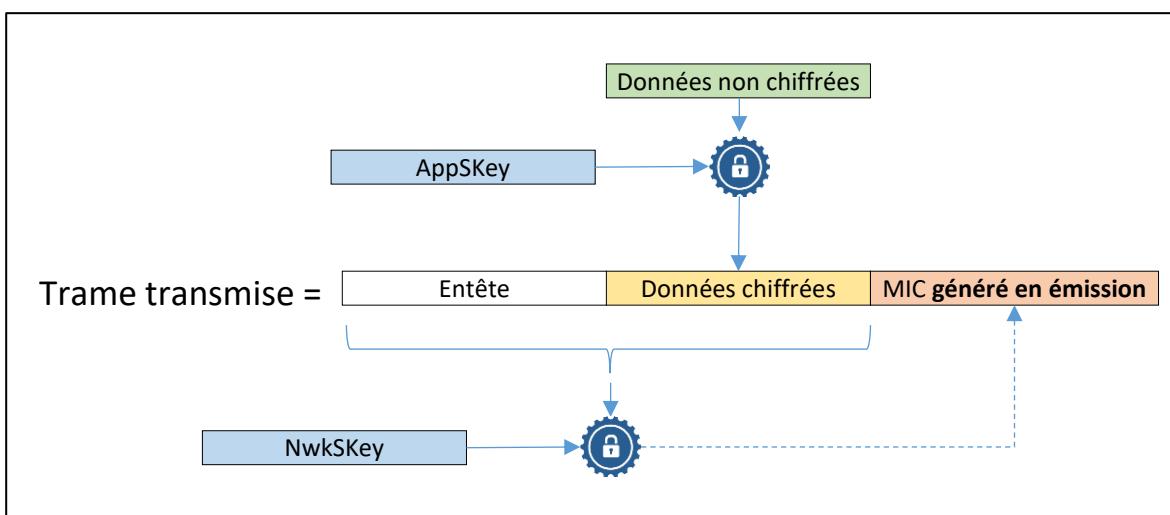


Figure 30 : Chiffrement, puis Authentification

### 4.3 Classes des Devices LoRaWAN

Les Devices LoRa sont classés en 3 catégories (A, B, C) en fonction de leur consommation et de leur accessibilité en **Downlink**, c'est-à-dire la facilité qu'un utilisateur aura à transmettre une trame au Device LoRa.

#### 4.3.1 Classe A (All) : Minimal power Application

Tous les Devices LoRaWAN sont de classe A. Chaque Device peut transmettre (Uplink) à la Gateway sans vérification de la disponibilité du récepteur. Si la transmission échoue, elle sera réémise après un certain temps. Cette transmission est suivie de 2 fenêtres de réception très courtes. La Gateway peut alors transmettre pendant le « RX Slot 1 » ou le « RX Slot 2 », mais pas les deux.

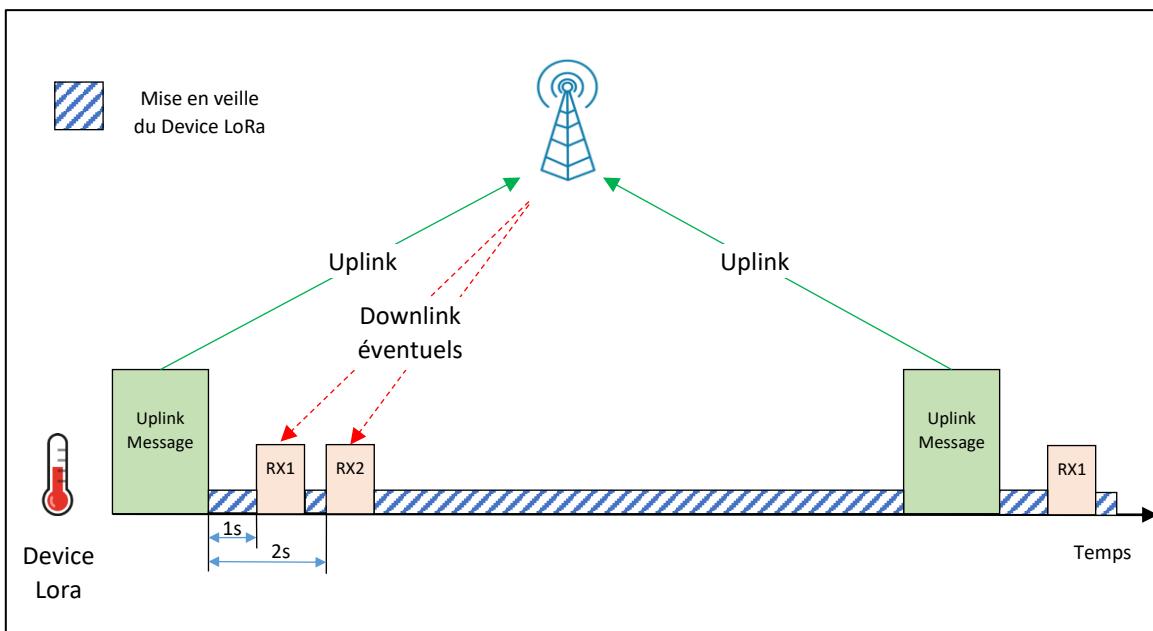


Figure 31 : Slots de réception pour un Device LoRa de classe A.

La durée des fenêtres doit être au minimum la durée de réception d'un préambule. Un préambule dure  $12.25 T_{\text{symbole}}$  et dépend donc du Data Rate (DR : voir paragraphe 4.6 pour plus d'information sur le DR) . Lorsque qu'un préambule est détecté, le récepteur doit rester actif jusqu'à la fin de la transmission. Si la trame reçue pendant la première fenêtre de réception était bien à destination du Device LoRa, alors la deuxième fenêtre n'est pas ouverte.

Première fenêtre de réception :

- Le Slot RX1 est programmé par défaut à 1 seconde +/- 20 µs après la fin de l'émission Uplink.
- La fréquence et le Data Rate (DR) sont les mêmes que ceux choisis lors de la phase d'émission (Uplink).

Seconde fenêtre de réception :

- Le Slot RX2 est programmé par défaut à 2 secondes +/- 20 µs après la fin de l'émission Uplink.
- La fréquence et le Data Rate (DR) sont configurables mais fixes.

 **Un Device LoRa qui est uniquement de classe A ne peut pas recevoir s'il n'a pas émis. Il n'est donc pas joignable facilement.**

#### 4.3.2 Classe B (Beacon) : Scheduled Receive Slot

Les Devices de classe B ont le même comportement que les Devices de classe A, mais d'autres fenêtres de réceptions sont programmées à des périodes précises. Afin de synchroniser les fenêtres de réception du Device LoRa, la Gateway doit transmettre des balises (Beacons) de façon régulière.

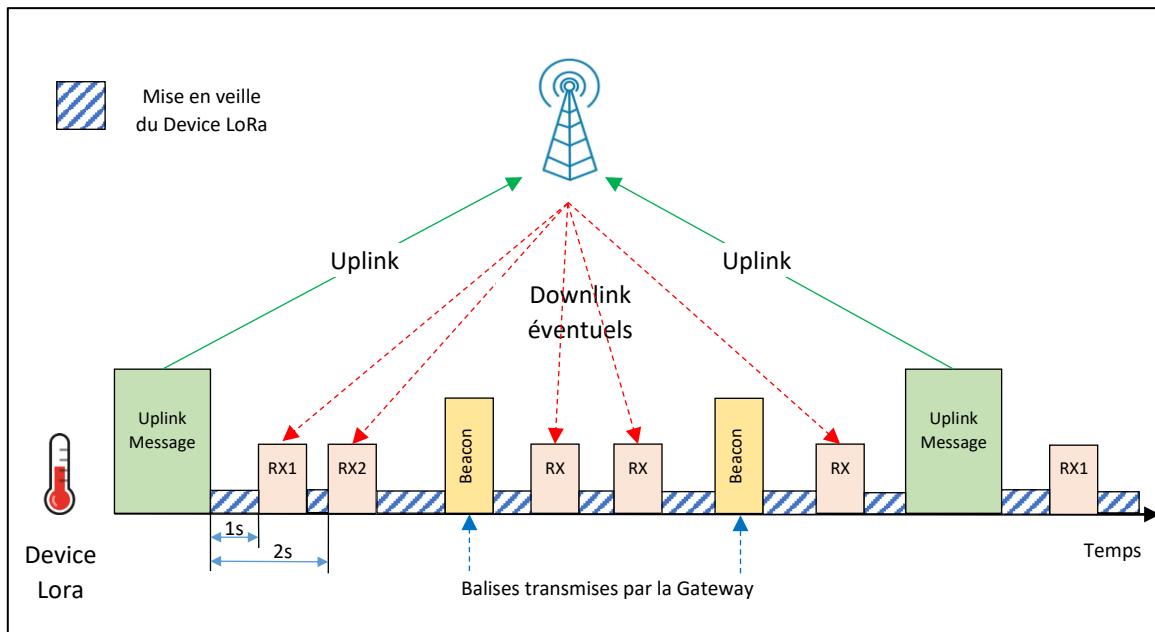


Figure 32 : Slots de réception pour un Device LoRa de classe B



Un Device LoRa de classe B est joignable régulièrement sans qu'il soit nécessairement obligé d'émettre. En revanche, il consomme plus qu'un Device de classe A.

#### 4.3.3 Classe C (Continuous) : Continuously Listening

Les Devices de classe C ont des fenêtres de réception constamment ouvertes entre 2 Uplinks. Ces Devices consomment donc beaucoup plus.

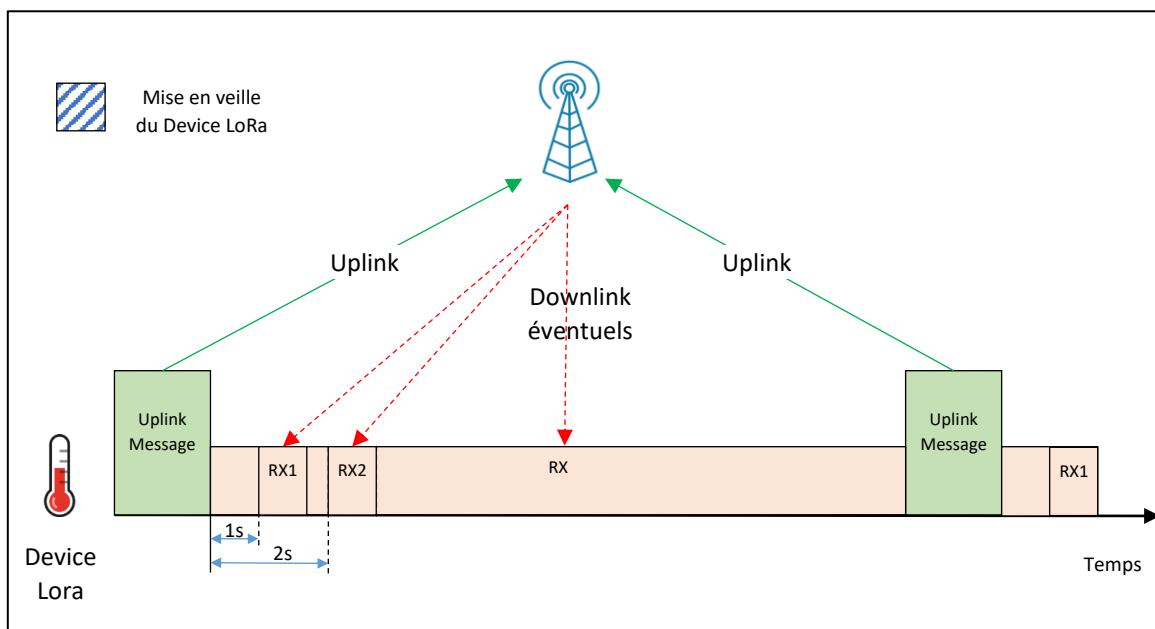


Figure 33 : Slots de réception pour un Device LoRa de classe C



- 💡 Un Device LoRa de classe C est joignable en permanence. En revanche, c'est la classe de Device qui est la plus énergivore.

#### 4.3.4 Résumé sur les classes de Device LoRa

- Un Device LoRa de classe B est aussi de classe A.
- Un Device LoRa de classe C est aussi de classe A.

On peut alors représenter les classes de Device LoRa par la suivante.

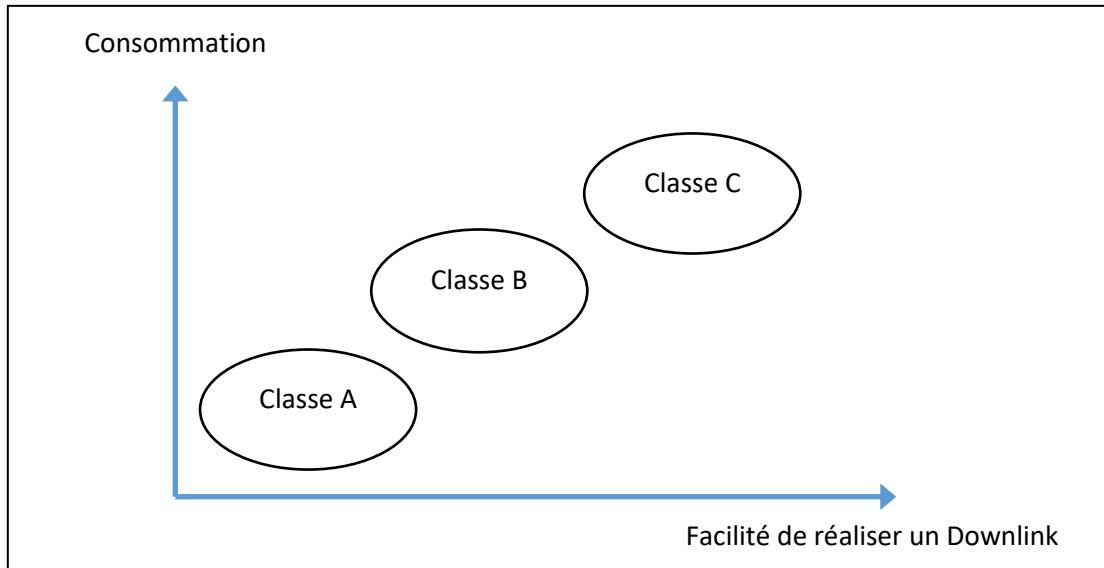


Figure 34 : Consommation et Downlink en fonction de la classe de Device

#### 4.3.5 Quelle Gateway pour les flux Downlink ?

Nous avons donc vu que le fonctionnement classique du protocole LoRaWAN était de récupérer de l'information depuis le Device LoRa : C'est le flux Uplink.

Dans le cas où l'utilisateur transmet des informations au Device LoRa (flux Downlink), on peut se demander quelle Gateway sera choisie pour transférer les données au Device LoRa. En effet, l'endroit où est situé le Device LoRa n'est pas nécessairement connu à l'avance.



- 💡 La Gateway utilisée pour le Downlink est celle qui a reçu le dernier message du Device LoRa. Un message ne pourra jamais arriver à destination d'un Device LoRa si celui-ci n'a jamais transmis, quel que soit sa classe A, B ou C.

### 4.4 Activation des Devices LoRa : ABP ou OTAA

En LoRaWAN, les trois éléments indispensables pour la communication sont le **DevAddr** pour l'indentification du Device, ainsi que deux clés : le **NwkSKey** pour l'authentification et l'**AppSKey**

pour le chiffrement. Deux méthodes sont possibles pour fournir ces informations à la fois au Device LoRa et au serveur :

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

#### 4.4.1 ABP : Activation By Personalization

C'est la plus simple des méthodes. C'est donc peut-être celle que nous avons tendance à utiliser lors d'un test de prototype et d'une mise en place de communication LoRaWAN.

- Le Device LoRa possède déjà le **DevAddr**, l'**AppSKey** et le **NwkSKey**
- Le Network server et application serveur possède déjà le **DevAddr**, **NwkSKey** et **AppSKey**

 En ABP, toutes les informations nécessaires à la communication sont déjà connues par le Device LoRa, par le Network Server et par l'Application Server.

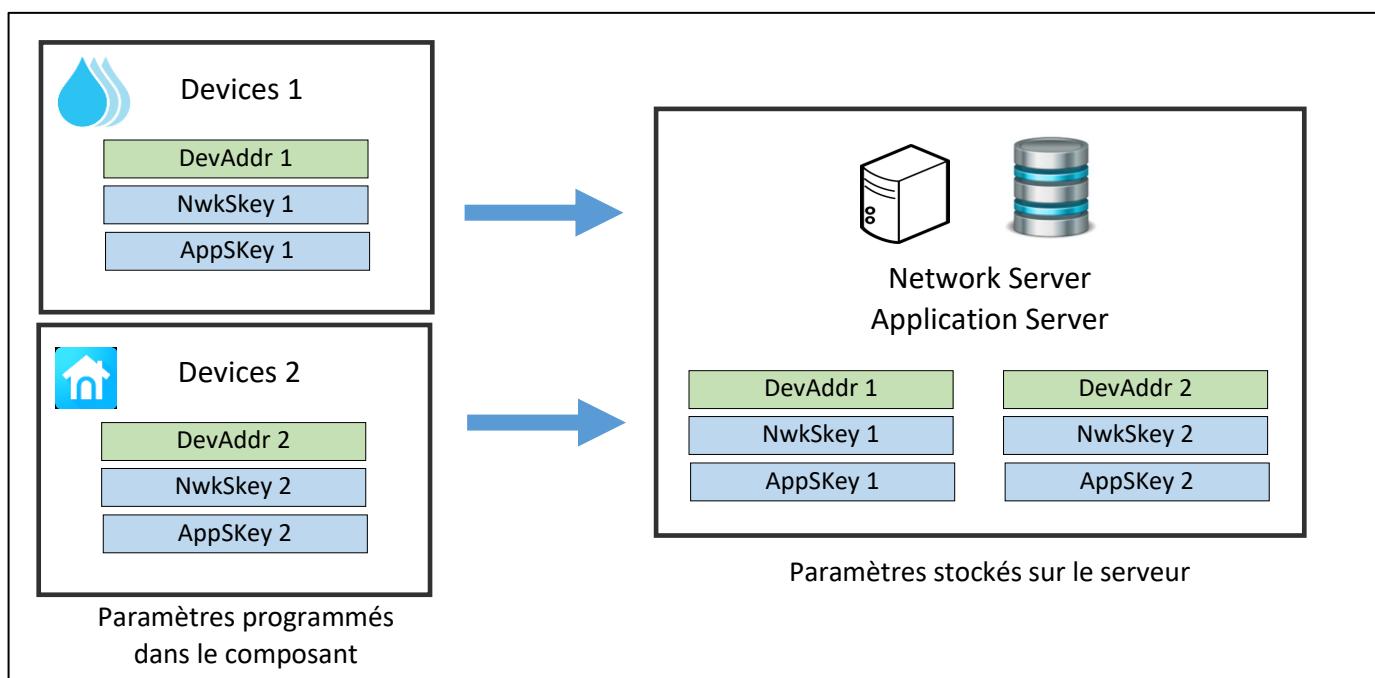


Figure 35 : Paramétrage de DevAddr, NwkSKey et AppSKey en ABP

#### 4.4.2 OTAA : Over The Air Activation :

Cette fois-ci le **DevAddr**, l'**AppSKey** et le **NwkSKey** vont être générés lors d'une procédure de négociation (Join Request) dès lors que le Device LoRa se connectera pour la première fois au serveur.

Au préalable, le Device LoRa doit connaître : le **DevEUI**, l'**AppEUI**, et l'**Appkey**. Le Network Server doit lui connaître le **DevEUI**, l'**AppEUI**, et l'**Appkey**.



⚠️ Tous les éléments notés EUI (Extended Unique Identifier) sont toujours sur une taille de 64 bits.

C'est donc uniquement grâce à ces informations de départ et à la négociation avec le Network Server (Join Request) que le Device LoRa et le serveur vont générer les informations essentielles : **DevAddr**, **NwkSKey** et **AppSKey**.

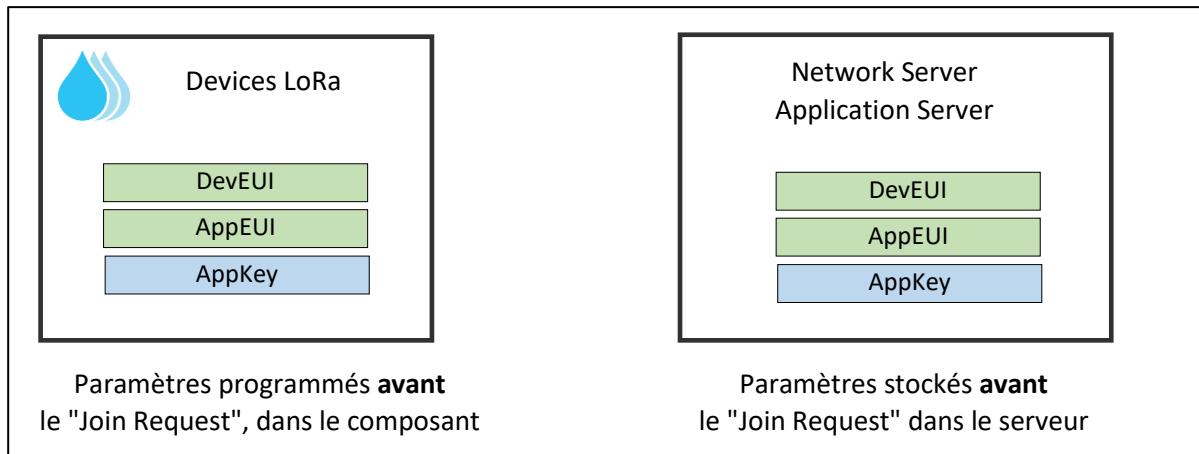


Figure 36 : Paramétrage du Device LoRa et du Serveur **avant** le Join Request (OTAA)

Lorsque le Join Request a eu lieu, alors les paramètres générés (**DevAddr**, **NwkSKey** et **AppSKey**) sont stockés dans le Device LoRa et dans les serveurs.

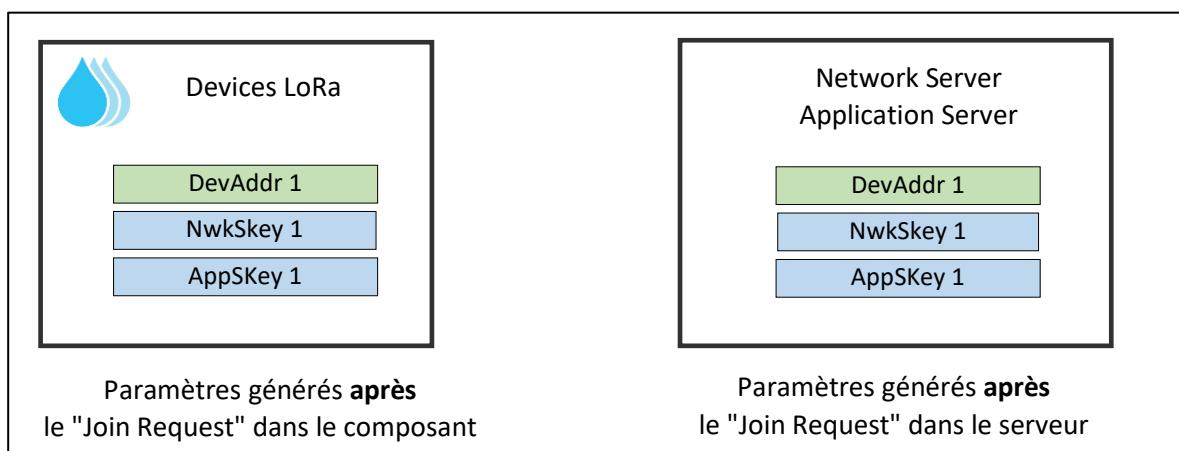


Figure 37 : Paramétrage du Device LoRa et du Serveur **après** le Join Request (OTAA)

Informations possédées par le Device LoRa **avant** le Join Request :

- **DevEUI** : Unique Identifier pour le device LoRa (Equivalent à une @MAC sur Ethernet). Certains Device LoRa ont déjà un DevEUI fourni en usine.
- **AppEUI** : Unique Identifier pour l'application server.
- **AppKey** : AES 128 clé utilisée pour générer le MIC (Message Code Integrity) lors de la Join Request. Il est partagé avec le Network server.

Informations possédés par le Device LoRa **après** le Join Request :

- **NwkSKey** : Utilisé pour l'authentification avec le Network Server
- **AppSKey** : Utilisé pour le chiffrement des données
- **DevAddr** : Identifiant sur 32 bits unique au sein d'un réseau LoRa.

La Figure 38 présente le déroulement de la négociation de ces paramètres :

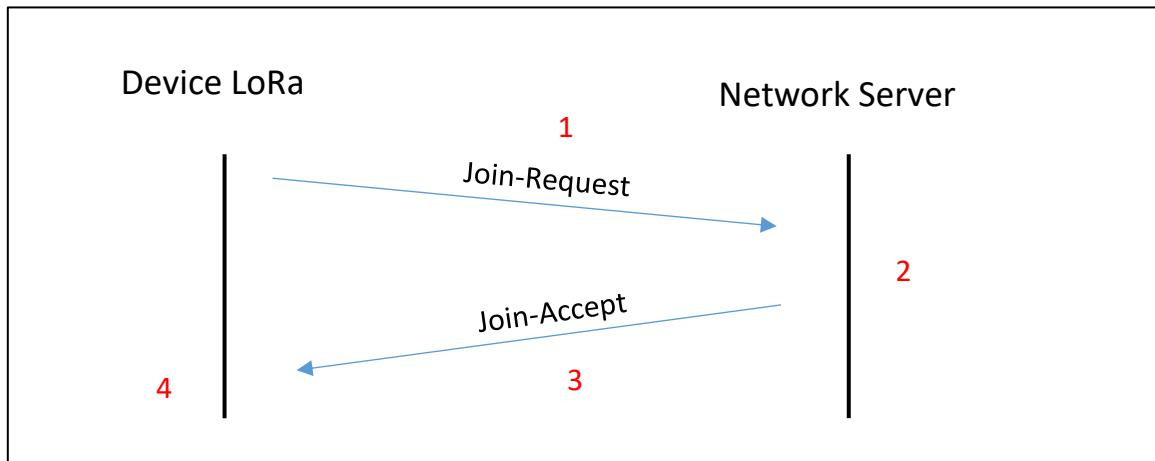


Figure 38 : Join Request – Join Accept en OTAA

1. Le Device LoRa émet un Join-Request à l'aide des informations **DevEUI**, **AppEUI** et **AppKey** qu'il possède.
2. Le Network Server authentifie le Join Request et le valide. Il génère alors une **NwkSKey**, une **AppSKey**, et un **DevAddr**.
3. Le Network Server retourne le **DevAddr**, ainsi qu'une série de **paramètres**.
4. Les **paramètres** fournis lors du Join-Accept, associé à l'**AppKey**, permettent au Device LoRa de générer le même **NwkSKey** et le même **AppSKey** qui avait été initialement généré sur le Network Server.

## 4.5 Choix de la technique d'activation : ABP ou OTAA ?

On se pose souvent la question de la pertinence d'utiliser l'une ou l'autre des méthodes d'activation. Nous allons voir quels sont les avantages de l'une et l'autre des deux méthodes.

### 4.5.1 Sécurité

D'un point de vue de la sécurité globale, les méthodes d'activation ABP et OTAA sont équivalentes. Le point faible de chacune des méthodes sont les clés stockées en permanence dans le Device LoRa:

- **NwkSKey et AppSKey** en ABP
- **AppKey** en OTAA

Elles devront donc être stockées dans des mémoires sécurisées.

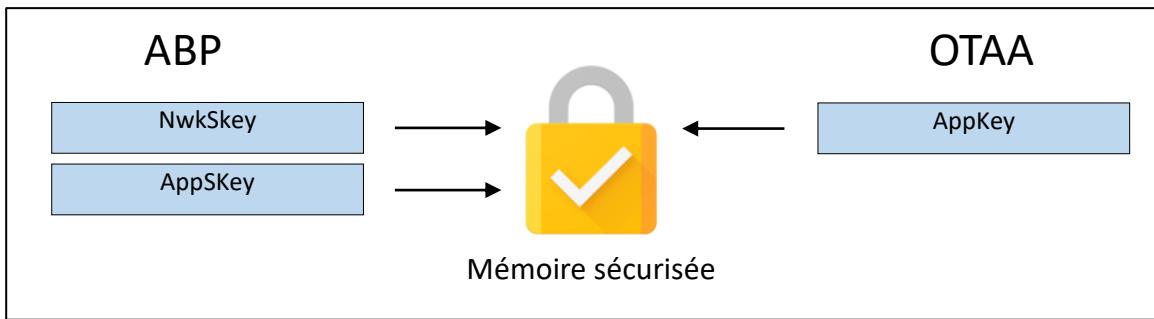


Figure 39 : Clés stockées dans une mémoire sécurisée

#### 4.5.2 Protection contre l'attaque par Replay

Les clés AES 128 bits permettent le chiffrement des informations transmises en LoRaWAN. Malgré ce chiffrement, une attaque connue en Wireless est celle du REPLAY. C'est-à-dire que le Hacker enregistre des trames chiffrées circulant sur le réseau LoRa pour les réémettre plus tard. Même si le Hacker ne comprend pas le contenu (les trames sont chiffrées), les données qui sont transportées sont, elles, bien comprises par l'Application Server. Des actions peuvent donc être réalisées simplement par mimétisme.

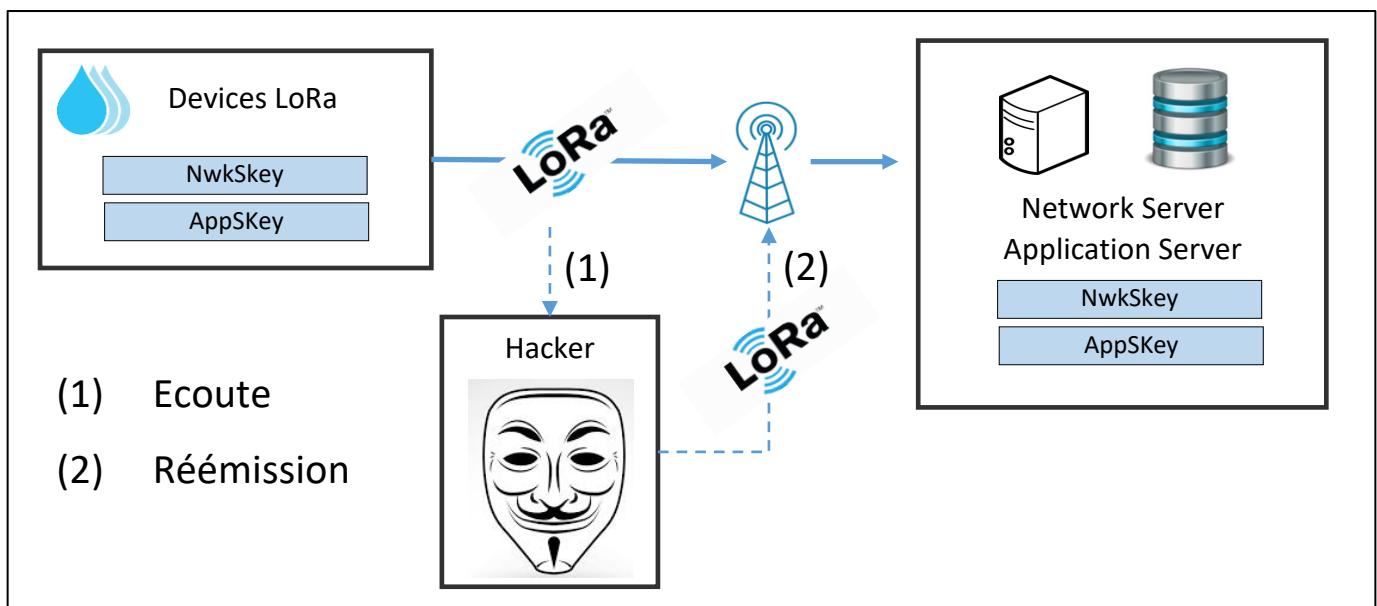


Figure 40 : Attaque par REPLAY

Pour éviter cela, la trame LoRaWAN intègre un champ variable appelé « Frame Counter ». Il s'agit d'un nombre sur 2 octets numérotant la trame. Le Server acceptera une trame uniquement si le « Frame Counter » reçu est supérieur au « Frame Counter » précédemment. Donc si le Hacker, retransmet la trame telle qu'elle, le « Frame Counter » sera équivalent, donc la trame sera refusée. Si le Hacker décide de modifier le champs "Frame Counter" avec une valeur au hasard, l'authentification échouera car le calcul du MIC avec le Network Session Key ne sera plus valide.

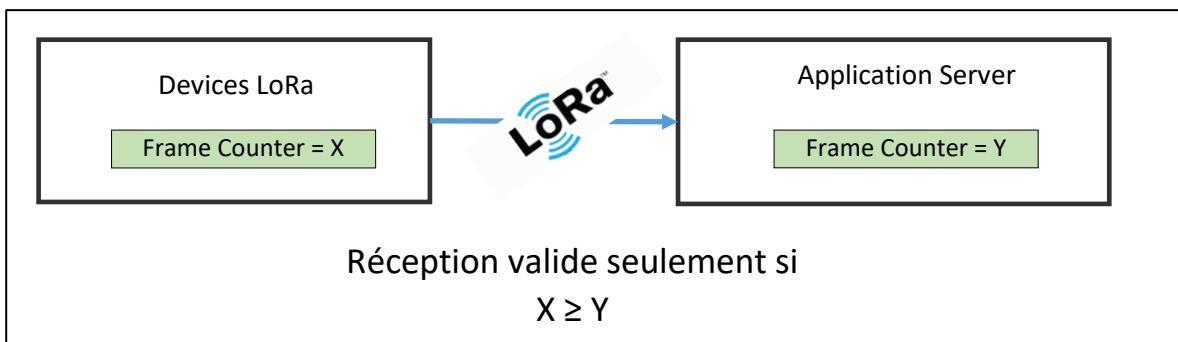


Figure 41 : Utilisation d'un "Frame Counter" pour éviter l'attaque par Replay

Cette sécurité implémentée par les « Frame Counter » est intéressante pour s'affranchir d'une attaque par REPLAY, mais dans nos tests cela peut poser problème. En effet, à chaque fois que nous redémarrons le microcontrôleur, notre « Frame Counter » revient à zéro, alors que celui de l'Application Server continue de s'incrémenter. Cela peut être résolu par différentes manières :

1. Désactivation du « Frame Counter Check » : Dans certains "Application Server", cette option peut être proposée. Bien sûr, il faudra garder en tête la faille de sécurité que cela engendre.

**Frame Counter Checks**

⚠ Disabling frame counter checks drastically reduces security and should only be used for development purposes

Figure 42 : Désactivation du "Frame Counter Check" dans TTN

2. Utiliser l'activation par OTAA au lieu de ABP. En effet, à chaque ouverture de session en OTAA, le « Frame Counter » est réinitialisé côté serveur.
3. Conserver la valeur du « Frame Counter » dans une mémoire non volatile et récupérer sa valeur au démarrage du Device LoRa.

#### 4.5.3 L'attribution du devAddr

Le devAddr nécessaire à une communication LoRa est généré par le Network Server. Les devAddr ne sont pas uniques. C'est l'association DevAddr / NwSKey qui permet d'identifier le Device parmi les milliards de systèmes LoRaWAN existants. Si le Network server que vous utilisez est répertorié auprès de la "LoRa Alliance" [ <https://lora-alliance.org> ], alors les devAddr qui seront attribués auront un préfixe. Par exemple, "The Things Network fondation" possède des devAddr qui commenceront toujours par 0x2601.

Que se passe-t-il si vous décidez de changer de Network Server, en passant par exemple de TTN à Loriot [ [www.loriot.io](http://www.loriot.io) ]? Cela dépend de votre mode d'activation :

Si vous avez choisi le mode OTAA : Il faut que le Device LoRa réémette un "Join Request" au server pour se voir attribuer un nouveau jeu de paramètres (DevAddr, NwSKey, AppSKey). Une nouvelle adresse DevAddr sera donc disponible et le Device pourra continuer à fonctionner.

Si vous fonctionnez en ABP : Ce cas est problématique car vous avez le DevAddr fixé de façon permanente au Device LoRa. La seule solution est donc de reprogrammer le firmware du Device, ce qui peut être très compliqué si la flotte de Device est déjà en service.

#### 4.5.4 RX Delay and CFList

Lorsque nous fonctionnons en OTAA, le Device Lora émet un "Join Request". Un "Join Accept" est retourné par le Network Server si le Device a été autorisé à échanger avec le serveur. Le "Join Accept" comporte :

- Un DevAddr pour l'adressage du Device LoRa
- Des paramètres permettant au Device de générer le NwSKey et AppSKey (voir chapitre 4.4.2)

En plus de ces éléments que nous avons déjà vu, le "Join Accept" comporte

- Un champ **RX Delay**
- Un champ **CFList**.

Le **RX Delay** permet de modifier le temps entre la fin de l'émission (Uplink) et le début de la fenêtre de réception (Downlink). **RX Delay** est de 1s par défaut.

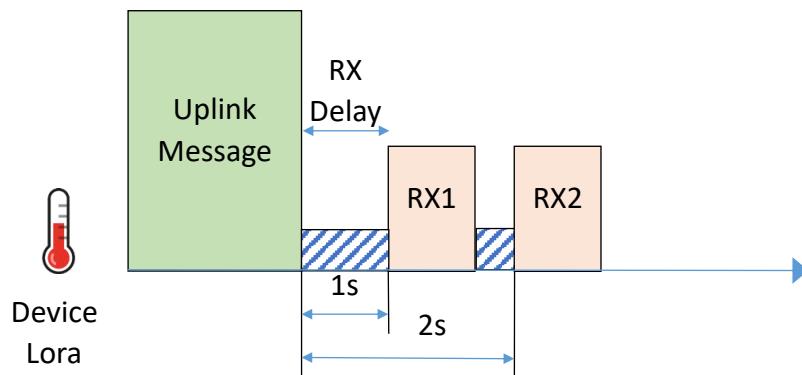


Figure 43 : Le RX Delay

**CFList** est une liste permettant de spécifier au Device LoRa les canaux disponibles pour ce réseau en plus des canaux 0 à 2 (qui ne sont pas modifiables) : 868.1 Mhz, 868.3 Mhz et 868.5 Mhz. Les autres canaux peuvent être préprogrammés dans le Device LoRa ou ils peuvent être redéfinis dans la trame "Join Accept" (canaux 3 à 7) comme le montre la Figure 44.

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

Figure 44 : Canaux 4 à 8 pouvant être définies dans la CFList du Join Accept

Ces deux paramétrages étant présents uniquement dans la trame "Join Accept", un Device fonctionnant en ABP ne pourra pas en bénéficier.

#### 4.5.5 Résumé

Nous pouvons donc résumer les deux méthodes d'activation dans le Tableau 6.

	<b>ABP</b>	<b>OTAA</b>
<b>Sécurité globale</b>	Stockage en mémoire sécurisée : <b>NwSkey et AppSKey</b>	Stockage en mémoire sécurisée : <b>AppKey</b>
<b>Gestion du Frame Counter</b>	Sauvegarde en mémoire non volatile obligatoire Possibilité de désactiver la vérification du compteur en risquant des attaques par replay.	Pris en charge par l'OTAA
<b>Changement de Network</b>	Pas possible	Pris en charge par l'OTAA
<b>Modification du RX Delay</b>	Pas possible	Pris en charge par l'OTAA
<b>Ajout de canaux</b>	Pas possible	Pris en charge par l'OTAA

Tableau 6 : Comparaison des méthodes d'activation ABP et OTAA

## 4.6 Canaux et Data Rate (DR) et puissance

### 4.6.1 Les DR (Data Rate)

Comme nous l'avons vu au paragraphe 3.1.2, le Spreading Factor (SF) et la bande passante (BW) sont les deux paramètres qui ont des conséquences sur le débit de la transmission. L'association d'un SF et d'un BW sont notés DR (Data Rate) et sont normalisés de DR0 à DR6 pour la modulation LoRa :

<b>Data Rate</b>	<b>Spreading Factor</b>	<b>Bandwidth</b>
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

Tableau 7 : Dénomination DR en fonction du SF et de la Bande Passante

### 4.6.2 Les canaux

Le LoRa utilise des bandes de fréquences différentes suivant les régions du monde. En Europe, la bande utilisée est celle des 868 Mhz [ De 863 Mhz à 870 Mhz ]. Parmi cette bande, le serveur LoRaWAN définit un plan avec un certain nombre de canaux et Data Rate à utiliser en Uplink et en Downlink. Un Device LoRa doit **obligatoirement** connaître les canaux 868.1 Mhz, 868.3 Mhz et 868.5 Mhz de DR0 à DR5. Les autres canaux dépendent du réseau sur lequel vous exploitez les Devices LoRa. Le Tableau 8 représente les canaux obligatoires ainsi que les canaux spécifiques à The Things Network (TTN).

Canaux	Canaux	Spreading Factor	Bandwidth	Direction
Obligatoires	868.1 Mhz	De SF7 à SF12	125 kHz	Uplink / Downlink
	868.3 Mhz	De SF7 à SF12	125 kHz	Uplink / Downlink
	868.5 Mhz	De SF7 à SF12	125 kHz	Uplink / Downlink
Spécifiques à TTN	868.3 Mhz	SF7	250 kHz	Uplink / Downlink
	867.1 Mhz	De SF7 à SF12	125 kHz	Uplink / Downlink
	867.3 Mhz	De SF7 à SF12	125 kHz	Uplink / Downlink
	867.5 Mhz	De SF7 à SF12	125 kHz	Uplink / Downlink
	867.7 Mhz	De SF7 à SF12	125 kHz	Uplink / Downlink
	867.9 Mhz	De SF7 à SF12	125 kHz	Uplink / Downlink
	869.5 Mhz	SF9	125 kHz	Downlink

Tableau 8 : Canaux, SF et Bandwidth de notre Gateway LoRaWAN



💡 Pour écouter tout le flux Uplink d'un Device enregistré sur TTN, une Gateway LoRa doit écouter sur 8 canaux simultanément avec les 6 Spreading Factor différents, soit 48 combinaisons possibles.

#### 4.6.3 La puissance d'émission

La puissance d'émission maximum ( $P_E$ ) est de 14 dBm (25 mW) sur la bande des 868 Mhz en Europe. Le protocole LoRaWAN définit des puissances sous forme d'index (de 1 à 5) qui correspondent aux valeurs présentées dans le

TX Power Index	Valeur en dBm
1	14 dBm
2	11 dBm
3	8 dBm
4	5 dBm
5	2 dBm

Tableau 9 : Correspondance entre l'index TX Power et sa valeur en dBm

#### 4.6.4 L'ADR (Adaptive Data Rate)

La consommation énergétique d'un Device LoRa est directement liée à deux paramètres de la transmission LoRa :

- Le "Time On Air" (voir chapitre 3.4). En effet, plus le message est long à être envoyé, plus la radio sera alimentée pendant un long moment.
- La puissance d'émission ( $P_E$ )

Une solution simple pour réduire le "Time On Air" est de réduire le SF (Spreading Factor). En effet, lorsqu'on réduit le SF de 1, le "Time On Air" est divisé par deux (voir chapitre 3.1.2). Pour la puissance d'émission ( $P_E$ ), on peut la réduire directement dans la configuration du Transceiver. Bien sûr, ces modifications ont des conséquences, et il faut donc les ajuster avec cohérence. Le Tableau 10 répertorie les meilleures sensibilités, ainsi que le pire SNR acceptable sur le récepteur, en fonction du SF utilisé à l'émission. Ces données sont une compilation des données de la documentation du SX1272 (SNR) et de calcul effectué par le LoRa Calculator (Sensibilité).

Emetteur	Récepteur	
Spreading Factor	Sensibilité	SNR minimum
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

Tableau 10 : Sensibilité et SNR acceptable en fonction du Spreading Factor

Le choix du SF et de la puissance d'émission n'est pas simple, et c'est souvent par une méthode empirique que nous déterminons leurs valeurs. Il faut sans cesse vérifier la bonne réception des trames, et s'assurer que les marges sur le SNR et sur le RSSI sont suffisantes pour les trames reçues par les Gateways.

Pour pallier à cette difficulté, une méthode d'ajustement automatique a été mise en place par le protocole LoRaWAN : c'est l'**Adaptive Data Rate (ADR)**. L'idée est de faire en sorte que le Network Server soit lui-même à l'initiative du meilleur choix combiné du SF et de  $P_E$ .

Pour effectuer cette opération, le Network Serveur vérifie :

- Le RSSI (puissance reçue sur le récepteur) : Le RSSI sera comparé à la sensibilité du récepteur. Il faut que le RSSI soit supérieur à la sensibilité pour que la transmission soit valide. Dans le cas de l'ADR, on prendra en plus une marge.
- Le SNR de la transmission : Cette valeur est comparée au SNR minimum du récepteur. Il faut que le SNR de la transmission soit supérieur au SNR mini du récepteur. Dans le cas de l'ADR, on prendra en plus une marge (par exemple TTN choisi une marge de 5 dB).

La vérification n'est pas faite uniquement sur une seule trame mais sur plusieurs en faisant une moyenne. On peut appliquer la démarche suivante :

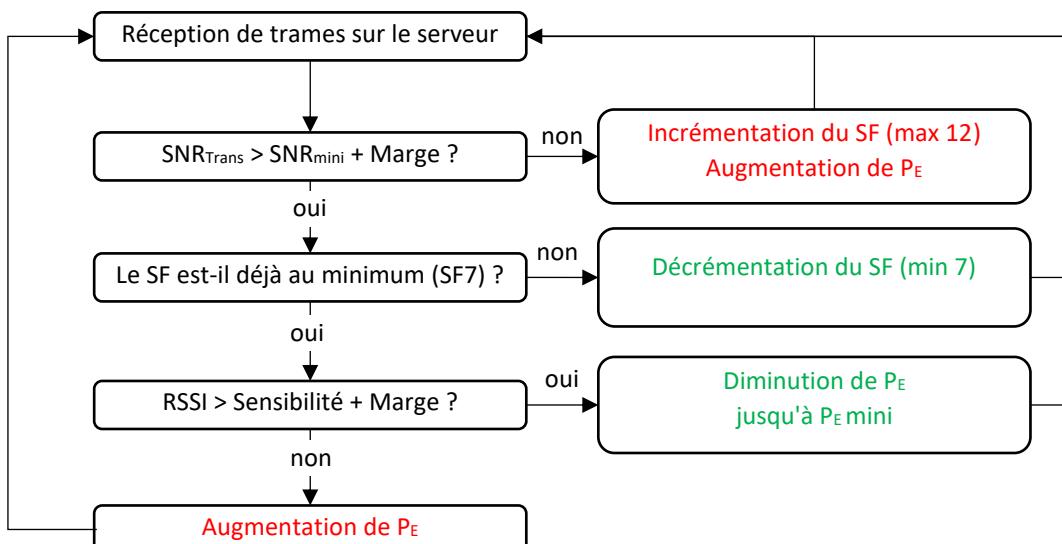


Figure 45 : Ajustement du SF et de PE grâce à l'Adaptive Data Rate



⚠ Dans tous les cas, le Device LoRa doit se configurer en mode ADR pour pouvoir utiliser cette fonctionnalité.

Dans la Figure 45, on parle d'incrémentation / décrémentation du SF, et d'augmentation / diminution de  $P_E$ . Mais cette démarche est effectuée sur le Network Server, il faut donc que le Network Server demande explicitement au Device LoRa d'effectuer ces actions. Pour cela, deux scénarios sont possibles :

1. Le Network Server transmet lui-même les informations au Device LoRa via une commande appellée **LinkADRReq**, en profitant d'une trame Downlink existante.
2. Le Network Server n'a pas pu profiter d'une trame Downlink pour envoyer la commande **LinkADRReq**. Dans la prochaine trame Uplink, le Device LoRa va donc envoyer une option appelée **ADRACKReq** afin de faire une demande explicite au Network Server. Le Server générera une trame **LinkADRReq**.

1. Le Network Server est à l'initiative de l'envoi de la commande **LinkADRReq** :

Le Device LoRa émet des trames en Uplink en direction du serveur. Après un certain nombre de trames reçues et lorsque celui-ci considère que la marge sur le SNR et sa sensibilité est suffisante, il va insérer dans une trame de donnée en Downlink une commande **LinkADRReq** au Device (requête de changement de SF /  $P_E$ ). Dès que le Device la reçoit, il va modifier sa configuration en conséquence comme le montre la Figure 46.

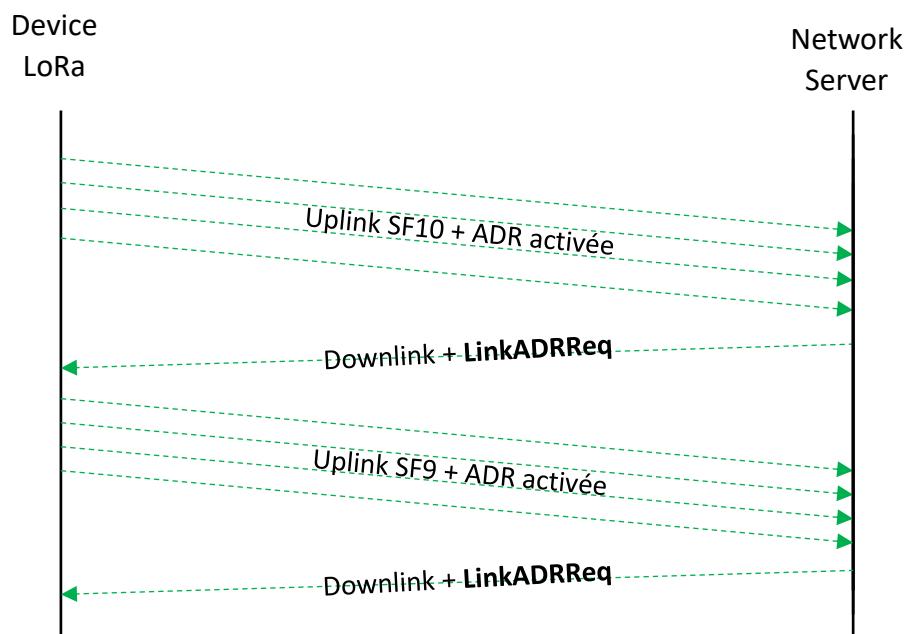


Figure 46 : Optimisation du SF et de  $P_E$  grâce à l'ADR

La trame **linkADRReq**, n'est pas envoyée seule, elle est mutualisée avec une trame de données en Downlink. S'il n'y a pas de trame de données Downlink disponible alors le Network Serveur attendra avant d'envoyer cette commande.

## 2. Le Device LoRa demande au Network Server l'envoi de la commande **linkADRReq**:

Lorsque le Device LoRa n'a pas reçu de trame Downlink, il sait que le Network Server n'a pas eu d'opportunité de lui transmettre sa commande **linkADRReq**. Il va donc explicitement lui faire une demande de **linkADRReq** en utilisant l'option **ADRACKReq** dans sa trame. S'il n'obtient pas de réponse dans un temps imparti, alors la communication avec le serveur est considérée comme perdue et le Device LoRa va incrémenter SF / PE, jusqu'à rétablir la communication avec le serveur comme le montre la Figure 47.

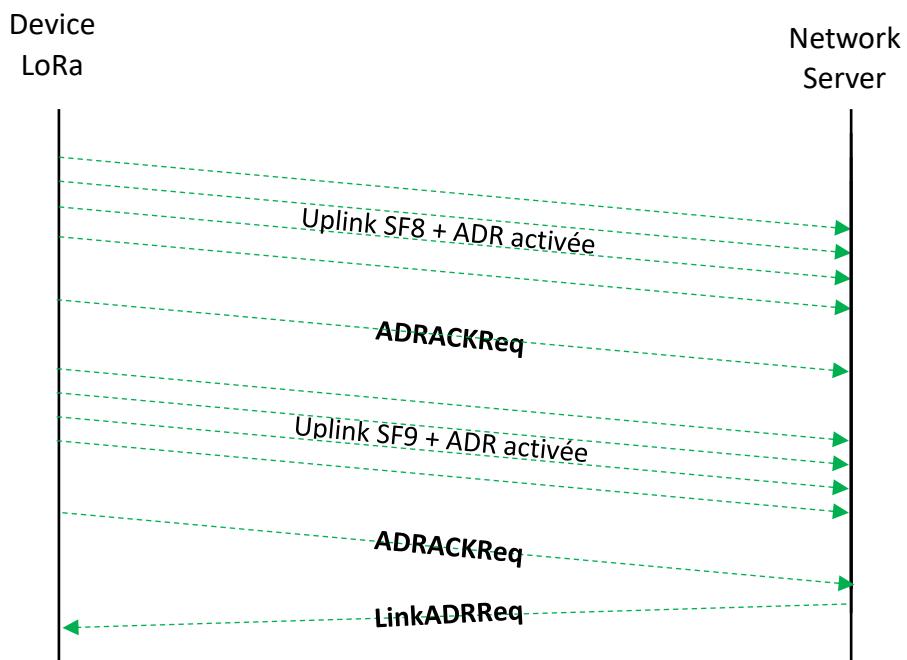


Figure 47 : Requête ADRACKReq du Device pour obtenir un LinkADRReq

### Résumé sur l'ADR :

- L'Adaptive Data Rate est n'est actif que si le Device le demande explicitement.
- On peut choisir le SF et la  $P_E$  par défaut pour les premiers envois de trames.
- En utilisant des trames de données Uplink "Confirmed" (c'est-à-dire en demandant un acquittement au Network Server), on génère un flux Downlink qui facilite l'envoi de la commande **linkADRReq**. Le Device LoRa converge donc plus vite vers la valeur optimale.

## 5 Les réseaux et serveurs LoRaWAN

### 5.1 Les différents types de réseaux

Les réseaux LoRaWAN peuvent être utilisés suivant deux méthodes :

- En utilisant les réseaux LoRaWAN opérés proposés par les opérateurs de télécoms
- En utilisant votre propre réseau LoRaWAN privé
- En utilisant une infrastructure intermédiaire appelée réseau dédié

#### 5.1.1 Les réseaux LoRaWAN opérés

Ce sont des réseaux LoRaWAN proposés par les opérateurs comme par exemple Objenious (filiale de Bouygues) ou encore Orange. A eux deux, ils couvrent la quasi-totalité du territoire. Dans le cas de l'utilisation des réseaux opérés, l'utilisateur a juste besoin de s'occuper des Devices LoRa et de l'application utilisateur. Les Gateways, le Network Server et l'Application Server sont gérés par l'opérateur.

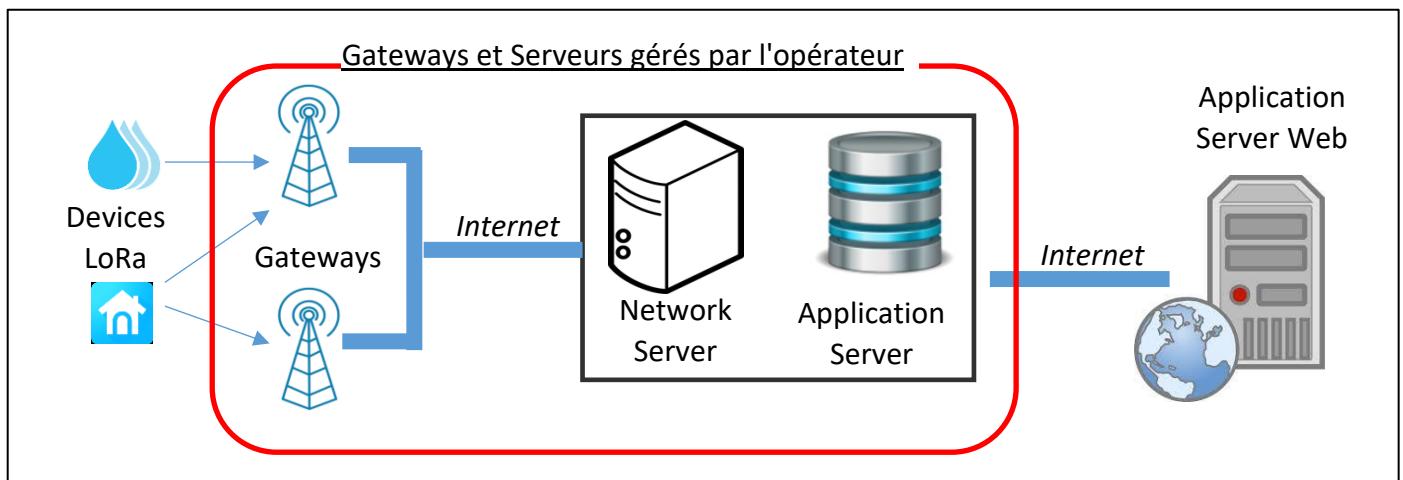


Figure 48 : infrastructure d'un réseau LoRaWAN opéré

A titre d'exemple, voici en 2020 les abonnements proposés par Bouygues et Orange pour avoir accès à leur réseau LoRaWAN :

Orange :

- Illimité en Uplink (dans le respect du Duty Cycle)
- Prix de chaque message Uplink de 5 cts
- Le prix de l'abonnement par Device LoRa est présenté Figure 49

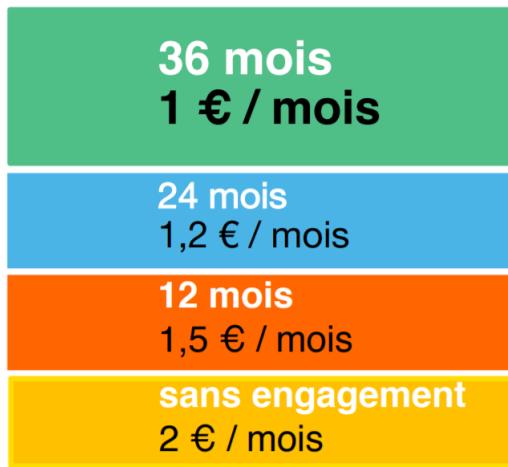


Figure 50 : Tarification d'un abonnement par Device LoRaWAN chez Orange

Bouygues Objenious :

- 144 messages par jour en Uplink
- 6 messages par jour en Downlink
- Le prix de l'abonnement par Device LoRa :

**20€ TTC**  
/CAPTEUR/AN

### 5.1.2 Les réseaux LoRaWAN privés

Chacun est libre de réaliser son propre réseau privé en implémentant son propre réseau de Gateways, ainsi que sa propre infrastructure de serveurs pour communiquer avec ses Devices LoRa. L'entreprise doit prendre en charge la mise en place d'un Network Server et Application serveur privés. Dans certaines Gateways, une implémentation de ces deux serveurs est proposée. Il existe des serveurs (Network et Application) open source, c'est le cas par exemple de Chirp Stack [[www.chirpstack.io](http://www.chirpstack.io)] ou [<https://thethingsstack.io/>]. La présentation de ces stacks permettant d'implémenter ces serveurs LoRaWAN est réalisée au chapitre 9.

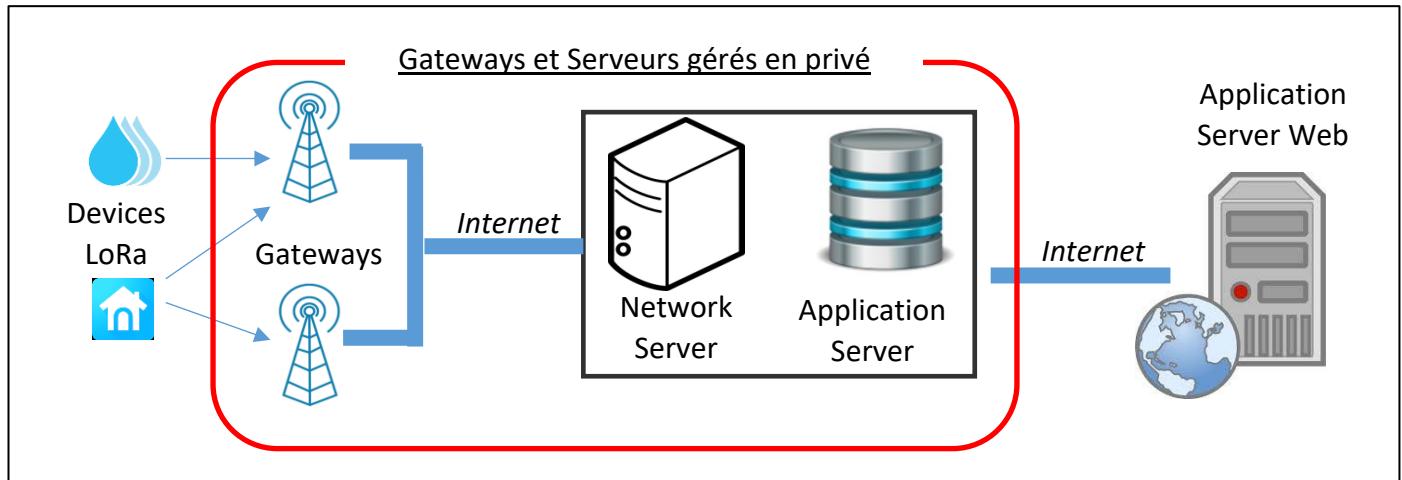


Figure 51 : Infrastructure d'un réseau LoRaWAN privé



### 5.1.3 Choix du réseau : opéré ou privé ?

Nous pouvons résumer les avantages et inconvénients de chacun de ces types de réseau dans le Tableau 11.

	Réseau privé	Réseau opéré
<b>Coût d'abonnement</b>	Gratuit	Environ 1,5 € par Device LoRa
<b>Coût de l'infrastructure</b>	Investissement important au début (Gateways et Serveurs)	Compris dans l'abonnement
<b>Compétences requises</b>	Demande des compétences en interne à l'entreprise pour la mise en place, et pour la maintenance.	Tout est géré par l'opérateur
<b>Couverture</b>	Optimisée suivant les besoins	Dépendante de l'opérateur choisi. Possibilité de Roaming avec l'international.
<b>Flux Uplink</b>	Illimité dans le respect du Duty-Cycle	Limité suivant l'abonnement
<b>Flux Downlink</b>	Illimité dans le respect du Duty-Cycle	Limité en nombre ou payant à l'unité

Tableau 12 : Choix entre un réseau privée et un réseau opéré

#### 5.1.4 Une alternative, le réseau LoRaWAN dédié (hybride)

Dans le cas où aucune des solutions extrêmes (réseau opéré ou réseau privé) ne convient, il est possible d'avoir une solution intermédiaire appelée réseau dédié. Elle a l'avantage de gérer la couverture réseau LoRa en utilisant nos propres Gateways, mais de confier l'infrastructure des serveurs LoRaWAN à un prestataire afin de limiter les investissements et la maintenance.

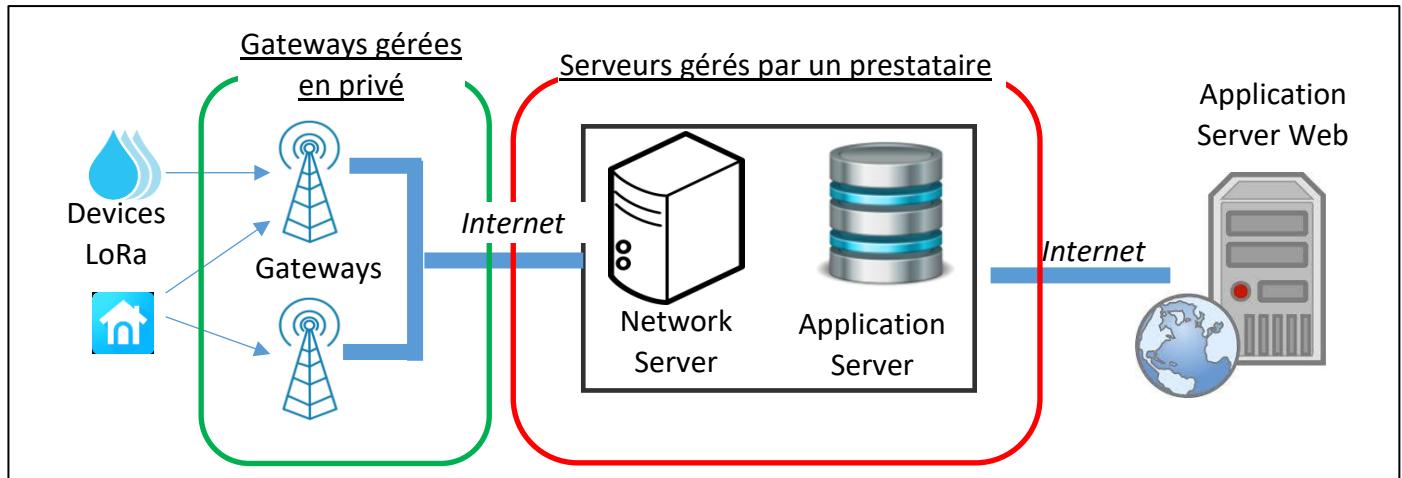


Figure 52 infrastructure d'un réseau LoRaWAN dédié

Network Server et Application Server en ligne : De très nombreux Network Server et d'Application Server sont proposés. Ce sont des services payants ou avec des contreparties :

- Loriot [ [www.loriot.io](http://www.loriot.io) ]
- ResIoT [ [www.resiot.io](http://www.resiot.io) ]
- The Things Network [ [www.thethingsnetwork.org](http://www.thethingsnetwork.org) ]

Dans le cadre de ce cours, la plupart du temps, nous sommes dans le cas d'un réseau dédié. En effet, nous utilisons nos propres Gateways associées à des serveurs LoRaWAN fournis par TTN.

#### 5.1.5 Les zones de couvertures

Pour connaître la zone de couverture de notre Gateway utilisée avec TTN, nous pouvons utiliser l'application TTN Mapper : <https://ttnmapper.org/>. L'idée est de promener son Device LoRa associé à un GPS dans la zone de couverture des Gateways qui nous entourent. A chaque trame reçue par le Serveur, on note les coordonnées GPS du Device LoRa ainsi que la Gateway qui a transmis le message. Toutes ces informations sont alors retracées sur une carte.

Les zones de couvertures des réseaux opérés sont mises à jour sur le site web des opérateurs.

## 5.2 Exemple de serveur : The Things Network (TTN)

### 5.2.1 Présentation de TTN

Pour la suite, nous utiliserons le Network Server et l'Application Server fourni par TTN : [www.thethingsnetwork.org](http://www.thethingsnetwork.org)

TTN est gratuit et Open Source. En revanche, le fait d'utiliser TTN impose à ceux qui enregistrent des Gateways de les mettre à disposition à tous les autres utilisateurs de TTN. L'objectif est de réaliser un réseau global ouvert.

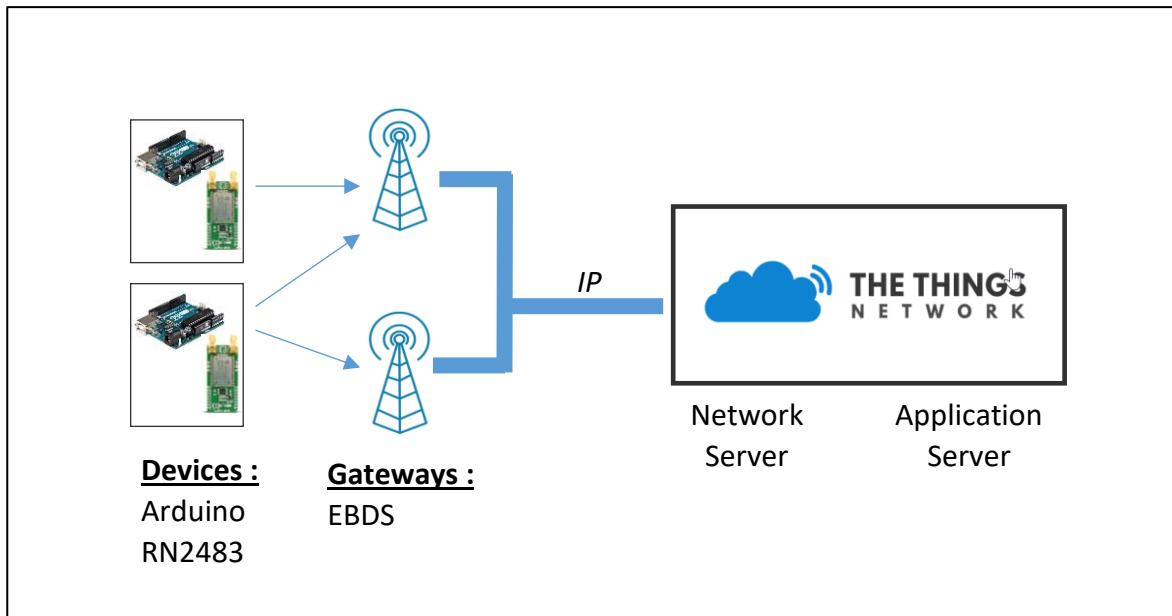


Figure 53 : Schéma de notre application

### 5.2.2 Configuration de la Gateway

A l'université, nous avons deux Gateways LoRa (Marque EBDS). Elles sont toutes les deux situées sur le Campus de Technolac. Une Gateway est à l'intérieur du bâtiment ISERAN et sert essentiellement pour les Travaux Pratiques et formations. L'autre est sur le toit du bâtiment Chablais et est mise à disposition du public via le service de TTN.

Les deux Gateway écoutent sur tous les canaux et sur tous les Spreading Factor : SF7 > SF12.

- L'@ IP de la Gateway LoRa à l'université (bâtiment ISERAN) est 192.168.140.196
- L'@ IP de la Gateway LoRa à l'université (bâtiment CHABLAISS) est 192.168.138.151
- Serveur DNS Univ SMB : 193.48.120.32 / 193.48.129.137
- Network Server (TTN) : 13.76.168.68 / router.eu.thethings.network
- Port Upstream : 1700 / Port Downstream : 1700

### 5.2.3 Enregistrement des Gateways, Applications et Devices

La configuration de TTN se fait en ligne. La première opération à réaliser lorsqu'une nouvelle Gateway est déployée, c'est de l'enregistrer dans TTN.

- Enregistrement d'une Gateway LoRa : **TTN > Console > Gateway > register gateway**

GATEWAYS		<a href="#">register gateway</a>
eui-b827ebfffeae26f5	Gateway EBDS 1	connected EU_863_870
eui-b827ebfffeae26f6	Gateway EBDS 2	connected EU_863_870

Figure 54 : Enregistrement d'une Gateway LoRa sur TTN

- Enregistrement d'une application : **TTN > Console > Application > add application**

APPLICATIONS		<a href="#">add application</a>
test_app_lorawan_sylvainmontagny	test	ttn-handler-eu 70 B3 D5 7E D0 00 A7 97

Figure 55 : Enregistrement d'une application sur TTN

- Enregistrement des Devices LoRa dans l'application : **Nom de l'application > register device**

DEVICES		<a href="#">register device</a>
onebee_rpihat	00 DC ED 8A 32 55 89 08	●
stm32lorawan_1	00 56 AB 22 5B F4 98 77	●
stm32lorawan_10	00 E1 A8 EC 00 56 AF 3F	●

Figure 56 : Enregistrement des Devices LoRa dans une application

#### 5.2.4 Configuration des Devices LoRa

Lorsque nous avons enregistré des Devices LoRa dans TTN, nous devons choisir entre les deux modes d'authentification. Dans un premier temps nous utiliserons le mode le plus simple qui est **ABP**. Nous générerons dans TTN les **DevAddr**, **NetwkSKey** et **AppSKey**.

**Activation Method**

OTAA   **ABP**

**Device Address**

26 01 4C 8E 4 bytes

**Network Session Key**

FB EC 51 50 27 00 02 01 45 3B 07 04 75 67 02 16 bytes

**App Session Key**

63 A9 1D 80 C7 A5 F5 8C E0 FA EC C6 E7 88 A5 BE 16 bytes

Figure 57 : Configuration des Devices LoRa en ABP dans TTN

### 5.2.5 Simulation d'Uplink dans l'Application Server

Dans beaucoup de situations, nous souhaitons valider le fonctionnement de l'Application Server de TTN ou/et de notre propre Application. Il est donc très utile de pouvoir simuler l'envoi d'une trame LoRa sur l'Application Server, sans que celle-ci soit réellement émise par un Device LoRa. Pour cela, un outil existe dans les Settings de chaque Device enregistré dans TTN. Nous avons juste besoin de spécifier le **Frame Payload** en clair en hexadécimal et le numéro de Port.

**SIMULATE UPLINK**

FPort	Payload
1	<span style="float: right;">0 bytes</span>

Figure 58 : Simulation d'une trame envoyée par un Device LoRa

### 5.2.6 Downlink : De l'Application Server au Device LoRa

La communication LoRa est bidirectionnelle. Des données peuvent être transmises au Device LoRa depuis l'Application Server. Depuis TTN, un outil existe dans les Settings de chaque Device enregistré. L'interface est représentée ci-dessous.



Figure 59 : Transmission de données de l' « Application Server » jusqu'au Device LoRa

- **Replace** scheduling Mode : Par défaut le **Frame Payload** transmis remplacera le Frame Payload en attente (si il existe). Dans ce mode, un seul Frame Payload est planifié.
- **First** scheduling Mode : Le Frame Payload est mis en file d'attente, en première position.
- **Last** scheduling Mode : Le Payload est mis en file d'attente, en dernière position.

Une demande de confirmation peut être demandée au Device LoRa pour qu'il précise s'il a bien reçu les données. L'option « confirmed » permet donc de spécifier le type de trame dans le MAC Header (voir paragraphe 6.1.2) : « Confirmed Data Down » ou « Unconfirmed Data Down » dans notre cas.

### 5.3 Mise en application

Nous avons dans notre application 10 Devices LoRa d'enregistrés : aduino0 à aduino9. Ils sont tous configurés en ABP.

- ➔ Installer la librairie TheThingsNetwork : **Arduino IDE > Outils > Gérer les bibliothèques**. Rechercher la bibliothèque **TheThingsNetwork** et l'installer.
- ➔ Dans le dossier de la librairie, ouvrir le code d'exemple : **/example/Workshop**. Enregistrer sous votre projet dans vos documents.
- ➔ Modifier votre configuration (**DevAddr**, **NetwkSKey** et **AppSKey**) en fonction de votre numéro de binôme (voir document "informations seminaire.txt" dans votre répertoire de travail).
- ➔ Vérifier que votre Device LoRa arrive bien à transmettre des informations (Uplink) vers la Gateway, puis vers le Network Serveur de TTN (The Things Network).
- ➔ Regarder la documentation dans **/docs/thethingsNetwork.md** afin de modifier le spreading factor (method 'sendBytes').

## 6 La Trame LoRa / LoRaWAN

### 6.1 Les couches du protocole LoRaWAN

Lorsque nous avons fait une communication en Point à Point, nous avons simplement utilisé la couche physique de la modulation LoRa. Lorsque nous souhaitons utiliser le protocole LoRaWAN, des couches protocolaires supplémentaires se rajoutent.

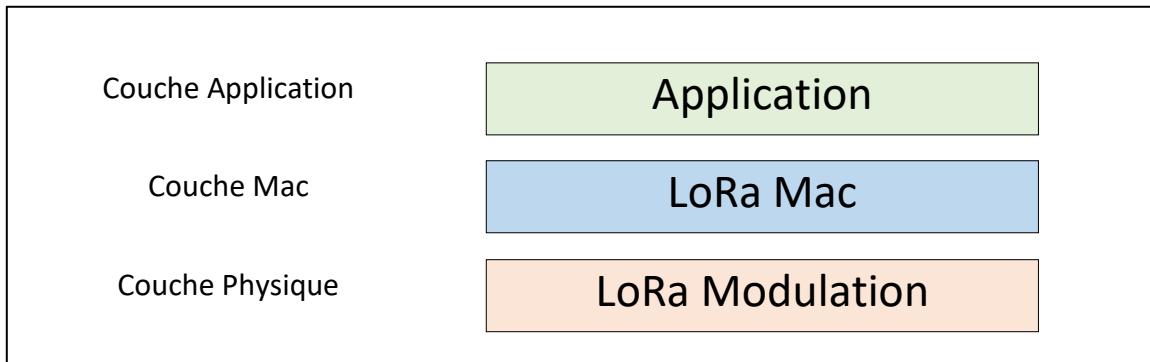


Figure 60 : Les couches protocolaires du LoRaWAN

Chaque couche rajoute un service. Lors de l'envoi de la trame, les données utilisateurs sont donc encapsulées dans chaque couche inférieure jusqu'à la transmission. Le détail de l'ensemble de la trame LoRaWAN par couche est décrit sur la figure Figure 61 :

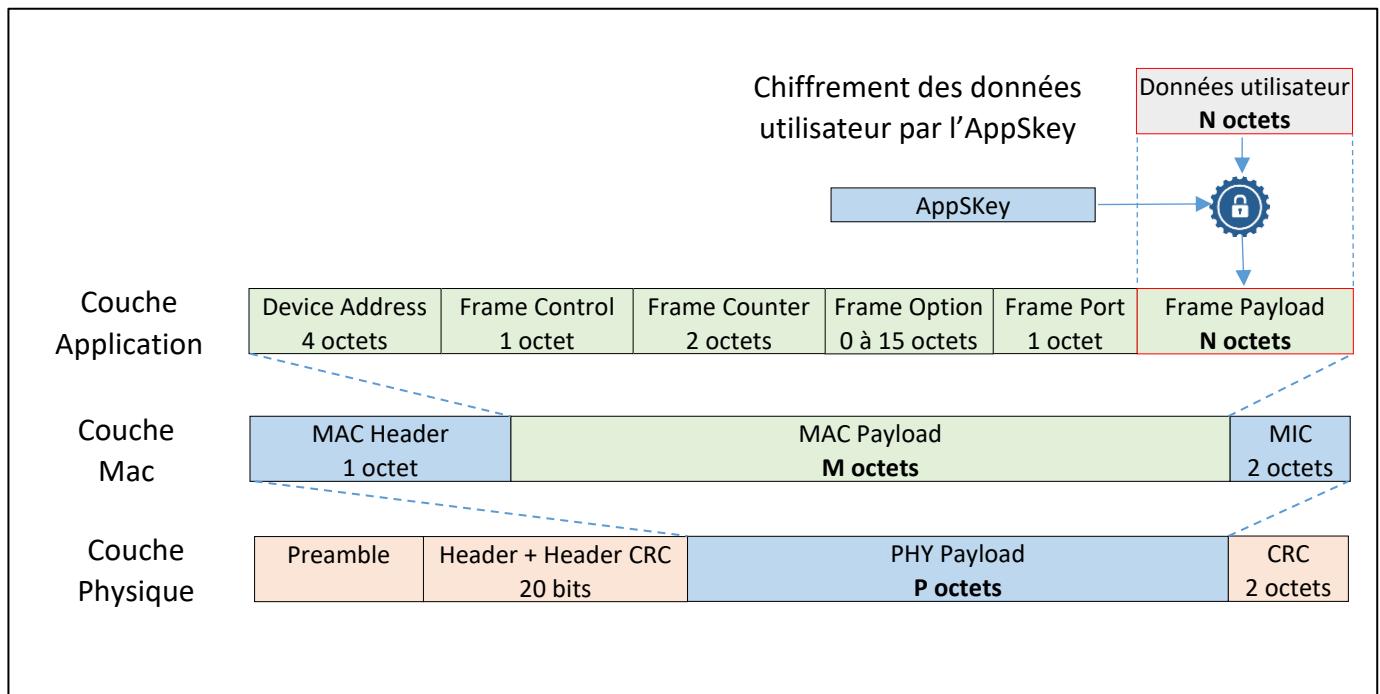


Figure 61 : Trame LORAWAN complète par couche protocolaire

### 6.1.1 Détail de la couche Application

La couche Application accueille les données de l'utilisateur. Avant de les encapsuler, elles sont chiffrées avec l'AppSKey afin de sécuriser la transaction.

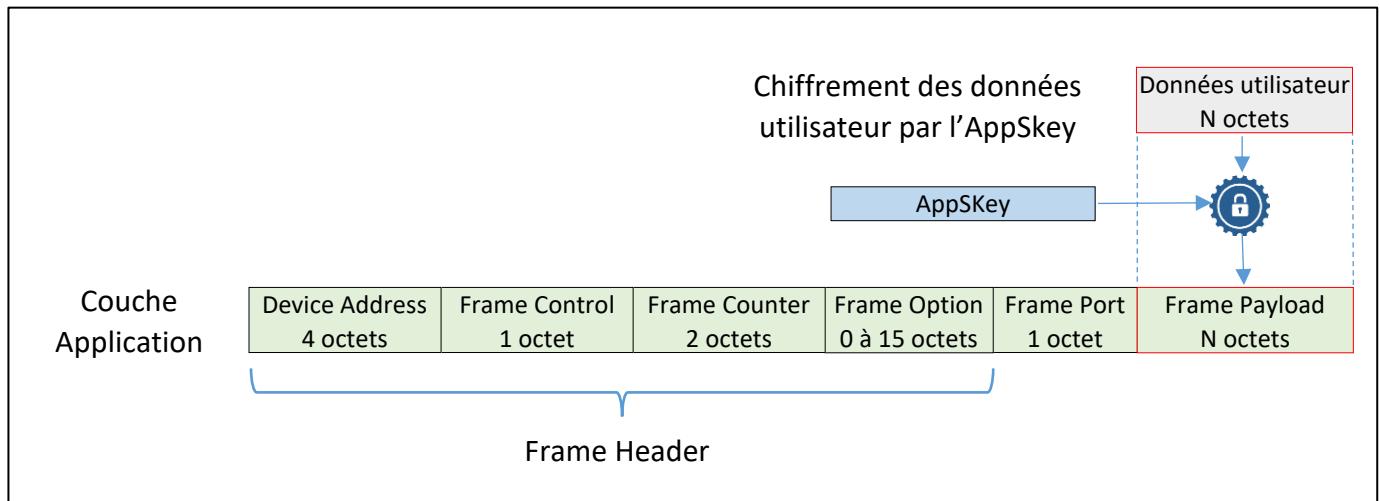


Figure 62 : Trame LORA couche Applicative

Un ensemble de champs nommé **Frame Header** permet de spécifier le **DevAddr**, le **Frame Control**, le **Frame Counter**, et le **Frame Option**.

Le **Frame Port** dépend du type d'application et sera choisi par l'utilisateur.

Le **Frame Payload** contient les données chiffrées à transmettre. Le nombre d'octets maximum pouvant être transmis (N octets) est donné dans le tableau suivant :

Data Rate	Spreading Factor	Bandwidth	Max Frame Payload (Nombre N)
DR 0	SF12	125 KHz	51 octets
DR 1	SF11	125 KHz	51 octets
DR 2	SF10	125 KHz	51 octets
DR 3	SF9	125 KHz	115 octets
DR 4	SF8	125 KHz	222 octets
DR 5	SF7	125 KHz	222 octets
DR 6	SF7	250 KHz	222 octets

Tableau 13 : Taille maximum du Frame Payload en fonction du Data Rate

### 6.1.2 Détail de la couche LoRa MAC

Cette trame est destinée au Network Server. Elle est authentifiée grâce au champ MIC (Message Integrity Protocol) selon la méthode expliquée au paragraphe 4.2.6.

Le protocole LoRa MAC est composé de :

1. MAC Header : Version de protocole et type de message :

Type de Message	Description
000	Join-Request
001	Join-Accept
010	Unconfirmed Data Up
011	Unconfirmed Data Down
100	Confirmed Data Up
101	Confirmed Data Down
110	Rejoin-request
111	Proprietary

Tableau 14 : Les types de messages transmis en LoRaWAN

2. MAC Payload : Contient tout le protocole applicatif.
3. MIC : Message Integrity Code, pour l'authentification de la trame.

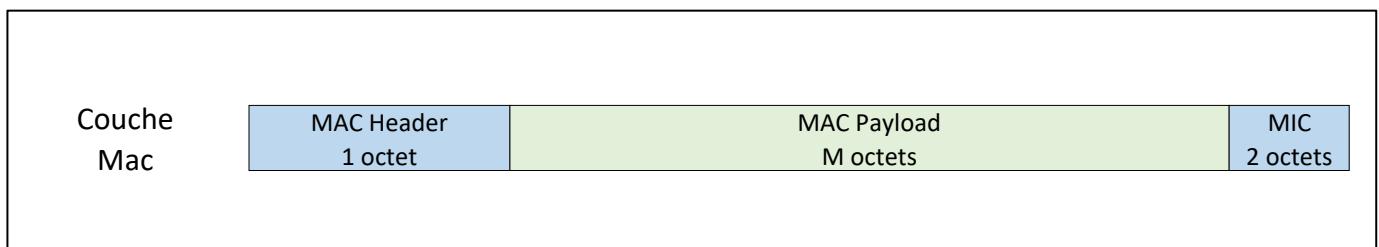


Figure 63 : Trame LORA couche LORA MAC

### 6.1.3 Détail de la couche physique : Modulation LoRa

Le choix du moment d'émission des Devices LoRa se fait de façon simple : Lorsqu'un équipement doit émettre, il le fait sans contrôle et ne cherche pas à savoir si le canal est libre. Si le paquet a été perdu, il le retransmettra simplement au bout d'un temps aléatoire. La couche physique est représentée par l'émission de la trame suivante :

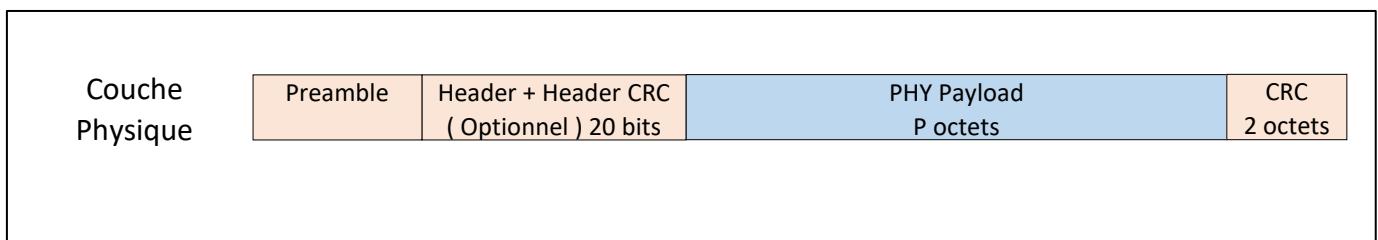


Figure 64 : Trame LORA couche Physique

Le Préambule est représenté par 8 symboles + 4.25. Le temps du Préambule est donc de  $12.25 T_{\text{symbole}}$  (voir chapitre 3.1 pour rappel de la définition d'un symbole).

L'en-tête (Header optionnel) est seulement présent dans le mode de transmission par défaut (explicite), il est transmis avec un Coding Rate de 4/8. Il indique la taille des données, le Coding Rate pour le reste de la trame et il précise également si un CRC sera présent en fin de trame.

Le PHY Payload contient toutes les informations de la Couche LoRa MAC.

Le CRC sert à la détection d'erreur de la trame LoRa.

## 6.2 La Gateway : du LoRa à la trame IP

La Gateway reçoit d'un côté un message radio modulé en LoRa et transmet de l'autre côté une trame IP à destination du Network Server.

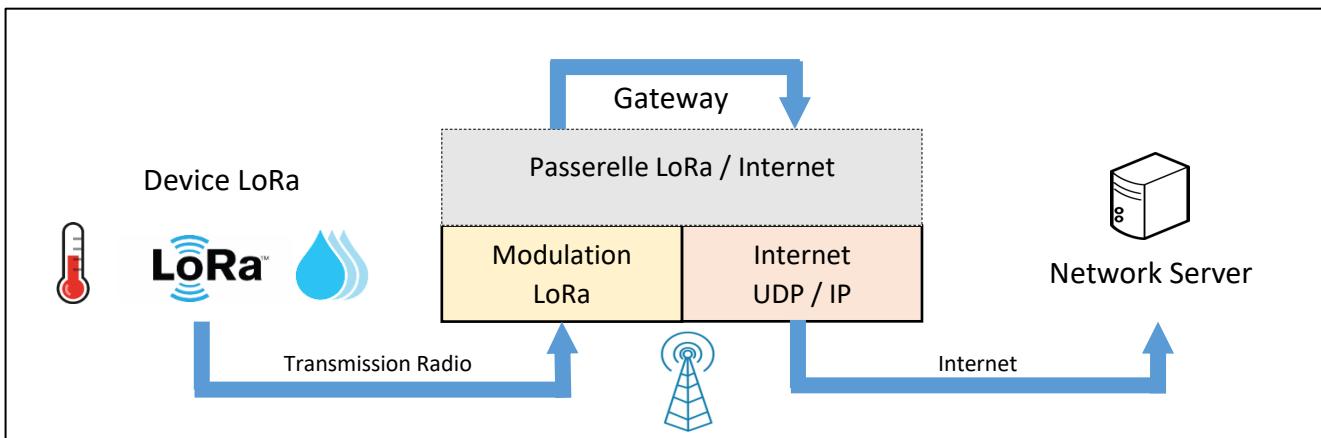


Figure 65 : Rôle de la Gateway LoRa

Coté interface Radio : La Gateway réceptionne une trame LoRaWAN et extrait le PHY Payload. Elle va coder ce PHY Payload en format ASCII en base 64 (voir le paragraphe 6.3.2). La Gateway extrait aussi toutes les informations utiles sur les caractéristiques de la réception qui a eu lieu : SF, Bandwidth, RSSI, Time On Air...etc...

Coté interface réseau IP : La Gateway transmet l'ensemble de ces informations dans un paquet IP (UDP) au Network Server. Les données transmises sont du texte en format JSON (voir paragraphe 6.3). La Gateway a donc bien un rôle de passerelle entre le protocole **LoRa** d'un côté et un réseau **IP** de l'autre.

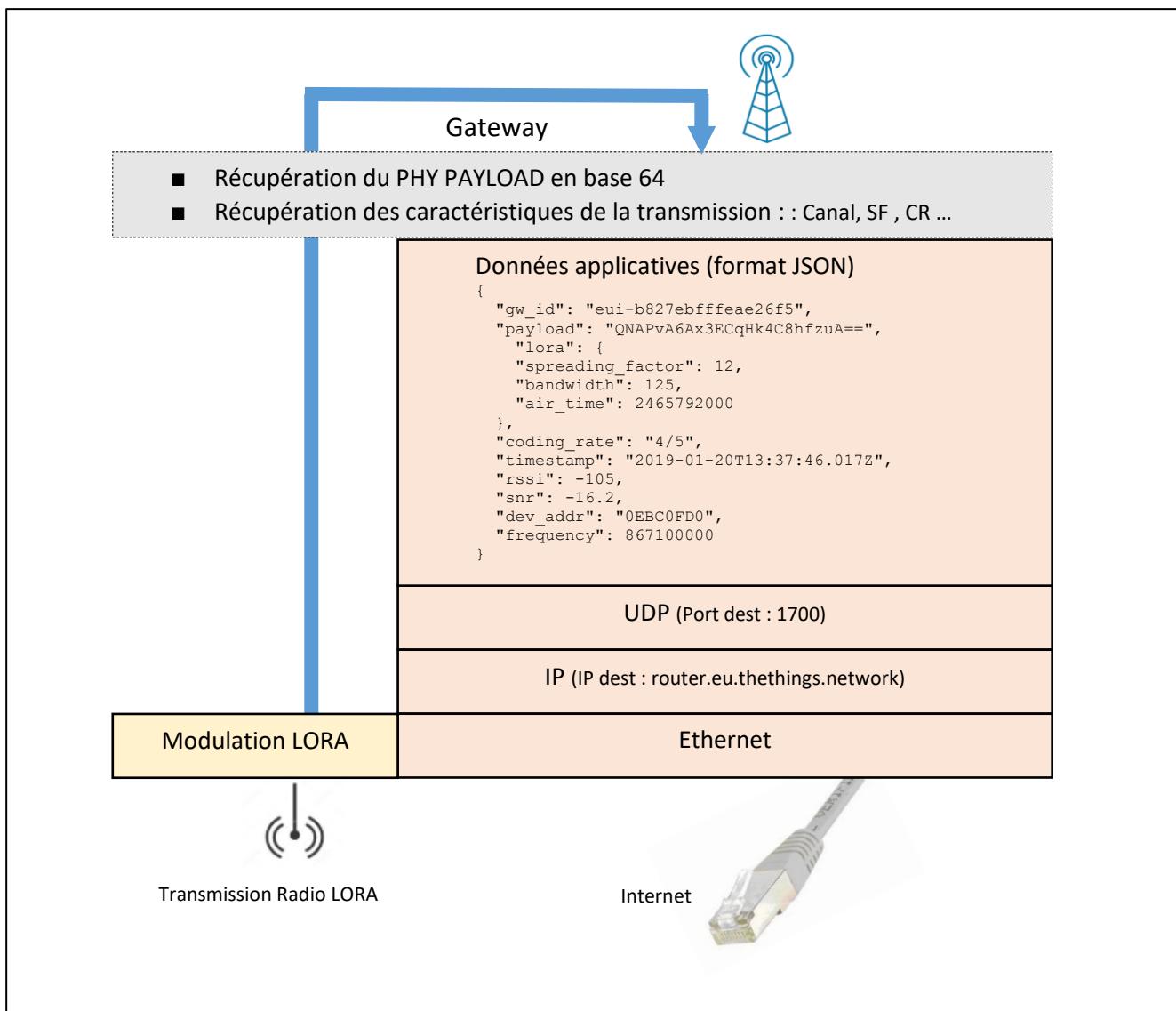


Figure 66 : Passerelle (Gateway) LORAWAN

## 6.3 Analyse des trames IP

### 6.3.1 Le format JSON

Les données applicatives sont formatées en JSON. Le format JSON est un format texte composé d'une succession de couple nom/valeur. Dans l'exemple de la figure précédente, "gw\_id" est un nom et "eui-b827ebfffeae26f5" est la valeur associée. Dans cette exemple, la valeur est un string. Les objets sont délimités par un couple d'accolades { et }.

Une valeur peut être :

- Un string      exemple :      "coding\_rate" : "4/5"
- Un nombre    exemple :      "spreading\_factor" : 12
- Un objet      exemple :      "lora": { "spreading\_factor": 12, "air\_time": 2465792000 }
- Un Boolean    exemple :      "service" : true

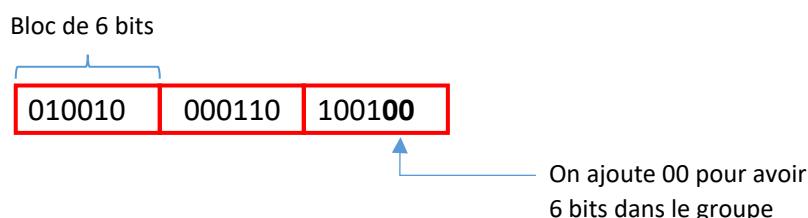
### 6.3.2 Utilisation de la base 64

Dans les données applicatives, un champ important est à notre disposition : Payload. Le Payload correspond aux données transmises dans la trame LoRa. Elles sont brutes et non déchiffrées. Notre Gateway nous présente le Payload PHY en base64. L'explication de la méthode de représentation en base 64 est fourni au travers d'un exemple : Le code hexadécimal 0x4869 représente nos données binaires que nous souhaitons transmettre en base 64.

1. On écrit les données à transmettre en binaire

$$0x4869 = 0100\ 1000\ 0110\ 1001$$

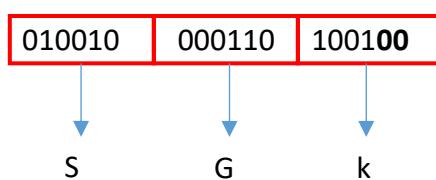
2. On regroupe les éléments binaires par des blocs de 6 bits. Le nombre de bloc de 6 bits doit être un multiple de 4 (minimum 4 blocs). S'il manque des bits pour constituer un groupe de 6 bits, on rajoute des zéros.



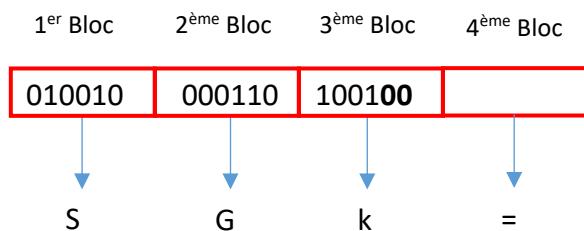
3. S'il manque des blocs pour faire un minimum de 4 blocs, des caractères spéciaux seront ajoutés.
4. Chaque groupe de 6 bits est traduit par le tableau suivant. (Source Wikipédia)

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

Figure 67 : Codage en base 64



5. Si un bloc de 6 bits manque (ils doivent être un multiple de 4), on rajoute un ou plusieurs compléments (caractère « = » )



Résultat : Le codage de 0x4869 en base 64 est « SGk= »

### 6.3.3 Intérêt et inconvénient de la base 64

L'utilisation de la base 64 est un choix qui a été fait pour le protocole LoRa / LoRaWAN afin de rendre les données binaires lisibles. Le problème du code ASCII c'est qu'il est composé d'un certain nombre de caractères non imprimables (EOF, CR, LF,...). Pour éviter ces soucis, la base 64 ne comporte que 64 caractères imprimables (voir tableau ci-dessus).

La restriction à 64 caractères a cependant un inconvénient, c'est que nous ne pouvons coder que 6 bits ( $2^6=64$ ) au lieu de 8. La base 64 est donc moins efficace d'une façon générale.



☞ Nous cherchons à coder le code ASCII « AA » en base 64. Retrouver la démarche en montrant que le résultat en base 64 est « QUE= ».

### 6.3.4 Uplink : Du Device LoRa au Network Server

Le Network Server de TTN reçoit des trames IP en provenance de la Gateway. Comme nous l'avons vu précédemment, un certain nombre d'informations peuvent être retrouvées avec cette trame (DevAddr, SF, Bandwidth, etc...) mais les données Applicatives sont bien sûr chiffrées (avec l'**AppkSKey**). A ce niveau de réception (sans connaître l'**AppSKey**), il n'est donc pas possible de comprendre la totalité du message reçu.

Imaginons que la trame IP reçue par le Network Server de TTN est la suivante :

```
{
  "gw_id": "eui-b827ebffffae26f6",
  "payload": "QNMaASYABwAPlobuUHQ=",
  "f_cnt": 7,
  "lora": {
    "spreading_factor": 7,
    "bandwidth": 125,
    "air_time": 46336000
  },
}
```

```
        "coding_rate": "4/5",
        "timestamp": "2019-03-05T14:00:42.448Z",
        "rss": -82,
        "snr": 9,
        "dev_addr": "26011AD3",
        "frequency": 867300000
    }
```

Le Network Server affiche donc les informations de la façon suivante :

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 1AD3 payload size: 14 bytes

Figure 68 : Trame récupérée sur le « Network Server » de TTN

Nous retrouvons bien les valeurs fournies par la Gateway :

- timestamp (à 1 heure près en fonction du fuseau horaire),
  - frequency : 867,3 Mhz
  - modulation : Lora
  - Coding Rate : 4/5
  - data Rate : SF 7 / 125 kHz (DR5)
  - air time : 46,3 ms

D'autres informations proviennent de l'analyse du Payload. Le Payload inscrit ici est le **PHY Payload**. Il y a donc une partie chiffrée (**Frame Payload**), mais les entêtes sont en claires (voir paragraphe 6.1). Ce PHY Payload est « QNMaASYABwAP1obuUHQ= ». Lorsque celui-ci est exprimé en hexadécimal au lieu de la base 64, il vaut : « 40D31A01260007000FD686EE5074 », comme le montre le Network Server de TTN.

## Physical Payload

Figure 69 : PHY Payload présenté dans notre Network Server

Sa taille est bien de 14 octets (en hexadécimal) comme précisé sur la Figure 68

Nous reprenons le format de la trame LoRaWAN vu à la Figure 61. Nous pouvons alors retrouver tous les champs de toute la trame :

```

PHYPayload = 40D31A01260007000FD686EE5074

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

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

Frame Header = DevAddr[4] | FCtrl[1] | FCnt[2] | FOpts[0..15]

```

DevAddr	=	<b>26011AD3</b> (Big Endian)
FCtrl (Frame Control)	=	<b>00</b> (No ACK, No ADR)
FCnt (Frame Counter)	=	<b>0007</b> (Big Endian)
FOpts (Frame Option)	=	

- ➔ Vous pouvez vérifier l'ensemble de ces informations grâce au décodeur de trame LoRaWAN (LoRaWAN packet decoder) : <https://bit.ly/2szdXtv>

## LoRaWAN 1.0.x packet decoder

A frontend towards [lora-packet](#).

Base64 or hex-encoded packet

Decode

Secret NwkSKey (hex-encoded; optional)

Secret AppSKey (hex-encoded; optional)

Figure 70 : LoRaWAN packet decoder

### 6.3.5 Uplink : Du Network Server à l'Application Server

Nous reprenons l'exemple de la trame ci-dessus (Figure 69). Pour information, les clés NwkSKey et AppSKey suivantes ont été utilisée :

- NwkSKey : E3D90AFBC36AD479552EFEA2CDA937B9
- AppSKey : F0BC25E9E554B9646F208E1A8E3C7B24

Le Network Server a décodé l'ensemble de la trame. Si le MIC est correct (authentification de la trame par le **NwkSKey**) alors le Network Server va passer le contenu du message chiffré (Frame Payload) à l'Application Server. Dans notre cas le Frame Payload est (d'après le décodage effectué au chapitre précédent) :

FramePayload	=	D6
--------------	---	----

**D6** est le contenu chiffré, lorsqu'il est déchiffré avec **l'AppSkey** on trouve **01**. Vous pouvez vérifier l'ensemble de ces informations grâce au décodeur de trame LoRaWAN : <https://bit.ly/2szdXtv>

A noter que l'Application Server recevra les données chiffrées seulement si le Device LoRa a bien été enregistré. On peut aussi vérifier ci-dessous que l'Application Serveur de TTN nous retourne bien un payload (Frame Payload) de **01**.

time	counter	port	
▲ 15:00:42	7	15	payload: 01

Figure 71 : Trame récupérée sur l'Application Server de TTN

```
{
  "time": "2019-03-05T14:00:42.379279991Z",
  "frequency": 867.3,
  "modulation": "LORA",
  "data_rate": "SF7BW125",
  "coding_rate": "4/5",
  "gateways": [
```

```
{  
    "gtw_id": "eui-b827ebfffeae26f6",  
    "timestamp": 2447508531,  
    "time": "",  
    "channel": 4,  
    "rssи": -82,  
    "snr": 9,  
    "latitude": 45.63647,  
    "longitude": 5.8721523,  
    "location_source": "registry"  
}  
]
```

## 7 La récupération des données

### 7.1 Les services rendus par notre Application

Nous avons vu jusqu'à maintenant comment envoyer des données depuis un Device LoRa à destination de l'Application Server, en passant par les Gateway et le Network Server. Ces données ne sont pas directement disponibles pour un utilisateur. Elles doivent être récupérées, présentées sous différentes formes (tableaux, graphiques...), enregistrées dans des BDD (Bases De Données) et enfin mises à disposition par l'intermédiaire d'un service Web que l'utilisateur pourra interroger.

Toute cette partie est totalement indépendante du protocole LoRa et LoRaWAN que nous avons étudié jusqu'ici, et n'a donc aucun lien avec celle-ci. Les explications qui viendront peuvent donc être aisément transposées à tout autre protocole lié à l'Internet des Objets. On a donc d'un côté la communication entre le Device LoRa et les serveurs LoRaWAN (par l'intermédiaire des Gateways). Et de l'autre côté, on a la communication entre les Serveur LoRaWAN et notre Application. Notre Application fera le lien avec l'utilisateur. Elle devra donc réaliser les actions suivantes :

- Recevoir des données en provenance des Serveurs LoRaWAN
- Transmettre des données à destination des Devices LoRaWAN
- Faire la gestion du serveur LoRaWAN : par exemple Ajouter / Supprimer une Application, Ajouter / Supprimer un Device LoRa, etc ...

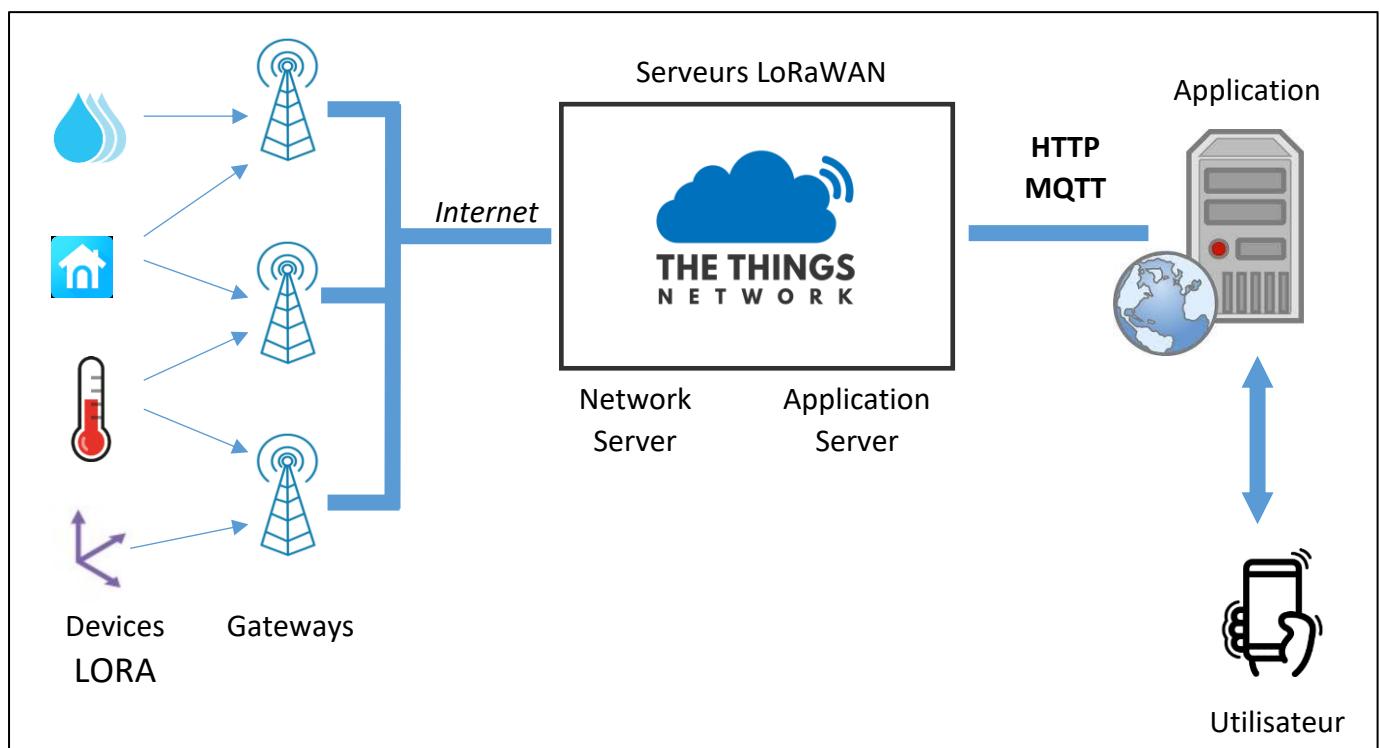


Figure 72 : Structure globale d'un réseau LORAWAN

Attention, les termes utilisés ici sont très proches malgré le fait qu'ils désignent des entités complètement différentes :

- On parle d'**Application Server** lorsque que nous parlons du serveur Application LoRaWAN. Ce terme est défini dans la spécification du protocole LoRaWAN.
- On parle d'**Application** lorsqu'on définit le serveur coté utilisateur. Ce serveur n'a aucun lien avec le protocole LoRaWAN.

Le dialogue entre les serveur LoRaWAN et l'Application peut se faire à l'aide de différents protocoles que nous étudierons dans les prochains chapitres.

Nous utiliserons à nouveau le Network Server et Application Server de "The Things Network". Dans le sens Uplink, notre Application utilisateur aura pour rôle :

- **La récupération des données**
- Le stockage et traitement
- La mise en forme (graphiques, tableaux...)
- L'envoi à l'utilisateur

Dans le sens Downlink, notre Application utilisateur devra :

- Présenter une interface utilisateur (Bouton, champ texte, ...)
- Traiter les commandes de l'utilisateur
- **Envoyer ces commandes aux Serveur LoRaWAN**

Dans ce chapitre, nous traiterons seulement les deux éléments **en gras** de la liste précédente. C'est-à-dire : "**Gérer la récupération des données (Uplink)**" et "**Envoyer les commandes de l'utilisateur aux Serveur LoRaWAN (Downlink)**".

Nous allons voir plusieurs méthodes pour communiquer entre notre Application et TTN. Ces deux méthodes sont le protocole **HTTP** et le protocole **MQTT** comme le montre la Figure 73.

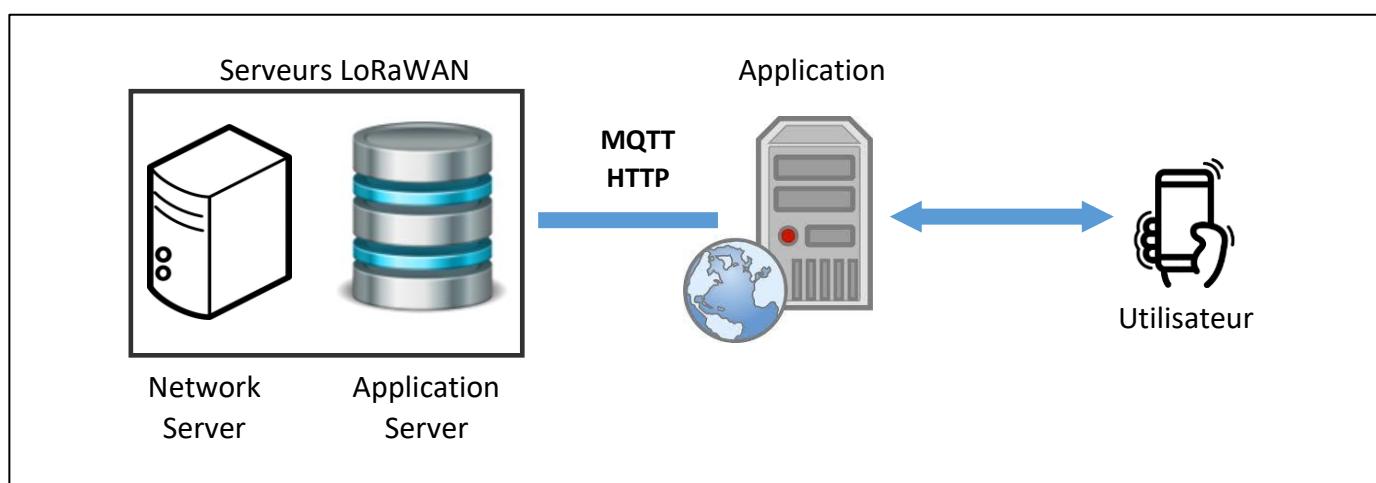


Figure 73 : Application Server (LoRaWAN) et Application

## 7.2 Récupération des données avec le protocole HTTP (GET)

### 7.2.1 Présentation du principe Client - Serveur

Comme la majorité des protocoles, HTTP fait appel à un transfert d'information entre un client et un serveur. Le client et le serveur sont deux entités éloignées qui souhaitent dialoguer entre elles. Le client fait des requêtes et le serveur lui répond. Cette notion de Client - Serveur est assez analogique à celle que vous avez dans un restaurant. Le client interpelle le serveur pour préciser son besoin, il émet donc une requête. Le serveur traite cette requête en apportant le contenu. Le client et le serveur peuvent être de tout type : mail, web, fichiers ...

La Figure 74 représente un client, un serveur et les deux types de trames qui circulent entre eux : les requêtes et les réponses. Il est très important de savoir qui va jouer le rôle du client et qui va

jouer le rôle du serveur. En effet, il faudra affecter un rôle (client ou serveur) au serveur LoRaWAN (TTN dans notre cas) et à notre Application.

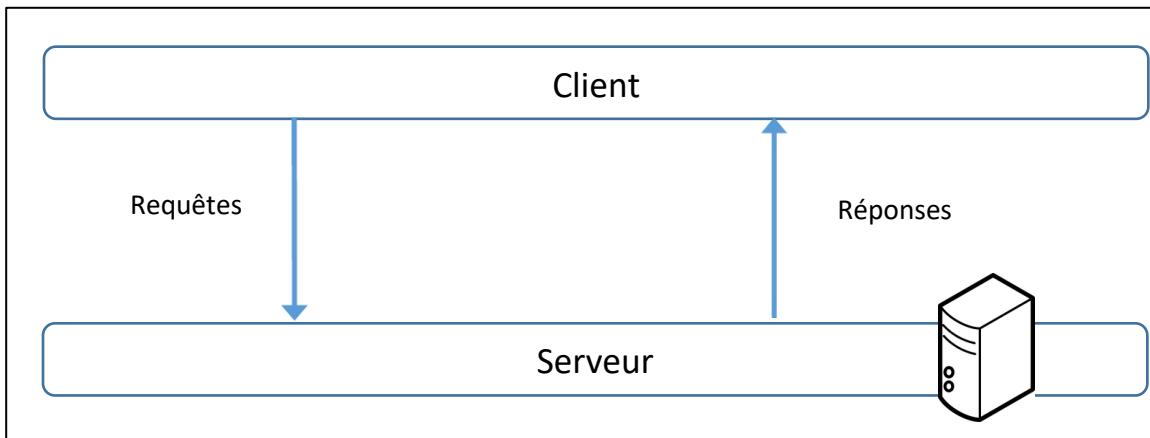


Figure 74 : Client - Serveur et Requêtes - Réponses

### 7.2.2 Désignation du client et du serveur

Le dialogue s'effectue entre le serveur LoRaWAN notre Application. La Figure 75 représente ces deux entités ainsi que les échanges réalisés. En haut de la figure, nous avons les serveurs LoRaWAN de TTN (bien sûr, cela pourrait être n'importe quel autre serveur LoRaWAN) et en bas, nous avons notre Application. On remarque ici que le serveur LoRaWAN possède déjà l'intitulé "serveur". Il faut absolument faire abstraction de ce terme "serveur" dans le nom "serveur LoRaWAN". Cela nous permettra de bien reprendre la démarche depuis le début, et de bien comprendre "qui est le serveur de qui?" et "qui est le client de qui?"

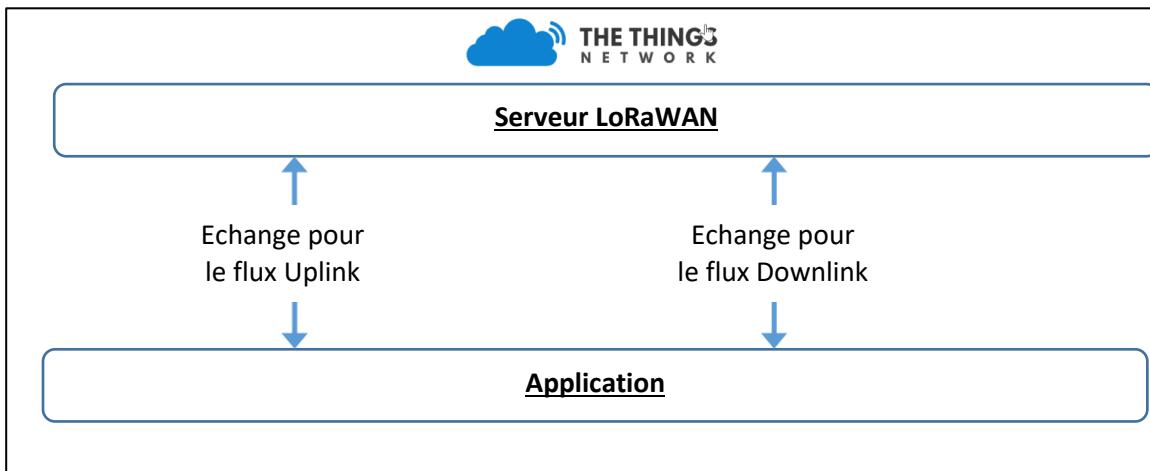


Figure 75 : Communication entre le serveur LoRaWAN et notre Application avec HTTP

Nous allons maintenant étudier les deux situations qui sont le flux Uplink et le flux Downlink représentés sur la Figure 76.

Commençons par la situation la plus courante qui est celle du flux Uplink, c'est-à-dire que nous cherchons à récupérer sur notre Application, les données du serveur LoRaWAN (TTN). La première idée, c'est de faire depuis notre Application une requête (1) à TTN pour qu'il nous fournissent ces données. Cette requête du protocole HTTP s'appelle une requête HTTP GET. Quand vous faites une requête HTTP GET à un serveur Web, il vous retourne le contenu de la page HTML qu'il contient. Ici,

TTN qui aura donc le rôle de serveur retournera les données LoRa **(2)**. Nous avons donc dans ce cas, TTN qui joue le rôle du serveur, et notre Application qui joue le rôle du client.

Si on parle maintenant du flux Downlink, c'est-à-dire que nous cherchons à récupérer sur le serveur LoRaWAN les données que l'utilisateur souhaite envoyer au Device LoRa. Nous allons ici faire en sorte le serveur LoRaWAN fasse des requêtes HTTP GET **(3)** à notre Application pour qu'elle nous fournissent ces données lors de la réponse **(4)**. Nous avons donc dans ce cas, le serveur LoRaWAN qui joue le rôle du client, et notre Application qui joue le rôle de serveur.

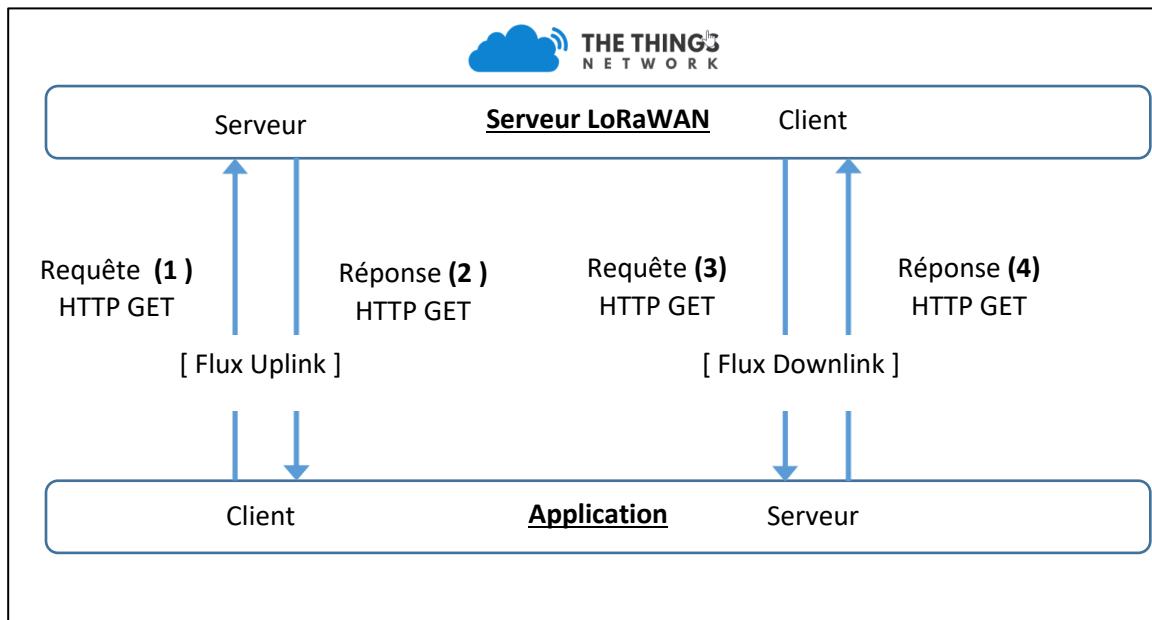
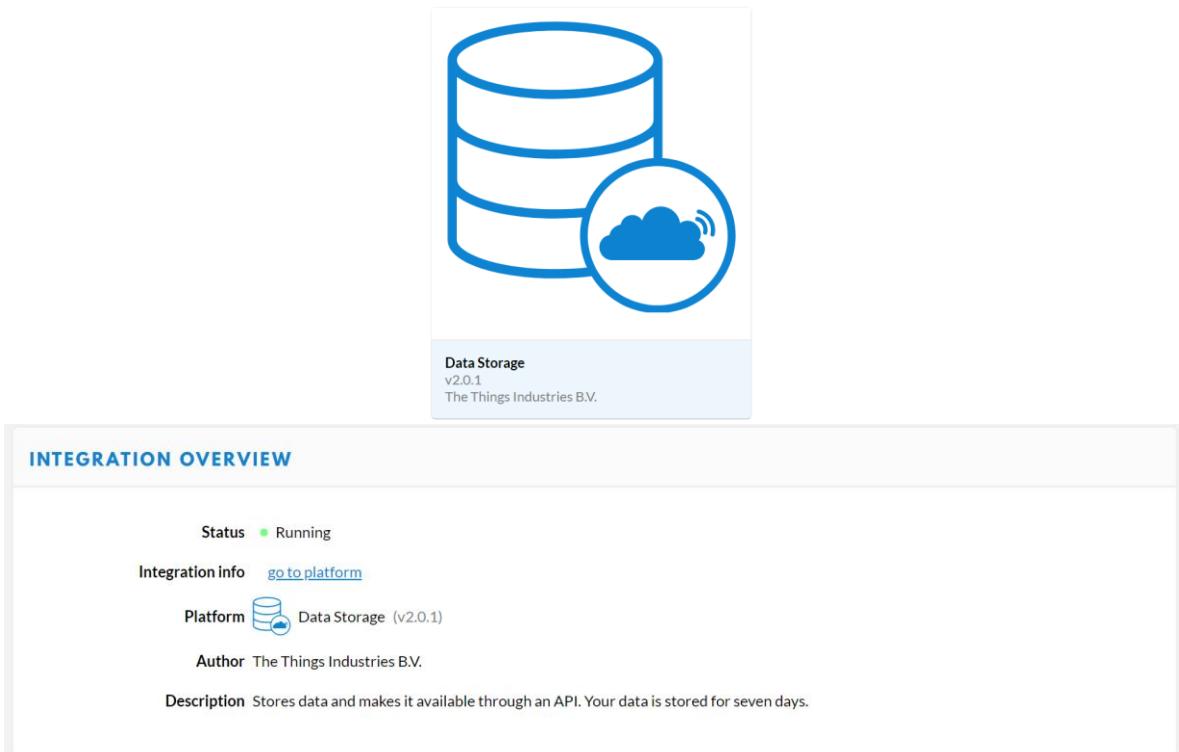


Figure 76 : Flux Uplink et Downlink en HTTP GET

### 7.2.3 Installation des services HTTP pour le flux Uplink

Nous allons mettre en place la récupération des données présentes sur les serveurs LoRaWAN afin de les rapatrier sur notre Application. Sur la Figure 76, cela représente les échanges Uplink, donc les trames **(1)** et **(2)**. Comme le montre la Figure 76, nous devons donc mettre en place un serveur HTTP sur nos serveurs LoRaWAN, et un client HTTP sur notre Application.

Nous commençons par la mise en place du serveur HTTP sur TTN. Pour cela, nous devons nous rendre dans notre console TTN. **TTN > Applications > Nom de l'Application > Intégration > Data Storage**. Le serveur est alors disponible ainsi qu'une sauvegarde temporaire des données pendant 7 jours.



*Figure 77 : Intégration du service HTTP dans TTN*

Pour prendre connaissance de l'API disponible pour récupérer les données, cliquer sur "go to platform" comme indiqué sur la Figure 77. La Figure 78 présente l'ensemble des APIs, qui rendent les requêtes simples et intuitives.

devices		Show/Hide   List Operations   Expand Operations
GET	/api/v2/devices	Query the devices for which data has been stored
<b>query</b>		Show/Hide   List Operations   Expand Operations
GET	/api/v2/query	Query data
<b>query/{device-id}</b>		Show/Hide   List Operations   Expand Operations
GET	/api/v2/query/{device-id}	Query data for a specific device

*Figure 78 : API REST disponible sur le serveur HTTP*

Nous allons donc maintenant générer les requêtes en suivant la documentation de l'API disponible pour récupérer les éléments souhaités. Cela est possible de plusieurs façons. Nous pouvons faire un premier test en testant les commandes directement sur la page de la documentation de l'API. Il faut d'abord autoriser la page à générer des commandes, puis tester les différentes API disponibles.

La deuxième méthode, est un peu plus complexe à mettre en œuvre, mais est beaucoup plus générique et nous servira dans de nombreux cas, c'est donc celle que j'expliquerai plus en détail. Pour cela nous allons utiliser un logiciel nommé POSTMAN [ [www.postman.com](http://www.postman.com) ] qui permet de générer toutes sortes de requêtes HTTP. Il nous suffira de nous référer à la documentation pour choisir les bons formats. En résumé, nous avons mis en place la structure présentée à la Figure 79 :

- "Data Storage Integration" pour la mise en place du serveur HTTP
- POSTMAN pour la mise en place du client HTTP

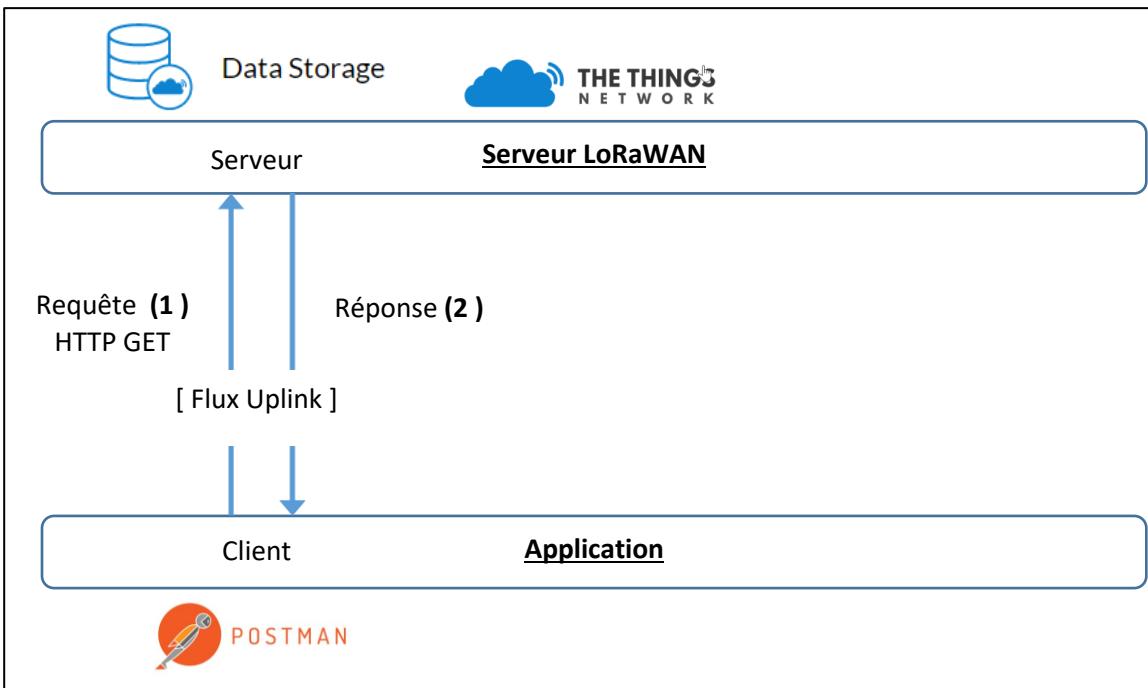


Figure 79 : Flux Uplink et Downlink en HTTP GET

Dans POSTMAN, nous allons maintenant réaliser la requête HTTP GET. Nous nous intéressons à la requête /api/v2/query permettant de connaître les données reçues par tous les Devices de l'application. Vérifier que vos Devices LoRa émettent bien et générer la requête suivante avec POSTMAN :

- ➔ **POSTMAN > New > Request > Save Request**
- ➔ Créer la requête suivante :
  - Type : HTTP GET
  - URL : [https://seminaire\\_lorawan.data.thethingsnetwork.org/api/v2/query](https://seminaire_lorawan.data.thethingsnetwork.org/api/v2/query)
  - Authorization : Récupérer la clé de votre application dans TTN [ **TTN > Applications > Nom de votre Application > Overview > Access Key** ], puis insérer la dans la commande HTTP : Onglet Authorization > TYPE : API Key . Dans le champs Key mettre : Authorization. Dans le champs Value mettre votre clé, précéder du mot : key (avec un espace entre key et votre clé).
- ➔ Envoyer la requête : **Send**

Si vous avez la moindre difficulté pour insérer votre commande vous pouvez l'importer directement dans POSTMAN via la commande Curl proposée dans la documentation de l'API :

- ➔ **POSTMAN > Import > Poste Raw Text > Mettre la commande Curl**

Vous devriez avoir la réponse à votre requête au format Json. Dans la réponse suivante, le Device "Arduino0" de mon application à reçu 0x01 ( AQ== en Base64 ) à l'heure indiquée.

```
{
  "device_id": "arduino0",
  "raw": "AQ==",
  "time": "2020-08-24T11:26:43.140932111Z"
}
```

#### 7.2.4 Remarques sur la méthode HTTP GET

Cette méthode est intéressante pour sa simplicité de mise en place : De simples requêtes HTTP GET permettent d'avoir les informations sur les données reçus sur le serveur LoRaWAN.

Le premier inconvénient est que nous avons travaillé uniquement sur le flux Uplink. La partie de droite de la Figure 76 n'a pas pu être implémentée car il n'y a pas de façon simple d'installer un client HTTP générant des requêtes GET sur TTN.

Le deuxième inconvénient est que pour le flux Uplink, nous passons notre temps à demander des données qui n'existent potentiellement pas. En effet, nous faisons des requêtes pour des données sans savoir si elles sont vraiment arrivées. Si un capteur émet de façon non régulière des valeurs, alors nous devrons faire des requêtes périodiques avec une forte chance d'avoir des réponses vides (car le Device n'aura rien émis).

Pour le flux Downlink, si nous avions pu installer le client sur TTN, le problème serait le même. Nous passerions notre temps à demander des commandes alors qu'il y a de fortes chances que l'utilisateur n'en ait passé aucune.

On voit bien qu'on a ici des objectifs contradictoires : On aimerait avoir une réception des données rapides sur notre Application, mais cette exigence impose de faire des demandes très fréquentes au serveur et donc d'augmenter considérablement la charge réseau.

La solution est donc de réorganiser les clients, les serveurs et les requêtes pour essayer d'optimiser la façon dont on transfère les données entre le serveur LoRaWAN et notre Application. Cela sera permis grâce aux requêtes HTTP POST.

### 7.3 Récupération des données avec le protocole HTTP (POST)

On reprend le schéma de la Figure 75 permettant de montrer les échanges entre le serveur LoRaWAN (TTN) et notre Application. Nous allons étudier les deux situations qui sont le flux Uplink et le flux Downlink représentés sur la Figure 80.

Pour le flux Uplink, nous allons répartir les rôles de façon différentes en imaginant que l'Application ne va pas demander au serveur LoRaWAN les informations, mais que le serveur LoRaWAN va plutôt les fournir elle-même à l'Application. Le serveur LoRaWAN va donc transmettre à l'Application une requête qui s'appelle HTTP POST **(1)** pour "poster" les données. La réponse **(2)** est un simple acquittement et ne contient pas de données. Donc ici, TTN jouera le rôle de client, et notre application, le rôle de serveur.

Inversement, pour le flux Downlink, l'utilisateur qui souhaite transmettre des données à destination du Device LoRa doit être capable de fournir une requête HTTP POST **(3)** vers TTN. TTN jouera le rôle de Serveur, et notre application, le rôle de client.

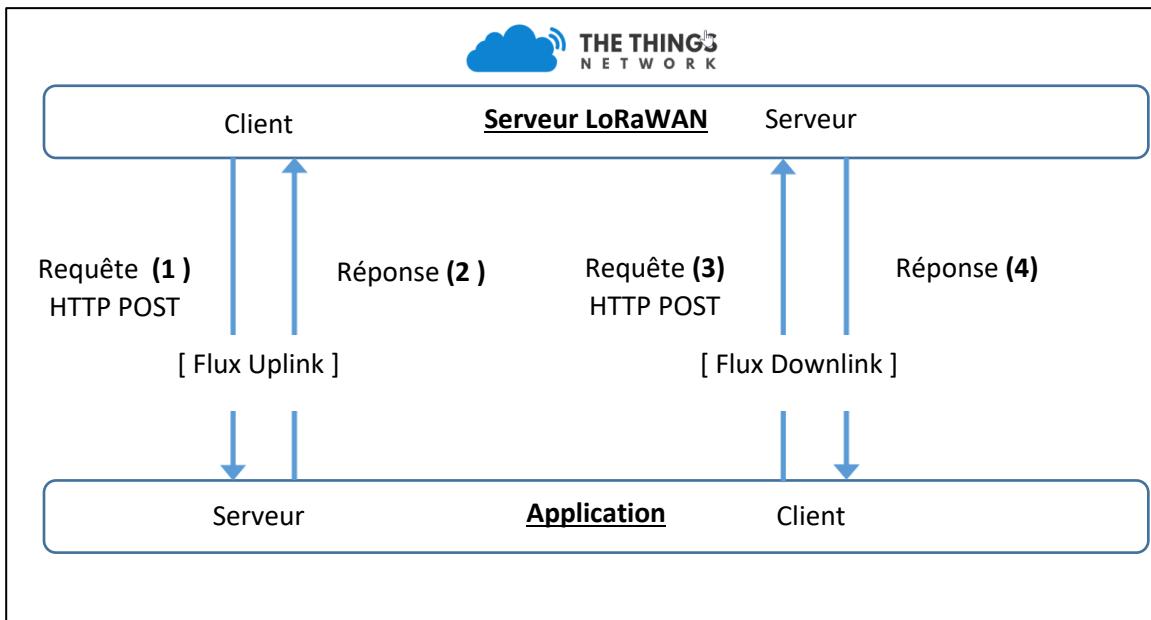


Figure 80 : Flux Uplink et Downlink en HTTP POST

### 7.3.1 Installation des services HTTP pour le flux Uplink

Nous allons mettre la place la récupération des données présentes sur les serveur LoRaWAN afin de les rapatrier sur notre Application. Sur la Figure 80, cela représente les échanges Uplink, donc les trames (1) et (2). Comme le montre la Figure 80, nous devons donc mettre en place un client HTTP sur nos serveurs LoRaWAN, et un serveur HTTP sur notre Application. Tous les deux ne traiteront plus des requêtes et des réponses HTTP GET comme nous l'avons vu au paragraphe 7.2, mais des requêtes et des réponses HTTP POST.

Nous commençons par la mise en place du serveur. Nous utiliserons un serveur HTTP disponible sur le web, gérant les requêtes HTTP POST [ <https://rbaskets.in/> ] ou [ <https://beeceptor.com/> ] par exemple.

- ➔ Aller sur <https://rbaskets.in/>
- ➔ Créer un nouveau Basket (Endpoint serveur)
- ➔ Conserver le token si vous souhaitez revenir sur ce même serveur plus tard et cliquer sur "Open Basket".
- ➔ L'adresse vers laquelle vous devez envoyer vos requêtes est alors précisée, conservez-là. Elle nous servira lorsque nous configurerons le client.
- ➔ Vous êtes donc dans l'attente de données, votre basket est vide et les requêtes que vous recevrez apparaîtront ici.

Nous devons maintenant mettre en place un client sur TTN. Pour cela, nous devons nous rendre dans notre console TTN. **TTN > Applications > Nom de l'Application > Integration > HTTP Integration**.

**Process ID**  
The unique identifier of the new integration process

✓

**Access Key**  
The access key used for downlink

 devices
  messages
 ✓

**URL**  
The URL of the endpoint

✓

**Method**  
The HTTP method to use

✓

Figure 81 : Ajout d'un client HTTP POST dans TTN

Pour valider le fonctionnement de notre architecture, nous pouvons soit :

- Envoyer une trame depuis un Device LoRa vers TTN
- Simuler l'envoi d'une trame d'un Device LoRa vers TTN (grâce à l'outil proposé par TTN vu paragraphe 5.2.5)

Dans les deux cas, voici un exemple de requête POST reçue sur notre serveur HTTP :

```
{
  "app_id": "test_app_lorawan_sylvainmontagny",
  "dev_id": "stm32lorawan_1",
  "hardware_serial": "0056A.....F49877",
  "port": 1,
  "counter": 0,
  "payload_raw": "qg==",
  "metadata": {
    "time": "2019-01-24T15:24:37.03499298Z"
  },
  "downlink_url": "https://integrations.thethingsnetwork.org/ttn-eu/api/v2/down/test_app_lorawan_sylvainmontagny/rbaskets?key=ttn-account-v2.....8ypjj7ZnL3KieQ"
}
```

Comme nous pouvons le voir, le contenu fourni à notre serveur est formaté en JSON (voir paragraphe 6.3). Voici quelques compléments d'information :

- payload\_raw : Frame Payload déchiffré, c'est donc les données en claires dans la base 64.
- downlink\_url : URL du serveur HTTP POST qui serviront à transmettre des données au Device LoRa (flux Downlink)

Nous récupérerons alors les données de nos capteurs au format JSON sur notre serveur POST. Reste à les traiter, les stocker (BDD, etc...), et les mettre à disposition d'un utilisateur comme nous le verrons au chapitre 10 lors de la création complète de notre propre Application.

### 7.3.1 Installation des services HTTP pour le flux Downlink

Nous allons mettre en place l'envoi des données utilisateur depuis l'Application en direction du serveur LoRaWAN. Sur la Figure 80, cela représente les échanges Downlink, donc les trames **(3)** et **(4)**. Comme le montre la Figure 80, nous devons donc mettre en place un client HTTP sur notre Application, et un serveur HTTP sur notre serveur LoRaWAN (TTN).

Nous commençons par la mise en place du serveur. C'est très simple puisqu'il existe déjà dans TTN et a été installé en même temps que le client au paragraphe précédent. Il n'y a donc rien de plus à faire pour la mise en place du serveur.

Pour le client, nous utiliserons à nouveau POSTMAN et nous allons générer un requête HTTP POST :

➔ **POSTMAN > New > Request > Save Request**

➔ Créer la requête suivante :

- Type : HTTP POST
- URL : L'adresse du server HTTP vers lequel il faut émettre les requêtes est donnée dans la trame reçue en Uplink précédemment. Dans l'exemple précédent cela correspond à l'URL :

```
"downlink_url": "https://integrations.thethingsnetwork.org/ttn-eu/api/v2/down/test_app_lorawan_sylvainmontagny/rbaskets?key=ttn-account-v2.....8ypjj7ZnL3KieQ"
```

- Body : **Body > Pretty > JSON** , avec le contenu :

```
{  
    "dev_id": "YourDeviceID",  
    "payload_raw": "aGVsbG8=",  
    "port": 1,  
    "confirmed": false  
}
```

➔ Envoyer la requête : **Send**

Vous devez envoyer le texte (payload\_raw) en base 64. Dans l'exemple ci-dessus « aGVsbG8= » correspond à la chaîne de caractère « hello ». Vous pouvez utiliser les nombreux encodeur/decodeur en ligne pour vous aider. Le texte doit s'afficher sur votre moniteur série de votre Device LoRa.

En résumé, l'Application que nous avons mise en place est conforme à la Figure 82 ci-dessous. Sur notre serveur LoRaWAN, c'est l'intégration du service "HTTP Intégration" qui joue le rôle de client (Uplink) et de serveur (Downlink). Sur notre Application, c'est rbasket qui joue le rôle de serveur (Uplink) et POSTMAN qui joue le rôle de client (Downlink).

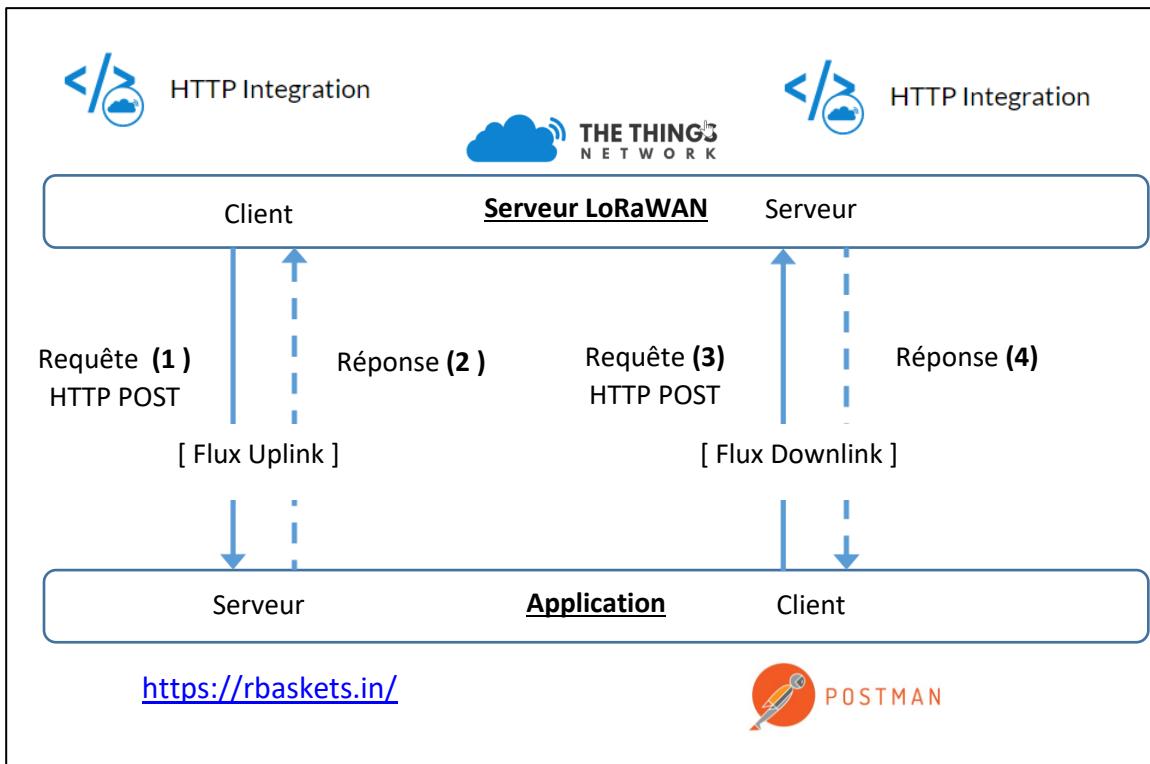


Figure 82 : Flux Uplink et Downlink en HTTP POST

## 7.4 Récupération des données avec le protocole MQTT

### 7.4.1 Présentation du protocole MQTT

MQTT est un protocole léger qui permet de s'abonner à des flux de données. Plutôt que l'architecture Client / Serveur classique qui fonctionne avec des Requêtes / Réponses, MQTT est basé sur un modèle Publisher / Subscriber. La différence est importante, car cela évite d'avoir à demander (Requête) des données dont on n'a aucune idée du moment où elles vont arriver. Une donnée sera donc directement transmise au Subscriber dès lors que celle-ci a été reçue dans le Broker (serveur central).

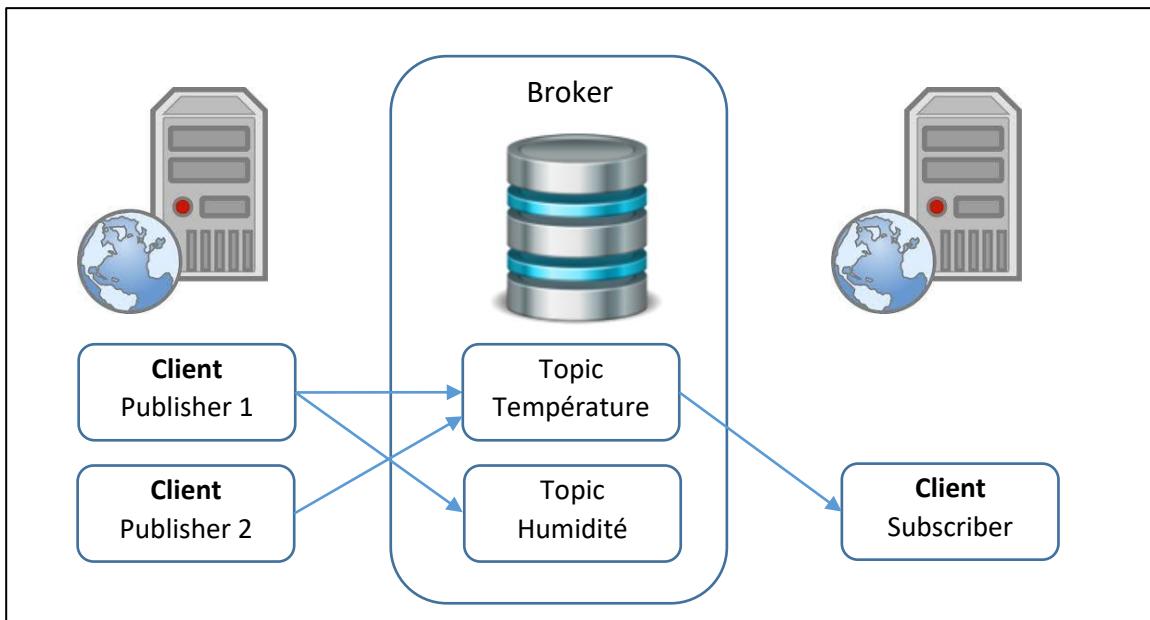


Figure 83 : Modèle Publisher / Subscriber du protocole MQTT

Pour recevoir les données appartenant à un Topic, un Subscriber doit souscrire (comme son nom l'indique) au préalable sur ce Topic.

MQTT est un protocole qui repose sur TCP. L'encapsulation des trames sur le réseau est donc la suivante :

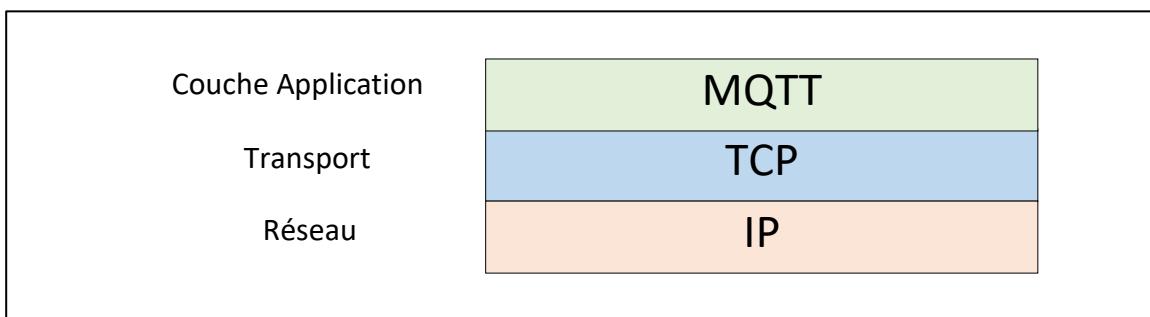


Figure 84 : Protocoles utilisés pour la communication avec MQTT

On peut le vérifier par une capture de trame sur Wireshark.

```
> Ethernet II, Src: Raspber_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 85 : Capture d'une trame MQTT avec Wireshark

On peut noter que le port TCP utilisé pour le protocole MQTT (non chiffré) est le 1883.



- ➲ Les Publishers et les Subscribers n'ont pas besoin de se connaître.
- ➲ Les Publishers et les Subscribers ne sont pas obligés de s'exécuter en même temps.

#### 7.4.2 Connexion au Broker MQTT

Nous nous intéresserons essentiellement aux options de connexion qui permettront de gérer la Qualité de Service (QoS). Pour se connecter, un client MQTT envoie deux informations importantes au Broker :

keepAlive : C'est la période la plus longue pendant laquelle le client Publisher ou Subscriber pourra rester silencieux. Au-delà, il sera considéré comme déconnecté.

cleanSession : Lorsque le Client et le Broker sont momentanément déconnectés (au-delà du keepAlive annoncé), on peut donc se poser la question de savoir ce qu'il se passera lorsque le client sera à nouveau connecté :

- Si la connexion était non persistante (cleanSession = True) alors les messages non transmis sont perdus. Quel que soit le niveau de QoS (Quality of Service).
- Si la connexion était persistante (cleanSession = False) alors les messages non transmis seront éventuellement réémis, en fonction du niveau de QoS. Voir le chapitre 7.4.4.

#### 7.4.3 Qualité de Service au cours d'une même connexion

A partir du moment où le Client se connecte au Broker, il est possible de choisir un niveau de fiabilité des transactions. Le Publisher fiabilise l'émission de ces messages vers le Broker, et le Subscriber fiabilise la réception des messages en provenance du Broker. On parle ici du cas d'une même connexion, c'est-à-dire entre le moment où le Client se connecte, et le moment où :

- Soit il se déconnecte explicitement (Close connexion)
- Soit il n'a rien émis, ni fait signe de vie pendant le temps "keepAlive"

Lors d'une même connexion la Qualité de Service (QoS) qui est mise en œuvre dépend uniquement de la valeur du QoS selon les valeurs suivantes :

QoS 0 "At most once" (au plus une fois) : Le premier niveau de qualité est "sans acquittement". Le Publisher envoie un message une seule fois au Broker et le Broker ne transmet ce message qu'une seule fois aux Subscribers. Ce mécanisme **ne garantit pas** la bonne réception des messages MQTT.

QoS 1 "At least once" (au moins une fois) : Le deuxième niveau de qualité est "avec acquittement". Le Publisher envoie un message au Broker et attend sa confirmation. De la même façon, le Broker envoie un message à ces Subscribers et attend leurs confirmations. Ce mécanisme **garantit** la réception des messages MQTT.

Cependant, si les acquittements n'arrivent pas en temps voulu, ou s'ils se perdent, la réémission du message d'origine peut engendrer une duplication du message. Il peut donc être reçu plusieurs fois.

QoS 2 "Exactly once" (exactement une fois) : Le troisième niveau de qualité est "garanti une seule fois". Le Publisher envoie un message au Broker et attend sa confirmation. Le Publisher donne alors l'ordre de diffuser le message et attend une confirmation. Ce mécanisme **garantit** que quel que soit le nombre de tentatives de réémission, le message ne sera délivré **qu'une seule fois**.

La Figure 86 montre les trames émises pour chaque niveau de QoS.

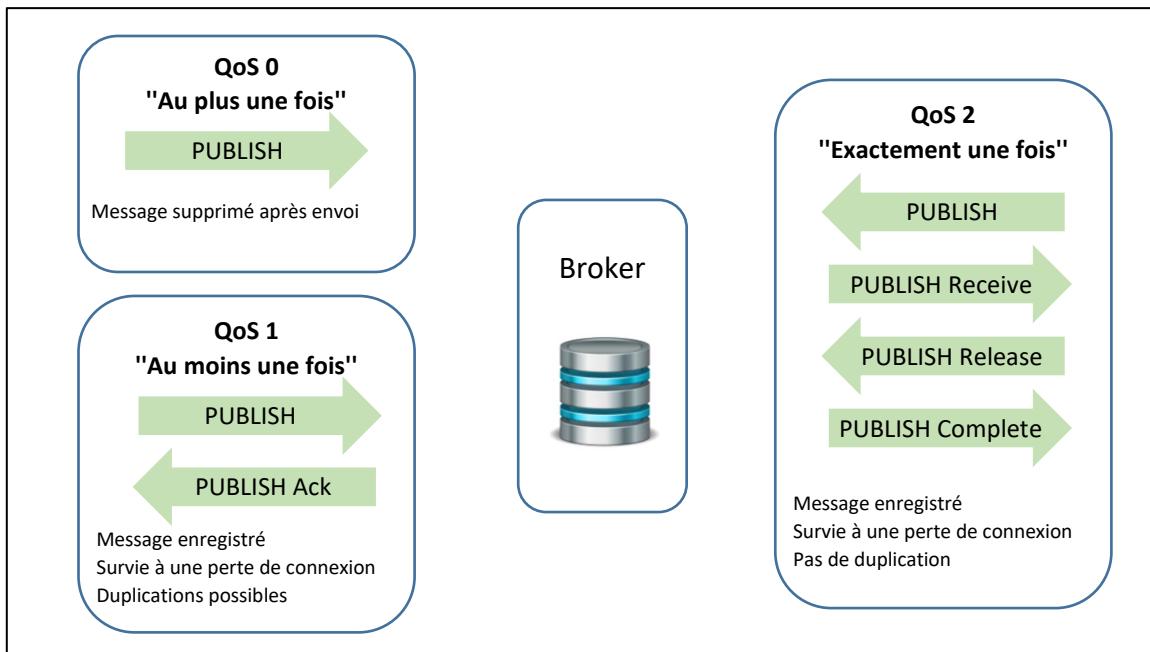


Figure 86 : Qualité de Service en MQTT

La Figure 87 représente les 3 captures de trames sur Wireshark représentant les 3 QoS (1, 2 et 3) que nous venons d'expliquer.

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 87 : Capture de trame avec QoS = 0, puis QoS = 1, puis QoS = 2

#### 7.4.4 Qualité de Service après une reconnexion

La question que nous nous posons est de savoir ce que deviennent les messages publiés sur le Broker lorsqu'un ou plusieurs Subscribers sont momentanément inaccessibles. Il est possible de conserver les messages qui ont été publiés sur le Broker afin de les retransmettre lors de la prochaine connexion. Cette possibilité de sauvegarde doit être activée à l'ouverture de connexion grâce au flag `cleanSession = 0`. La connexion sera alors persistante, c'est-à-dire que le Broker enregistre tous les messages qu'il n'a pas réussi à diffuser au Subscriber.

Le Tableau 15 résume l'effet du flag `cleanSession` et du QoS.

Clean Session Flag	Subscriber QoS	Publisher QoS	Comportement
True (= 1)	0 / 1 / 2	0 / 1 / 2	Messages perdus
False (= 0 )	0	0 / 1 / 2	Messages perdus
False (= 0 )	0 / 1 / 2	0	Messages perdus
False (= 0 )	1 / 2	1 / 2	Tous les messages sont retransmis

Tableau 15 : Qualité de Service en fonction de la valeur du QoS et du flag cleanSession

#### 7.4.5 Les Topics du protocole MQTT

Les chaînes de caractères décrivant un sujet forment une arborescence en utilisant la barre oblique "/" comme caractère de séparation. La Figure 88 et le Tableau 16 donne un exemple d'organisation de Topic.

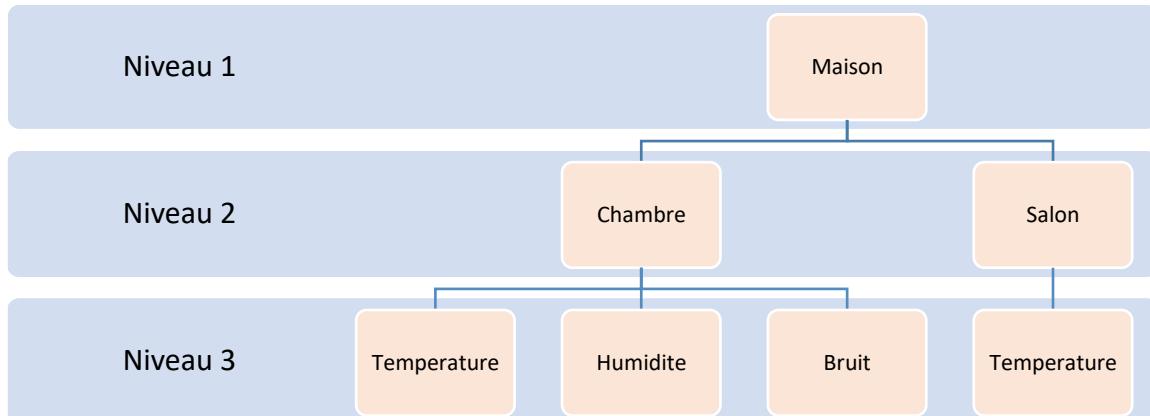


Figure 88 : Exemple de hiérarchie de Topic MQTT

Nom du Topic	Détail du Topic
Maison/Chambre/Temperature	La température de la chambre de la maison
Maison/Chambre/Bruit	Le bruit de la chambre de la maison
Maison/Salon/Temperature	La température du Salon de la maison

Tableau 16 : Exemple de Topic

Un client peut s'abonner (ou se désabonner) à plusieurs branches de l'arborescence à l'aide de "jokers" englobant plusieurs Topics. Deux caractères "jokers" existent :

- Le signe plus "+" remplace n'importe quelles chaînes de caractères sur le niveau où il est placé.
- Le dièse "#" remplace n'importe quelles chaînes de caractères sur tous les niveaux suivants. Il est obligatoirement placé à la fin.

Nom du Topic	Détail du Topic
Maison/+/Temperature	Les températures de toutes les pièces de la maison
Maison/#	La température, l'humidité et le bruit de toute la maison.

Tableau 17 : Exemple de Topic

#### 7.4.6 Mise en place d'un Broker MQTT

Afin de bien comprendre le fonctionnement du protocole de MQTT, on réalise des tests indépendamment de TTN. Nous allons mettre en place :

- Un Broker : Mosquitto <https://test.mosquitto.org/>
- Un client Publisher : MQTT Box sur Windows

- Un client Subscriber : MQTT Box sur Windows

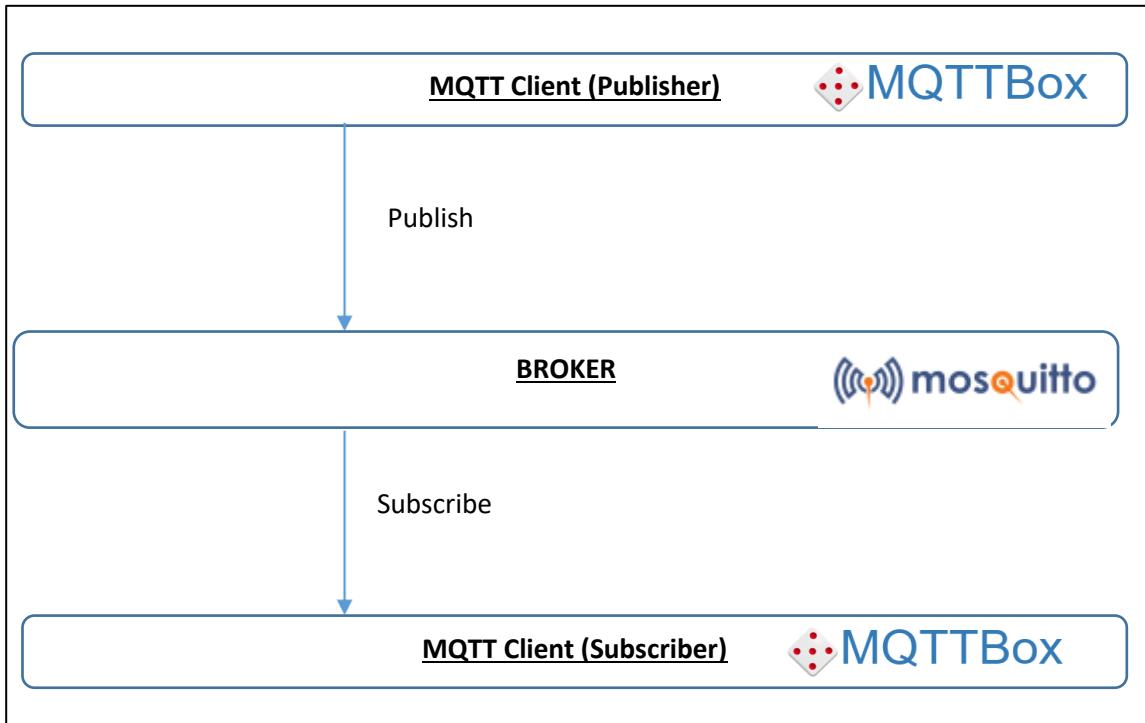


Figure 89 : Test du protocole MQTT

Le Broker MQTT est commun à tout le monde. Nous pouvons soit le réaliser nous-même, soit utiliser un Broker MQTT public de test. Nous utiliserons le Broker de test de Mosquitto disponible sur <https://test.mosquitto.org/>.

Si nous décidions de le réaliser nous-même à l'aide d'une Raspberry Pi (par exemple). L'installation se ferait en deux lignes de commande :

- apt-get install mosquitto // Installation
- sudo systemctl start mosquitto.service // Lancement du service

Il est nécessaire d'installer *mosquitto-clients* si on souhaite que le Raspberry PI joue aussi le rôle de client, ce qui n'est pas notre cas.

#### 7.4.7 Mise en place d'un Publisher et d'un Subscriber MQTT

- ➔ Lancer le logiciel MQTTBox.
- ➔ MQTTBox > Create MQTT Client.
- ➔ Dans ce client les seuls champs indispensables sont :

  - Protocol : MQTT /TCP
  - Host : test.mosquitto.org

MQTT Client Name	MQTT Client Id
mosquito public	5e31ac5f-50ef-4015-8979-de30f6f 
Protocol	Host
mqtt / tcp	test.mosquitto.org

Figure 90 : Configuration d'un Client MQTT dans MQTT Box

- ➔ Tester l'envoi et la réception sur les Topics de votre choix.

#### 7.4.8 Récupérer des données depuis l'Application Server avec MQTT

La récupération des données fait référence au flux Uplink. Dans ce cas :

- TTN joue aussi le rôle de Broker
- Notre application joue le rôle de Subscriber

Nous pouvons donc représenter l'application globale par la Figure 91. Nos explications feront référence à la trame (1).

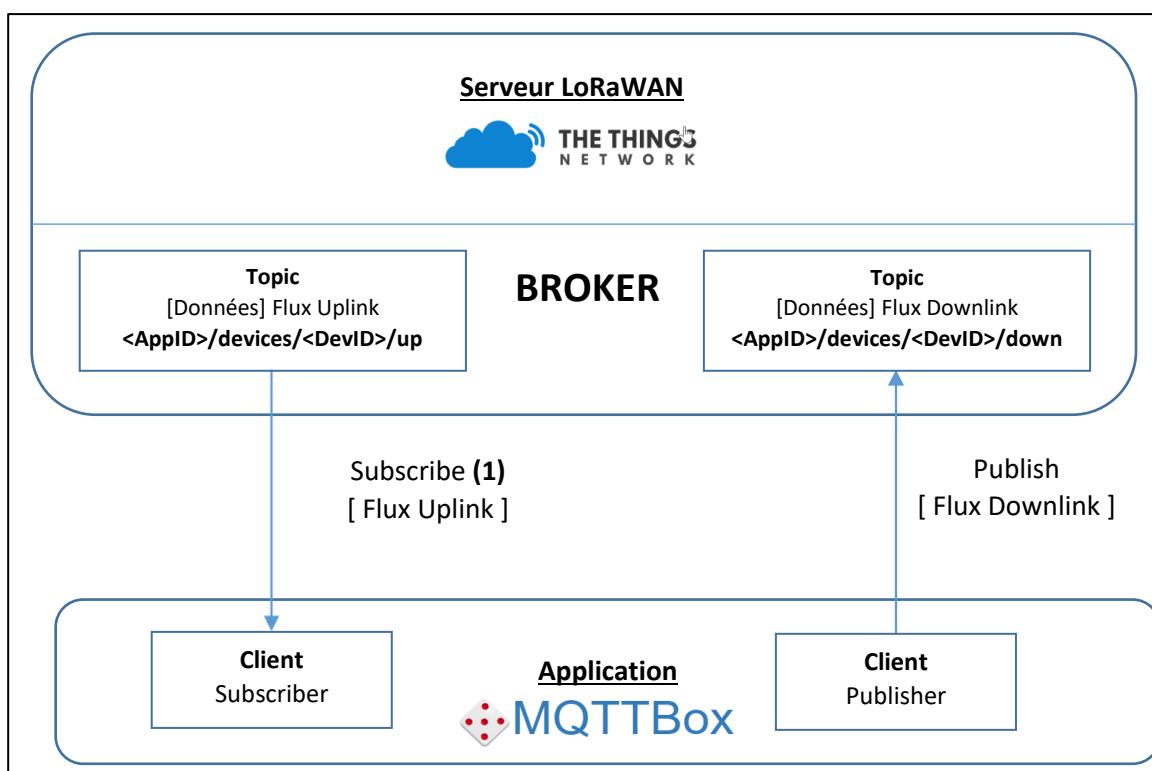


Figure 91 : Mise en place du Broker, des Publishers et des Subscriber

Le BROKER est déjà intégré dans TTN, c'est d'ailleurs le cas dans tous les serveurs LoRaWAN existants. Il nous reste à configurer le client Subscriber pour gérer le flux Uplink. Nous utiliserons à nouveau MQTT Box mais cette fois avec la configuration suivante :

- Protocol : **mqtt / tcp**
- Host : C'est l'@IP du Broker vers lequel il faut se connecter. Dans notre cas il s'agit de celui de TTN dont l'adresse est : **eu.thethings.network**

- **Username** : La connexion vers le broker MQTT est soumis à une authentification Username / Password. Dans notre cas le Username correspond au nom de notre application, c'est-à-dire : **seminaire\_lorawan**. Si vous avez choisi une autre application, il faut bien mettre la vôtre.
- **Password** : Le Password est nommé « Access Key » par TTN. Elle vous sera donnée par l'enseignant dans le cadre de ce test. Si vous avez enregistré votre propre application, vous trouverez l'Access Key dans : **TTN > Application > Nom\_de\_votre\_application > Overview**.



Figure 92 : Access Keys de l'application dans TTN

<b>MQTT Client Name</b>	<b>MQTT Client Id</b>
MQTT The Things Network	c029dd54-55e3-4fef-b55e-798fbaf
<b>Protocol</b>	<b>Host</b>
mqtt / tcp	eu.thethings.network:1883
<b>Username</b>	<b>Password</b>
seminaire_lorawan	*****

Figure 93 : Configuration du Client dans MQTT Box

Le client MQTT étant configuré, il est maintenant capable de se connecter au Broker. Il reste donc à définir le fait que le client sera Subscriber. Les informations que recevra le Subscriber dépendent du Topic auquel nous souscrivons. Voici les Topic disponibles sur le Broker de TTN :

- <**AppID**> correspond au nom de votre Application
- <**DevID**> correspond au nom de votre Device LoRa

Détail du Topic	Nom du Topic
[Données] Flux Uplink	< <b>AppID</b> >/devices/< <b>DevID</b> >/up
[Données] Flux Downlink	< <b>AppID</b> >/devices/< <b>DevID</b> >/down
[Activation Events] Activation d'un Device	< <b>AppID</b> >/devices/< <b>DevID</b> >/events/activations
[Management Events] Création d'un Device	< <b>AppID</b> >/devices/< <b>DevID</b> >/events/create
[Management Events] Update d'un Device	< <b>AppID</b> >/devices/< <b>DevID</b> >/events/update
[Management Events] Suppression d'un Device	< <b>AppID</b> >/devices/< <b>DevID</b> >/events/delete
[Downlink Events ] Message programmé	< <b>AppID</b> >/devices/< <b>DevID</b> >/events/down/scheduled
[Downlink Events ] Message envoyé	< <b>AppID</b> >/devices/< <b>DevID</b> >/events/down/sent
[Erreurs] Uplink erreurs	< <b>AppID</b> >/devices/< <b>DevID</b> >/events/up/errors
[Erreurs] Downlink erreurs	< <b>AppID</b> >/devices/< <b>DevID</b> >/events/down/errors
[Erreurs] Activations erreurs	< <b>AppID</b> >/devices/< <b>DevID</b> >/events/activations/errors

Tableau 18 : Topics enregistrés dans TTN

Dans un premier temps nous souscrivons au Topic : **+/devices/+/up**. Cela signifie que nous souscrivons à **tous** les Devices LoRa de **toutes** nos applications pour le flux Uplink.

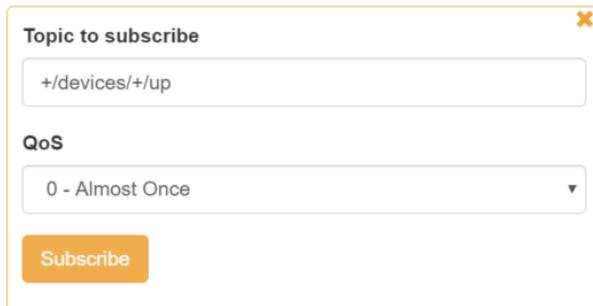


Figure 94 : Configuration du Subscriber

Les éléments émis par le Broker seront alors affichés dans MQTTBox.

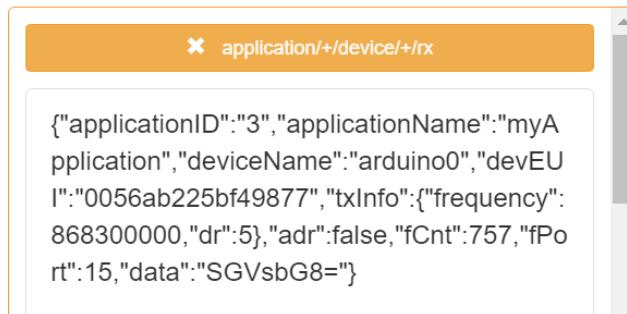


Figure 95 : Trame LoRaWAN reçue sur le Subscriber MQTT

Ces données sont écrites en JSON, nous pouvons les remettre en forme :

```
{
  "applicationID": "3",
  "applicationName": "myApplication",
  "deviceName": "arduino0",
  "devEUI": "0056xxxxxxxxx877",
  "txInfo": {
    "frequency": 868500000,
    "dr": 5
  },
  "adr": false,
  "fCnt": 792,
  "fPort": 15,
  "data": "SGVsbG8="
}
```

Le "Frame Payload" déchiffré est fourni dans le champ "data" en base 64. La valeur fournie par "data": "SGVsbG8=" correspond bien à la chaîne de caractères "hello" que nous avons émise avec le Device LoRa.

#### 7.4.9 Envoyer des données depuis notre Application avec MQTT

L'envoi des données fait référence au flux Downlink. Dans ce cas :

- Notre application joue le rôle de Publisher
- TTN joue le rôle de Broker

Nos explications feront référence à la trame (2) de la Figure 91.

Le BROKER étant toujours réalisé par TTN, il est déjà configuré. Il nous reste à configurer le client Subscriber. Nous utiliserons à nouveau MQTT Box. Le rôle de client dans MQTT Box a été fait au paragraphe 7.4.8. Il nous reste simplement à ajouter le rôle de Publisher. La configuration est la suivante :

- Topic du Subscriber : **seminaire\_lorawan/devices/arduino0/down** où seminaire\_lorawan est à remplacer par le nom de votre application, et arduino0 par le nom du Device LoRa vers lequel vous souhaitez envoyer une donnée.
- Le Payload doit être au format JSON. L'exemple suivant envoie «hello» :

```
{  
    "dev_id": "arduino0",  
    "payload_raw": "aGVsbG8=",  
    "port": 1,  
    "confirmed": false  
}
```

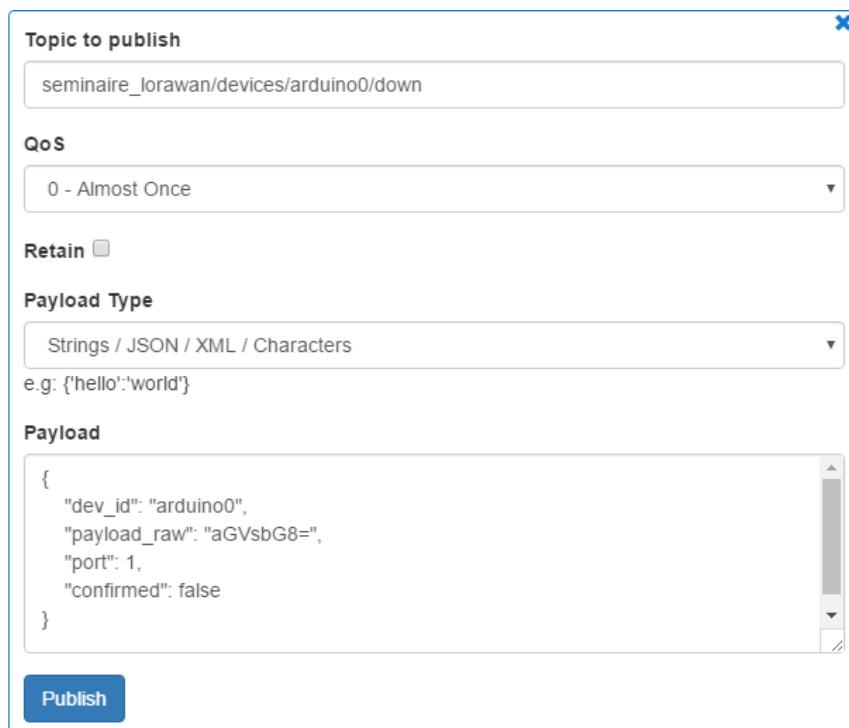


Figure 96 : Configuration du Publisher

Vous devriez voir les données arriver sur votre Device LoRA.

## 8 La création de notre propre Device LoRa

Pour fonctionner convenablement, il est indispensable que le Device LoRa possède :

- Un espace pour stocker le Firmware de l'application utilisateur
- Une pile de protocole LoRaWAN
- Une interface radio (Transceiver) LoRa

A partir de là, on peut imaginer plusieurs architectures qui possèdent chacune des avantages et des inconvénients. Nous allons les étudier puis nous les résumerons dans le Tableau 19 à la fin de ce chapitre.

### 8.1 Architecture Microcontrôleur + Transceiver

#### 8.1.1 Présentation de l'architecture

Dans ce type d'architecture, un microcontrôleur gère à la fois la pile LoRaWAN et le Firmware de l'application, nous avons donc qu'un seul microcontrôleur. En revanche, cela nécessite d'avoir une pile de protocole LoRaWAN (stack LoRaWAN) à disposition.

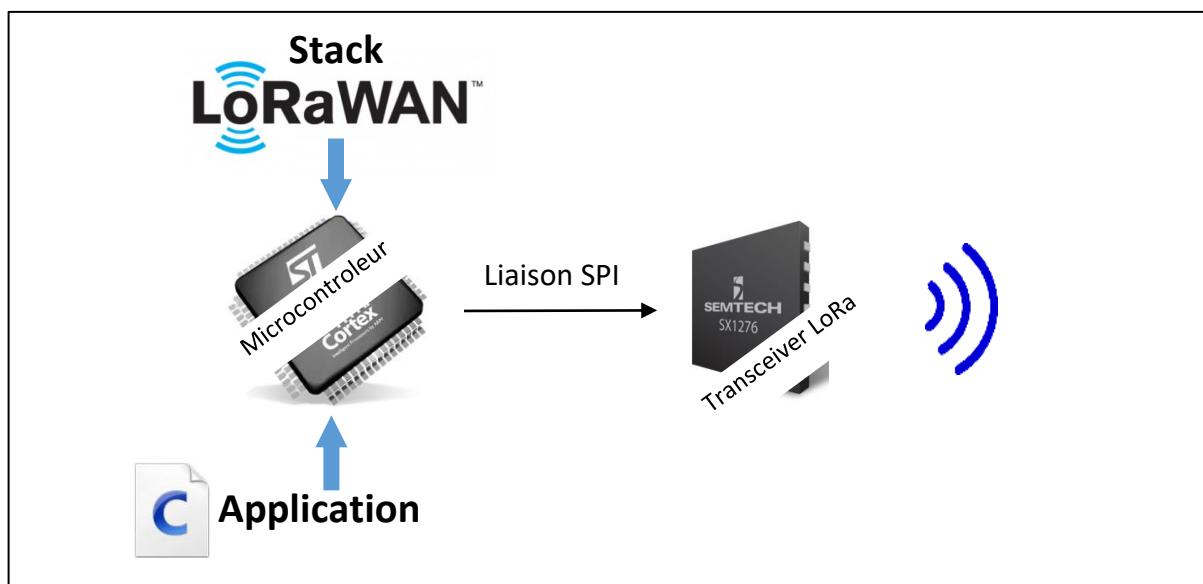


Figure 97 : Device LoRa avec microcontrôleur + Transceiver

Le SX1272 (ou SX1276, ou SX1262) sont des transceivers LoRa de chez SEMTECH.. Ils gèrent la partie physique du protocole : Modulation, détection de préambule.... La gestion complète du protocole LoRaWAN est donc réalisée par une pile logicielle qui est implémentée dans le microcontrôleur. Les différentes stack LoRaWAN sont présentées au paragraphe 8.6. Les Transceivers sont pilotés en SPI.

Ce choix-là est assez abouti et optimisé en terme de consommation, car il n'y a qu'un seul microcontrôleur qui gère tout. En revanche, c'est une solution beaucoup plus complexe en ce qui concerne la partie logicielle. En effet, il faudra se plonger dans la Stack. Même si la partie Application est bien différenciée, il peut être assez compliqué de réaliser un système très modulaire car la stack LoRAWAN et l'application se déroulent en même temps. Il faut en permanence faire très attention à ne pas se supprimer des ressources mutuellement.

### 8.1.2 Exemple de carte de développement

Chez ST, la carte de développement P-NUCLEO-LRWAN1 permet d'associer un microcontrôleur STM32L073 (Cortex M0+) avec un Transceiver SX1272.

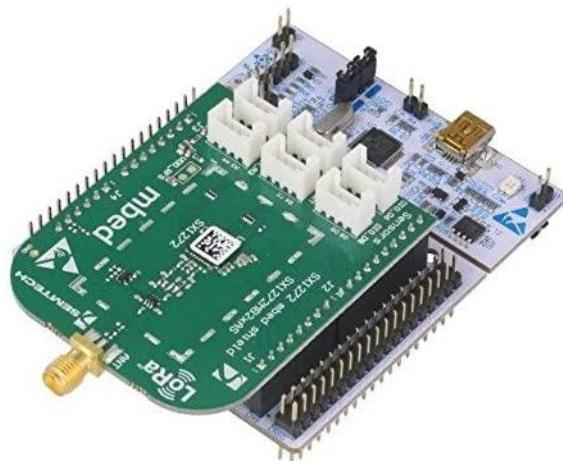


Figure 98 : P-NUCLEO-LRWAN1 package

### 8.1.3 Solution d'implémentation

Après la période de prototypage sur des cartes comme la P-NUCLEO-LRWAN1, on peut passer à la réalisation de son propre PCB. Si on souhaite réaliser la soudure soit même dans un atelier raisonnablement équipé, il sera très difficile de souder un Transceiver comme le SX1272/76. La solution intéressante est d'utiliser des cartes modules (breakout) qui intègre en plus déjà un certain nombre de composants.



Figure 99 : Exemple de module chez NiceRF

## 8.2 Architecture Module LoRaWAN Standalone

Dans ce type d'architecture, nous utilisons un module autonome qui regroupe à la fois un microcontrôleur qui possède le Firmware de l'application et la stack LoRaWAN, et un Transceiver. Attention, derrière ce module, il y a en fait plusieurs composant. Le Transceiver n'est pas intégré dans le microcontrôleur comme nous le verrons au paragraphe 8.4, mais du point de vue du programmeur, c'est presque la même chose.

Cette solution possède l'avantage de simplifier la partie hardware de la solution précédente. En revanche, on a toujours la proximité du Firmware applicatif et de la stack LoRaWAN à gérer. Le

module possède un certain nombre de périphérique à disposition (ADC, I2C, UART, GPIO...) pour mettre en œuvre l'application du Device LoRa.

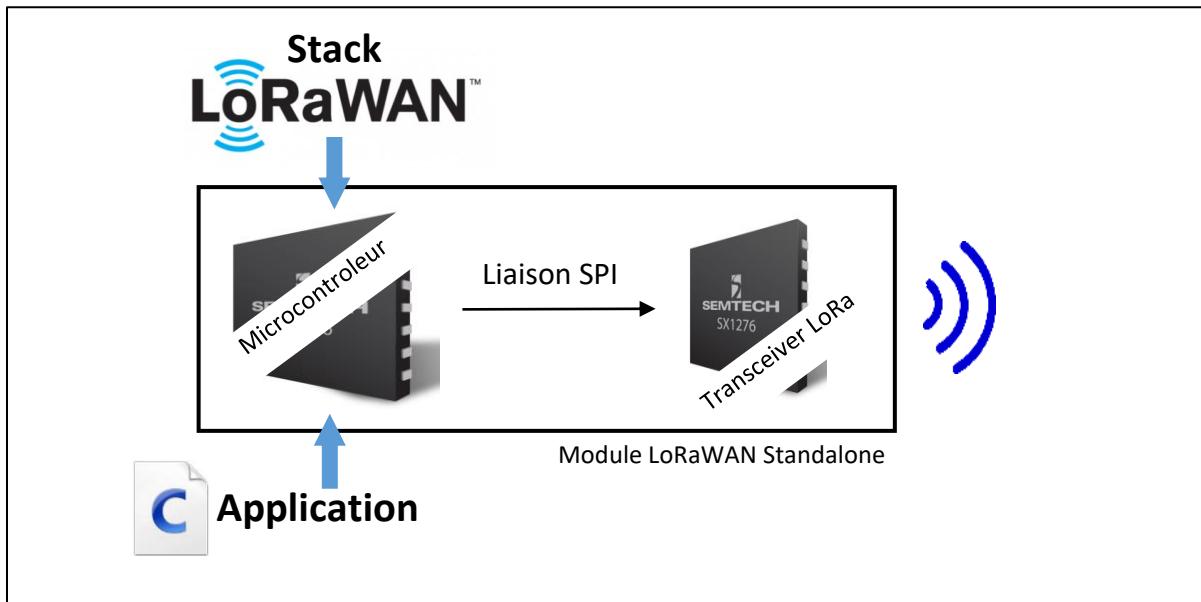


Figure 100 : Module LoRaWAN Standalone

### 8.2.1 Exemple de module

Le Murata : CMWX1ZZABZ peut fonctionner en standalone. Comme nous l'avons vu, c'est-à-dire qu'il n'aura pas besoin d'autre composant pour fonctionner. Seul l'antenne avec adaptation d'impédance est à rajouter.



Figure 101 : Module intégrant stack LoRaWAN et Transceiver

### 8.2.2 Exemple de carte de développement

Le module Murata intègre un microcontrôleur STM32, ST a donc développé une carte de développement pour le composant CMWX1ZZABZ. Il s'agit de la carte discovery B-L072Z-LRWAN1.



Figure 102 : Carte discovery B-L072Z-LRWAN1 de chez ST

### 8.3 Architecture microcontrôleur + Module LoRaWAN

Dans ce type d'architecture, le microcontrôleur ne gère que le Firmware de l'application. Un module externe, piloté par une liaison série gère l'ensemble du protocole LoRaWAN. Ce module est lui-même une combinaison d'un microcontrôleur et d'un Transceiver en interne comme le montre la Figure 103.

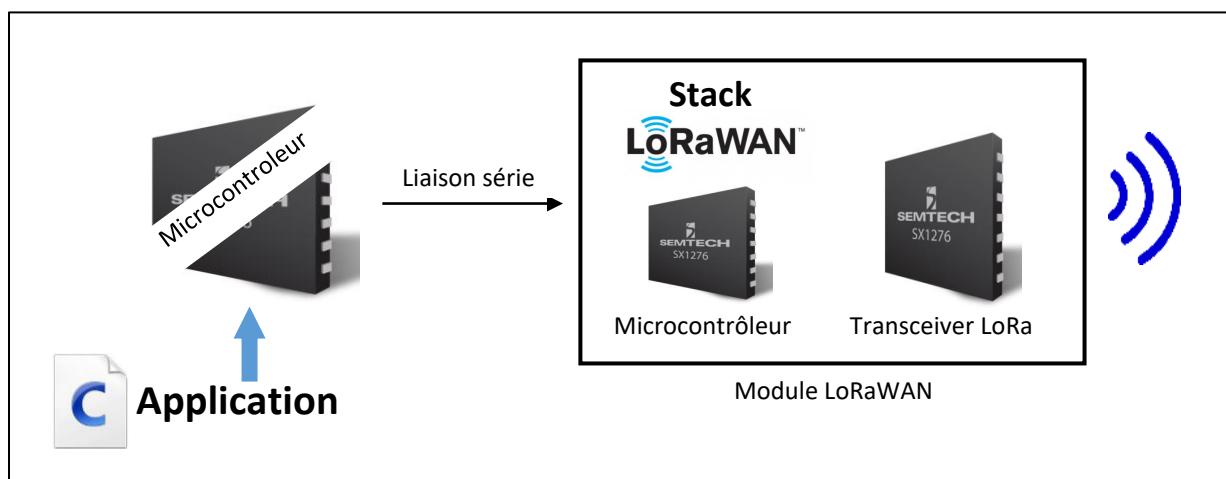


Figure 103 : Device LoRa avec microcontrôleur et module LoRaWAN

Nous pouvons choisir n'importe quel  $\mu$ C 32 bits (ou même 8 bits). La stack LoRaWAN qui prend beaucoup de ressource est intégrée au module.

#### 8.3.1 Exemple de module LoRaWAN

Voici deux modules couramment utilisés :

- Microchip : RN2483
- Murata : CMWX1ZZABZ : Il s'agit du même module que celui vu au paragraphe 8.2, mais le Firmware est différent. Il est ici configuré pour répondre à des commandes d'un maître en liaison série au niveau applicatif, alors que dans l'exemple du paragraphe 8.2, il fonctionne en spi au niveau LoRa MAC.

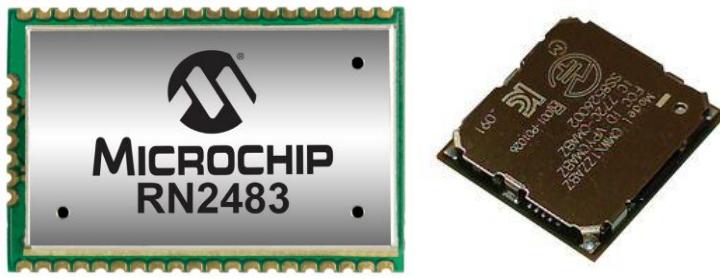


Figure 104 : Module intégrant stack LoRaWAN et Transceiver

Ces modules se pilotent à l'aide d'un jeu de commande AT sur une liaison série. Il existe des librairies pour Arduino pour ces deux modules. ST propose aussi une librairie de gestion des commandes AT pour le CMWX1ZZABZ. En effet, le CMWX1ZZABZ est en réalité une combinaison d'un STM32L0 et d'un SX1276.

Ce choix-là possède l'avantage considérable de sa simplicité. Vous n'avez rien à gérer sur le protocole LoRa/LoRaWAN car tout est fait dans le module. Le fait d'envoyer des commandes applicatives simples permet au concepteur du Device LoRa de se focaliser sur son application.

La contrepartie est évidemment le fait que le système global possède 2 microcontrôleurs : un pour le Firmware de l'application et un pour la gestion de la stack LoRaWAN. Cela aura des influences sur le prix de l'ensemble, et bien sûr, sa consommation.

A savoir que seul le module RN2483 peut être soudé simplement avec des solutions "amateurs", l'empreinte du CMWX1ZZABZ n'est pas adaptée.

### 8.3.1 Exemple de carte de développement

Voici deux exemples de cartes de développement couramment utilisées.



Figure 105 : Microcontrôleur ATMEL + Module RN2483



Figure 106 : Microcontrôleur ATMEL + Module CMWX1ZZABZ

## 8.4 Architecture Microcontrôleur Wireless LoRaWAN

ST vient de développer en 2020 le premier microcontrôleur avec LoRa intégré. Il s'agit du STM32WLE5 basé sur une STM32L4. A l'heure où j'écris ces lignes il n'y a pas de carte de développement associée mais cela devrait être le cas très prochainement. Dans cette architecture, la stack LoRaWAN, le Firmware applicatif et le Transceiver LoRa sont tous intégrés dans un seul et unique microcontrôleur. C'est la solution la plus intéressante en terme de cout, de consommation et de surface utilisée.

## 8.5 Résumé des architectures

Un résumé comparatif des solutions est donné Tableau 19. Les caractéristiques fournies ne sont pas chiffrées.

	<b>Microcontrôleur + Transceiver</b>	<b>Module LoRa Standalone</b>	<b>Microcontrôleur + Module LoRa</b>	<b>Microcontrôleur Wireless LoRa</b>
<b>Encombrement</b>	Moyen	Faible	Elevé	Très faible
<b>Coût</b>	Elevé	Moyen	Très élevé	Faible
<b>Complexité logicielle</b>	Elevé	Elevé	Faible	Elevé

Tableau 19 : Résumé des avantages et inconvénients des architectures

## 8.6 Les stacks LoRa à disposition

Si nous utilisons un Device qui intègre déjà une stack LoRa, alors nous n'avons pas à nous soucier de l'intégration de celle-ci dans notre composant. En revanche, si on reprend par exemple le cas de l'architecture du paragraphe 8.1, il faut intégrer nous même une stack LoRaWAN. Voici ci-dessous une courte description de 3 stacks à disposition :

- SEMTECH propose sa propre stack appelée "LoRa MAC Node" pour l'ensemble des microcontrôleurs. En savoir plus ici : <https://lora-developers.semtech.com/resources/tools/>
- ST propose sa propre stack, qui reprend la "LoRa MAC Node" de SEMTECH, mais qui a été améliorée pour mieux correspondre aux spécificités des microcontrôleurs STM32.
- La stack "LMIC" est celle est utilisée lorsqu'on utilise des arduino : <https://github.com/matthijskooijman/arduino-lmic>

## 9 La création de notre propre Network et Application Server

⚠ Ce chapitre est en cours de rédaction et ne doit donc pas être pris en compte. Il sera mis en ligne le 1<sup>er</sup> octobre 2020.

### 9.1 Les stacks disponibles

Lors de la mise en œuvre du réseau LoRaWAN au chapitre 5.2, nous avons utilisé TTN (The Things Network) pour jouer le rôle de Network Server et d'Application Server. Pour de multiples raisons (sécurité, autonomie, coût...), il peut être intéressant de monter soi-même un Network Server et un Application Server. Cela est possible seulement si vous possédez vos propres Gateway puisque celles-ci devront forwarder leurs paquets à une destination précise. Il faudra donc les reconfigurer pour qu'elles pointent vers votre nouvelle architecture.

Nous verrons dans ce document l'installation d'un Network Server et d'un Application Server appelé ChirpStack [ [www.chirpstack.io](http://www.chirpstack.io) ]. Dans le cours proposé en vidéo [ <https://cutt.ly/lorawan> ] vous verrez aussi l'installation de "The Things Stack" qui est le serveur LoRaWAN de The Things Network [ <https://thethingsstack.io> ].

La configuration que nous avons utilisée jusqu'ici était donc un **réseau dédié hybride** (voir chapitre 5.1.4) et les explications suivantes nous permettront de réaliser un **réseau privé** (voir chapitre 5.1.2)

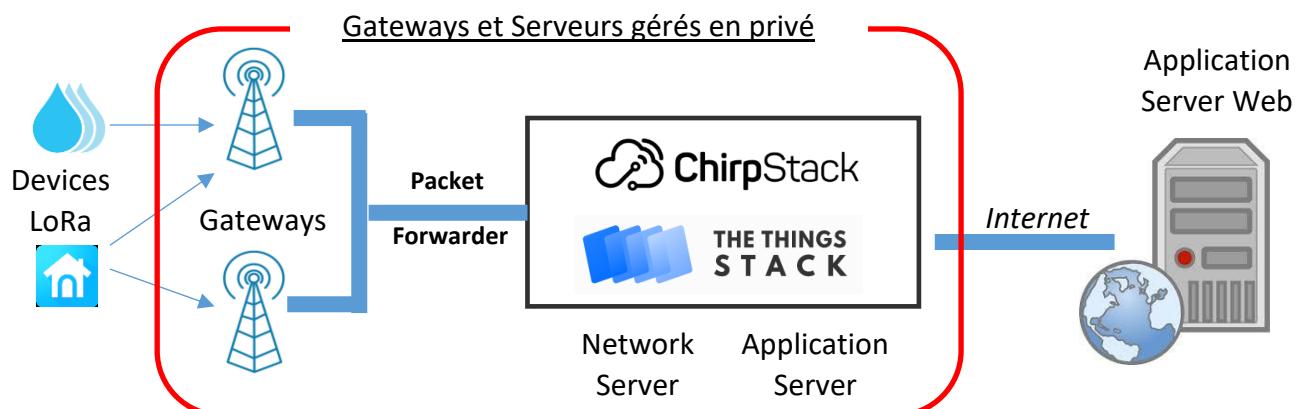


Figure 107 : Infrastructure d'un réseau LoRaWAN privé avec ChirpStack ou The Things Stack

### 9.2 Présentation de ChirpStack

#### 9.2.1 Le projet ChirpStack

ChirpStack est un projet Open-Source répondant exactement au besoin que nous avons. Nous l'installerons sur un serveur qui devra être joignable par les Gateways. Dans certaines configurations, notamment en phase de prototypage, ChirpStack peut être mis en place dans le même système que la Gateway, mais dans une configuration normale, les Gateways et les serveurs

LoRaWAN sont deux entités distinctes et distantes. Pour des raisons pédagogiques, il est donc important de séparer les systèmes pour bien différencier les problématiques.

L'architecture et le fonctionnement de ChirpStack est présentée dans sa documentation et est simplifié sur la Figure 108 :

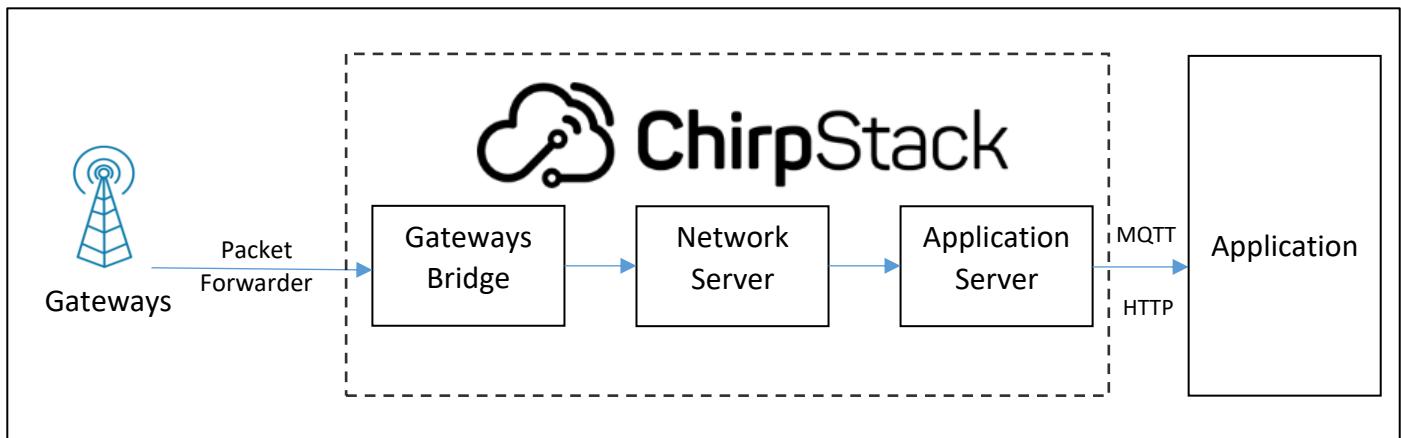


Figure 108 : Architecture détaillée de ChirpStack

ChirpStack est constitué d'un **Network Server** et d'un **Application Server** dont nous avons déjà détaillé les rôles. Il est aussi constitué d'une entité appelée **Gateway Bridge**.

La communication entre la Gateway et le serveur LoRaWAN doit être normalisée afin que les deux entités se comprennent. Le protocole le plus connu et le plus utilisé à ce jour est appelé "UDP Packet Forwarder" et a été développé par Semtech.

### 9.2.2 LoRa Gateway Bridge

Nous avons décrit au chapitre précédent (9.4.1) le fonctionnement du Packet Forwarder. Le Network Server(Lora Server) que nous allons mettre en place aurait très bien pu s'interfacer directement à ce protocole. Mais cela aurait plusieurs impacts dont un qui est important dans la conception de logiciel : Si le Packet Forwarder change, ou si un autre type de Forwarder est utilisé, alors l'utilisation du LoRa Server devient impossible. Il a donc été choisi de créer une étape intermédiaire. Le Network Server que nous mettrons en place utilisera le protocole MQTT plutôt que le protocole UDP packet Forwarder.

En d'autres termes, le LoRa Gateway Bridge est un service qui fera abstraction du Packet Forwarder pour s'interfacer plus simplement avec le Network Server.

### 9.2.3 LoRa Server (Network Server)

Il s'agit de notre Network Server qui a le même rôle que celui que nous avons étudié au chapitre 4.2.3. Cependant, il s'interfacerà au protocole MQTT en recevant les informations à partir du Broker au lieu de recevoir directement les trames en provenance de la Gateway.

### 9.2.4 LoRa App Server (Application Server)

Il s'agit de notre Application Server qui a le même rôle que celui que nous avons étudié au chapitre 4.2.4.

### 9.2.5 Application

Il s'agit de notre Application qui a le même rôle que celui que nous avons étudié au chapitre 4.2.5. Elle ne fait pas partie du projet LoRaServer, c'est donc bien toujours à nous de proposer notre propre Application.

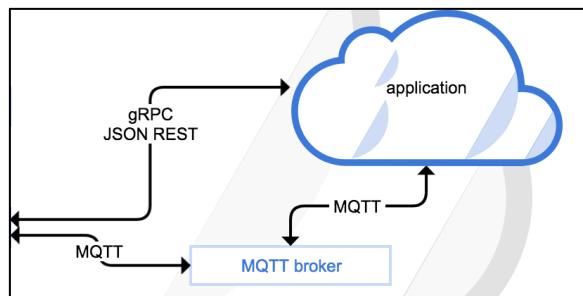


Figure 109 : Les interfaces possibles entre LoRaServer et l'Application utilisateur

Comme nous pouvons le voir sur la figure précédente, deux possibilités sont offertes pour interfaçer notre application : JSON REST (incluant HTTP POST) et via un Broker MQTT. Ces deux méthodes ont déjà été traitée au chapitre 7.3 (HTTP POST ) et au chapitre 7.4 (MQTT). Nous verrons aussi d'autres possibilités pour personnaliser notre application au chapitre 8.

## 9.3 Installation de LoRaServer

### 9.3.1 Mise en place de l'environnement

Nous allons installer une carte SD avec la dernière version de Raspbian. Nous ne détaillerons pas la démarche pour cela car elles sont largement documentées et mises à jour sur le site [www.raspberrypi.org](http://www.raspberrypi.org).

- ➔ Installer une carte SD avec la dernière version de Raspbian

Plutôt que de fonctionner avec un clavier et une souris, nous préconisons de travailler à distance avec la RPI via une connexion SSH. Depuis quelques années Raspbian a supprimé le démarrage du service SSH au boot. Heureusement il est possible de le réactiver par la méthode suivante :

- ➔ Avec Windows ou Linux, ouvrir la partition BOOT de votre carte SD et placer un fichier à la racine nommé ssh.txt (ou ssh).
- ➔ Mettre la carte SD dans votre RPI et la démarrer

➊ Pour accéder à la RPI en SSH depuis Windows, nous utiliserons de préférence le logiciel client : MobaXterm, mais tout autre client SSH est valable.

- ➔ Installer MobaXterm sur votre PC et connectez-vous à votre Raspberry PI.

L'utilisation d'un espion de réseau type Wireshark peut être intéressant pour étudier les trames qui sont reçues et qui sont émises. Cela nous permettra de valider le fonctionnement des différents protocoles réseau qui permettent de communiquer à la Gateway d'une part, et à l'Application d'autre part. Nous utiliserons tshark.

- apt-get install tshark // installation
- tshark -n -i eth0 // Exemple d'une analyse sur l'interface eth0

### 9.3.2 Installation sur la Raspberry PI

La méthode d'installation de LoRaServer est documentée sur le site web à l'adresse suivante : [ <https://www.loraserver.io/guides/debian-ubuntu/> ]. Nous installerons le LoRa Gateway Bridge, le LoRa Server et le LoRa App Server.

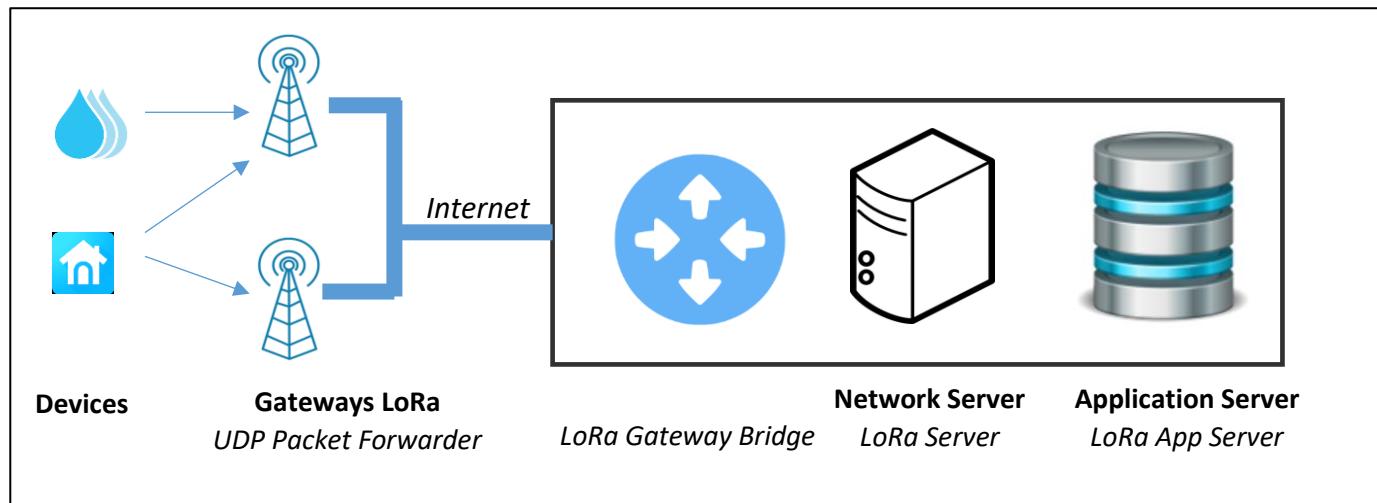


Figure 110 : Architecture globale après installation de LoRaServer

Après l'installation, la configuration se fait par une interface graphique accessible par le réseau sur le port 8080.

- Si vous travaillez sur la RPI, connectez-vous sur : <http://localhost:8080/>
- Si vous travaillez à distance (SSH), connectez-vous sur : [http://@IP\\_RPI:8080](http://@IP_RPI:8080)

Les Usernames et Password par défaut sont :

- Username: admin
- Password: admin

L'écran d'accueil sera donc le suivant :

Écran d'accueil de LoRaServer après authentification. L'interface est divisée en deux panneaux. Le panneau de gauche affiche une liste de liens : Network-servers, Gateway-profiles, Organizations, All users et Select... . Le panneau de droite affiche une liste d'applications avec des colonnes pour ID, Name, Service-profile et Description. Un bouton '+ CREATE' est visible dans le coin supérieur droit du tableau d'applications. En haut de l'écran, il y a un menu avec un logo, une barre de recherche et un bouton 'admin'.

Figure 111 : Ecran d'accueil de LoRaServer après authentification

## 9.4 Configuration de LoRa Server pour l'Uplink

### 9.4.1 Enregistrement d'une nouvelle Organisation

Pour enregistrer une nouvelle Organizations : **Organizations > Create**. Puis entrer les configurations suivantes :

Organization name \*

myOrganization

The name may only contain words, numbers and dashes.

Display name \*

myOrganization

Gateways

Organization can have gateways

When checked, it means that organization administrators are able to add their own gateways to the network. Note that the usage of the gateways is not limited to this organization.

**CREATE ORGANIZATION**

Figure 112 : Enregistrement d'un nouvelle Organisation

### 9.4.2 Enregistrement d'une instance du Network Server

Pour enregistrer une nouvelle instance du Network Server : **Network Server > Create**. Puis entrer les configurations suivantes :

GENERAL   GATEWAY DISCOVERY   TLS CERTIFICATES

Network-server name \*

myNetworkServer

A name to identify the network-server.

Network-server server \*

localhost:8000

The 'hostname:port' of the network-server, e.g. 'localhost:8000'.

**UPDATE NETWORK-SERVER**

Figure 113 : Enregistrement d'un nouveau Network Server

Dans l'onglet Gateway Discovery, il est possible d'activer une option de découverte de Gateway aux alentours. Lorsqu'elle est configurée, le Network Server va envoyer une trame Downlink de configuration de la Gateway lui ordonnant de réaliser des PING en LoRa. La configuration du Gateway Discovery prévoit de spécifier :

- Le nombre de fois par jour que les PING LoRa sont envoyés
- Le canal d'émission
- Le Data Rate (Voir paragraphe 4.6)

Un PING LoRa reçu sur une Gateway sera retransmis sur le Network Server (Lora Server) et une carte sera affichée.

L'onglet TLS Certificates ne sera pas utilisé dans le cadre de notre démonstration.

#### 9.4.3 Enregistrement d'une Application (Sur l'Application Server)

Il faut tout d'abord faire un enregistrement d'un « service-profile ». Un « service profile permet de faire la connexion entre le Network Server et les Applications que nous enregistrerons dans notre Organisation : **Service-profiles > Create**.

- Service-profile name : **myServiceProfile**
- Network-server : **myNetworkServer**

On laissera par défaut toutes les autres options.

Pour enregistrer une nouvelle Application : **Applications > Create**. Puis entrer les configurations suivantes

The screenshot shows a form for creating a new application. The fields are as follows:

- Application name \***: myApplication
- Application description \***: myApplication
- Service-profile \***: myServiceProfile
- Payload codec**: None

Below the form, there is a note: "By defining a payload codec, LoRa App Server can encode and decode the binary device payload for you." At the bottom right of the form is a blue button labeled "CREATE APPLICATION".

Figure 114 : Enregistrement d'une nouvelle Application

#### 9.4.4 Enregistrement des Devices LoRa

Il faut tout d'abord faire l'enregistrement d'un Device-profile pour les Devices LoRa que vous souhaitez connecter. Dans notre cas, nous ferons un premier test en spécifiant que nous utilisons seulement le mode d'authentification ABP (voir chapitre 4.4.1).: **Devices profile > Create**

GENERAL JOIN (OTAA / ABP) CLASS-B CLASS-C

Device-profile name \*  
myDeviceProfile  
A name to identify the device-profile.

LoRaWAN MAC version \*  
1.0.0  
The LoRaWAN MAC version supported by the device.

LoRaWAN Regional Parameters revision \*  
A  
Revision of the Regional Parameters specification supported by the device.

Max EIRP \*  
20  
Maximum EIRP supported by the device.

**UPDATE DEVICE-PROFILE**

Figure 115 : Enregistrement d'un « Device profile »

Nous pouvons alors créer dans notre Application des nouveaux Devices : **Application > myApplication > Create**

Device name \*  
arduino0  
The name may only contain words, numbers and dashes.

Device description \*  
arduino

Device EUU \*  
00 56 AB 22 5B F4 98 77 MSB C

Device-profile \*  
myDeviceProfile

Disable frame-counter validation  
Note that disabling the frame-counter validation will compromise security as it enables people to perform replay-attacks.

**CREATE DEVICE**

Figure 116 : Enregistrement d'un nouveau Device LoRa

Dans la fenêtre Activation, on va alors configurer les 3 éléments indispensables à une authentification en ABP : Le **DevAddr**, le **NwkSKey**, et l'**AppSKey**.

CONFIGURATION KEYS (OTAA) ACTIVATION LIVE DEVICE DATA LIVE LORAWAN FRAMES

Device address \*  
26 01 1a d3 MSB C

Network session key (LoRaWAN 1.0) \*  
.....

Application session key (LoRaWAN 1.0) \*  
.....

Uplink frame-counter \*  
0

Downlink frame-counter (network) \*  
0

**(RE)ACTIVATE DEVICE**

Figure 117 : Configuration de l'enregistrement du Device LoRa (en ABP)

La configuration minimale de LoRaServer est maintenant terminée. Nous pouvons donc brancher un Device LoRa qui possède les attributs (DevAddr, NwkSKey et AppSKey) que nous avons enregistré dans LoRaServer et le faire émettre.

#### 9.4.5 Visualisation des trames reçues

Les trames émises par le Device LoRa vont donc parcourir le cheminement suivant :

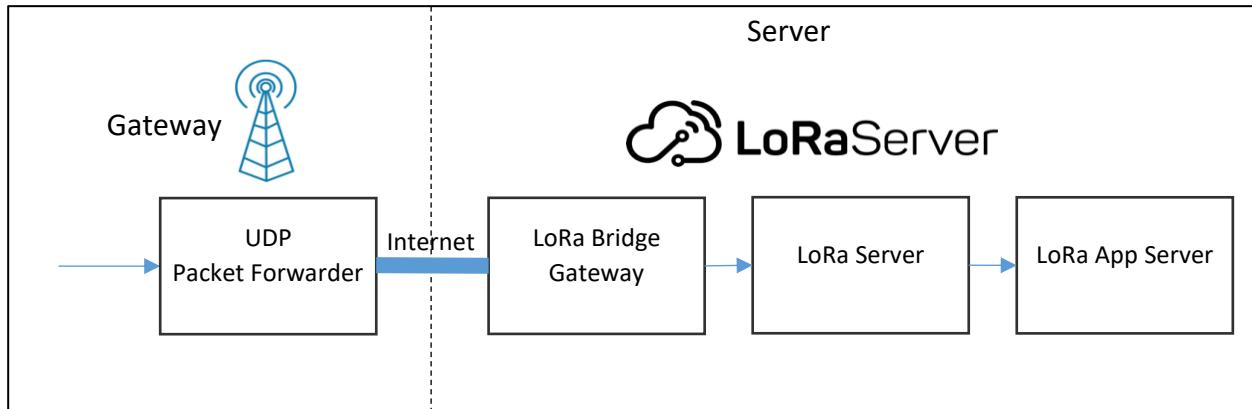


Figure 118 : Résumé du cheminement des trames LoRaWan en Uplink avec LoRaServer

En allant dans le Live Device Data (Application > Nom\_du\_Device > Live Device Data) on peut voir la liste des trames reçues ainsi que les données applicatives déchiffrées.

Time	Type
2:56:16 PM	uplink
2:56:07 PM	uplink
2:55:56 PM	uplink
2:55:46 PM	uplink
2:55:45 PM	uplink

Figure 119 : Réception des trames des Devices LoRa

```

{
  "adr": false,
  "applicationID": "3",
  "applicationName": "myApplication",
  "data": "SGVsbG8=",
  "devEUI": "0056ab225bf49877",
  "deviceName": "arduino0",
  "fCnt": 10262,
  "fPort": 15,
  "txInfo": {
    "dr": 5,
    "frequency": 868100000
  }
}
  
```

Figure 120 : Analyse des trames des Devices LoRa

Dans l'onglet LIVE LORAWAN FRAME, nous pouvons voir les trames LORAWAN, et si besoin étudier le contenu de ce qu'elles contiennent. En revanche, les données applicatives sont ici chiffrées.

CONFIGURATION	KEYS (OTAA)	ACTIVATION	LIVE DEVICE DATA	LIVE LORAWAN FRAMES
UPLINK	2:54:34 PM	UnconfirmedDataUp	26011ad3	▼
UPLINK	2:54:24 PM	UnconfirmedDataUp	26011ad3	▼
UPLINK	2:54:13 PM	UnconfirmedDataUp	26011ad3	▼
UPLINK	2:54:03 PM	UnconfirmedDataUp	26011ad3	▼

Figure 121 : Réception des Trames LoRaWAN

#### 9.4.1 UDP "Packet Forwarder"

Comme nous l'avons vu au chapitre 4.2.2, les Gateways LoRa sont des passerelles entre la modulation LoRa et un réseau IP. Sur le réseau IP, le protocole utilisé pour communiquer avec les serveurs LoRaWAN est appelé "UDP Packet Forwarder". Il a été développé par Semtech : [https://github.com/Lora-net/packet\\_forwarder](https://github.com/Lora-net/packet_forwarder). Vous trouverez dans ce dossier github, un fichier nommé PROTOCOL.TXT qui explique parfaitement ce protocole qui travaille au-dessus de UDP.

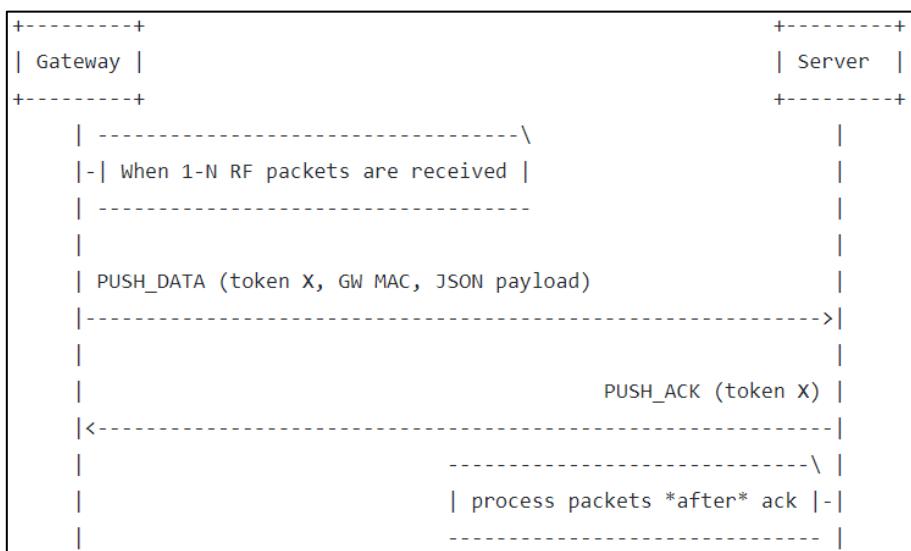


Figure 122 : Protocole Uplink (PUSH\_DATA du Packet Forwarder)

On remarque que les paquets sont envoyés en UDP au Network Server dans une trame appelée PUSH\_DATA. Cette trame est acquittée par le Network Server par une trame appelée PUSH\_ACK.

Sur notre Network Server, nous pouvons réaliser une capture de trame pour vérifier si le protocole est bien celui présenté dans le document :

```

Ethernet II, Src: Raspberry_ae:26:f5 (b8:27:eb:ae:26:f5), Dst: Raspberry_5b:ce:07 (b8:27:eb:5b:ce:07)
Internet Protocol Version 4, Src: 192.168.138.151, Dst: 192.168.138.168
User Datagram Protocol Src Port: 39579 Dst Port: 1700
Data (197 bytes)
Data: 02f93000b827ebffffae26f57b227278706b223a5b7b2274...
[Length: 197]
  
```

Figure 123 : Trame capturée par Wireshark lors d'un Uplink

D'après la capture précédente, on remarque que le Packet Forwarder fonctionne bien avec UDP sur le port 1700.

Le contenu des données (champ Data) est détaillé dans le tableau suivant :

Champ	Num octet	Fonction
[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 }

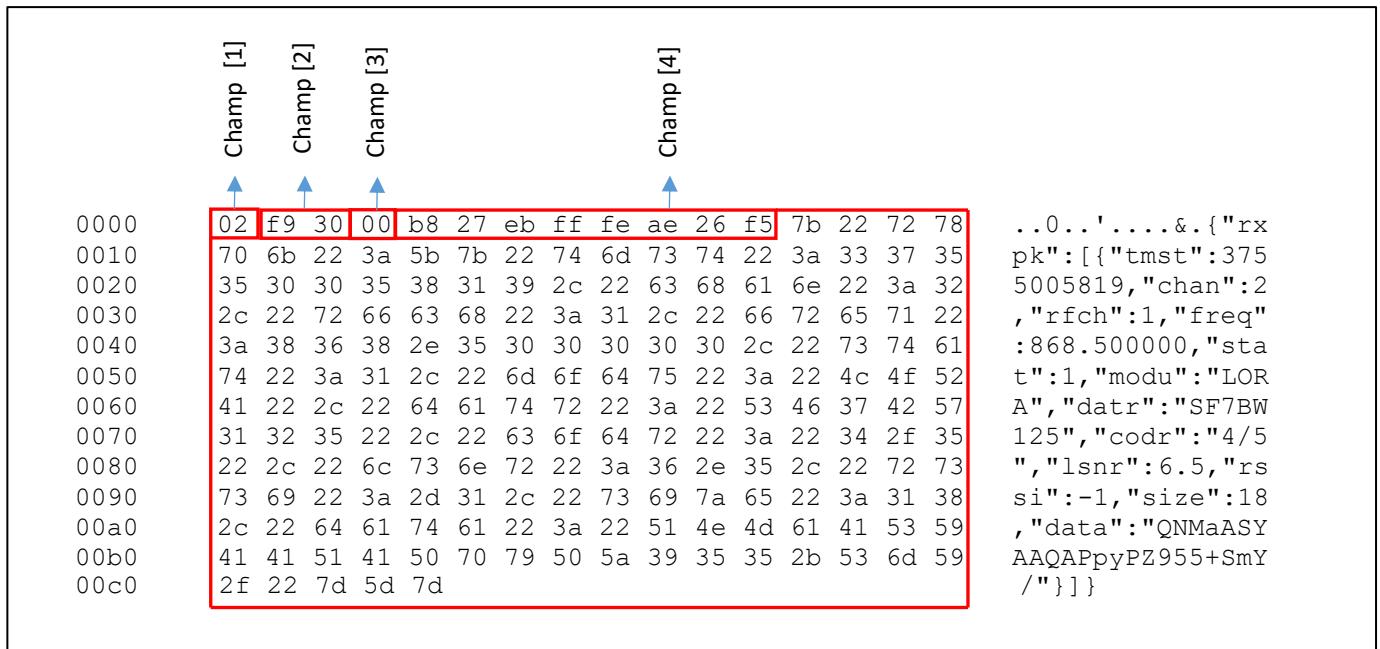


Figure 124 : Analyse du champ Données du protocole « Packet Forwarder »

L'objet JSON de la transmission réécrit plus proprement est celui-ci :

```
{
  "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": "QNMaASYAAQPyPZ955+SmY"
    }
  ]
}
```

Le champ "data" correspond au PHY Payload.

De la même façon on retrouve la Trame d'acquittement :

```

> Ethernet II, Src: Raspberry_5b:ce:07 (b8:27:eb:5b:ce:07), Dst: Raspberry_ae:26:f5 (b8:27:eb:ae:26:f5)
> Internet Protocol Version 4, Src: 192.168.138.168, Dst: 192.168.138.151
-> User Datagram Protocol, Src Port: 1700, Dst Port: 39579
  Data (4 bytes)
    Data: 02f93001
    [Length: 4]

```

Figure 125 : Trame capturée par Wireshark lors de l'acquittement d'un Uplink

Champs	Num octet	Fonction
[1]	0	protocol version = 0x02
[2]	1-2	same token as the PUSH_DATA to acknowledge
[3]	3	PUSH_ACK identifier = 0x01

Nous retrouvons bien tous ces champs dans la trame Wireshark.

## 9.5 Configuration de LoRaServer pour l'intégration d'Application

Nous avons vu dans le chapitre 9.2.5 que l'interface entre LoRaServer et notre Application pouvait être réalisée soit par MQTT, soit par les méthodes REST (HTTP POST). Nous avons déjà implémenté et expliqué ces protocoles dans le cadre de son utilisation avec TTN. Nous allons donc juste configurer LoRaServer pour tester ces deux méthodes.

### 9.5.1 Récupérer des données sur notre Application avec HTTP POST

Nous utiliserons exactement la même méthode que celle que nous avons utilisé avec TTN au chapitre **Erreur ! Source du renvoi introuvable..**. Les différents Clients/Serveurs disponibles pour ces tests sont expliqués au paragraphe **Erreur ! Source du renvoi introuvable..**.

Dans LoRaServer, ajouter une intégration HTTP : **LoRaServer > Applications > myApplication > Integrations > Create > HTTP Integration** et la configurer comme le montre la Figure 126. Vous remplacerez évidemment le lien par votre propre Endpoints.



Figure 126 : Configuration de l'intégration HTTP POST dans LoRaServer

En envoyant des trames LoRa depuis votre Device, vous devriez voir les contenus JSON sur votre Endpoint.

### 9.5.2 Récupérer des données sur notre Application avec MQTT

Nous allons nous connecter au Broker MQTT de LoRaServer. Pour cela nous utiliserons la même méthode que celle que nous avons utilisé avec TTN au chapitre 7.4.8 . Nous utiliserons MQTTBox, comme nous l'avons déjà fait au paragraphe 7.4.7.

Le Broker de LoRaServer a enregistré les Topics suivant :

- **[applicationID]** correspond au nom de votre Application
- **[devEUI]** correspond au nom de votre Device LoRa

Détail du Topic	Nom du Topic
[Données] Flux Uplink	application/[applicationID]/device/[devEUI]/rx
[Données] Flux Downlink	application/[applicationID]/device/[devEUI]/tx
[Status] Statut d'un Device	application/[applicationID]/device/[devEUI]/status
[Ack] Acquittement des messages	application/[applicationID]/device/[devEUI]/ack
[Erreurs] Erreurs	application/[applicationID]/device/[devEUI]/error

Tableau 20 : Topics enregistrés dans LoRaServer

## 10 La création de notre propre Application

---

Dans le chapitre 7, nous avons étudié comment récupérer les données de notre Application Server (Uplink) en provenance du Device LoRa. Nous avons aussi vu comment fournir des données à l'Application Server (Downlink) à destination du Device LoRa. Nous allons maintenant nous intéresser à l'Application utilisateur globale et non plus simplement à la récupération des données. A la fin de ce chapitre, nous aurons implémenté une architecture fonctionnelle complète pour recevoir, stocker, traiter, et afficher ces données sur un serveur Web. Le Tableau 21 ressemble différents choix technologiques pour réaliser le prototypage de cette Application utilisateur. La liste des solutions est très loin d'être exhaustives mais le tableau permet de comprendre les différentes fonctionnalités qui devront être réalisées.

Nous voyons que dans ce tableau, certains choix technologiques rassemblent plusieurs couches. C'est le cas par exemple de Node-RED qui est capable de tout réaliser en une seule application. A savoir :

Dans le sens Uplink :

- De gérer la récupération des données
- De stocker et traiter les données
- De les mettre en forme par un système de monitoring (graphiques, tableaux, ...)
- De les mettre à disposition de l'utilisateur avec un Serveur Web

Dans le sens Downlink :

- Présenter un interface utilisateur (Bouton, champ texte, ...)
- Traiter les commandes de l'utilisateur
- Envoyer ces commandes à l'Application Server de TTN

Service à rendre	Solutions génériques	Choix technologiques possibles				
<i>Service web :</i> <i>Mise à disposition de contenu WEB</i>	<i>Serveur WEB</i> <i>Interface utilisateur</i>		 Grafana	 kibana	 NGINX	 THE APACHE SOFTWARE FOUNDATION
<i>Gestion de l'affichage des données :</i> <i>Courbes, histogrammes, tableaux, jauge, etc...</i>	<i>Solution de monitoring complète, librairies pour Dashboard...</i>		 chronograf		 plotly	 Dash
<i>Sauvegarde des données</i>	<i>Base de données</i> <i>Fichier texte</i>		 influxdb	 elasticsearch	 SQLite	 MySQL
<i>Récupération des données</i>	<i>Endpoint HTTP</i> <i>Subscriber ou Publisher MQTT</i>		 telegraf	 logstash	 python	 HTTP / MQTT

Tableau 21 : Choix technologiques disponibles pour la réalisation de l'Application utilisateur

## 10.1 Mise en place de l'environnement de travail

### 10.1.1 Installation de docker et docker-compose

Une grande partie des difficultés à réaliser les applications est l'installation de tous les logiciels, services, clients, etc... L'objectif de ce cours est donc de simplifier au maximum ce processus et de le rendre le plus possible indépendant des autres applications en cours sur votre système. Nous allons donc utiliser **docker** et **docker-compose** pour la mise en place de l'environnement de travail en quelques commandes. Cela vous permettra d'installer en une seule fois l'ensemble des services convenablement configurés pour toutes les manipulations à venir. Le travail fourni est uniquement réalisable sur une RPI (Linux / ARM64) ou PC (Linux / AMD64). Voici l'ensemble des services qui seront disponibles sous forme de conteneur docker :

- Node-RED (tcp 1880)
- InfluxDB (tcp 8680)
- Grafana (tcp 3000)
- Telegraf

Chaque conteneur est indépendant et peut être stoppé, en revanche, ils sont capables de communiquer entre eux grâce à des noms (adresse IP) comme le montre la Figure 127.

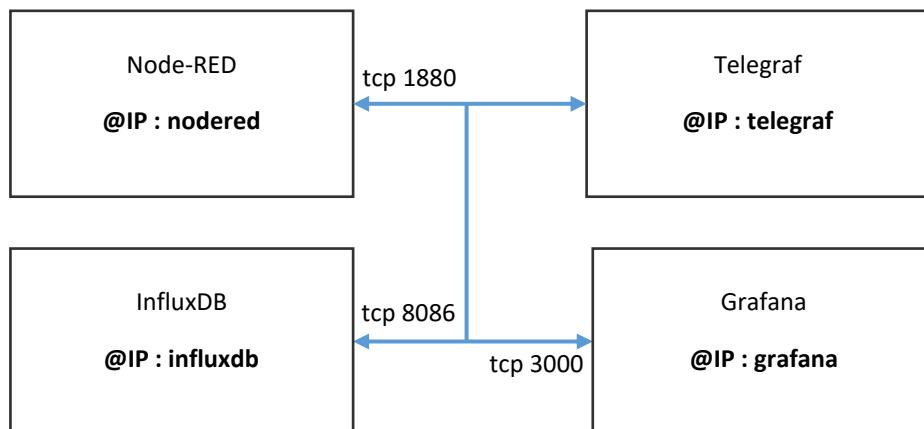


Figure 127 : Installation des services pour l'application

L'installation de docker et docker-compose se réalise de la façon suivante :

- ➔ Installer **docker** : <https://docs.docker.com/engine/install/>
- ➔ Installer **docker-compose** sur Raspberry PI (Raspbian): <https://bit.ly/3be8NqB>
- ➔ Installer **docker-compose** sur Linux : <https://docs.docker.com/compose/install/>

Nous devons maintenant récupérer les scripts de création des containers sur GitHub.

Télécharger <https://github.com/SylvainMontagny/lora-lorawan/archive/master.zip>

- ➔ Décompresser l'archive : **unzip master.zip**
- ➔ Placer vous dans le répertoire **RPI\_standalone** pour une installation sur RPI : **cd RPI\_standalone**.
- ➔ Lancer services Node-RED, Grafana, Telegraf, et InfluxDB : **docker-compose up -d**

Pour vérifier que les services sont actifs, vous devez pouvoir faire les tests suivants :

- ➔ Dans votre navigateur web, le lien [http://@IP\\_Raspeberry\\_PI:1880/](http://@IP_Raspeberry_PI:1880/) doit ouvrir node-RED.
- ➔ Dans votre navigateur web, le lien [http://@IP\\_Raspeberry\\_PI:3000/](http://@IP_Raspeberry_PI:3000/) doit ouvrir grafana.

Vérifier que les services soient actifs Une fois l'installation terminée, vous pouvez joindre voici la présentation de votre environnement de travail.

### 10.1.2 Cas d'étude

**TODO : Présentation Schéma des services en cours.**

Nous nous placerons dans un cas simple qui est l'envoi d'une température sur un octet par un capteur. Nous souhaitons récupérer cette valeur, la stocker dans une BDD et afficher sa variation en fonction du temps.

## 10.2 Utilisation de Node-RED

### 10.2.1 Présentation

Node-RED est un outil de programmation graphique qui permet de faire communiquer les périphériques matériels d'un système sans écrire de code. De nombreux protocoles sont aussi pris en charge au travers de bloc (Node) qu'il suffit de connecter entre eux pour réaliser simplement une application. Il peut s'exécuter localement sur un PC ou sur des cibles embarquées telle qu'une Raspberry PI (RPI).

Nous nous connectons au node-RED de votre système via un navigateur web sur le port 1880, en tapant dans la barre d'URL : [http://@IP\\_Raspeberry\\_PI:1880/](http://@IP_Raspeberry_PI:1880/).

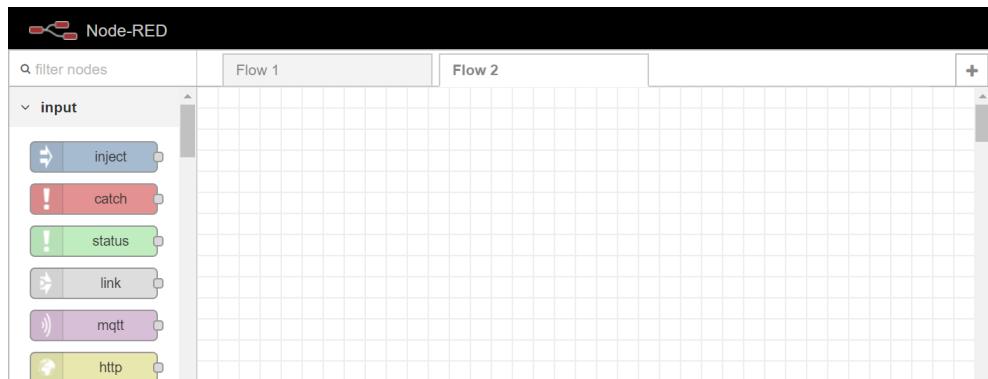


Figure 128 : Page d'accueil de Node-RED

En nous référant au Tableau 21, on peut lire que le choix de node-RED permet d'implémenter toutes les briques de notre application. Nous allons donc réaliser ces briques une par une avec la trame suivante :

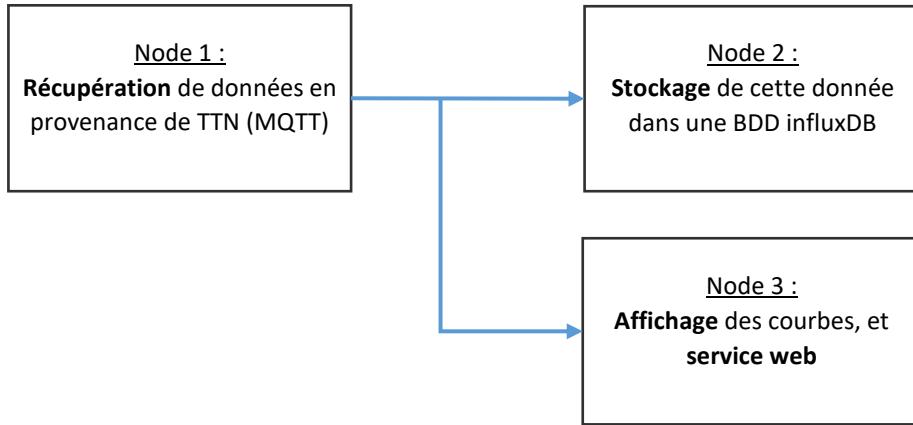


Figure 129 : Les briques du flow node-RED à réaliser

Les différentes étapes vont être expliquées une à une. Cependant, si vous souhaitez avoir l'application finale sans explication, il est possible de l'obtenir en important les nodes déjà configurés dans votre interface. Cette configuration se trouve dans le fichier flows.json dans le GitHub du cours présenté au début de ce document : <https://github.com/SylvainMontagny/lora-lorawan>. Pour importer le flow : **Menu > Import > Select a file to import**. La seule configuration à réaliser est celle du node "ttn Uplink" comme expliqué au paragraphe suivant.

### 10.2.2 Récupération des données avec le node TTN

Une librairie de node-RED fournie un node permettant de s'interfacer très facilement à TTN. Un node permet de gérer l'Uplink et un autre pour le Downlink.

Nous allons gérer le flux Uplink. Apportez sur votre schéma un node "ttn uplink" et un Node "Debug". Puis reliez-les. Le Node "ttn uplink" nous permettra de configurer la communication avec TTN **en MQTT**. Le Node "debug" écrira sur le terminal les informations brutes qui seront reçues par le Node "ttn uplink".

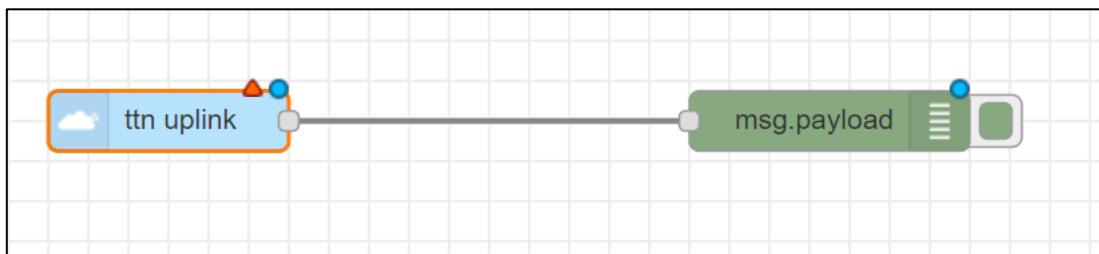


Figure 130 : Utilisation du Node TTN Uplink

Double cliquer sur le Node "ttn uplink" pour le configurer comme sur la Figure 131 :

Le champ "Name" sera le nom du node sur votre schéma. Le champ Device ID est le Device LoRa concerné dans votre application. De plus, il faut entrer les caractéristiques de votre application (Add new ttu app) pour que Node-RED puisse s'y connecter.

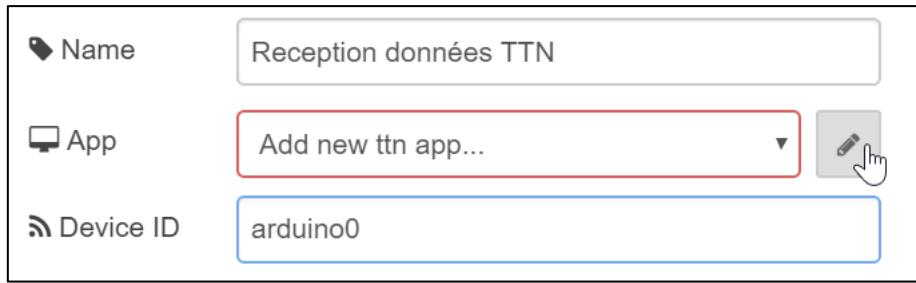


Figure 131 : Configuration du node "ttn uplink"

Configuration de votre nouvelle application :

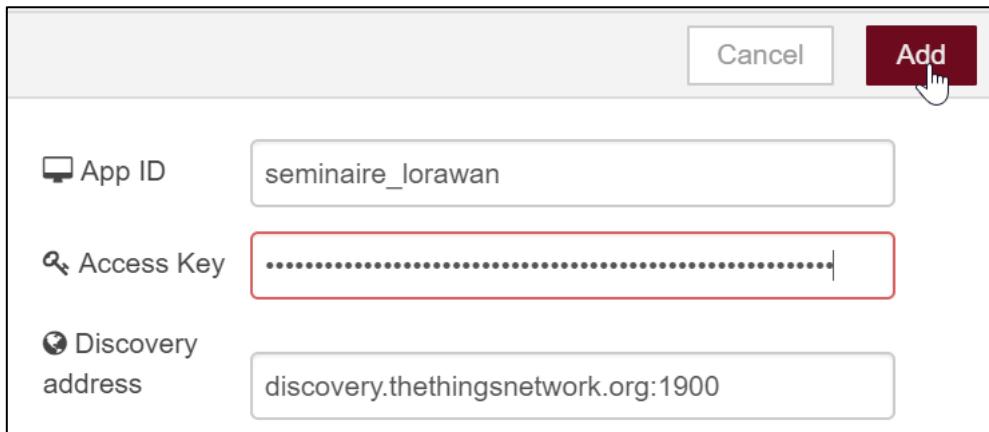


Figure 132 : Identifiant et mot de passe de notre application

- AppID : Nom de votre application. Dans notre cas "seminaire\_lorawan".
- Access Key : Comme nous l'avons déjà expliqué plus tôt, l' "Access Key" vous sera donné par l'enseignant dans le cadre de ce test. Si vous avez enregistré votre propre application, vous trouverez l'Access Key dans : **TTN > Application > Nom\_de\_votre\_application > Overview**.

Le node "ttn uplink" doit maintenant avoir le nom que vous avez configuré. Cliquer sur Deploy pour lancer votre projet. Le bloc "ttn uplink" doit maintenant être connecté à TTN. Cela est spécifié "connected" sous le bloc.

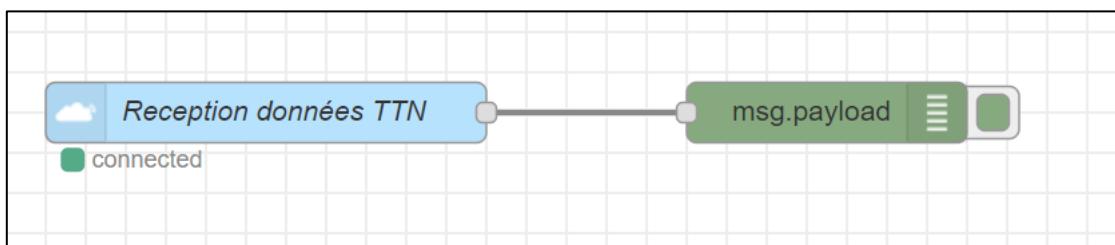


Figure 133 : Visualisation des trames publiées par TTN sur notre client MQTT

Vous pouvez alors vérifier que le message arrive bien dans la fenêtre de Debug à chaque réception d'une donnée en provenance du Device LoRa.

### 10.2.3 Modification du format du payload

Afin d'écrire dans la BDD, ou encore d'envoyer des données à notre Dashboard, il est nécessaire d'avoir un msg.payload représentant un nombre alors que le msg.payload reçu par TTN est un buffer (ou autre si vous avez utilisé le Decoder de TTN). La Figure 134 montre le format du msg.payload fourni par le node "TTN uplink" et le format du msg.payload que nous souhaitons avoir.



💡 Pour que node-RED montre l'ensemble des propriétés de l'objet dans la fenêtre de debug, il faut modifier les propriétés du node Debug : Output > Complete msg Object

```
▼ payload_raw: buffer[1] [raw]
  0: 0x3
▶ metadata: object
▼ payload: buffer[1] [raw]
  0: 0x3
```

```
▼ payload_raw: buffer[1] [raw]
  0: 0x3
▶ metadata: object
  payload: 3
```

Figure 134 : Format du msg.payload avant (gauche) et après transformation (droite)

On remarque que le msg.payload fourni par TTN est bien un buffer, alors que celui que nous souhaitons est un nombre. Nous devons donc réaliser l'affectation suivante :

```
▼ payload_raw: buffer[1] [raw]
  0: 0x3
▶ metadata: object
▼ payload: buffer[1] [raw]
  0: 0x3
```

```
▼ payload_raw: buffer[1] [raw]
  0: 0x3
▶ metadata: object
  payload: 3
```

Figure 135 : Transformation du format de msg.payload

Cette transformation peut être réalisée de plusieurs façon dans node-RED. La plus simple est d'utiliser le node "change" et de lui affecter la configuration suivante :



Figure 136 : Modification du msg.payload par un node "change"

La configuration ci-dessus dit : "Affecter la valeur de msg.payload, à la valeur de msg.payload\_raw[0]".



#### 10.2.4 Crédation d'un Dashboard

Nous allons maintenant réaliser une interface graphique très simple et la mettre à disposition des utilisateurs. Pour cela, il suffit de rajouter les nodes "chart" et "gauge" dans votre flow. La configuration par défaut du node "chart" et "gauge" fonctionne parfaitement. Vous pouvez néanmoins modifier les propriétés du node pour mettre des légendes, modifier les couleurs... etc.

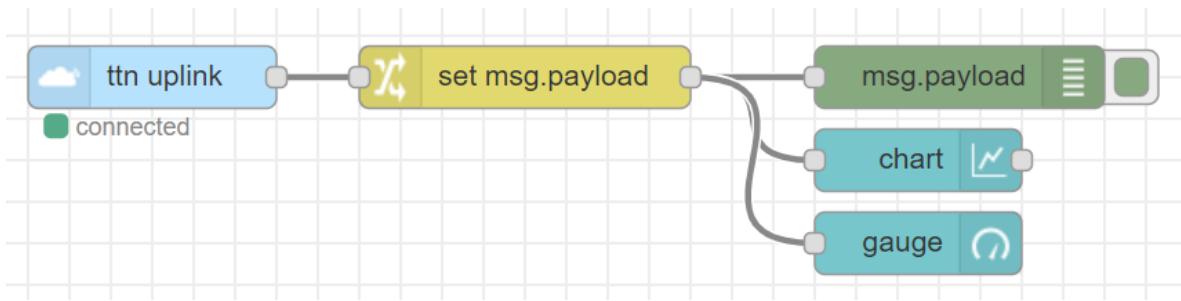


Figure 137 : Implémentation d'une interface graphique dans Node RED

L'URL pour joindre l'interface graphique de votre site web est disponible sur l'adresse [http://@IP\\_Serveur:1880/ui](http://@IP_Serveur:1880/ui).

Emettez une donnée en LoRa, ou faites une simulation d'Uplink (comme nous l'avons vu au paragraphe 5.2.5). L'interface graphique devrait se mettre à jour automatiquement.

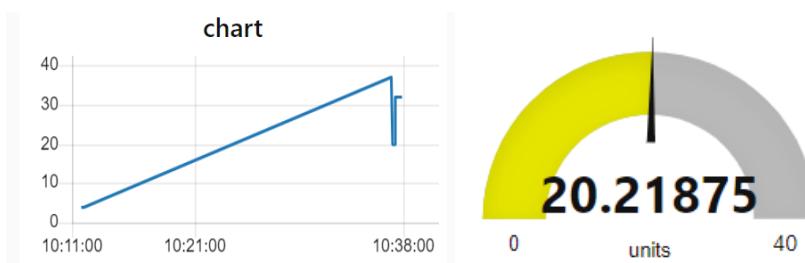


Figure 138 : Interface graphique du site web avec Node-RED

### 10.2.5 Stockage des informations dans une Base de données

Nous utiliserons la base de données InfluxDB. Une librairie de node-RED fournie un node permettant de s'interfacer très facilement à InfluxDB. Nous utiliserons principalement le node "influxdb out" pour écrire des données dans la BDD. La documentation des nodes influxdb est fournies par le lien suivant : <https://bit.ly/2Z1Uzqi>. Elle montre bien que si le msg.payload est un nombre, alors sa valeur sera enregistrée.

Les écritures dans la BDD se font grâce au protocole LINE dont la syntaxe est très bien expliquée ici : <https://bit.ly/3fB4LMu>. Dans notre cas, nous définirons un "measurement" appelé "**temperatures**" et les champs (fields) seront appelés "**values**".

La BDD est déjà installée d'après la mise en place de l'environnement de travail que nous avons vu au paragraphe 10.1, nous pouvons donc directement commencer à travailler. La BDD a été configurée avec une base appelée "maBase". Nous allons tout d'abord vérifier que nous pouvons bien nous connecter, et lire des valeurs dans cette BDD. Pour cela nous allons ouvrir un terminal dans le conteneur docker contenant la BDD influxDB en tapant la commande suivante dans le terminal de la machine contenant votre BDD :

```
docker exec -it influxdb /bin/bash
```

Nous sommes maintenant dans le conteneur docker influxdb. Les manipulations suivantes vont :

- Se connecter à la BDD influxDB [ influx ]
- Voir les bases présentes [ SHOW DATABASES ]
- Utiliser une base de donnée existante [ USE maBase ]

- Enregistrer une valeur dans un "measurement" appelée "temperatures" [ INSERT ]
- Voir les "measurements" présents [ SHOW MEASUREMENTS ]
- Lister les valeurs présentes dans le "measurement" temperatures [ SELECT ]

```
root@7bab25790b83:/# influx
Connected to http://localhost:8086 version 1.8.0
InfluxDB shell version: 1.8.0
> SHOW DATABASES
maBase
internal
> USE maBase
Using database maBase
> INSERT temperatures value=25
> SHOW MEASUREMENTS
name: measurements
temperatures
> SELECT * FROM temperatures
name: temperatures
time           value
----           -----
1589491733559187535 25
```

Nous allons maintenant placer les données reçues par TTN dans notre BDD. Il suffit de les donner à un "node influxdb out" comme le montre la Figure 139.

La configuration du node est la suivante :

- Host : influxdb
- Port : 8086
- Database : maBase
- Measurements : temperatures

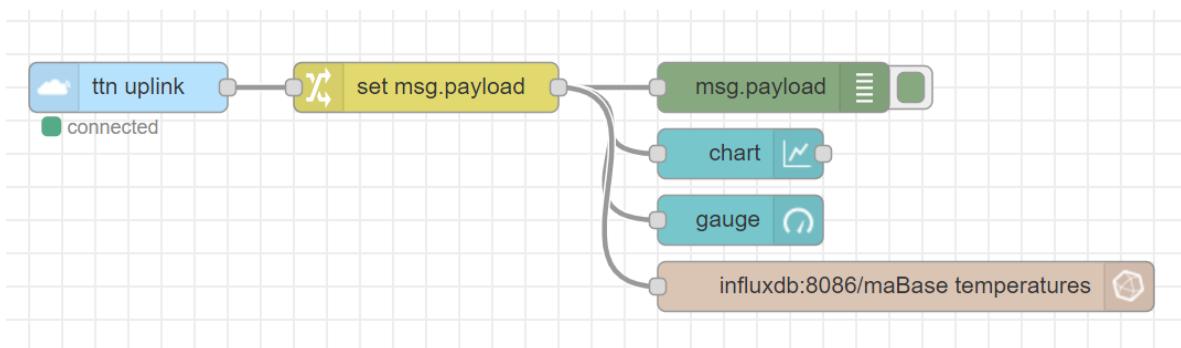


Figure 139 : Enregistrement des températures dans la BDD

On peut alors vérifier que les données sont bien stockées convenablement dans la BDD à chaque fois que TTN transfère une mise à jour de température.

## 10.3 Utilisation Telegraf, InfluxDB et Grafana

Comme le montre la Tableau 21, nous pouvons utiliser 3 services différents qui réaliseront chacun une brique de l'application. Chacun de ces services ont

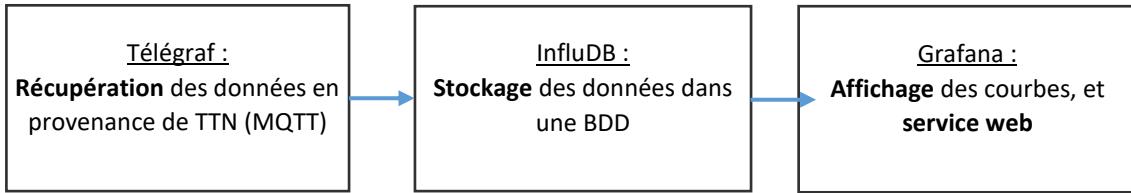


Figure 140 : Les briques d'une application basée sur Telegraf, InfluxDB et Grafana

Telegraf, InfluxDB et Grafana ont déjà été installés d'après la mise en place de l'environnement de travail que nous avons vu au paragraphe 10.1, nous pouvons donc directement commencer à travailler.

Information sur Telegraf : <https://www.influxdata.com/time-series-platform/telegraf/>

Information sur InfluxDB : <https://www.influxdata.com/time-series-platform/influxdb/>

Information sur Grafana : <https://grafana.com/>

### 10.3.1 Telegraf : Récupération des données

Telegraf est un agent qui permet de collecter des données depuis une multitude de sources (INPUT), et qui peut les retransmettre à une multitude de destination (OUTPUT). Le service Telegraf fonctionne à l'aide de plugins qui permettent la configuration de la source et de la destination.



Figure 141 : Fonctionnement de Telegraf

Nous devons donc choisir et configurer deux plugins, un pour la source, et un pour la destination.

### 10.3.2 InfluxDB : Sauvegarde dans une BDD

C'est Telegraf qui s'en ait chargé, nous avons donc déjà réalisé la sauvegarde des données. Visualisons cependant les types de données qui ont été sauvegardées. Pour cela nous nous connectons à la BDD de la même façon que nous l'avons vu en paragraphe 10.2.5.

Chapitre en cours de rédaction, mise à jour prévue pour le 1 octobre 2020

### 10.3.3 Grafana : Système de monitoring

Chapitre en cours de rédaction, mise à jour prévue pour le 1 octobre 2020

## 10.4 Workshop LoRa

```
docker-compose up -d --scale grafana=1 --scale nodered=1 --  
scale influxdb=0
```

Chapitre en cours de rédaction, mise à jour prévue pour le 1 octobre 2020

## 11 Versions du document

---

Version 1.0 : Février 2019

- **Version initiale.**

Version 2.0 : Février 2019

- **Ajout d'un paragraphe** : Les systèmes embarqués dans l'IoT / En introduction
- **Complément et amélioration** : Amélioration exercice étalement de spectre
- **Correction** : Câblage Arduino
- **Ajout d'un paragraphe** : Fonctionnement de l'Arduino et ajout de librairies.
- **Complément et amélioration** : Couche physique LoRa. Ajout de 4 figures pour l'explication de la modulation Chirp.
- **Ajout d'un paragraphe** sur les acronymes utilisés au début du cours : TTN, LoRa, MQTT...
- **Complément d'informations** sur les dB et dBm
- **Ajout LoRa Calculator**

Version 3.0 : Mars 2019

- **Ajout de figure** sur le rôle de la Gateway LoRa
- **Ajout de paragraphe** authentification avec le Network Server
- **Ajout de paragraphe** sur le chiffrement avec l'Application Server
- **Ajout d'un sommaire** en début de cours
- **Ajout de 2 schémas** : ABP et OTAA
- **Retrait explication** du détail de la « Join Request » en OTAA : Réalisation d'un schéma plus explicite (a faire).
- **Ajout d'un schéma** sur l'attaque par REPLAY
- **Ajout de paragraphe** des différents réseaux LoRaWAN existants : Opérés et Privés
- **Ajout des schémas** des trames LoRaWAN : Physique / LoRa MAC / Application
- **Complément d'information** sur les Classes de Devices
- **Ajout d'un paragraphe** sur la Gateway utilisée pour le Downlink
- **Ajout d'un paragraphe** sur le JSON
- **Complément d'information** sur le codage en Base64
- **Ajout d'information** dans le décodage des trames LoRaWAN
- **Restructuration des explications** sur l'envoi et la récupération des données en HTTP/MQTT

Version 4.0 : Mars 2019

- **Ajout d'un chapitre** : Création de notre propre Application
- **Ajout d'un chapitre** : Création de notre propre Network et Application Server
- **Ajout d'une figure** : Visualisation des Chirps LoRa par SDR (Radio Logicielle)
- **Compléments d'information** sur MQTT
- **Compléments d'information** sur les Topics de TTN en MQTT

Version 4.1 : Octobre 2019

- **Mise en page** : Suppression bordure de page.
- **Correction** : De très nombreuses corrections grâce à l'œil attentif de Alain Mouflet (Université de Rouen)

Version 4.2 : Février 2020

- **Modification** : Passage de l'arduino + RN2483 au module The Things UNO (Léonardo).
- **Complément d'information** : Des renvois sur les paraphes concernés ont été ajoutés.
- **Complément d'information** : Ajout du schéma de connexion en OTAA. Join-Request / Join-Accept.
- **Modification** du basket gérant le serveur HTTP POST : <https://rbaskets.in/lorawan> . Le Token pour s'y connecter est donné dans le fichier "Informations seminaire.txt".
- **Ajout** des champs JSON permettant de préciser le numéro de Port et la confirmation (ou non) et d'un message Downlink.

Version 5 : Juin 2020

- **Ajout** : Nouveaux schémas pour le partage du support en FDMA et CDMA
- **Amélioration** : De l'exemple sur le CDMA
- **Modification** : Figure 23 : Le rôle de la Gateway LoRa, la représentation en couche fait apparaître LoRaMAC pour être conforme à la norme et éviter les ambiguïtés.
- **Modification** : Figure 66 : Passerelle (Gateway) LORAWAN. La couche "Modulation LoRa" est laissée en couche basse.
- **Modification** : Création de schéma personnel pour les classes de Device (A, B, C).
- **Ajout** : Message d'attention suite au changement de nom de LoRaServer (maintenant ChirpStack Server). Mise à jour à faire.
- **Ajout** : Schéma de transmission radiofréquence. Exemple RSSI et SNR d'un signal reçu sur TTN.
- **Ajout** : Paragraphe sur le débit utile en LoRa en prenant en compte la trame LoRa.
- **Ajout** : Paragraphe sur le débit utile en LoRaWAN en prenant en compte la trame LoRaWAN.
- **Modification** : Le chapitre intitulé Time On Air est renommé en "Duty cycle"
- **Ajout** : Tableau 5 des Temps symboles et débit binaire en fonction du SF
- **Ajout** : Nouveau chapitre sur la création de notre propre Device LoRa : les différentes architectures, les stacks, les firmwares, les modules, les transceivers.
- **Modification** : Refonte complète du chapitre sur la création de notre application
- **Ajout** : Présentation du GitHub du cours : <https://github.com/SylvainMontagny/lora-lorawan>
- **Ajout** : Nouveau chapitre sur le choix entre OTAA et ABP

Version 6 : Juillet 2020

- Modification** : Le paragraphe "La Trame LoRa / LoRaWAN" est maintenant un chapitre entier.
- Modification** : Le chapitre "Mise en œuvre" d'un réseau LoRa est intégré au chapitre sur le LoRaWAN.
- Modification** : Séparation des paragraphes sur le DR, les canaux.
- Ajout** : Paragraphe sur le ADR (Adaptive Data Rate)

**Modification** : Nouveau chapitre "Les réseaux et serveurs LoRaWAN"

**Ajout** : Coût abonnement des réseaux LoRaWAN Bouygues et Orange (Juillet 2020)

**Ajout** : Tableau de choix entre réseau privé et opéré

**Ajout** : Les réseaux LoRaWAN dédiés

**Ajout** : Mise en ligne de la formation sur UDEMY

Version 7 : Septembre 2020

- **Udemy** : Ajout de Quizz en ligne sur chacune des fins de chapitre
- **Udemy** : Création des vidéos sur la récupération des données
- **Udemy** : Ajout du chapitre sur la récupération des données
- **Udemy** : Coupon de réduction 10% pour les vidéos aux lecteurs du livre
- **Modification** : Introduction plus complète sur le chapitre de la récupération des données
- **Ajout** : Récupération des données par HTTP GET
- **Modification** : Changement des schémas pour la présentation des protocoles MQTT et HTTP
- **Correction** : Contenu JSON du flux Downlink en MQTT corrigé
- **Modification en cours** : La création de notre propre Network Server et Application Server

Toutes remarques, modifications, améliorations, corrections peuvent être proposées par email :  
[sylvain.montagny@univ-smb.fr](mailto:sylvain.montagny@univ-smb.fr)

TODO : Encoder et Decoder de TTN: Mettre dans le chapitre de la récupération des données dans l'application

TODO LORA Alliance

TODO : Installation TTN en privé, aussi Chirpstack

TODO : Mettre en place l'application sur le web

Voir pour mettre des versions : alpine

TODO : Créer notre propre Gateway

Geolocalisation

Firmware Over The Air update (FOTA)

Multicast

TODO : A ton vraiment besoin de l'AppEUI dans le serveur pour faire fonctionner un device LoRa en OTAA. Dans chirpstack, on a pas besoin de la préciser dans le Network Serveur. Peut etre qu'il sert

juste à générer un AppEUI et donc qu'on est pas obligé de le citer dans le serveur.