Livre

Cette version PDF du livre est mise à disposition pour la communauté et peut être diffusée en l'état. Toutes remarques / modifications / améliorations / corrections peuvent être proposées à l'auteur via le lien suivant : https://cutt.ly/erreurs. L'auteur vous remercie par avance pour votre contribution.

Ce livre sur l'Internet des Objets et les protocoles LoRa / LoRaWAN est mis à jour régulièrement. Il fait partie d'une formation LoRa proposée par l'Université de Savoie Mont Blanc dans le cadre du Master Electronique Systèmes Embarqués et Télécommunications.

Formation LoRa-LoRaWAN en vidéo

Malgré le contenu complet de ce livre, une version plus interactive est disponible en vidéo sur une plateforme de E-Learning. La formation LoRa-LoRaWAN en vidéo est disponible sur UDEMY.

Cette formation est composée de 120 capsules vidéo d'une moyenne de 4mins. Elle reprend les chapitres du livre avec en plus:

- Des démonstrations.
- Des explications supplémentaires qu'il est plus opportun de faire en vidéo.
- Des détails sur les mises en ouvres et la configuration des Devices, Gateways et Serveurs.

Une heure de vidéo LoRa gratuite est disponible pour tous.

Vous pouvez poser vos questions sur la plateforme, les formateurs vous répondront.

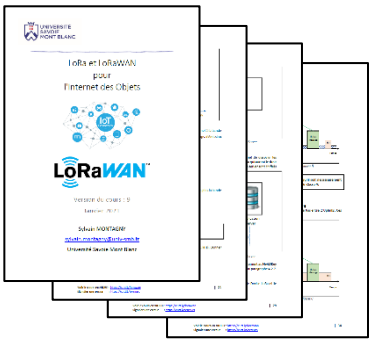
Formation LoRa avec matériel (100% en ligne)

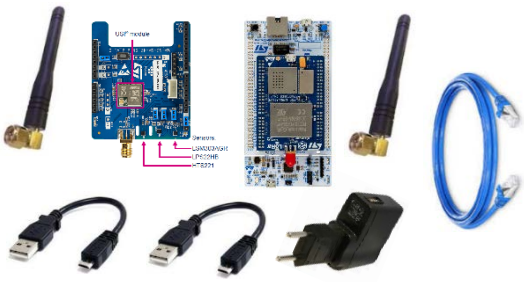
Enfin, vous pouvez faire cette formation LoRa – LoRaWAN sur 2 jours **en ligne**.

- Matériel fourni : Nous vous envoyons Device et Gateway afin de mettre en œuvre un réseau LoRaWAN. Vous conservez l'ensemble du matériel à l'issu de la formation.
- Effectif réduit : Nous souhaitons personnaliser au maximum le suivi afin de répondre au mieux à vos besoins (10 participants maximum)
- Suivi personnalisé : Nous sommes à votre disposition tout au long de la formation, et nous restons disponible à l'issu de celle-ci, pour répondre à vos éventuelles questions et besoins.
- Homologuée OPCO : La formation peut être prise en charge par vos organismes de formation professionnelle.

Avant la formation

- Accès à la plateforme pédagogique Udemy
- Envoi du matériel, du kit pédagogique et accès aux serveurs





Pendant la formation

- Accompagnement.
- Mise en œuvre pratique du LoRa.
- Test des paramètres de transmission (Bandwidth, canaux, Spreading Factor).
- Configuration de votre Gateway, utilisation d'un serveur LoRaWAN, création d'application et enregistrement de Devices.
- Test des différents modes d'activation (ABP et OTAA).
- Test et spécificité des Devices classe A, transfert en Uplink, Downlink, confirmé ou non.
- Test de l'Adaptive Data Rate (ADR).
- Récupération des données (HTTP et MQTT).
- Création de son propre Serveur LoRaWAN (TheThingsStack et Chirpstack).
- Création de sa propre Application avec Dashboard.

Après la formation

- Support pendant 7 jours.
- Utilisation des serveurs pendant 3 semaines.

Vos formateurs

Sylvain MONTAGNY

Je suis Professeur Agrégé à l'Université de Savoie Mont Blanc depuis 2006 et je me suis spécialisé dans les systèmes à microprocesseurs et l'internet des Objets. Je suis responsable du Master ESET (Electronique, Systèmes Embarqués et Télécoms) qui intègre principalement des étudiants en alternance proche du monde industriel. Le contexte actuel et les nouvelles tendances pédagogiques nous ont donné envie de proposer cette formation à distance. Nous relevons ce challenge et espérons que cette formation vous donnera pleinement satisfaction.

Florent Lorne

J'enseigne à l'Université de Savoie Mont Blanc depuis 2008 et je suis responsable du Master TRI (Télécoms et Réseaux Informatiques). En tant que Professeur Agrégé je me suis spécialisé dans l'architecture des réseaux et l'administration système. Je vous souhaite à tous la bienvenue dans cette formation.

Soutenue par ChipSelect

Cette formation est réalisée en partenariat avec ChipSelect, représentant de SEMTECH (France et Bénélux). La formation a été présentée lors du Webinaire (janvier 2021) et peut être vue en Replay.

Ressources

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

Légendes

Informations importantes

Exercices

A lire très attentivement

Sigles et Acronymes

ABP :	**A**ctivation **B**y **P**ersonalization
ADR :	**A**daptive **D**ata **R**ate
AppEUI :	**A**pplication **E**xtended **U**nique **I**dentifier
AppKey :	**A**pplication **K**ey
AppSKey :	**A**pplication **S**ession **K**ey
BDD :	**B**ase de **D**onnées
BW :	**B**and**W**idth
CDMA :	**C**ode **D**ivision **M**ultiple **A**ccess
CHIRP :	**C**ompressed **H**igh **I**ntensity **R**adar **P**ulse
CR :	**C**oding **R**ate
CRC :	**C**heck **R**edundancy **C**ycle
DevAddr :	**D**evice **A**ddress
DevEUI :	**D**evice **E**xtended **U**nique **I**dentifier
FDM :	**F**requency **D**ivision **M**ultiplexing
HTTP :	**H**yper**T**ext **T**ransfer **P**rotocol
IoT :	**I**nternet **O**f **T**hings
JSON :	**J**ava**S**cript **O**bject **N**otation
JoinEUI :	**J**oin **E**xtended **U**nique **I**dentifier
LoRa :	**Lo**ng **Ra**nge
LoRaWAN :	**Lo**ng **Ra**nge **W**ide **A**rea **N**etwork
LPWAN :	**L**ow **P**ower **W**ide **A**rea **N**etwork.
LTE-M :	**L**ong **T**erm **E**volution Cat **M**1
MIC :	**M**essage **I**ntegrity **C**ontrol
MQTT :	**M**essage **Q**ueuing **T**elemetry **T**ransport
NB-IoT :	**N**arrow**B**and **I**nternet **o**f **T**hings
NwkSKey :	**N**etwork **S**ession **K**ey
OTAA :	**O**ver **T**he **A**ir **A**ctivation
QoS :	**Q**uality **o**f **S**ervice
RSSI :	**R**eceived **S**ignal **S**trength **I**ndication.
SDR :	**S**oftware **D**igital **R**adio
SF :	**S**preading **F**actor
SNR :	**S**ignal **O**ver **N**oise **R**atio
TDM :	**T**ime **D**ivision **M**ultiplexing
TOA :	**T**ime **O**ne **A**ir
TTN :	**T**he **T**hings **N**etwork

Sommaire

1 Les systèmes embarqués et l'IoT

1.1 L'Internet des Objets (**I**nternet **o**f **T**hings / 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 :

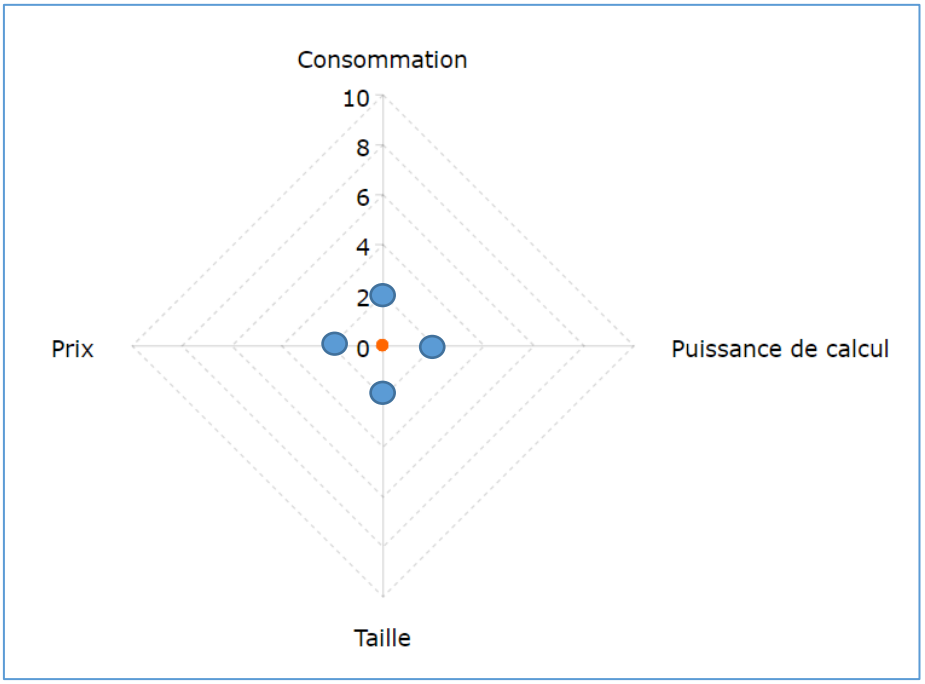


Figure 1 : Consommation, Puissance, Taille et Prix des objets connectés

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

Dans le monde l'IOT (Internet Of Things), on peut citer les protocoles suivants que nous pouvons classer en fonction de leur bande passante et de leur portée dans la Figure 2.

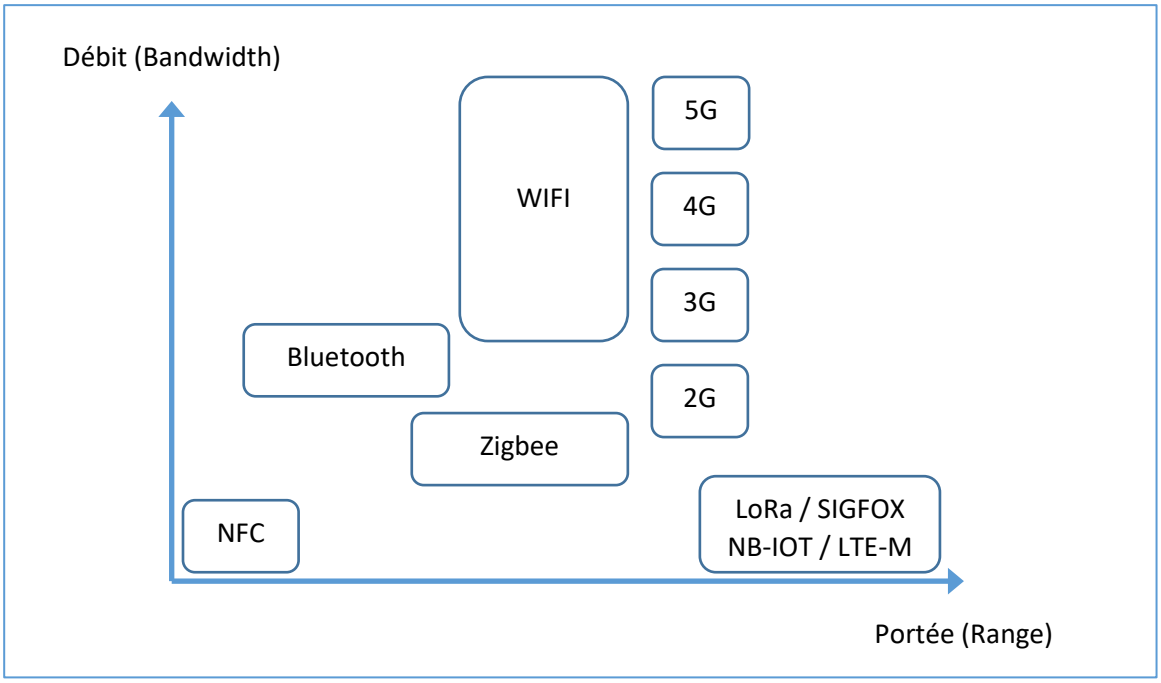


Figure 2 : Protocoles utilisés dans l'IoT en fonction du débit et de la portée

Dans ce cours, nous nous intéressons aux protocoles longue portée, faible débit et très faible consommation. L'ensemble de ces réseaux sont dénommés LPWAN : **L**ow **P**ower **W**ide **A**rea **N**etwork.

Ce graphique ne montre pas la consommation pour lesquelles les deux protocoles LoRa et Sigfox sont sans contestation les plus performants.

1.1.3 Bandes de fréquence utilisées

En Europe, certaines bandes de fréquence 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 fréquence 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 (**F**requency **D**ivision **M**ultiplexing) : 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.

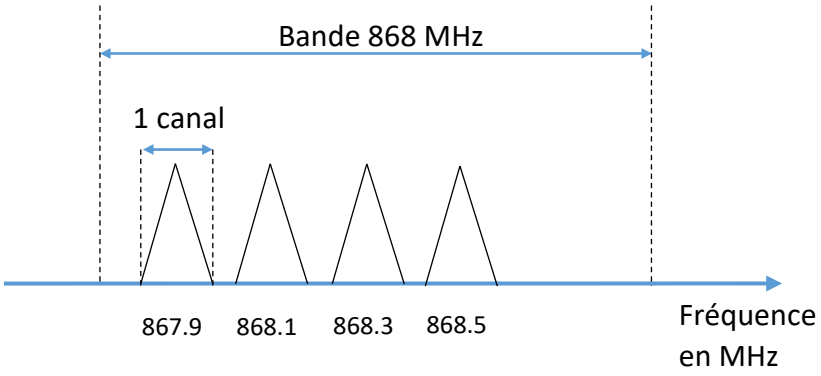


Figure 3 : Utilisation du FDMA en LoRa

- TDM (**T**ime **D**ivision **M**ultiplexing). 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 (**C**ode **D**ivision **M**ultiple **A**ccess) : 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**.

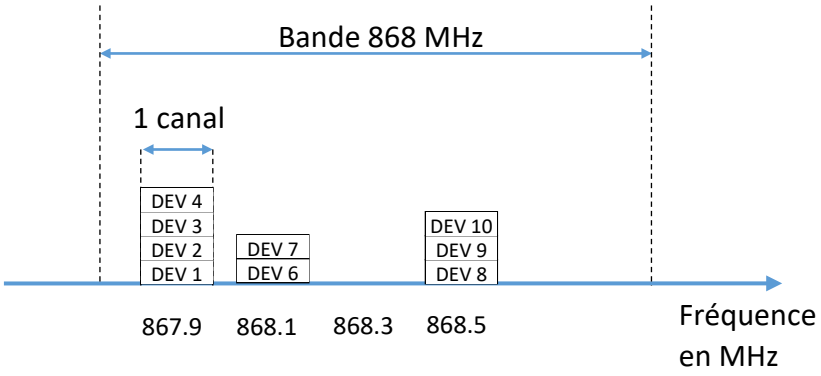


Figure 4 : Utilisation de l'étalement de spectre en LoRa

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 le détails la modulation LoRa.

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 Hadamart (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	**Message User 1**	**1**	**0**	**1**	**0**
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	**Message reçu (∑ (5) / nbr_bits)**	**1**	**0**	**1**	**0**

Les dernières colonnes des tableaux suivants sont laissées libres pour que vous puissiez tester les calculs par vous-même.

1’	**Message User 2**	**0**	**1**	**0**	**1**
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		
… 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		
6’	**Message reçu (∑ (5’) / nbr_bits)**	**0**	**1**		

1’’	**Message User 3**	**1**	**1**	**0**	**0**
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		
… 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		
6’’	**Message reçu (∑ (5’’) / nbr_bits)**	**1**	**1**		

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		
3’’	Symboles transmis User 3	1 -1 -1 1	1 -1 -1 1		

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.

1’’’	**∑ des symboles transmis (3 + 3’ + 3’’)**	**2 -2 0 0**	**2 0 -2 0**		

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		
4’’’	**Message reçu User 1 (∑ (3’’’)/ nbr_bits)**	**1**	**0**		

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		
4’’’	**Message reçu User 2 (∑ (3’’’)/ nbr_bits)**	**0**	**1**		

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		
4’’’	**Message reçu User 3 (∑ (3’’’)/ nbr_bits)**	**1**	**1**		

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. Le Spreading Factor sera définie avec précision dans le chapitre 3.

2 Transmission radio et propagation

2.1 Les unités et définitions

2.1.1 Le décibel (dB)

Le **dB** est un 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 environ 2
0 dB	Egalité
-3 dB	Division par environ 2
- 10 dB	Division par 10

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

2.1.2 La puissance en dBm

Le **dBm** est la puissance ramenée à 1mW : 0 dBm correspond à 1 mW. En reprenant les même rapports que ceux du Tableau 3 qui définissaient les dB, on peut remplir le Tableau 4 concernant les dBm.

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

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

Réponse :

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

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

Le Talkie-Walkie a une puissance d'émission de 33 dBm.

2.1.3 RSSI, Sensibilité, SNR, Link Budget

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

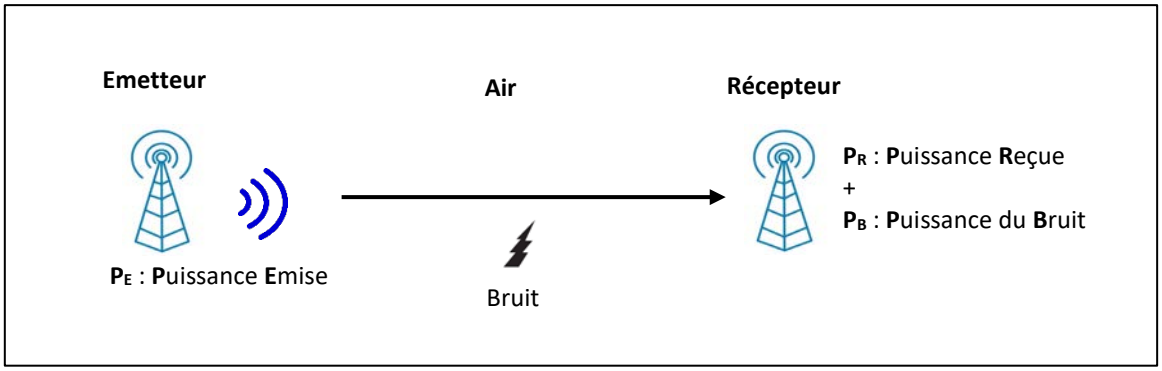


Figure 5 : Schéma d'une transmission radiofréquence

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** (**R**eceived Signal **S**trength **I**ndication), la puissance P_R du signal reçu.
- On appelle **Sensibilité**, la puissance P_R minimale (ou RSSI minimal) qui doit être présente sur le récepteur. Si le RSSI reçu est en dessous de la sensibilité, alors le signal est indétectable par le récepteur.
- On appelle **SNR** (**S**ignal over **N**oise **R**atio), 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 ne passe pas en dessous d'un certain seuil qui rendrait le signal impossible à démoduler.

Exercice : Un émetteur transmet à une puissance de 13dBm en utilisant une antenne dont le gain est de 2dB. Les pertes dans l'air sont de 60 dB. L'antenne réceptrice qui possède un gain de 2dB 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	La puissance reçue est de -43 dBm
-43 > -80 (sensibilité)	Donc **oui**, le signal pourra être reçu

La figure ci-dessous donne un exemple de RSSI et de SNR relevés sur une Gateway lors d'une transmission de donnée en LoRa. Les valeurs **"rssi": -13** et **"snr": 9.5** montre que dans cet exemple le signal reçu arrive avec une forte puissance et avec un très bon rapport signal sur bruit.

```
"gateways":
    {
      "time": "2020-04-29T12:09:45.563621044Z",
      "channel": 0,
      "rssi": -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.**

2.2 Distance de transmission en LoRa

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

$$Atténuation = 10.log_{10}(distance^2.frequence^2.1755)$$

- Atténuation : en **dB**.
- Distance : en **km**.
- Fréquence : en **MHz**.

On peut donc en déduire la distance maximale :

$$distance = \sqrt{\frac{10^{\frac{Atténuation}{10})}}{1755.frequence^2}}$$

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 :

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

- 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

La Figure 6 est un extrait de la documentation du Transceiver LoRa Sx1272 que nous utiliserons.

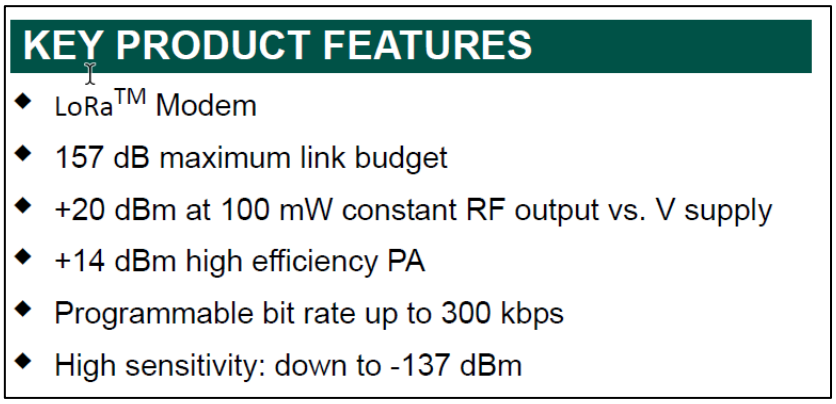


Figure 6 : Caractéristiques principales du composant radiofréquence utilisé

En reprenant la définition du **Link Budget** (P_E moins la sensibilité du récepteur), on retrouve les 157 dB annoncés dans cette documentation (20 dBm + 137 dBm).

En LoRa, plus le code d'étalement est grand, plus on est capable d'émettre dans un milieu bruité. La Figure 7 présente les rapports signal sur bruit avec lesquels nous seront capables de réaliser une transmission, en fonction des Spreading Factor utilisés.

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

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 (2ème colonne du tableau). Comme nous le verrons plus tard, cela impactera évidement 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 ses 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** : **C**ompressed **H**igh **I**ntensity **R**adar **P**ulse)

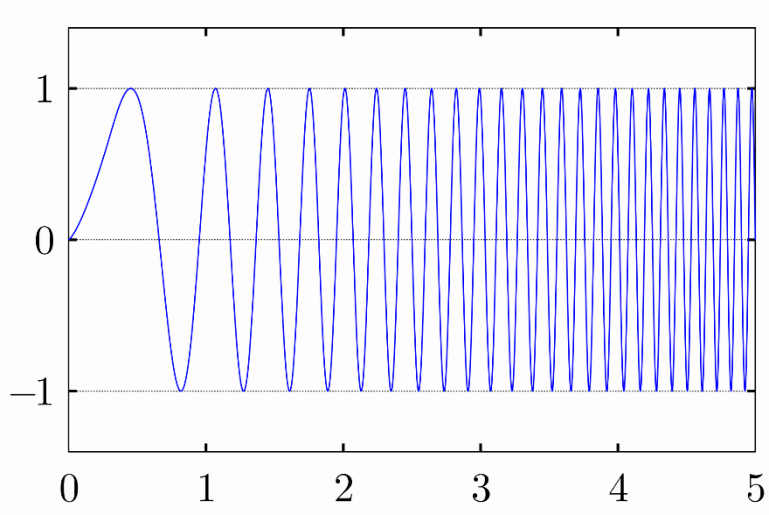


Figure 8 : Symbole de la modulation LoRa (source Wikipédia)

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 du canal **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.

Exercice : 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.

Réponse :

- Fréquence de début : 867 937 500 Hz
- Fréquence de fin : 868 062 500 Hz

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

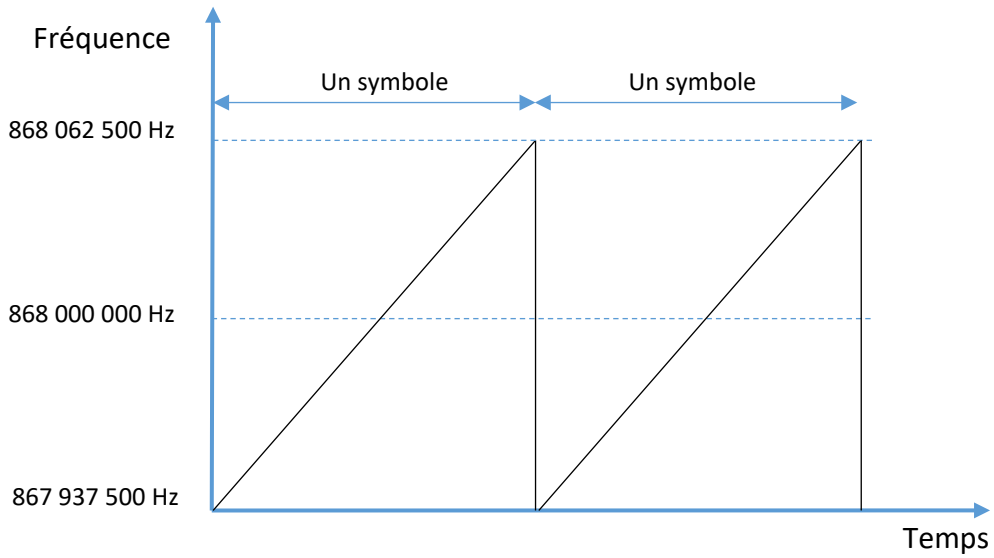


Figure 9 : Forme du symbole de base (CHIRP)

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.

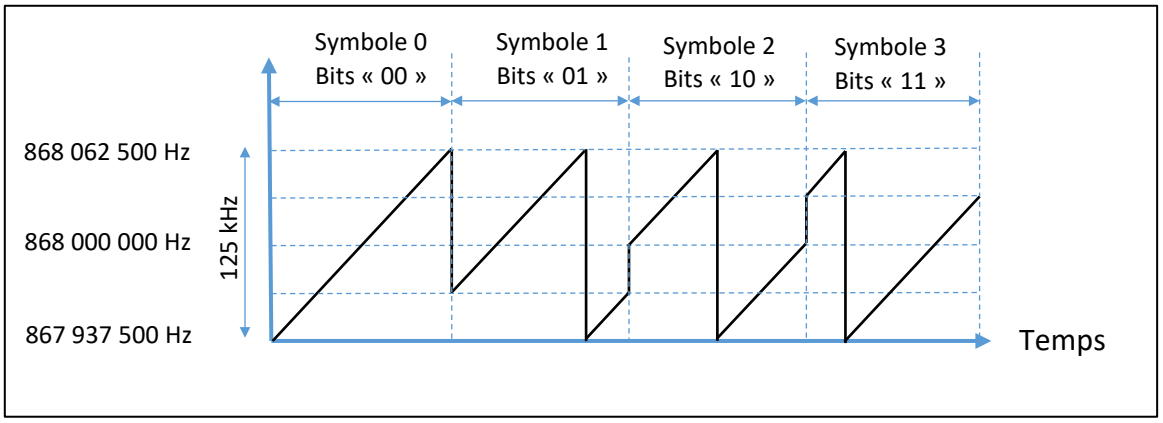


Figure 10 : Symboles émis en Modulation LoRa (Cas théorique en SF2)

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}).

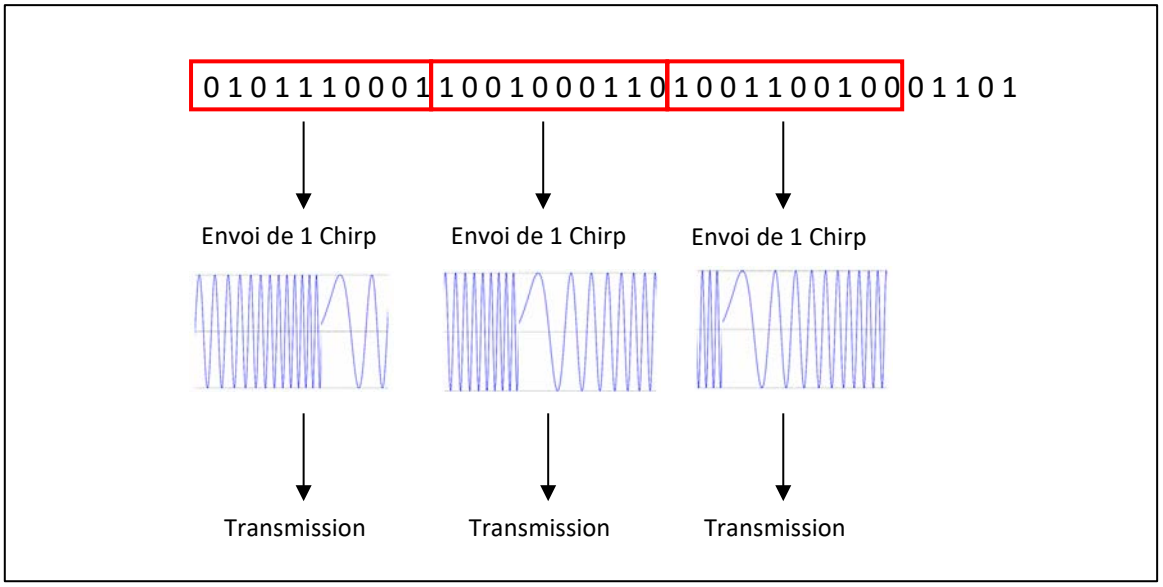


Figure 11 : Emission des Chirp en LoRa

Avec un outil d'analyse spectrale actif pendant l'émission LoRa d'un Device, on peut afficher les successions de symboles qui sont envoyées. Cet oscillogramme a été réalisé à l'aide d'un SDR (**S**oftware **D**igital **R**adio).

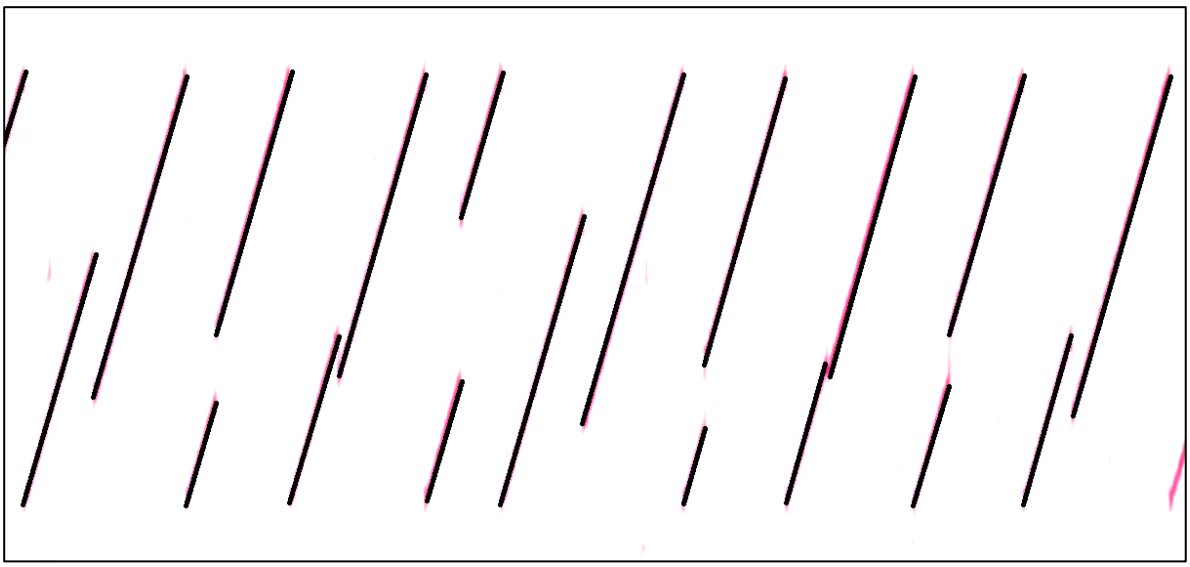


Figure 12 : Visualisation des Chirps LoRa réellement émis pendant une transmission

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.

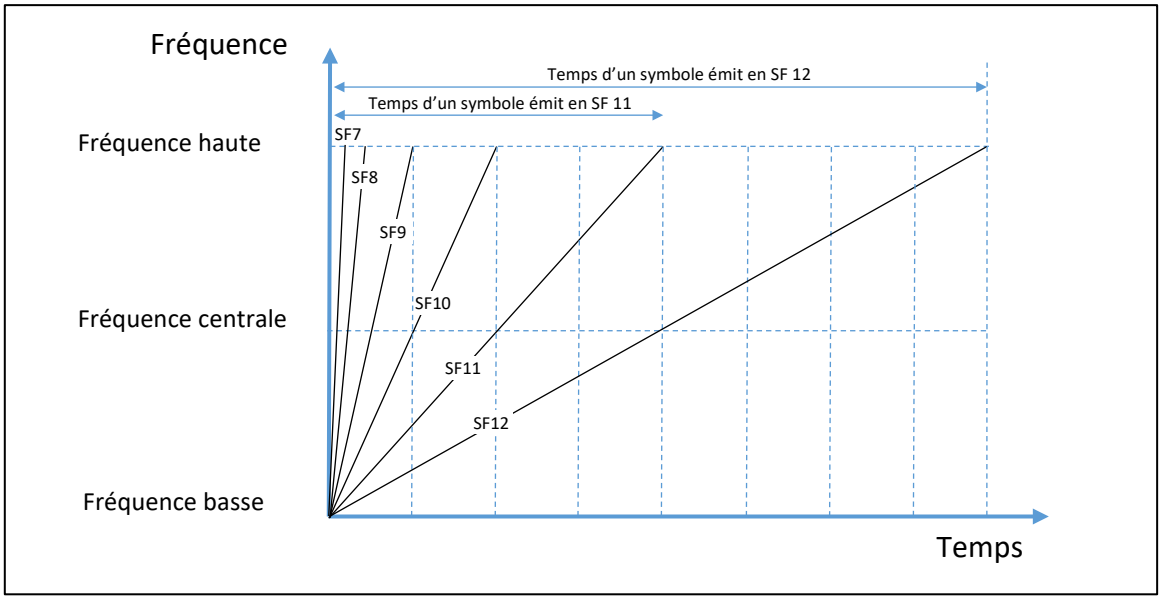


Figure 13 : Temps d'émission d'un symbole (Chirp) en fonction du SF

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 :

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

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

Exercice : On considère les deux cas suivants : cas 1 (SF7, 125 kHz) et cas 2 (SF12, 125 kHz). Donner le débit binaire correspondant.

Réponse :

- **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'erreurs. 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.

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

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

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

Réponse :

- **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…

Un simulateur LoRa en ligne équivalent est aussi disponible

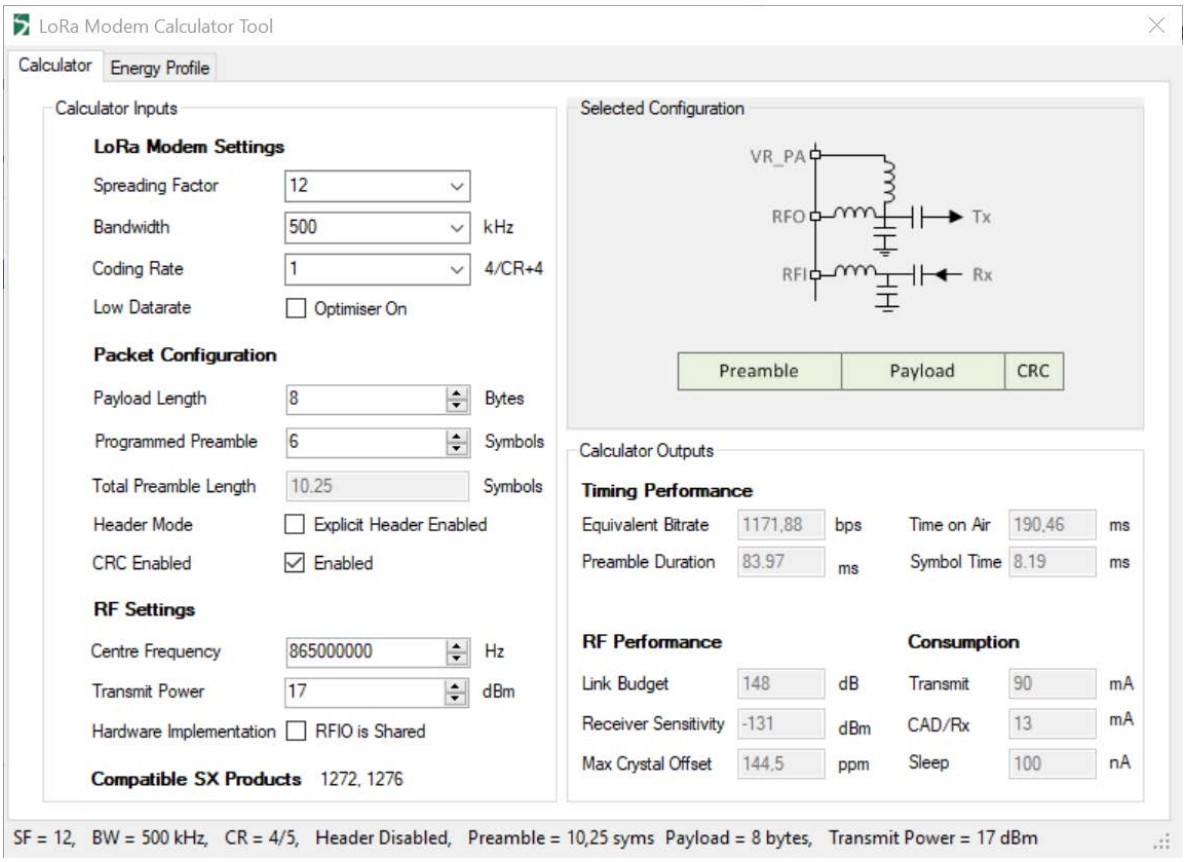


Figure 16 : Le logiciel LoRa Calculator

Exercice : En reprenant l'exemple des deux cas précédents [SF7, 125 kHz, CR 4/5] et [SF12, 125 kHz, CR 4/5], vérifier les calculs du "Equivalent Bitrate" avec le logiciel LoRa Calculator.

Réponse :

- **Cas 1 :** Pour SF7, 125 kHz et CR4/5 > Débit = **5468,75 bps**
- **Cas 2 :** Pour SF12, 125 kHz et CR4/5 > Débit = **292,97 bps**

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.

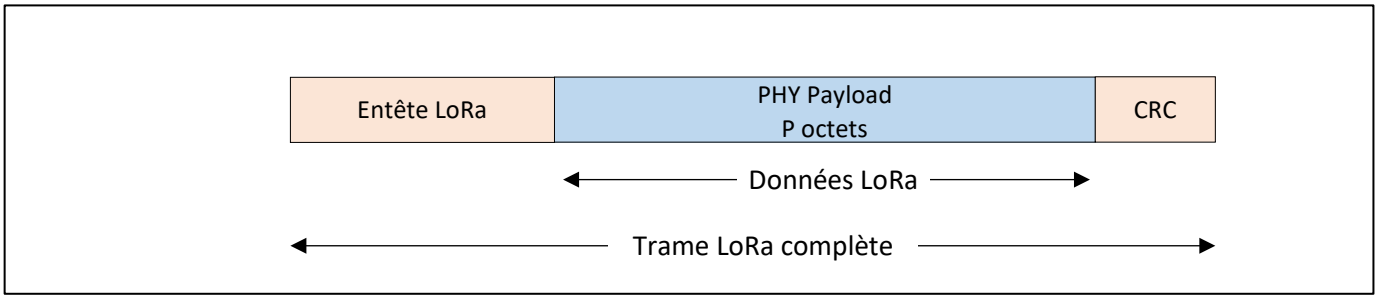


Figure 17 : Trame LoRa

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.

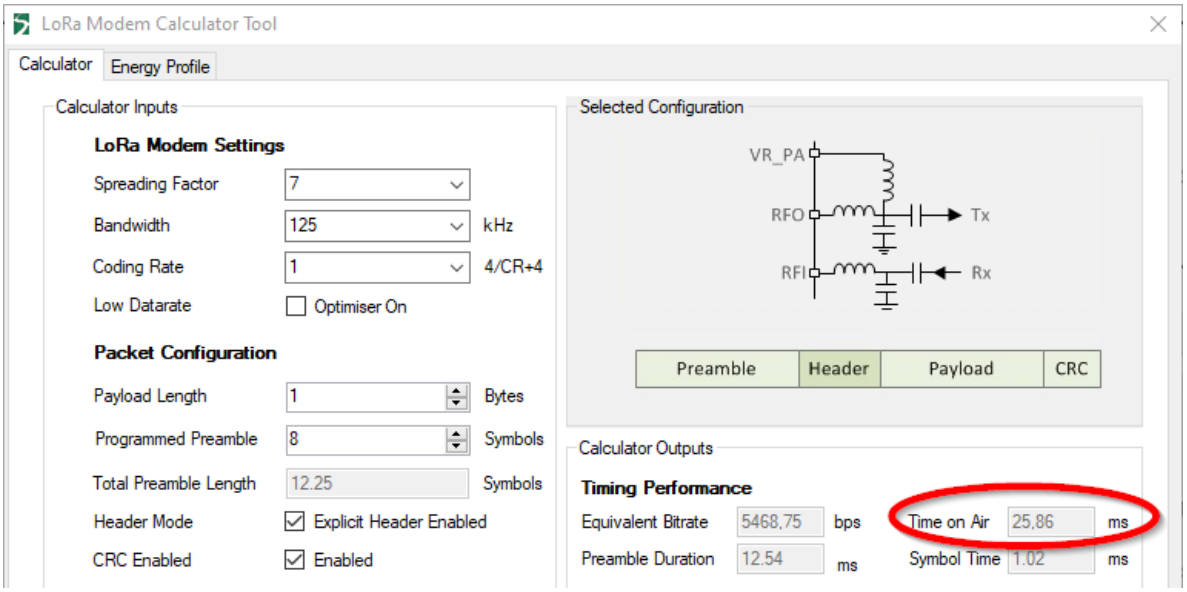


Figure 18 : Simulation du Time On Air en LoRa

Exercice : Donner le Time On Air pour l'envoi d'1 octet en SF7 et en SF12. En déduire En déduire le débit utile de cette transmission dans les deux cas.

Réponse :

- 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**

- **Cas 1 :** Pour SF7, 125 kHz et CR4/5 > $Débit_{utile\_LoRa}$ = 8 / 25,85 ms = **309,3 bps**
- **Cas 2 :** Pour SF12, 125 kHz et CR4/5 > $Débit_{utile\_LoRa}$ = 8 / 827.39 ms = **9,6 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 une 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.

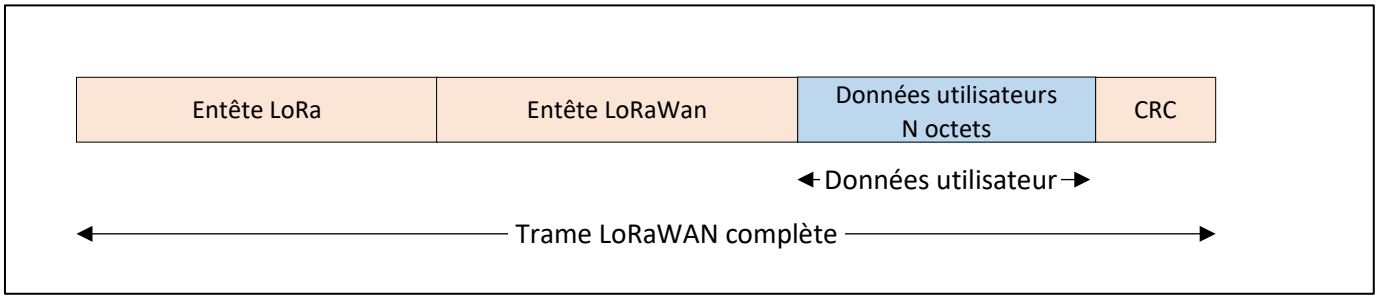


Figure 19 : Trame LoRaWAN

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

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.

Exercice : En déduire le débit utile de cette transmission pour les deux cas [SF7, 125 kHz, CR 4/5] et [SF12, 125 kHz, CR 4/5].

Réponse :

- **Cas 1 :** Pour SF7, 125 kHz et CR4/5 > $Débit_{utile\_LoRaWAN}$ = 8 / 46.3 ms = **172.7 bps**
- **Cas 2 :** Pour SF12, 125 kHz et CR4/5 > $Débit_{utile\_LoRaWAN}$ = 8/ 1155.1 ms = **6,9 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'émets 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 x 46,3ms = 4,58 secondes.

Exercice : En reprenant les exemples précédents [SF7, 125 kHz, CR 4/5] et [SF12, 125 kHz, CR 4/5], quels sont les débits moyens si on prend en compte le « Time On Air » et le Duty-Cycle de 1% de la norme LoRaWAN?

Réponse :

- **Cas 1 :** Pour SF7, 125 kHz et CR4/5 > $Débit_{final}$ = 172.7 bps / 100 = **1,73 bps**
- **Cas 2 :** Pour SF12, 125 kHz et CR4/5 > $Débit_{final}$ = 6,9 bps /100 = **0,07 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.

Un simulateur en ligne permet d'avoir une première approximation de la consommation et donc de l'autonomie d'un Device LoRa.

4 Le protocole LoRaWAN

4.1 La présentation du protocole

4.1.1 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 chaine 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.

4.1.2 La LoRa Alliance

La LoRa Alliance est une organisation née en 2015 qui a pour objectif de développer la technologie et l'ensemble de l'écosystème LoRaWAN. Ses membres sont des entreprises très impliquées qui investissent dans la LoRa Alliance pour la faire évoluer. N'importe quelle structure peut postuler pour faire partie de la LoRa Alliance (même gratuitement) et participer au développement du LoRaWAN.

4.1.3 Les versions du protocole

Un des principaux rôles de la LoRa Alliance est la spécification et l'évolution du protocole LoRaWAN. Voici les différentes versions et leurs évolutions au cours du temps.

- **Janvier 2015 : Version 1.0.0**
- **Février 2016 : Version 1.0.1**
- **Juillet 2016 : Version 1.0.2**. Première version stable avec de nombreuses régions supportées
- **Octobre 2017 : Version 1.1**. Amélioration de la sécurité et itinérance (roaming)
- **Juillet 2018 : Version 1.0.3**. Ajout de la spécification pour les Device de classe B
- **Octobre 2020 : Version 1.0.4**. AppEUI devient JoinEUI. Les Frames Counter sur 32 bits.
- **A venir courant 2021 : Version 1.1.1**. Basée sur le la version 1.1.

On remarque que très tôt une version 1.1 a été écrite, mais elle n'a pas été adoptée par les industriels. La LoRa Alliance a alors poursuivi la clarification de la version 1.0.X en ajoutant la version 1.0.3 et la version 1.0.4 qui est doit être la dernière de la série.

Sauf si cela est clairement spécifié, ce document traite essentiellement de la version 1.0.X du protocole 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 :

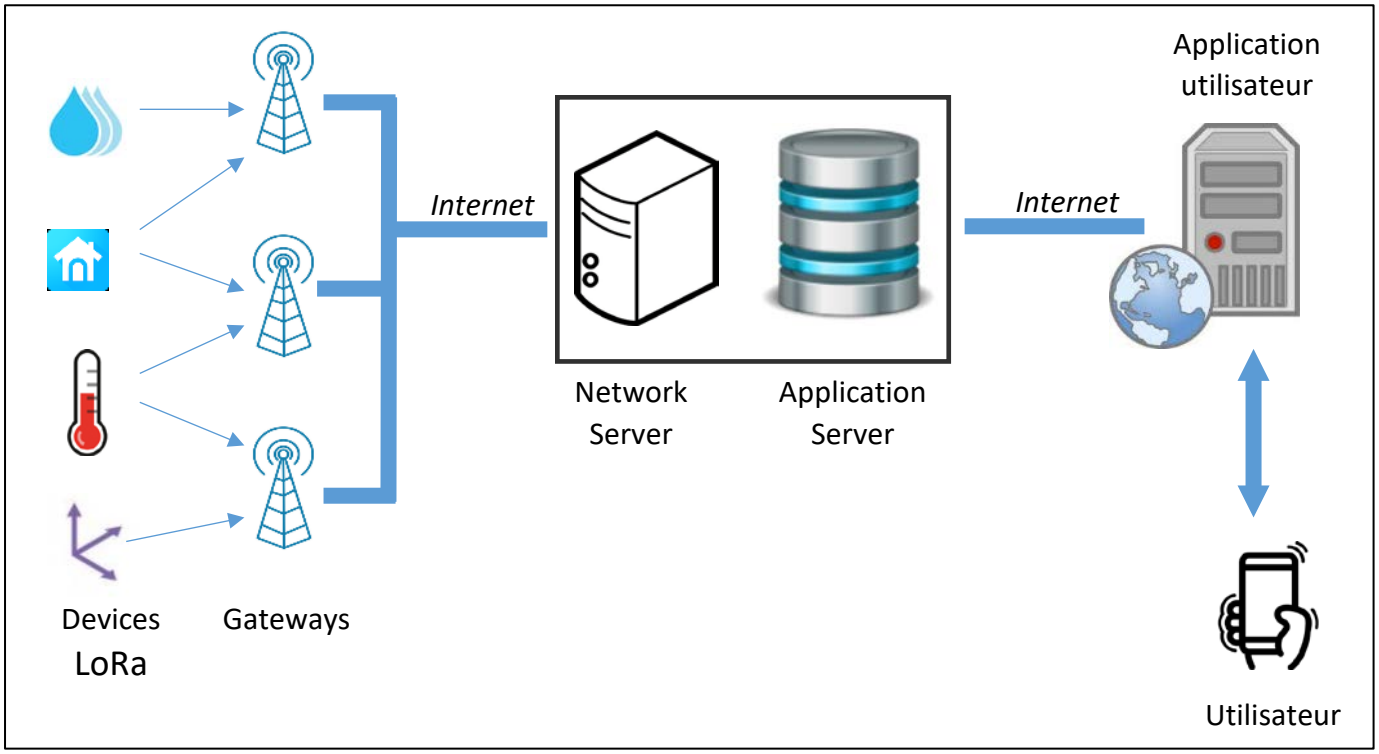


Figure 21 : Structure globale d'un réseau LORAWAN

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.

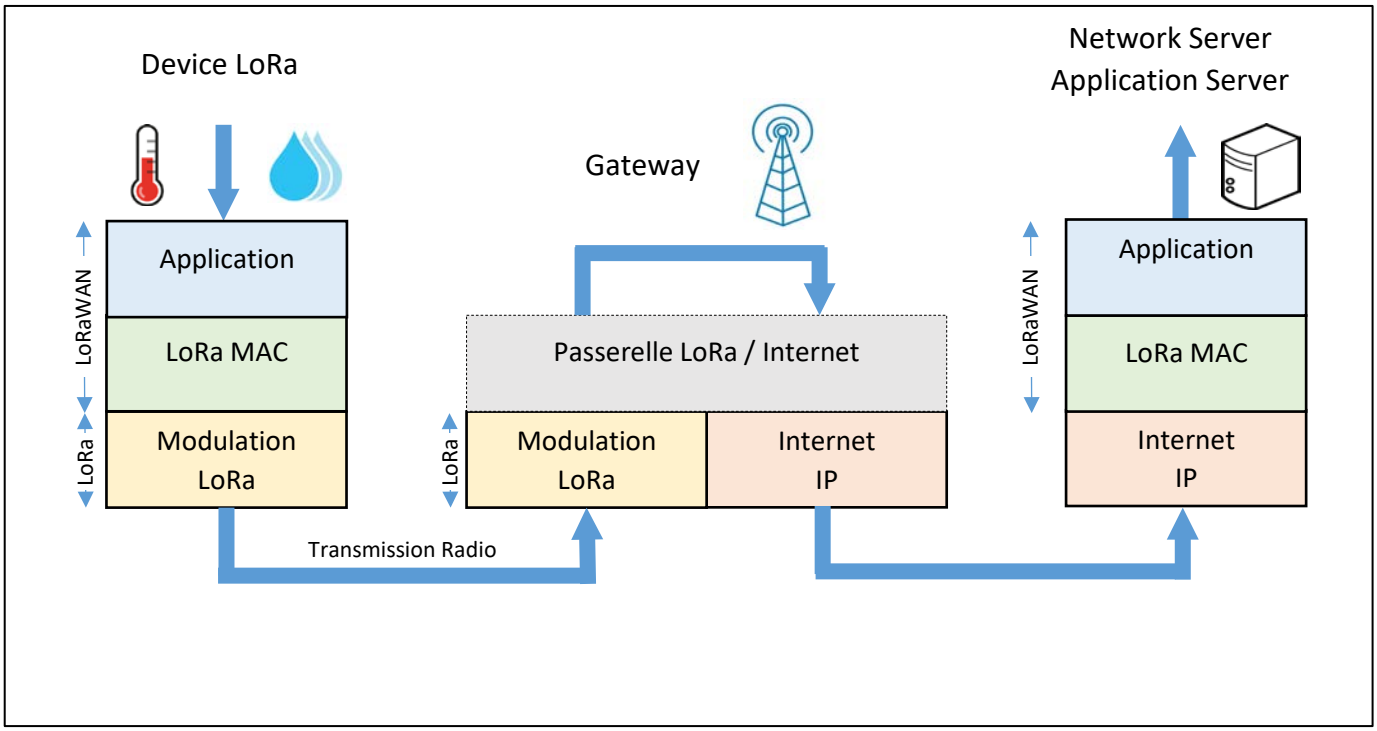


Figure 22 : Le rôle de la Gateway LoRa

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 **N**etwork **S**ession **K**ey **: NwkSKey**. Nous parlons bien ici d'authentification et non pas de chiffrement comme nous le verrons au paragraphe 4.2.6.

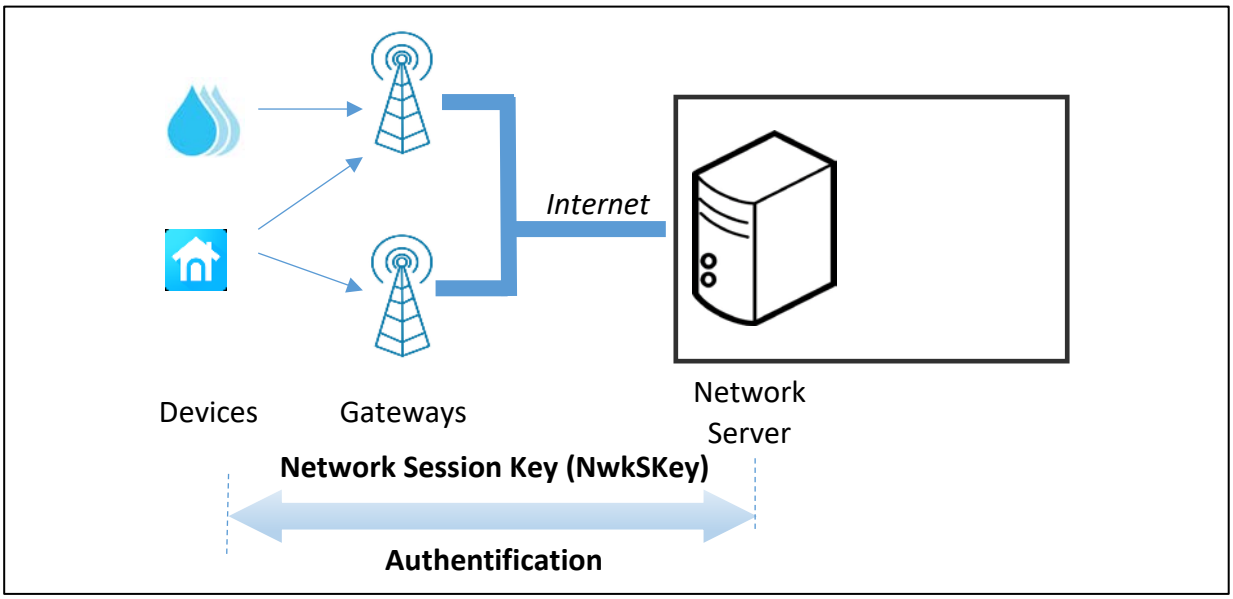


Figure 23 : Authentification entre le Device LoRa et le Network Server

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

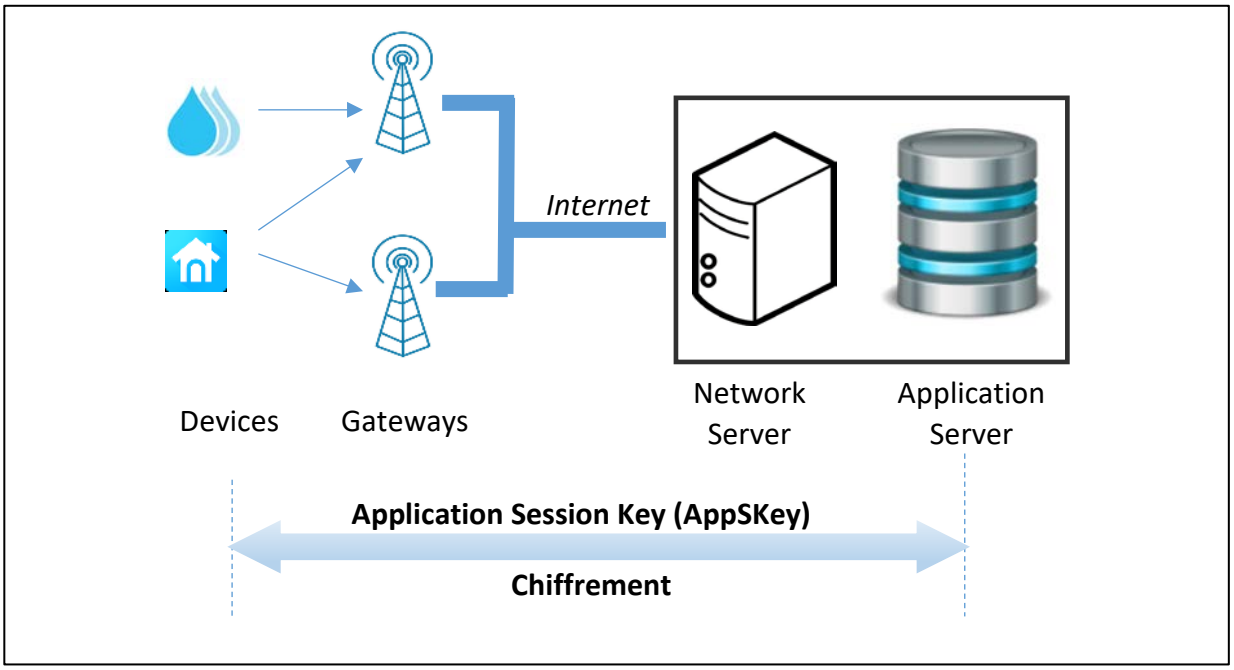


Figure 24: Chiffrement entre le Device LORA et l'Application Server

grâce à une clé AES 128 bits appelée Application Session Key : **AppSKey**. Contrairement au **NwkSKey (N**etwork **S**ession **K**ey), 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 **N**etwork **S**ession **K**ey (**NwkSKey**) pour l'authentification entre le Device LoRa et le Network Server

- De l'**A**pplication **S**ession **K**ey (**AppSKey**) pour le chiffrement entre le Device LoRa et l'Application Server.

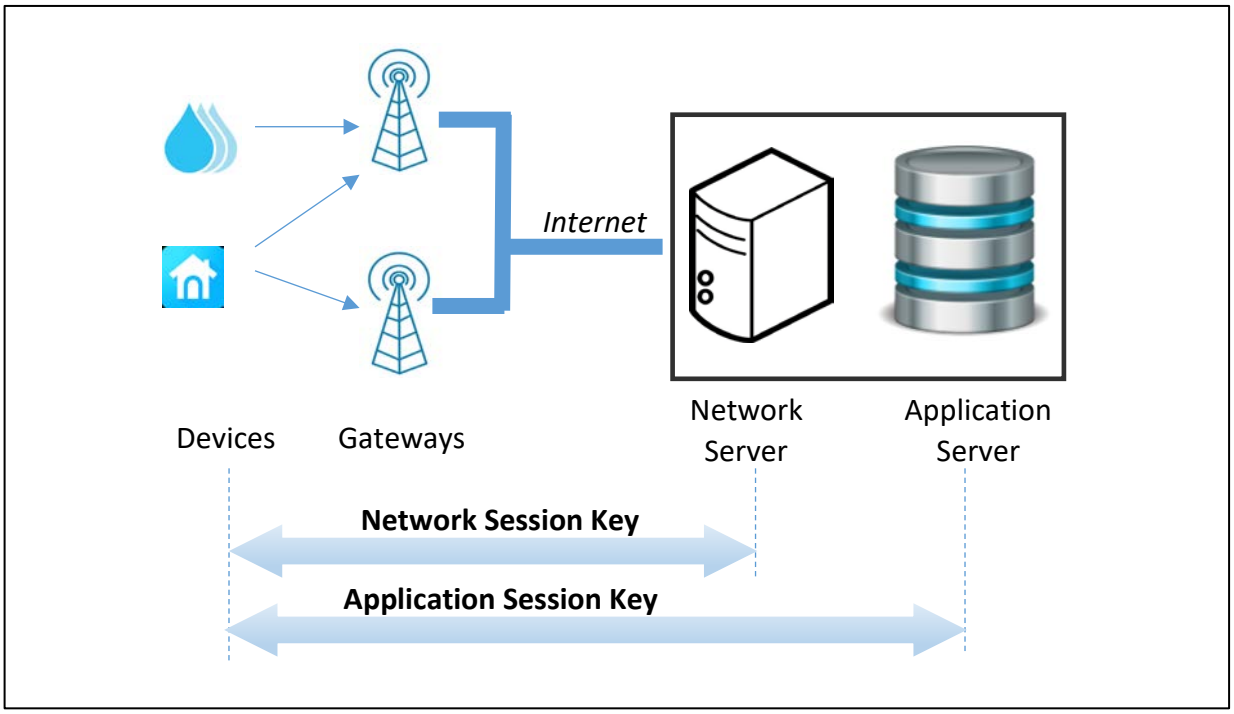


Figure 25 : Authentification et chiffrement en LoRaWan

4.2.5 Application utilisateur

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

- Avec le protocole HTTP
- Avec le protocole MQTT

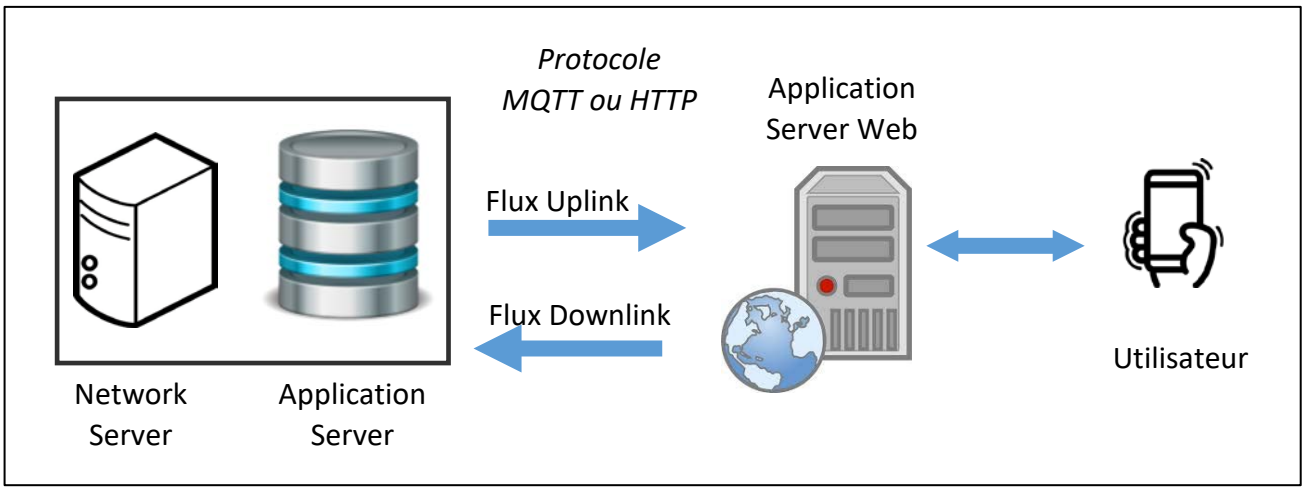


Figure 26 : Application / Web Server

> **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 tableau de bord 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** (**M**essage **I**ntegrity **C**ontrol) 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.

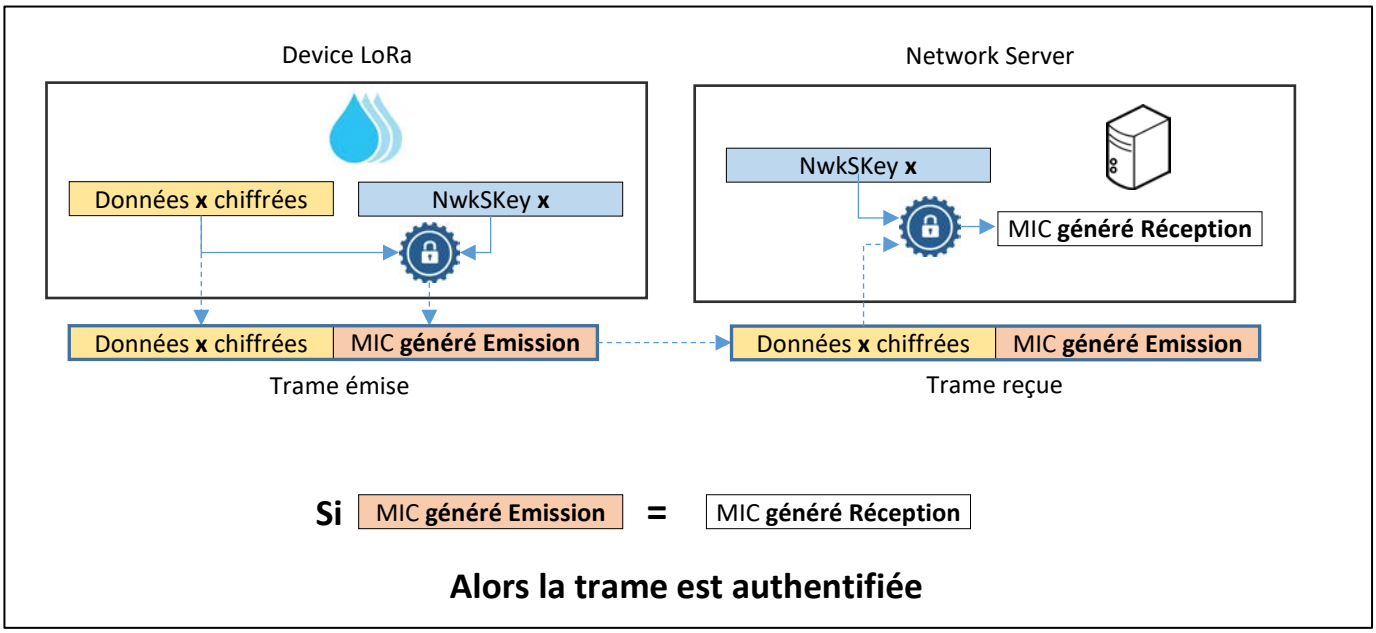


Figure 27 : Authentification d'un Device LoRa par le Network Server

Dans le schéma de la Figure 27, 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'**A**pplication **S**ession **K**ey (**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 :

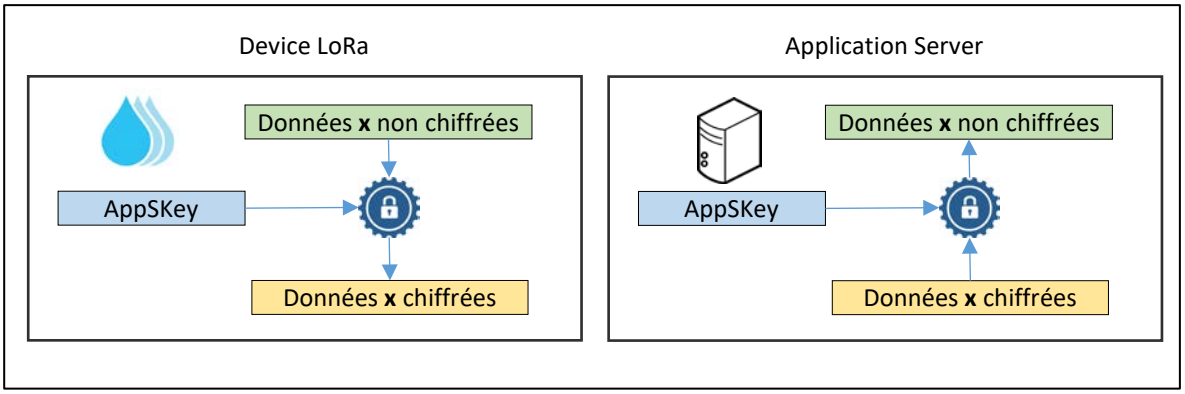


Figure 28 : Chiffrement des données par l'Application Session Key

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.

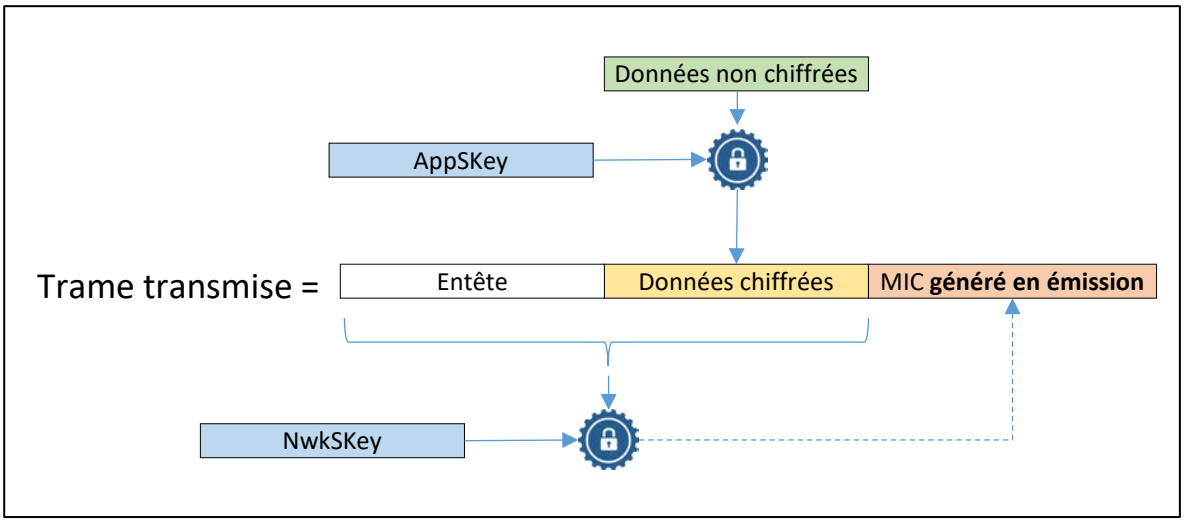


Figure 29 : Chiffrement, puis Authentification

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. 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.

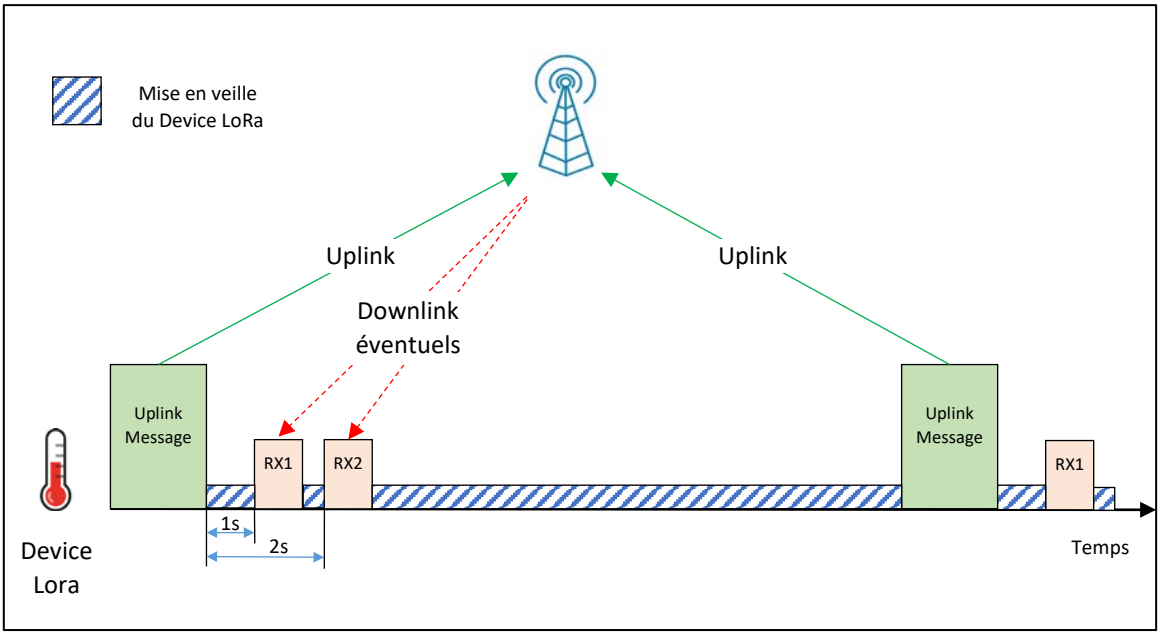


Figure 30 : Slots de réception pour un Device LoRa de classe A.

La durée des fenêtres doit être au minimum la durée de réception d'un préambule. Un préambule dure 12.25 $T_{symbole}$ 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 réception 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.

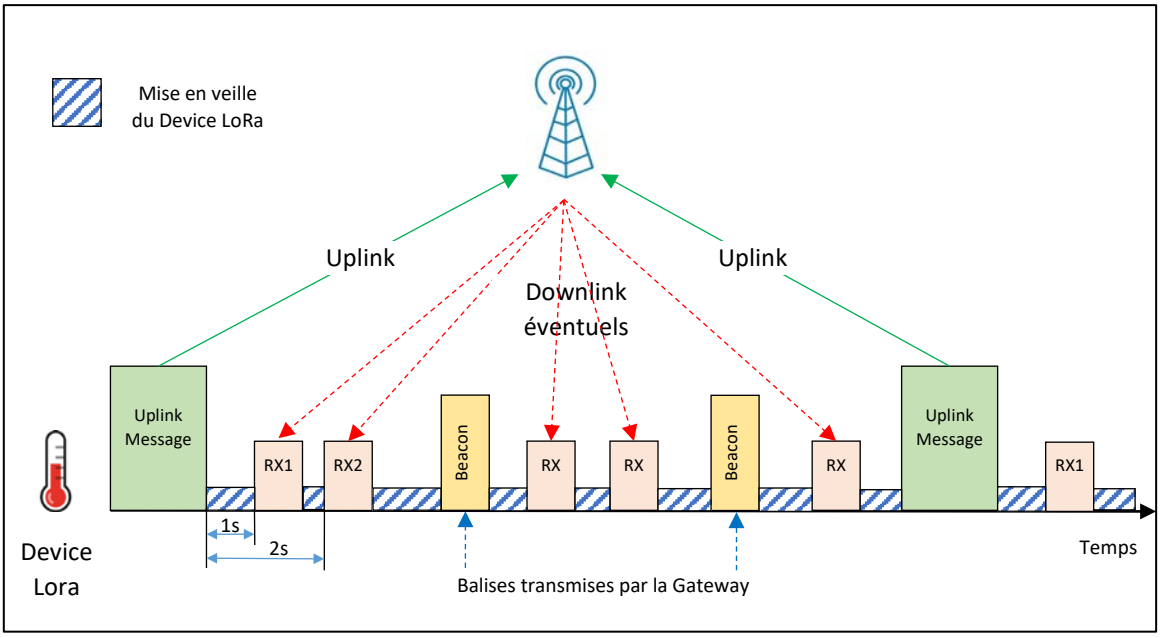


Figure 31 : Slots de réception pour un Device LoRa de classe B

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.

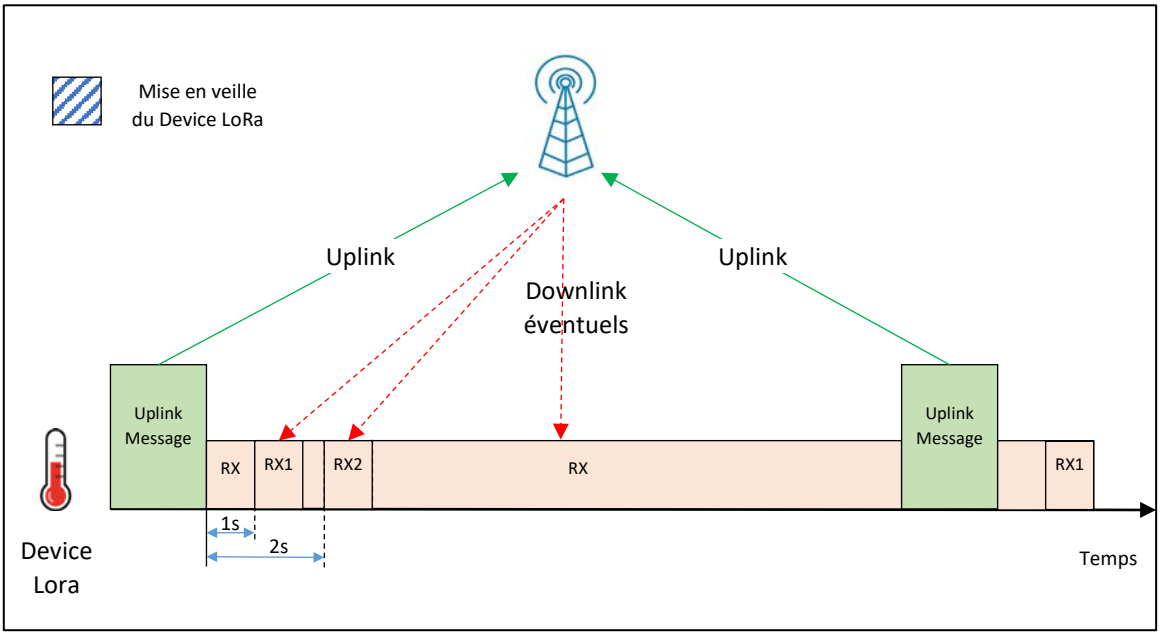


Figure 32 : Slots de réception pour un Device LoRa de classe C

Toutes les zones RX sont sur les même paramètres (canal et Spreading Factor) que RX2.

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 de la façon suivante :

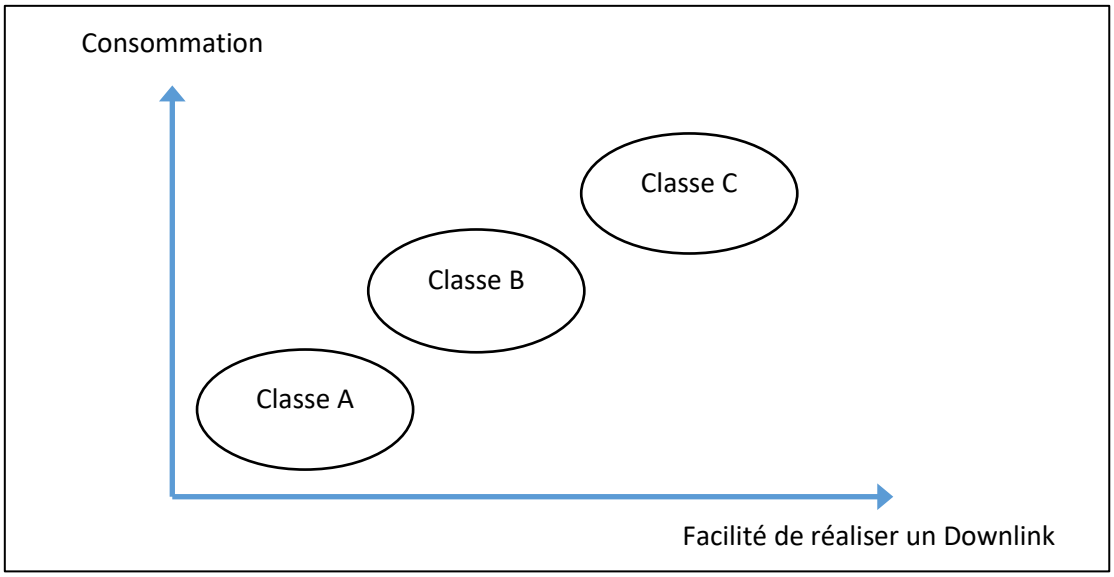


Figure 33 : Consommation et Downlink en fonction de la classe de Device

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 :

- **A**ctivation **B**y **P**ersonalization : **ABP**.
- **O**ver **T**he **A**ir **A**ctivation.

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.

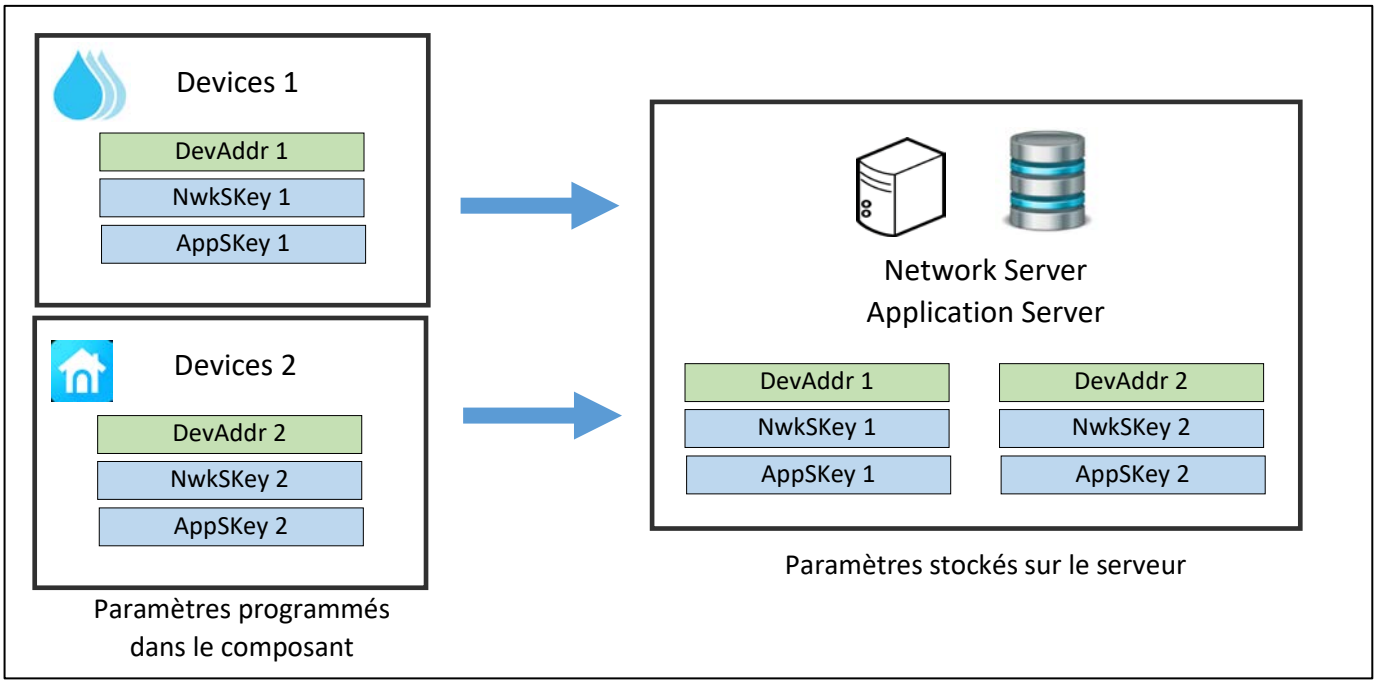


Figure 34 : Paramétrage de DevAddr, NwkSKey et AppSKey en ABP

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 connaitre : le **DevEUI**, **l'AppEUI**, et **l'Appkey**. Le Network Server doit, lui, connaitre 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**.

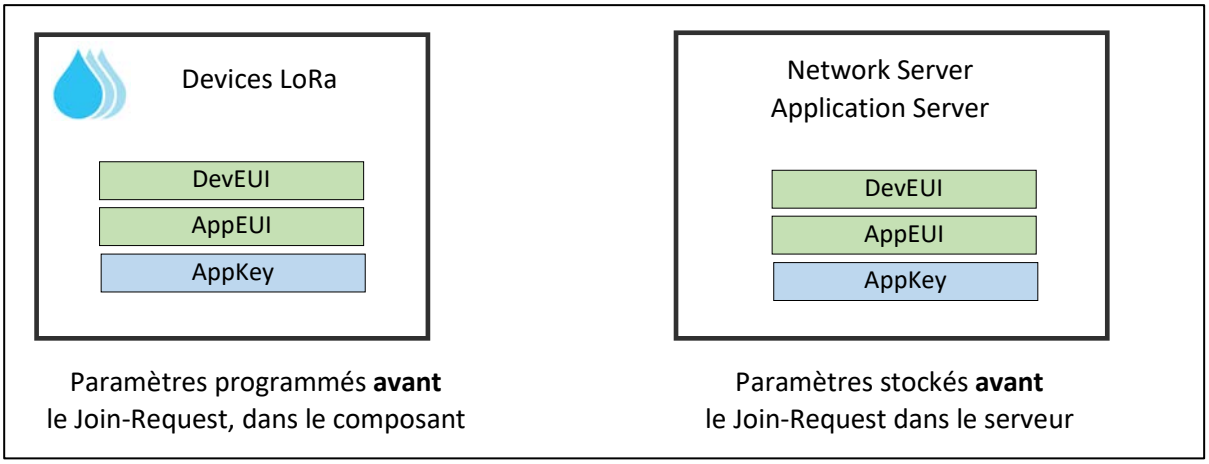


*Figure 35 : Paramétrage du Device LoRa et du Serveur **avant** le Join-Request (OTAA)*

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.

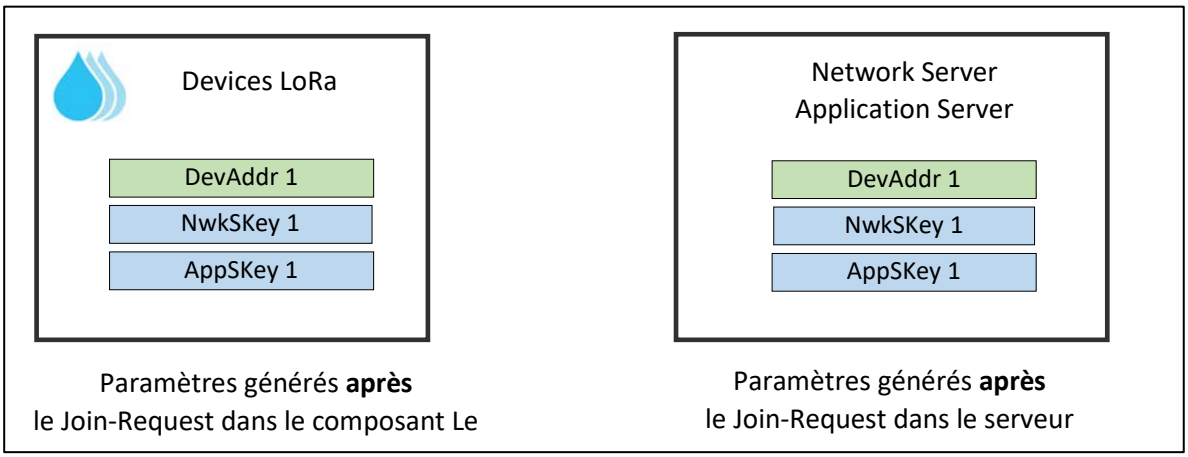


*Figure 36 : Paramétrage du Device LoRa et du Serveur **après** le Join-Request (OTAA)*

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 Resquest. Il est partagé avec le Network server.

Informations possédées 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 37 présente le déroulement de la négociation de ces paramètres :

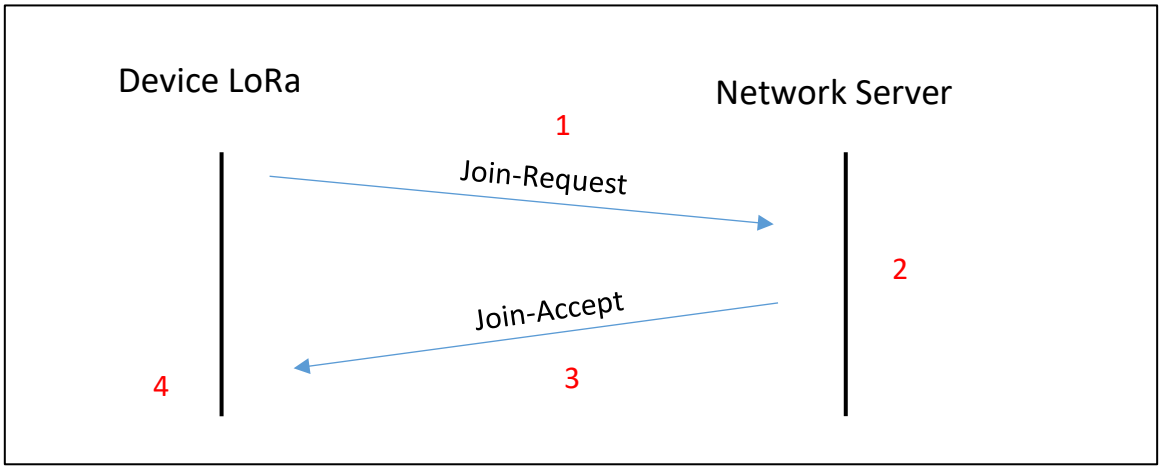


Figure 37 : Join-Request – Join-Accept en OTAA

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, associés à **l'AppKey**, permettent au Device LoRa de générer le même **NwkSKey** et le même **AppSKey** qui avaient été initialement générés 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 l'authentification et du chiffrement, 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:

- **NwSKey** et **AppSKey** en ABP.
- **AppKey** en OTAA.

Elles devront donc être stockées dans des mémoires sécurisées.

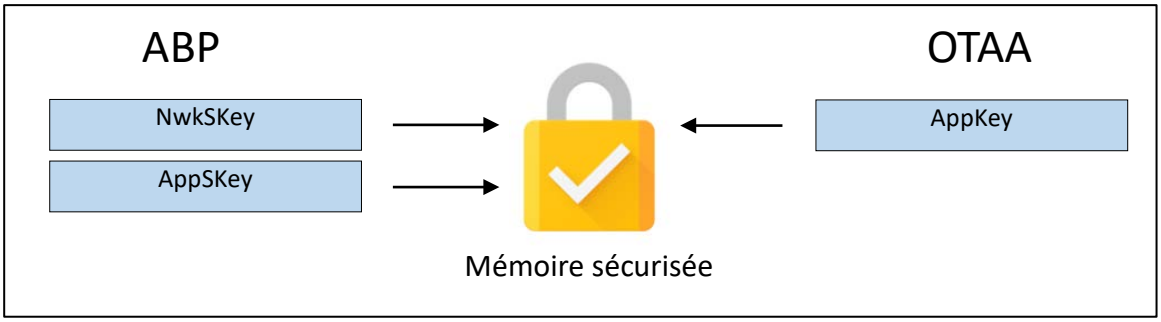


Figure 38 : Clés stockées dans une mémoire sécurisée

Néanmoins, comme la méthode ABP conserve les clés de session éternellement, il y a plus de chances de se les faire voler par force brute, notamment si le Device rejoue les mêmes séquences de nombreuses fois. De plus, la manipulation des clés de session en ABP (si on veut par exemple changer de réseau LoRaWAN) donne plus d'opportunités d'exposer les clés et donc de les rendre visibles.

D'un point de vue de la sécurité, on préférera donc toujours si possible le mode d'activation OTAA

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.

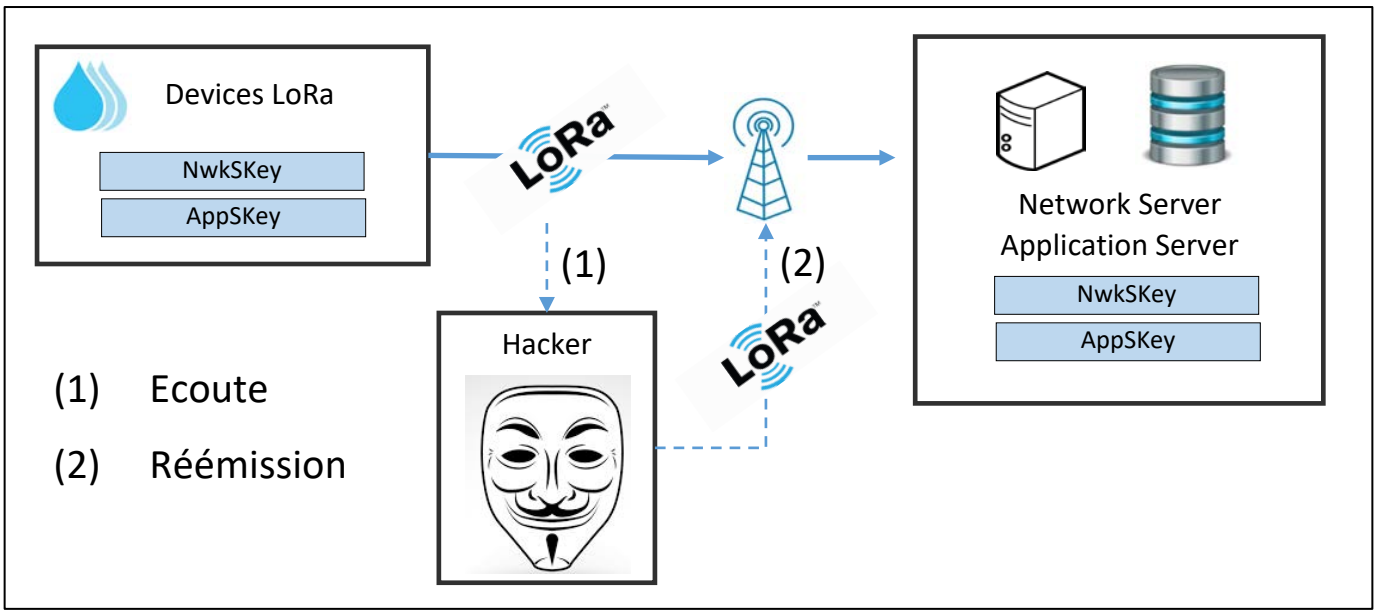


Figure 39 : Attaque par REPLAY

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.

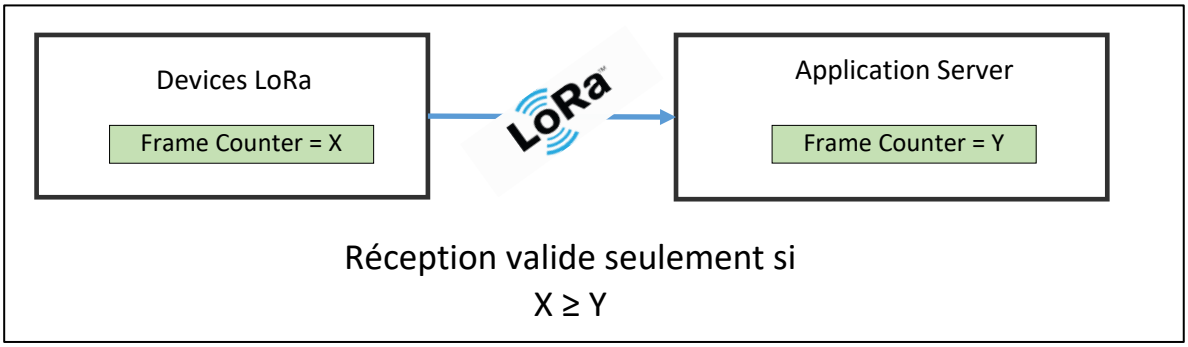


Figure 40 : Utilisation d'un "Frame Counter" pour éviter l'attaque par Replay

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 résolue 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.

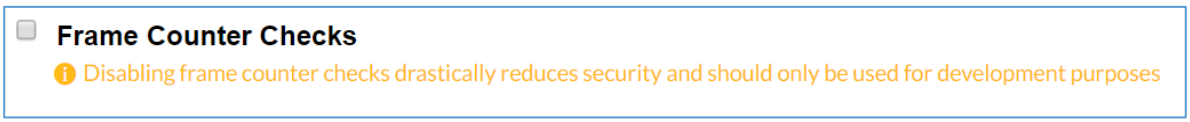


Figure 41 : Désactivation du "Frame Counter Check" dans TTN

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

A noter qu'un Hacker peut aussi utiliser une attaque par Replay pendant le transfert du Join-Request en OTAA. Un compteur dont le nom est "DevNonce" est donc aussi utilisé pendant cette phase. Depuis la version 1.0.4 de la spécification LoRaWAN, un Device en OTAA devra donc lui aussi faire une sauvegarde dans une mémoire non volatile de ce compteur.

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 possiblement existants. Si le Network server que vous utilisez est répertorié auprès de la LoRa Alliance, 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 et donc de passer sur un autre réseau? Cela dépends 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.

En plus de ces éléments que nous avons déjà vus, 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.

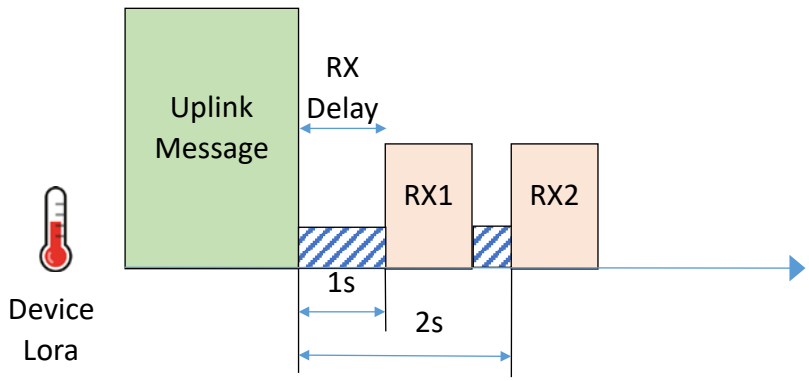


Figure 42 : Le RX Delay

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 43.

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

Figure 43 : Canaux 4 à 8 pouvant être définis 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 Les Devices ne supportant pas l'OTAA

La procédure d'activation en OTAA requiert que le Device soit capables de fonctionner en Downlink pour recevoir le Join-Accept. Si le flux Downlink n'est pas supporté, alors il devra forcément utiliser le mode d'activation ABP.

4.5.6 Résumé

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

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

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

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** connaitre 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
Exemple des canaux supportés par 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)Tableau 9

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 acceptables 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 leur valeur. 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'**A**daptive **D**ata **R**ate (**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 choisit 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 :

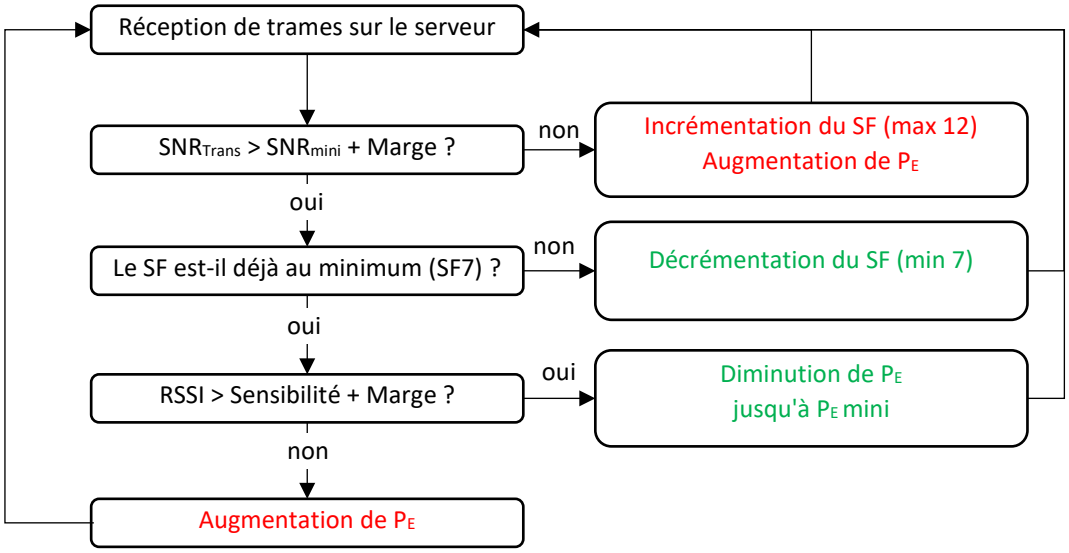


Figure 44 : Ajustement du SF et de PE grâce à l'Adaptive Data Rate

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

Dans la Figure 44, 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 45.

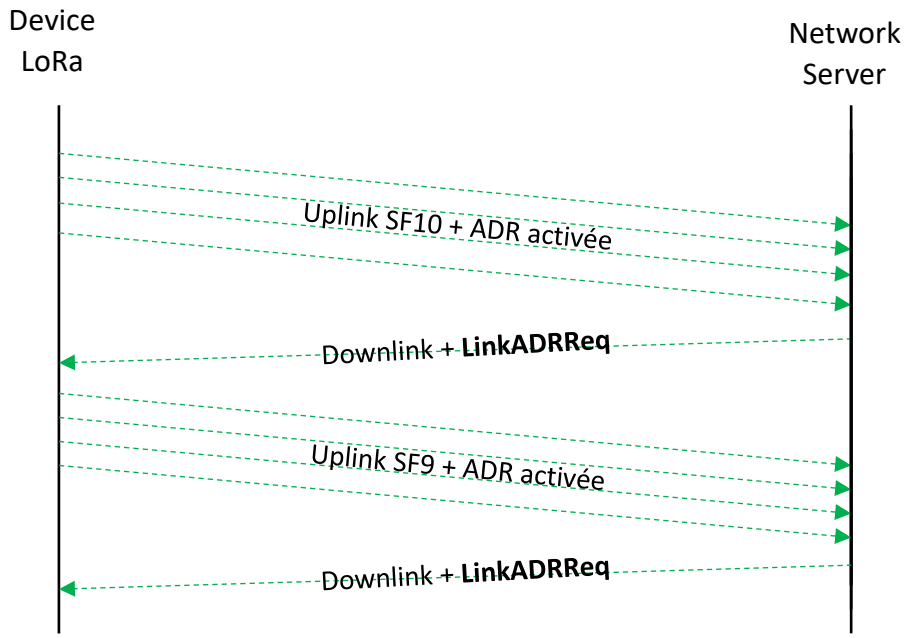


Figure 45 : Optimisation du SF et de P_E grâce à l'ADR

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 46.

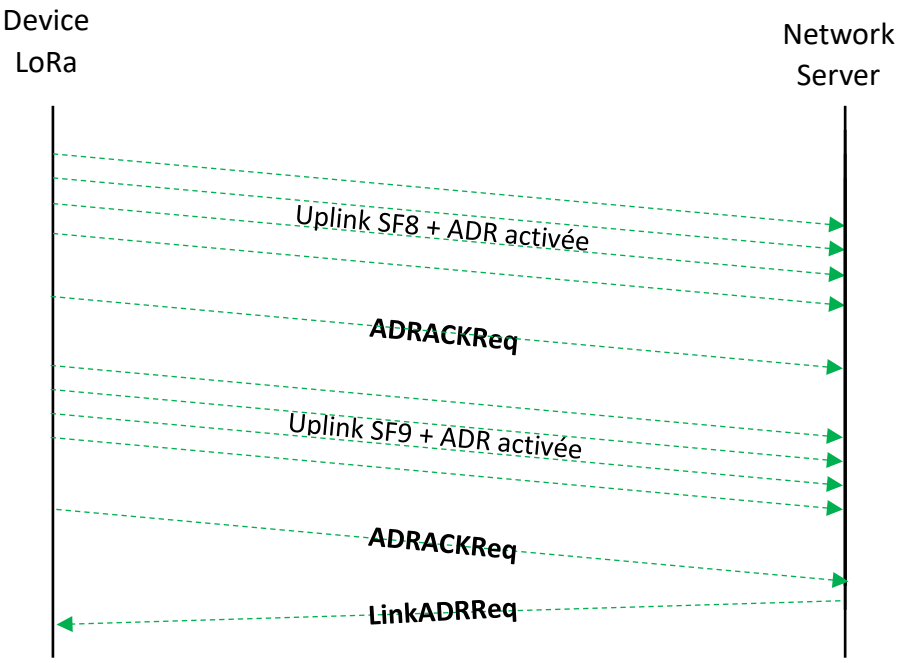


Figure 46 : Requête ADRACKReq du Device pour obtenir un LinkADRReq

Résumé sur l'ADR :

- L'Adaptive Data Rate 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 trois configurations :

- 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.

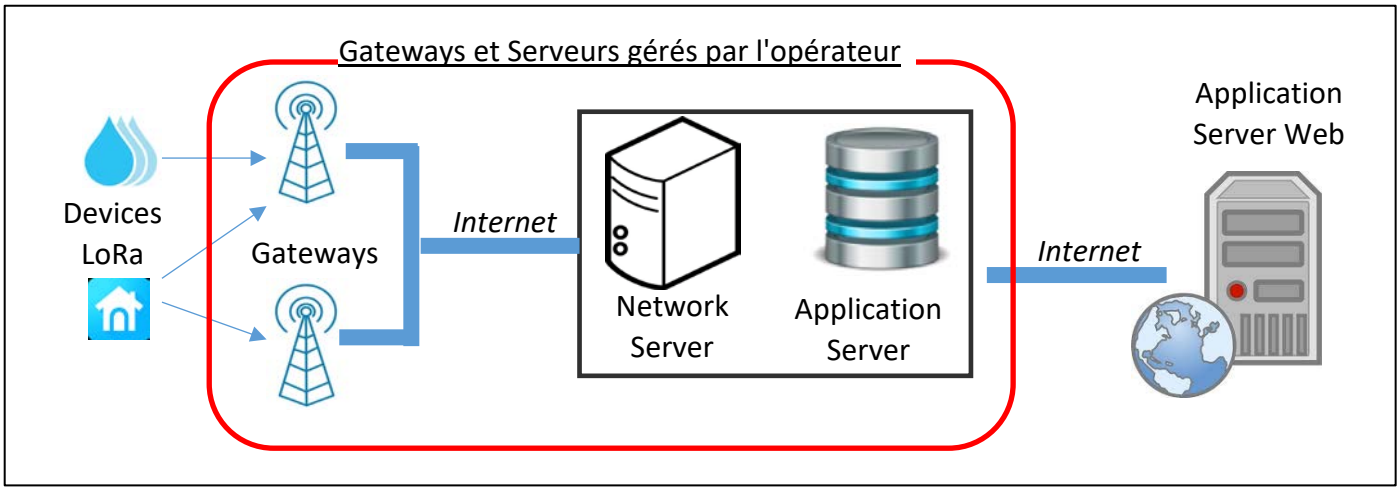


Figure 47 : infrastructure d'un réseau LoRaWAN opéré

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 48.

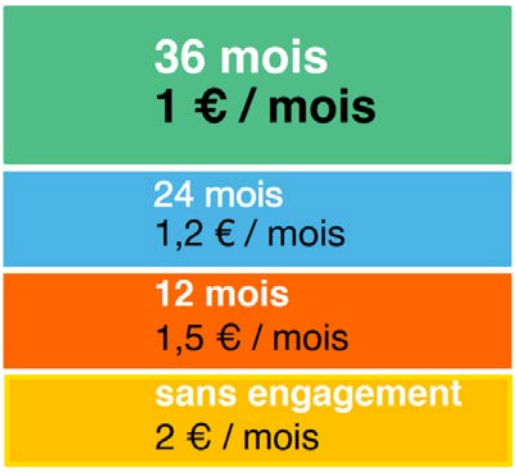


Figure 48 : Tarification d'un abonnement par Device LoRaWAN chez Orange

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 ChirpStack ou The Things Stack. La présentation de ces stacks permettant d'implémenter ces serveurs LoRaWAN est réalisée au chapitre 9.

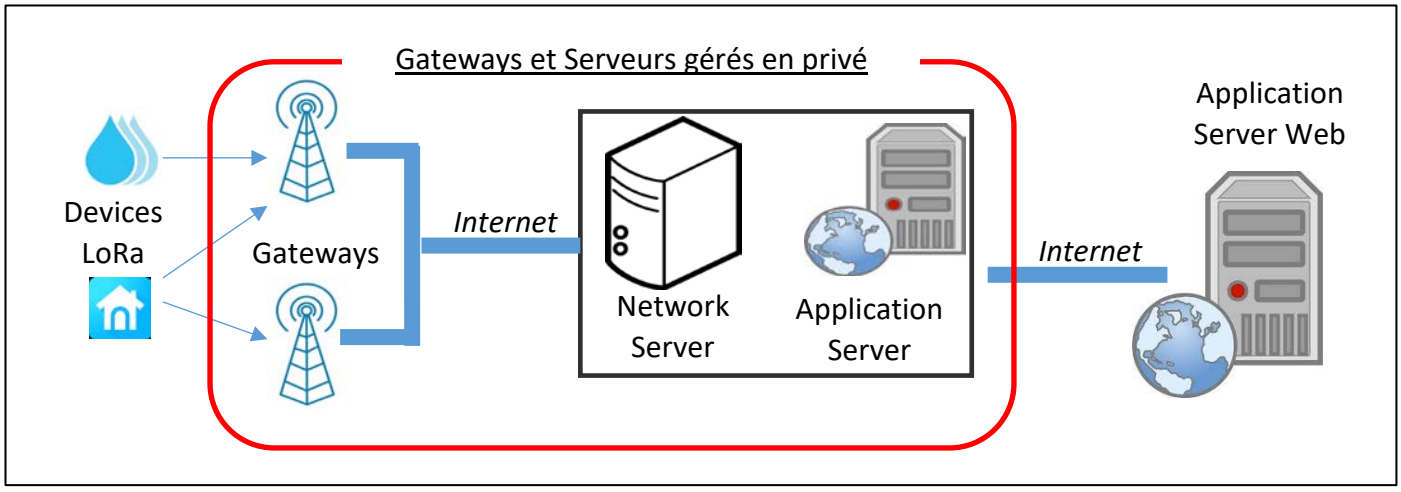


Figure 49 : Infrastructure d'un réseau LoRaWAN privé

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
Coût d'abonnement	Gratuit	Environ 1,5 € / mois par Device LoRa
Coût de l'infrastructure	Investissement important au début (Gateways et Serveurs)	Compris dans l'abonnement
Compétences requises	Demande des compétences en interne à l'entreprise pour la mise en place, et pour la maintenance.	Tout est géré par l'opérateur
Couverture	Optimisée suivant les besoins	Dépendante de l'opérateur choisi. Possibilité de Roaming avec l'international.
Flux Uplink	Illimité dans le respect du Duty-Cycle	Limité suivant l'abonnement
Flux Downlink	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 ses propres Gateways, tout en confiant l'infrastructure du Serveur LoRaWAN à un prestataire afin de limiter les investissements et la maintenance.

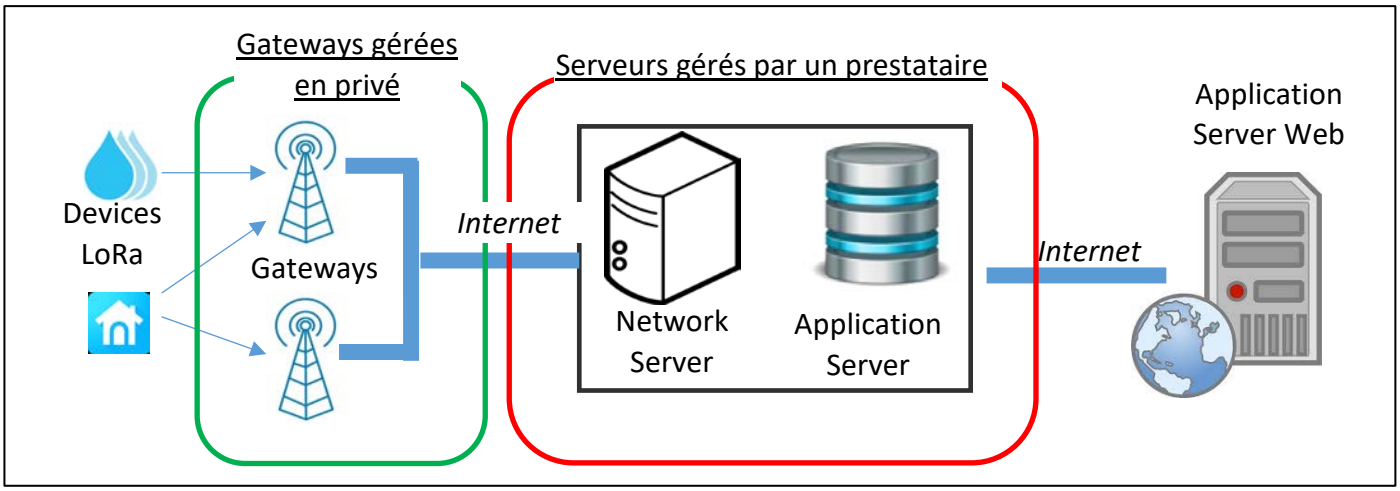


Figure 50 infrastructure d'un réseau LoRaWAN dédié

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 :

- Actility [www.actility.com]

- Loriot [www.loriot.io]
- ResIoT [www.resiot.io]
- The Things Network [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 que nous associons à des serveurs LoRaWAN (TTN, Chirpstack, LORIOT, LoRa Cloud…).

5.1.5 Les zones de couverture

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

Pour connaitre la zone de couverture de notre Gateway utilisée avec TTN, nous pouvons utiliser l'application TTN Mapper. 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 retranscrites sur une carte.

5.2 Paramétrage d'un Serveur LoRaWAN

5.2.1 Exemple de Serveur LoRaWAN

Pour la suite, nous utiliserons à plusieurs reprises le Serveur LoRaWAN nommé The Things Network. La version communautaire est gratuite et Open Source. En revanche, le fait d'utiliser TTN impose à ceux qui enregistrent des Gateways de les mettre à disposition de tous les autres utilisateurs. L'objectif est de réaliser un réseau global ouvert. Il existe aussi une version commerciale de TTN, nommée The Things Industries.

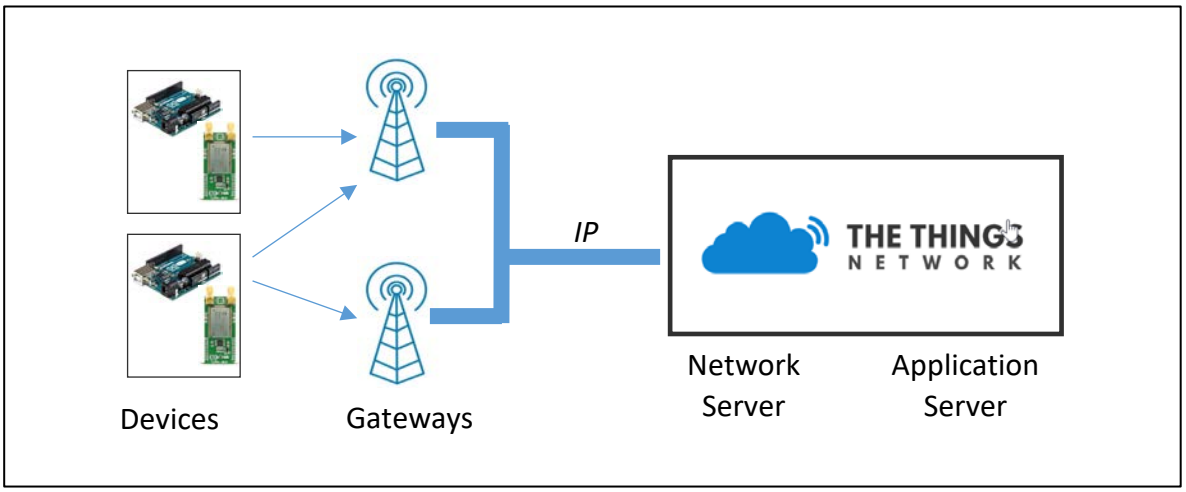


Figure 51 : Schéma de notre application

Il existe de très nombreux autres Serveurs LoRaWAN. Vous retrouvez dans les annexes de ce cours quelques-uns d'entre eux avec leurs adresses ainsi que les principales configurations qui leur sont spécifiques.

5.2.2 Configuration de la Gateway

Il existe de très nombreuses marques de Gateway LoRa. Chaque modèle est destiné à une utilisation particulière (Indoor, Outdoor, prototypage, production…). Dans tous les cas, voici les éléments que nous devons configurer :

- **Le type de Packet Forwarder** : Le Packet Forwarder est le logiciel interne qui déterminera aussi le protocole avec lequel la Gateway communiquera avec le Network Server. Nous donnerons plus d'information à ce sujet au chapitre 9.1.2.
- **L'adresse IP du Network Server** : La Gateway doit connaitre l'adresse IP vers laquelle la Gateway dirigera les données LoRa.
- **Le numéro de port Uplink distant** : La Gateway joindra le Network Server sur un numéro de port spécifique qui doit être connu.
- **Le numéro de port Downlink** : C'est le numéro de port source utilisé par le Network Server pour les trames Downlink à destination de la Gateway.

Par exemple, les Gateways LoRa (Marque ATIM, Modèle A-Gate) que nous avons à l'Université Savoie Mont Blanc ont la configuration suivante :

- Packet Forwarder : UDP Packet Forwarder (SEMTECH).
- Network Server (TTN) : 13.76.168.68 / router.eu.thethings.network
- Port Up : 1700
- Port Down : 1700

Les deux Gateway écoutent sur tous les canaux et sur tous les Spreading Factor (de SF7 à SF12).

5.2.3 Enregistrement des Gateways, Appplications et Devices

Les Gateway : Si vous avez une Gateway à disposition, vous pouvez l'enregistrer dans votre Serveur LoRaWAN. Elle aura alors l'autorisation de transférer des données au serveur. Si cette Gateway est mise à disposition de la communauté, alors tout le monde pourra l'utiliser. C'est par exemple le cas si vous utilisez TTN. En revanche, si vous utilisez un autre Serveur LoRaWAN, vous serez le seul à profiter de votre Gateway… mais vous ne profiterez pas de celles des autres non plus.

Les Applications : Avant d'enregistrer des Devices, il faut créer une application au sein de l'Application Server. C'est là que seront stockées toutes les AppSKey de tous les Devices.

Les Devices : Chaque Device LoRa doit être enregistré au sein d'une application. Pour chaque Device on renseignera la version du protocole LoRaWAN, le DevEUI, ainsi que les clés en fonction du mode d'activation ABP ou OTAA qui est utilisé.

5.2.4 Simulation de Downlink

La communication LoRa est bidirectionnelle. Des données peuvent être transmises au Device LoRa depuis le Serveur LoRaWAN. Dans tous les serveurs il existe un outil permettant de simuler un flux Downlink et donc de transmettre une donnée au Device. Cela est extrêmement utile lors de test de transmission.

Figure 52 : Transmission de données au Device LoRa (TTNv3)

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".

5.2.1 Simulation d'Uplink dans l'Application Server

Dans beaucoup de situations, nous souhaitons valider le fonctionnement de l'Application Server ou 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 parfois dans les paramètres des Devices. Nous avons juste besoin de spécifier le **Frame Payload** en clair et le numéro de Port. Voici un exemple dans TTNv3.

Figure 53 : Simulation d'une trame envoyée par un Device LoRa

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.

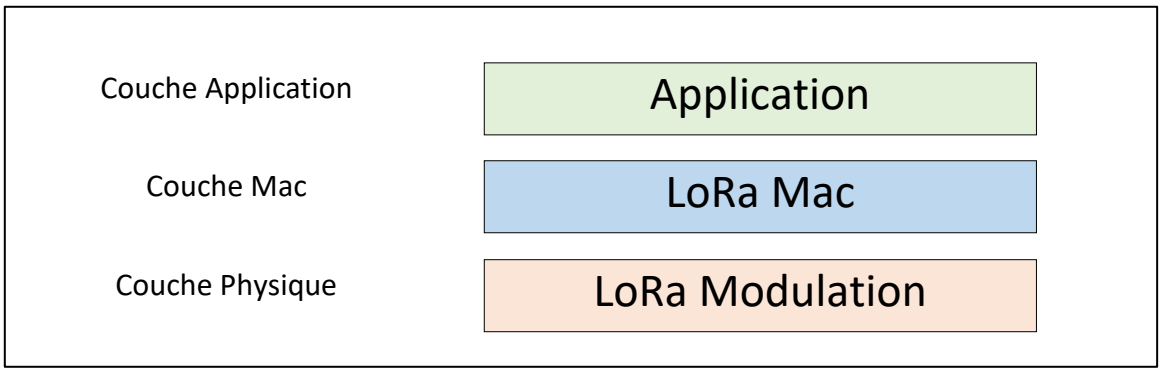


Figure 54 : Les couches protocolaires du LoRaWAN

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 55 :

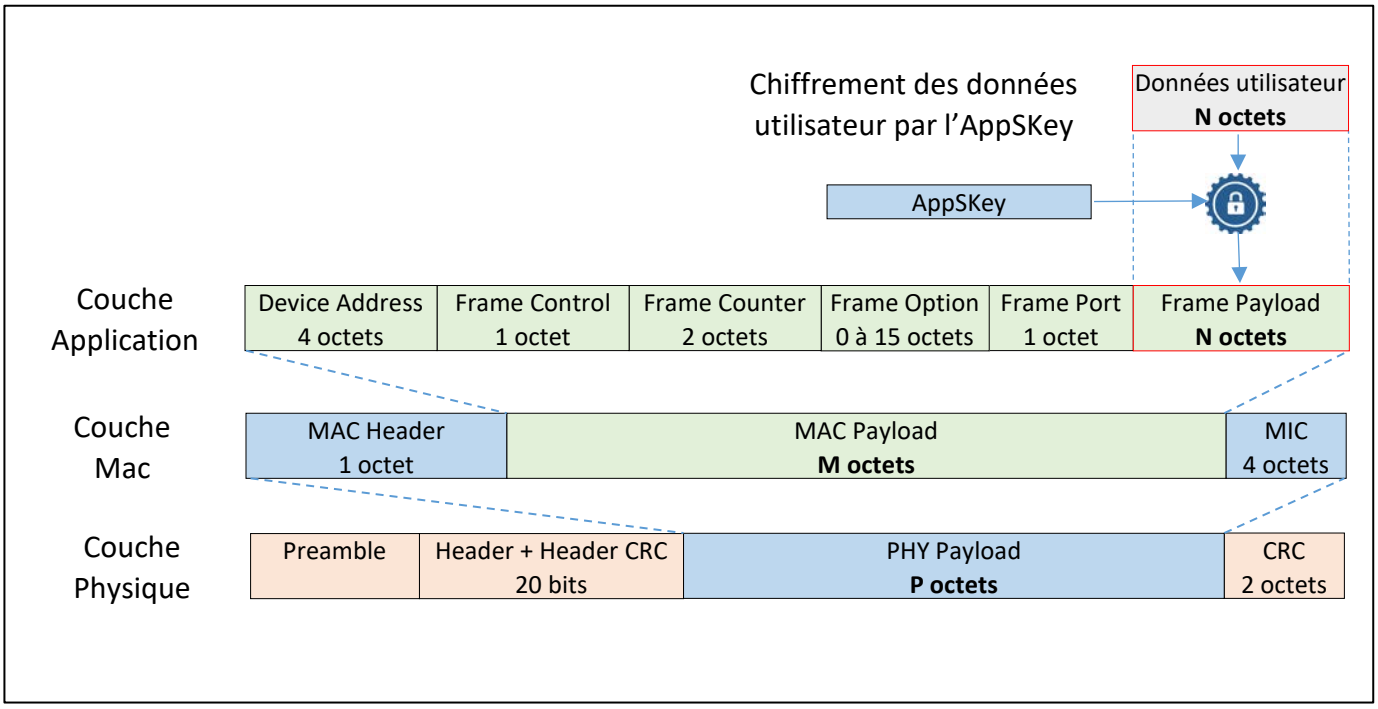


Figure 55 : Trame LORAWAN complète par couche protocolaire

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.

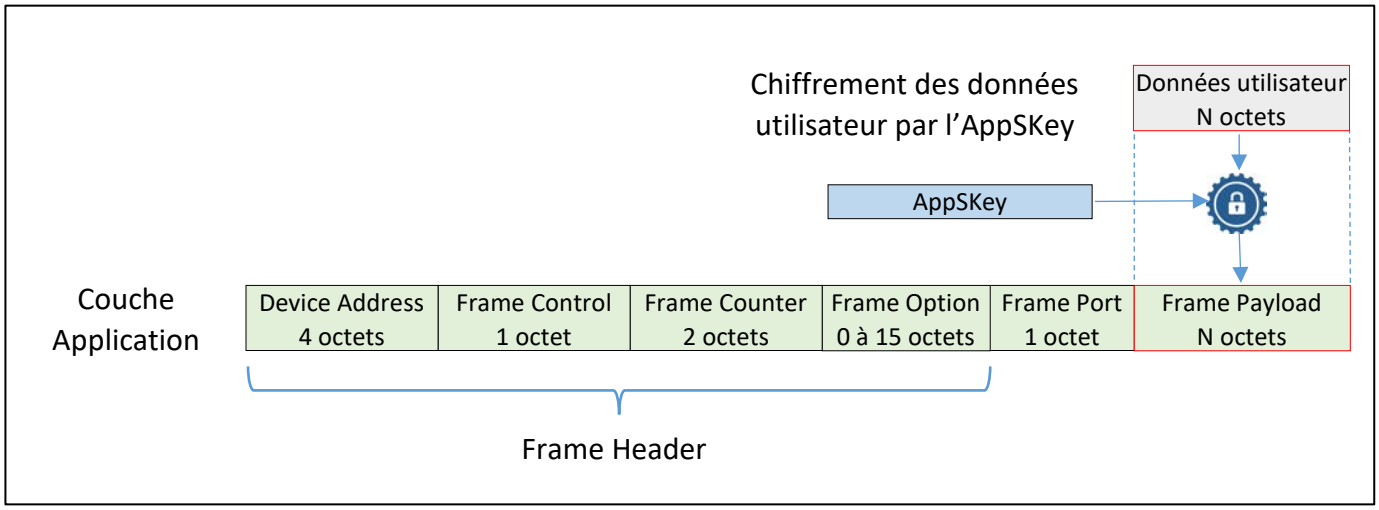


Figure 56 : Trame LORA couche Applicative

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	242 octets
DR 5	SF7	125 KHz	242 octets
DR 6	SF7	250 KHz	242 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	Re-Join-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.

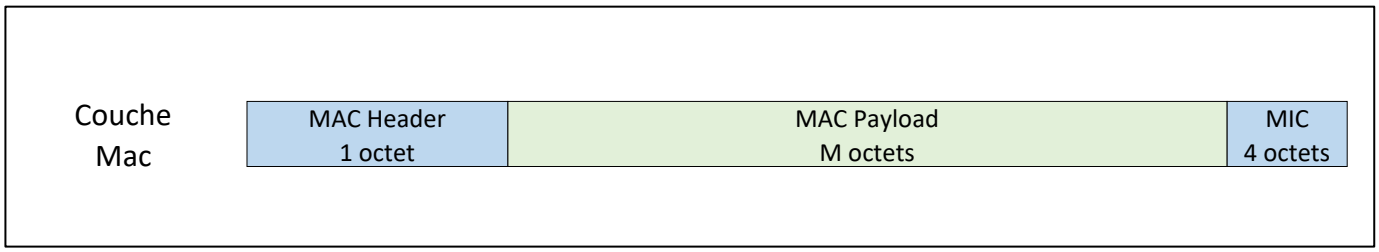


Figure 57 : Trame LORA couche LORA MAC

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 :

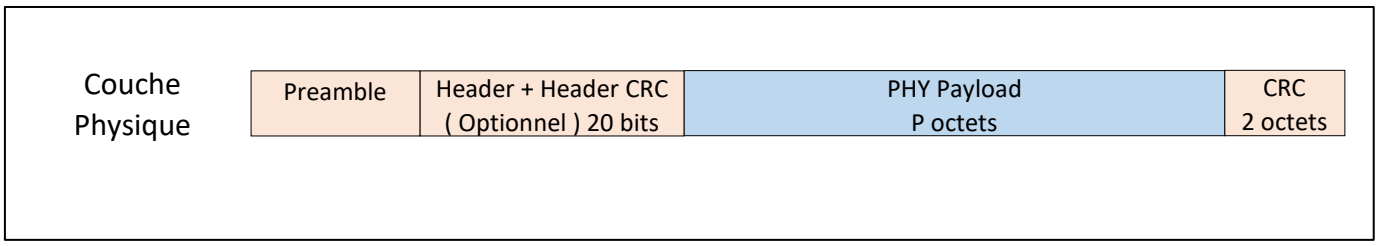


Figure 58 : Trame LORA couche Physique

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

L'entête (**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.

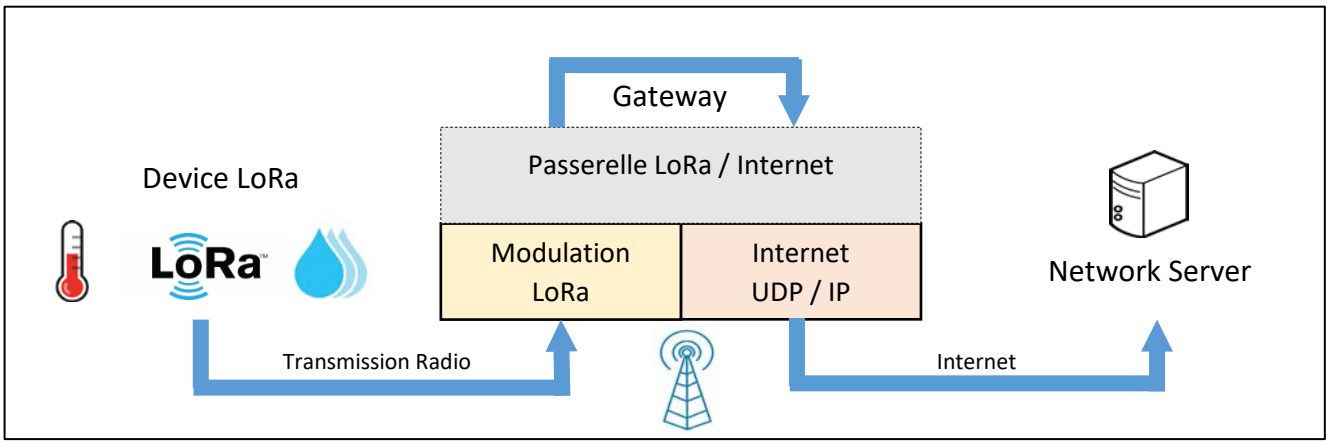


Figure 59 : Rôle de la Gateway LoRa

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.

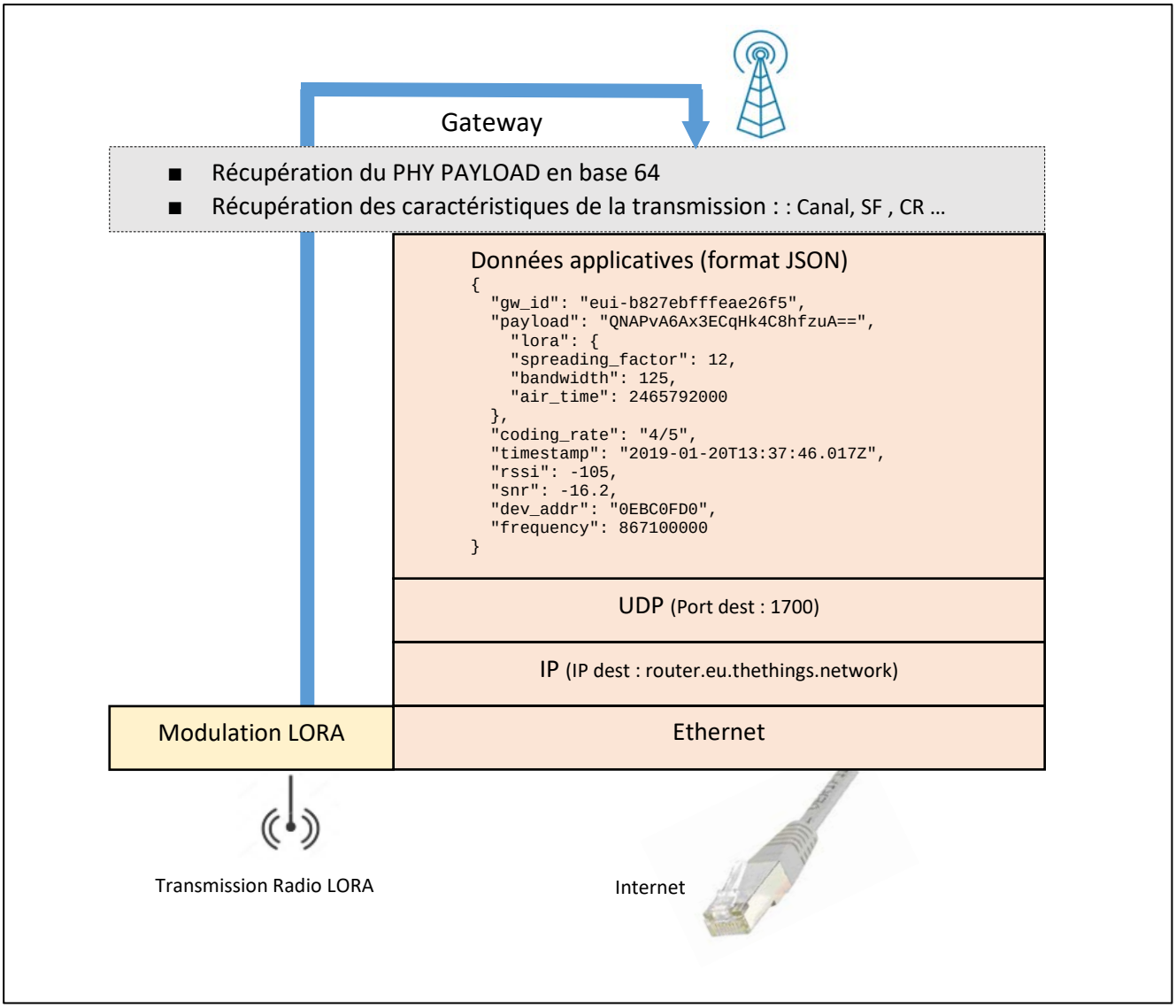


Figure 60 : Passerelle (Gateway) LORAWAN

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 booléen exemple : "service" : true

6.3.2 Utilisation de la base 64

Dans les données applicatives, un champ important est à notre disposition : le Payload, ou plus exactement le PHY Payload. Cela correspond à la représentation de l'ensemble des données binaires transmises dans la trame LoRa (voir Figure 58).

Quelles sont les méthodes à notre disposition pour représenter les données binaires ?

La méthode binaire (base 2): La façon la plus simple et de représenter chacun des éléments qui constitue la trame avec les caractères 0 et 1. Cette méthode est simple, mais elle a un très gros inconvénient : La représentation est visuellement complexe à lire car le nombre d'élément est considérable. Si on souhaite représenter un trame LoRa de 50 octets, il faudrait écrire 400 éléments '0' ou '1'.

La méthode hexadécimal (base 16): On regroupe les bits par 4, ce qui fait 16 combinaisons possibles. Les 16 caractères utilisés sont 0, 1, 2, …, 8, 9, A, B, …, E, F. Cette méthode a l'avantage d'être 4 fois plus compacte que la représentation binaire. Peut-on faire mieux ?

La base 64 : On regroupe les bits par 6, ce qui fait 64 combinaisons possibles. Les 64 caractères utilisés sont celle du tableau suivant :

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 61 : Codage en base 64

Cette méthode a l'avantage d'être 6 fois plus compacte que la représentation binaire. Peut-on faire mieux ?

La base 256 (ASCII) : On regroupe les bits par 8, ce qui fait 256 combinaisons possibles. Les 256 caractères utilisés sont ceux de la table ASCII que vous retrouverez facilement sur le web. Cette méthode à l'avantage d'être 8 fois plus compacte que la représentation binaire. Mais cette représentation a un énorme inconvénient : de très nombreux caractères de cette représentation sont des caractères non imprimables (retour à ligne, espace, EOF, …) et donc ne sont pas visibles. Cette représentation est donc utile pour le codage de texte, mais inutilisable si on souhaite représenter des données binaires.

Le meilleur compromis est donc la base 64. C'est cette méthode qui est souvent utilisé pour représenter le payload de la trame LoRa.

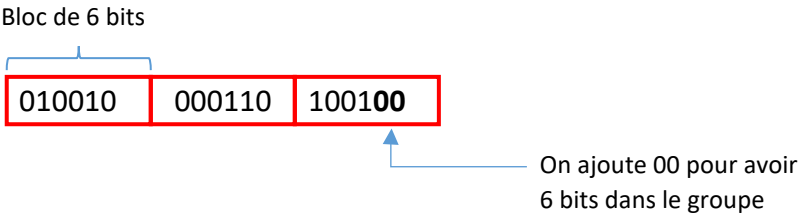
6.3.3 Exemple de codage en Base 64

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.

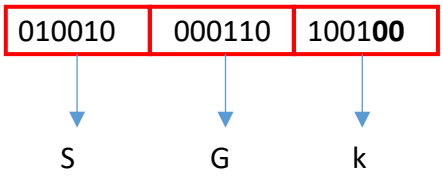
1. On écrit les données à transmettre en binaire

   0x4869 = 0100 1000 0110 1001

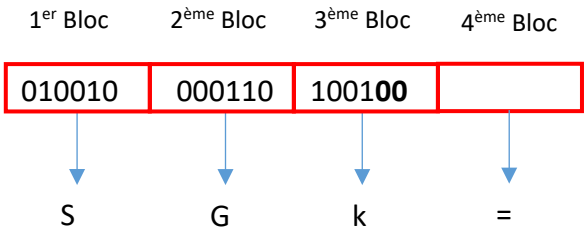
2. On regroupe les éléments binaires par des blocs de 6 bits. Le nombre de bloc de 6 bits doit être un multiple de 4 (minimum 4 blocs). S'il manque des bits pour constituer un groupe de 6 bits, on rajoute des zéros.



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)



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= »

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 **N**etwork **S**erver de TTN reçoit des trames IP en provenance de la Gateway. Comme nous l'avons vu précédemment, un certain nombre d'informations peuvent être retrouvés avec cette trame (DevAddr, SF, Bandwidth, etc…) mais les données Applicatives sont bien sûr chiffrées (avec l'**AppSKey**). A ce niveau de réception (sans connaitre 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-b827ebfffeae26f6",
  "payload": "QNMaASYABwAP1obuUHQ=",
  "f_cnt": 7,
  "lora": {
    "spreading_factor": 7,
    "bandwidth": 125,
    "air_time": 46336000
  },
  "coding_rate": "4/5",
  "timestamp": "2019-03-05T14:00:42.448Z",
  "rssi": -82,
  "snr": 9,
  "dev_addr": "26011AD3",
  "frequency": 867300000
}
```


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

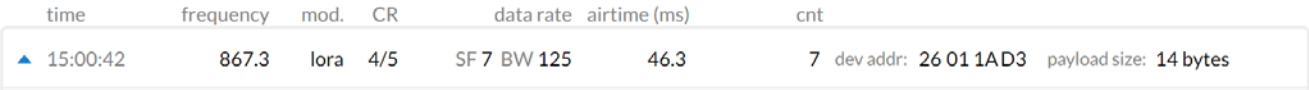


Figure 62 : Trame récupérée sur le « Network Server » de TTN

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 information 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 clairs (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.

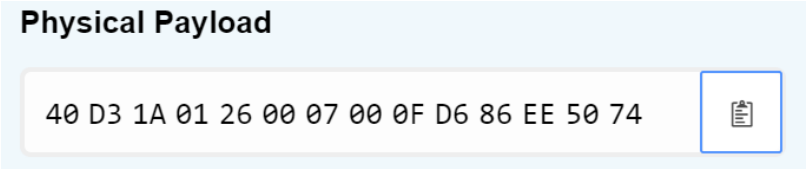


Figure 63 : PHY Payload présenté dans notre Network Server

Sa taille est bien de 14 octets (en hexadécimal) comme précisé sur la Figure 62.

Nous reprenons le format de la trame LoRaWAN vu à la. 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                     =    26011AD3 (Big Endian)
        FCtrl (Frame Control)       =    00 (No ACK, No ADR)
        FCnt (Frame Counter)        =    0007 (Big Endian)
        FOpts (Frame Option)        =
```

- Vous pouvez vérifier l'ensemble de ces informations grâce au [décodeur de trame LoRaWAN](#) (LoRaWAN packet decoder).

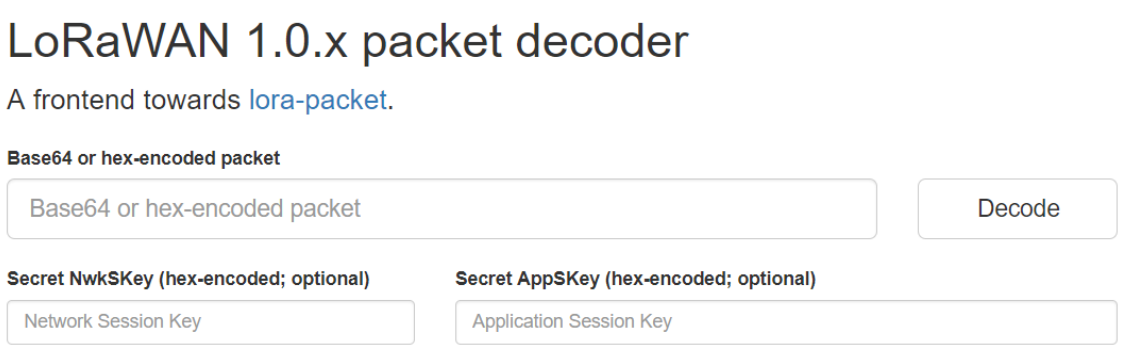


Figure 64 : LoRaWAN packet decoder

6.3.5 Uplink : Du Network Server à l'Application Server

Nous reprenons l'exemple de la trame ci-dessus (Figure 63). Pour information, les clés NwkSKey et AppSKey suivantes ont été utilisées :

- 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.

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**.

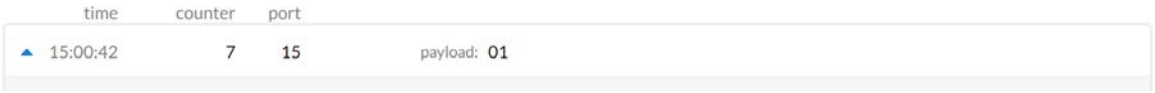


Figure 65 : Trame récupérée sur l'Application Server de TTN


```
{
  "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,
      "rssi": -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 celui-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 Devices LoRa (via le Serveurs LoRaWAN).
- Transmettre des données à destination des Devices LoRa (via le Serveur LoRaWAN).

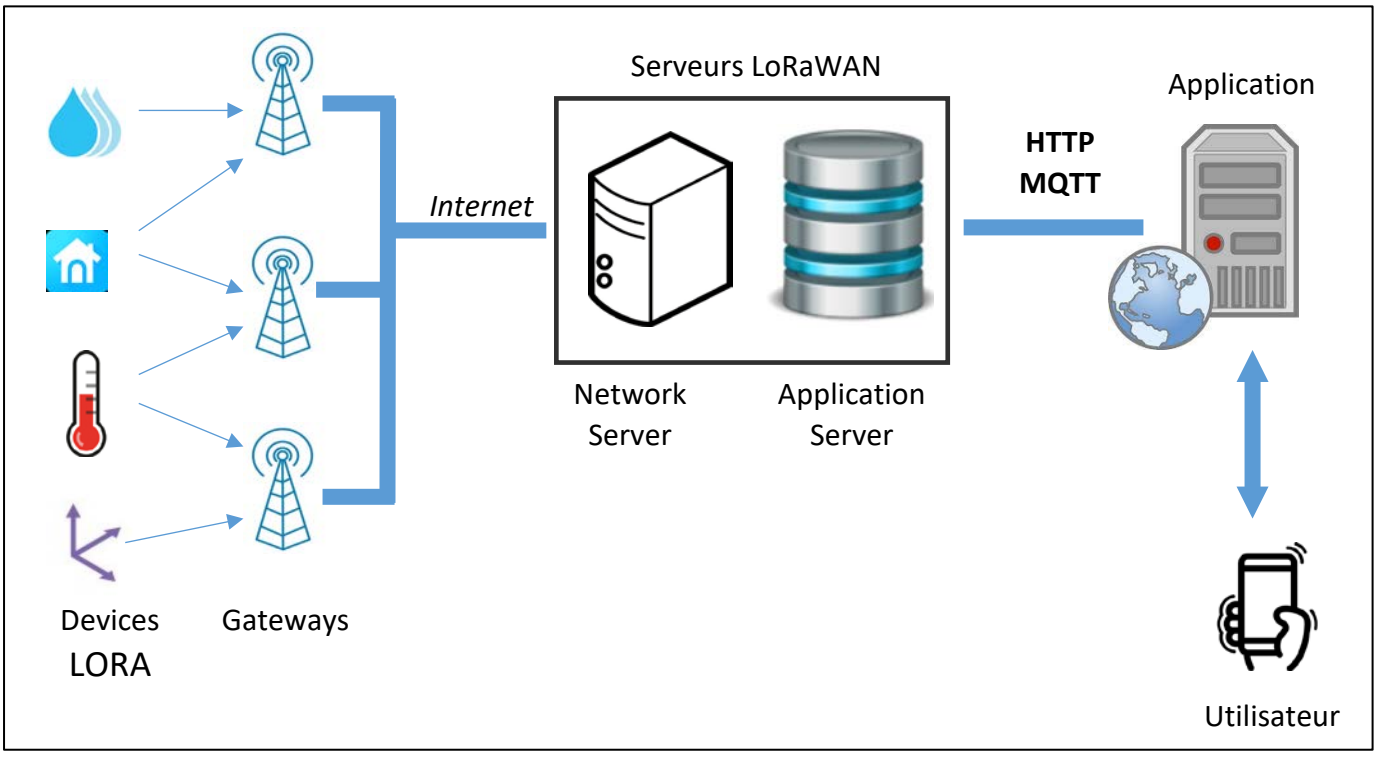


Figure 66 : Structure globale d'un réseau LORAWAN

Attention, les termes utilisés ici sont très proches malgré le fait qu'ils désignent des entités complétement 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 Serveurs LoRaWAN et l'Application peut se faire à l'aide de différents protocoles que nous étudierons dans les prochains paragraphes.

Dans le sens Uplink, notre Application utilisateur aura pour rôle :

1. **De gérer la récupération des données.**
2. De stocker les données (sauvegarde).
3. De les mettre en forme (graphiques, tableaux, …).
4. De les mettre à disposition (interface utilisateur)

Dans le sens Downlink, notre Application utilisateur devra :

1. Présenter une interface utilisateur (Bouton, champ texte, …).
2. Traiter la requête de l'utilisateur.
3. Stocker la requête (sauvegarde).
4. **Envoyer la requête au Serveur LoRaWAN.**

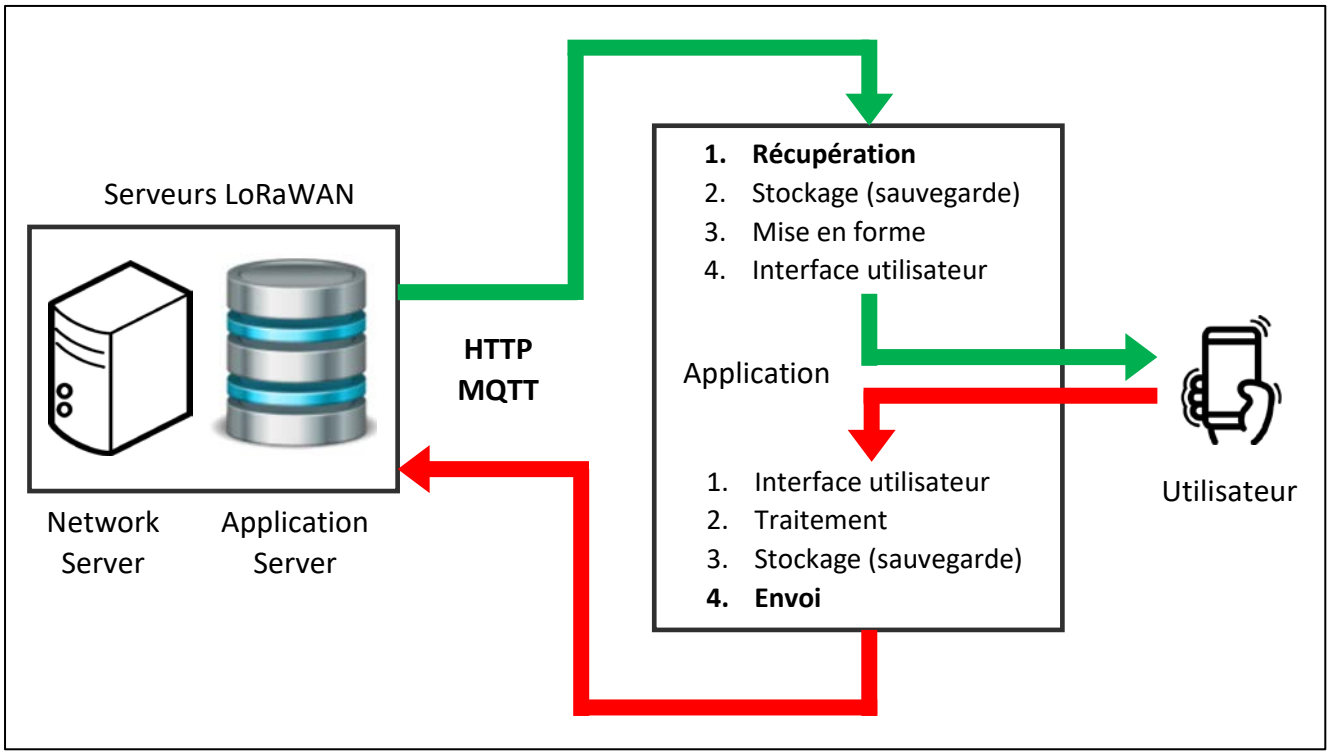


Figure 67 : Détail de l'Application

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 la requête au Serveur LoRaWAN (Downlink)".**

Nous allons voir deux méthodes pour communiquer entre notre Application et TTN : le protocole **HTTP** et le protocole **MQTT.**

Pour les explications de ce chapitre, nous utiliserons à nouveau le Network Server et Application Server de "The Things Network".

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 peut être de tout type : mail, web, fichiers …

La Figure 68 représente un client, un serveur et les deux types de trame 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.

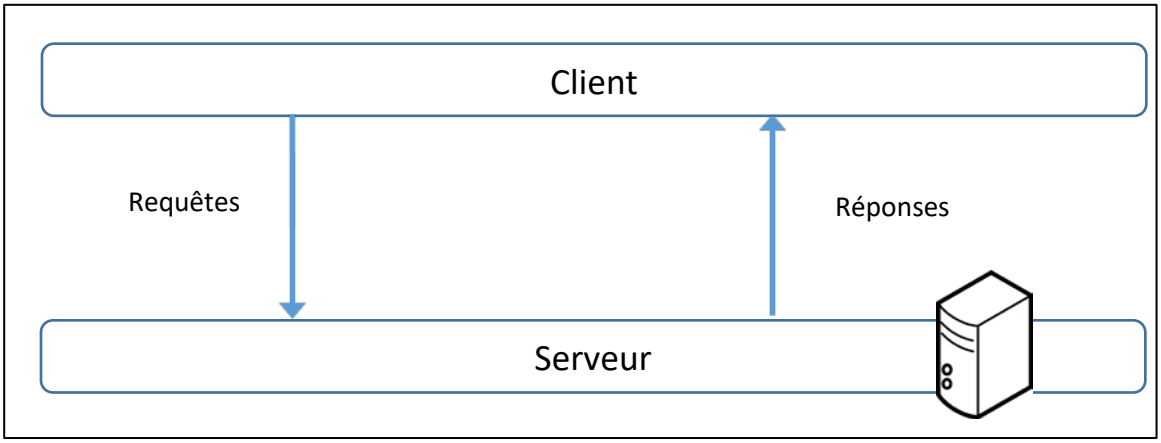


Figure 68 : Client - Serveur et Requêtes - Réponses

7.2.2 Désignation du client et du serveur

Le dialogue s'effectue entre le serveur LoRaWAN et notre Application. La Figure 69 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?"

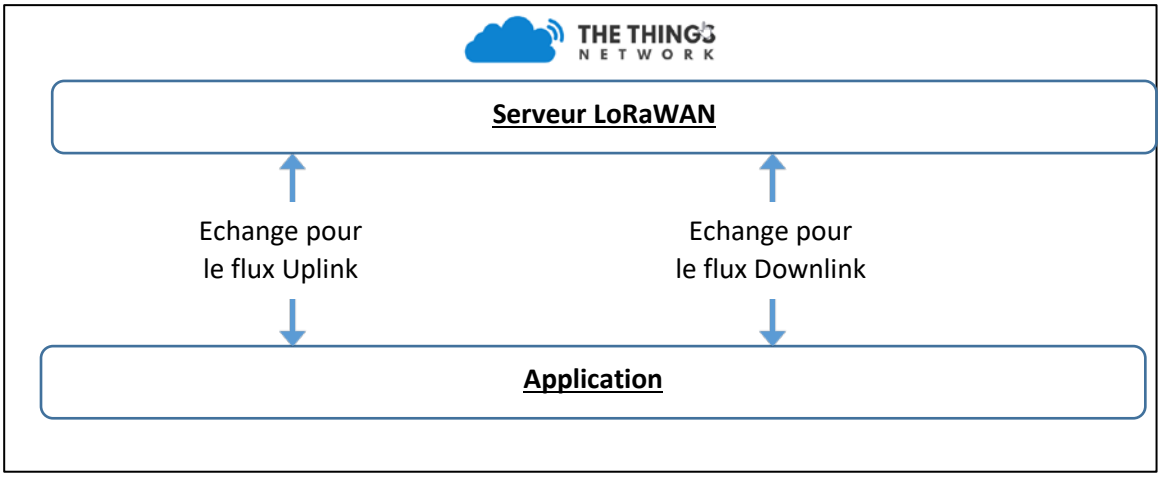


Figure 69 : Communication entre le serveur LoRaWAN et notre Application avec HTTP

Nous allons maintenant étudier les deux situations qui sont le flux Uplink et le flux Downlink représenter sur la Figure 70.

Commençons par la situation la plus courante qui est celle du flux Uplink : 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 fournisse 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.

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

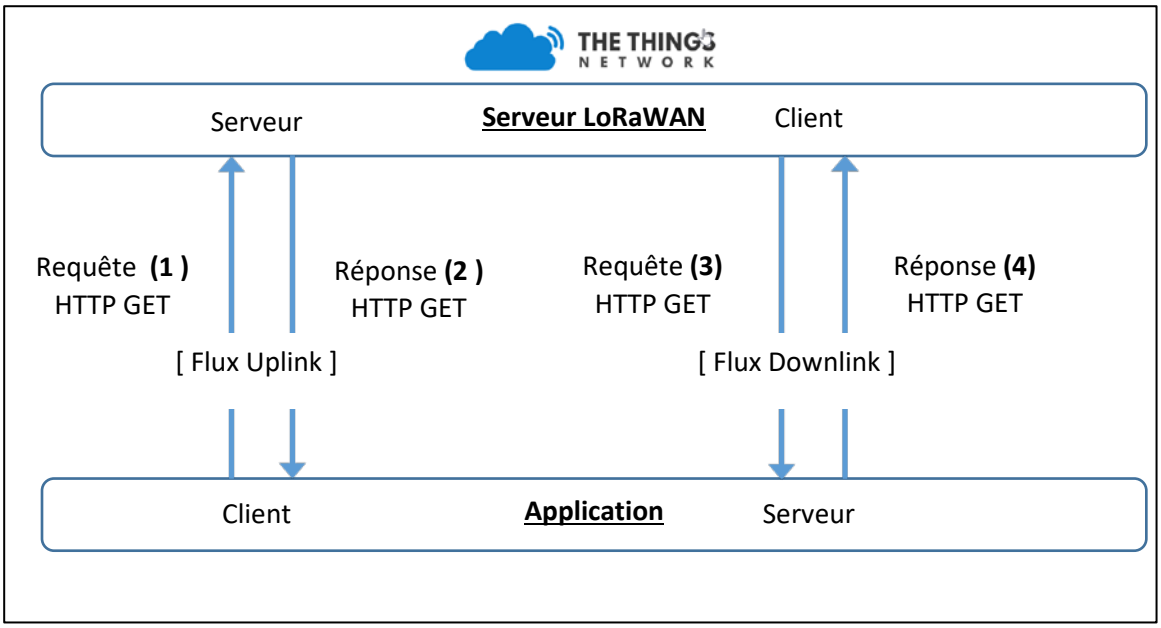


Figure 70 : Flux Uplink et Downlink en HTTP GET

7.2.3 Installation des services HTTP pour le flux Uplink

Nous allons mettre en place la récupération des données présentes sur les serveur LoRaWAN afin de les rapatrier sur notre Application. Sur la Figure 70, cela représente les échanges Uplink, donc les trames **(1)** et **(2)**. Comme le montre la Figure 70, 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.

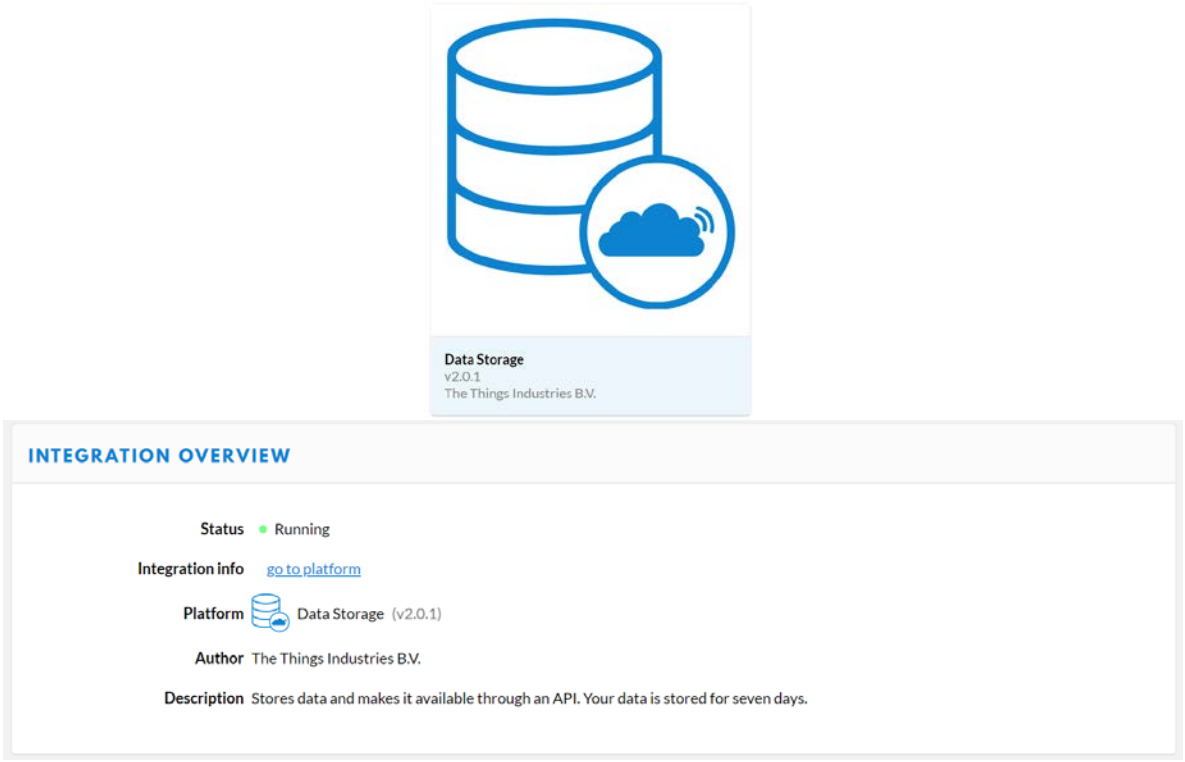


Figure 71 : Intégration du service HTTP dans TTN

Pour prendre connaissance de l'API disponible pour récupérer les données, cliquer sur "go to platform" comme indiqué sur la Figure 71. La Figure 72 présente l'ensemble des APIs, qui rendent les requêtes simples et intuitives.

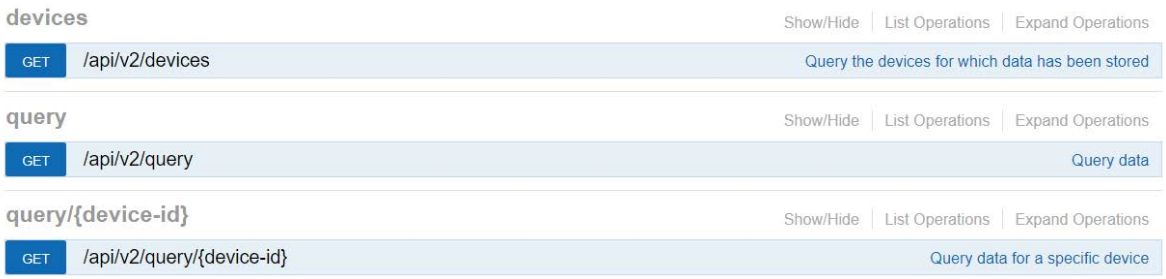


Figure 72 : API REST disponible sur le serveur HTTP

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 qui permet de généré toutes sortes de requête 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 73 :

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

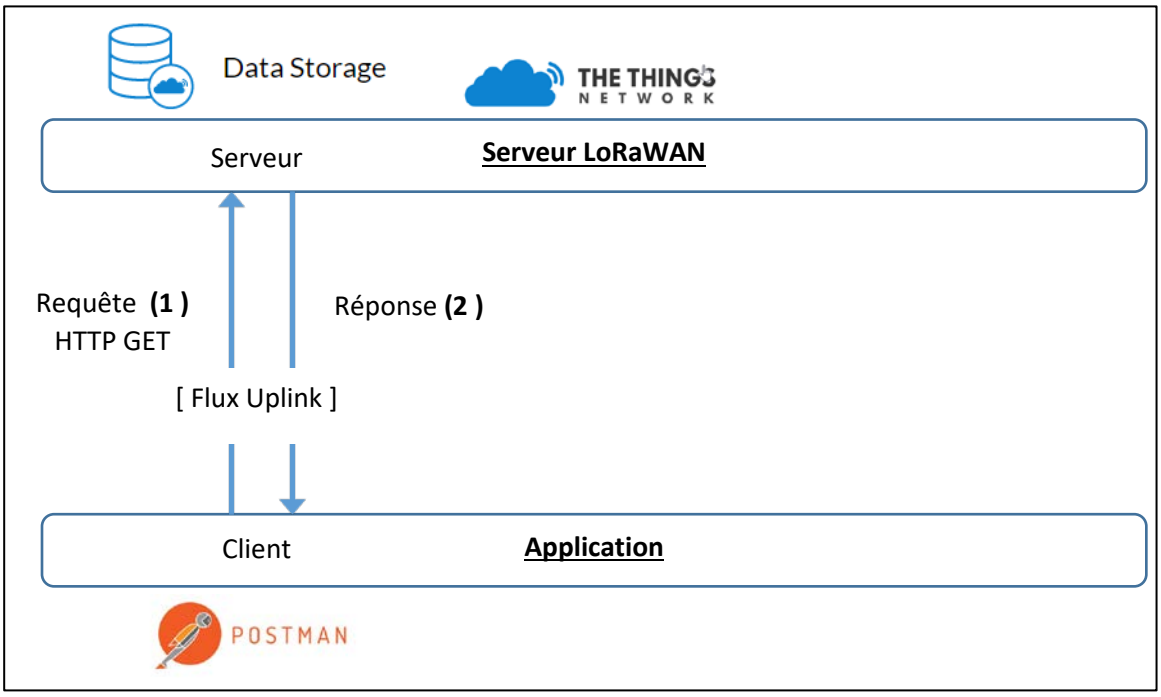


Figure 73 : Flux Uplink et Downlink en HTTP GET

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

- **POSTMAM > New > Request > Save Request**
- Créer la requête suivante :

- Type : HTTP GET
- URL : 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 champ **Key** mettre **Authorization**. Dans le champ **Value** mettre votre clé, précédé 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 > 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 70 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 chance 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 69 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 74.

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.

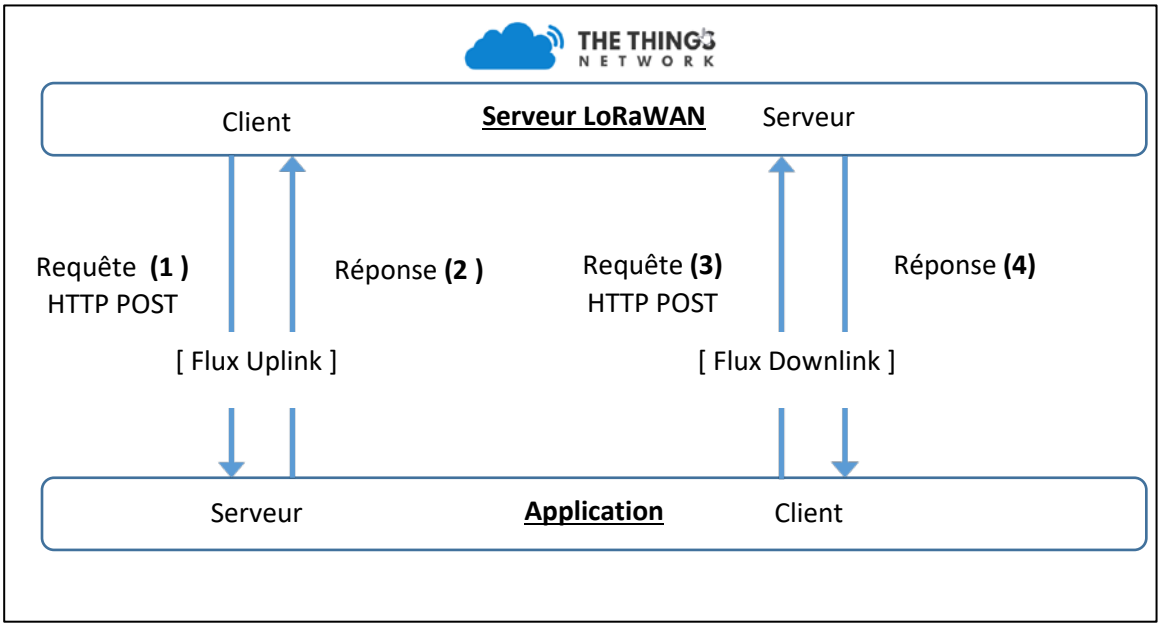


Figure 74 : Flux Uplink et Downlink en HTTP POST

7.3.1 Installation des services HTTP pour le flux Uplink

Nous allons récupérer les données présentes sur les serveur LoRaWAN afin de les rapatrier sur notre Application. Sur la Figure 74, cela représente les échanges Uplink, donc les trames **(1)** et **(2)**. Comme le montre la Figure 74, nous devons 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 apparaitront 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**.

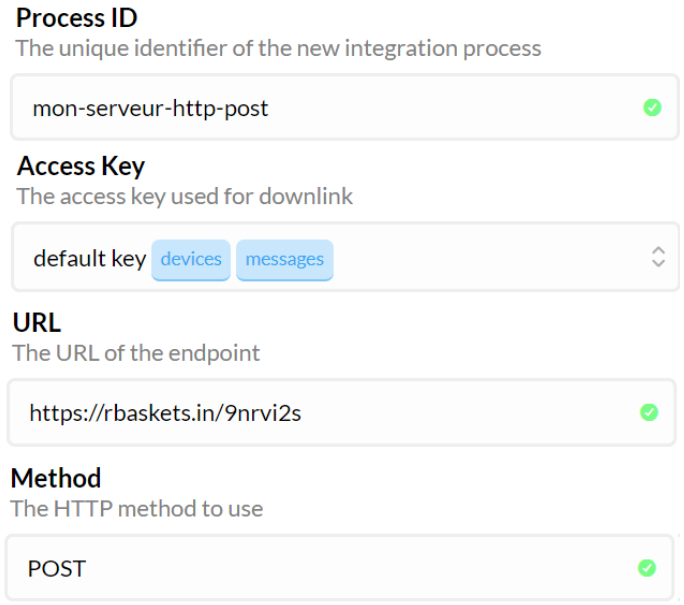


Figure 75 : Ajout d'un client HTTP POST dans TTN

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 au 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. Ce sont les données en 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 74, cela représente les échanges Downlink, donc les trames **(3)** et **(4)**. Comme le montre la Figure 74, 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 une requête HTTP POST :

- **POSTMAM > 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 chaine de caractère « hello ». Vous pouvez utiliser les nombreux encodeur/décodeur 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 76 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 que joui le rôle de serveur (Uplink) et POSTMAN qui joue le rôle de client (Downlink).

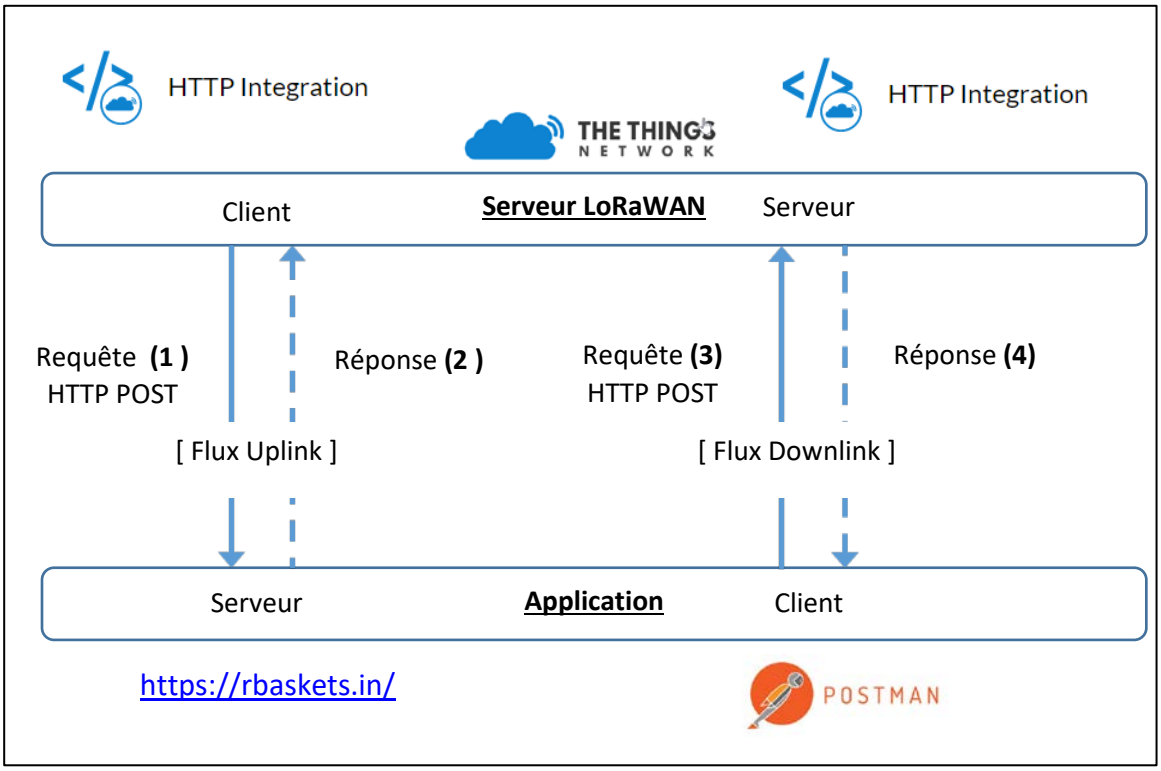


Figure 76 : Flux Uplink et Downlink en HTTP POST

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).

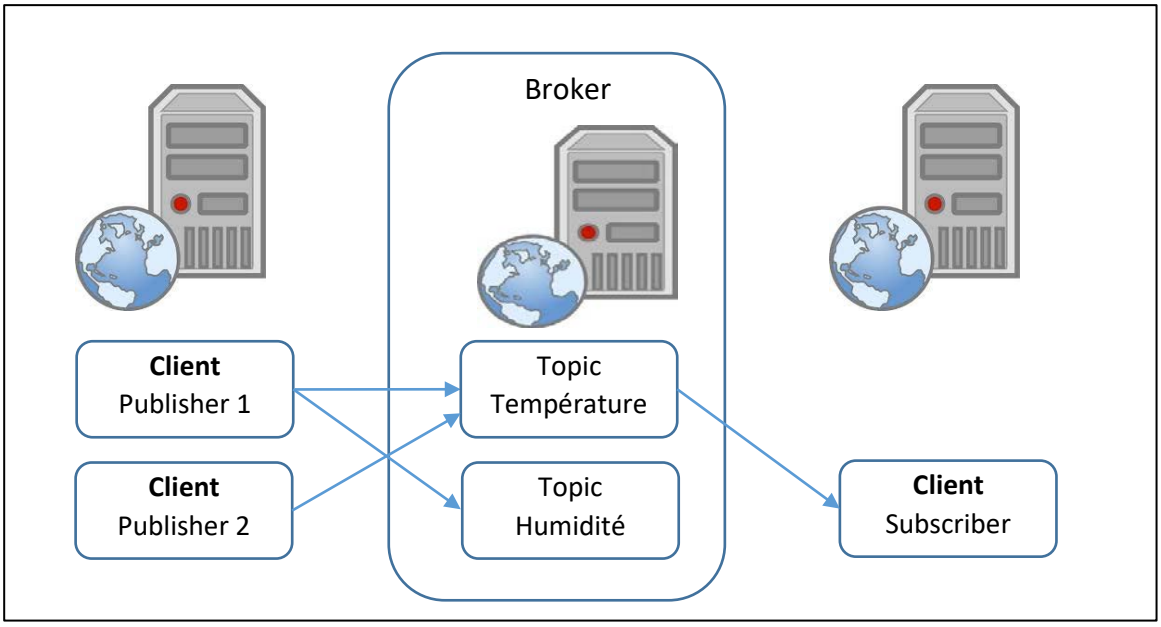


Figure 77 : Modèle Publisher / Subscriber du protocole MQTT

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 :

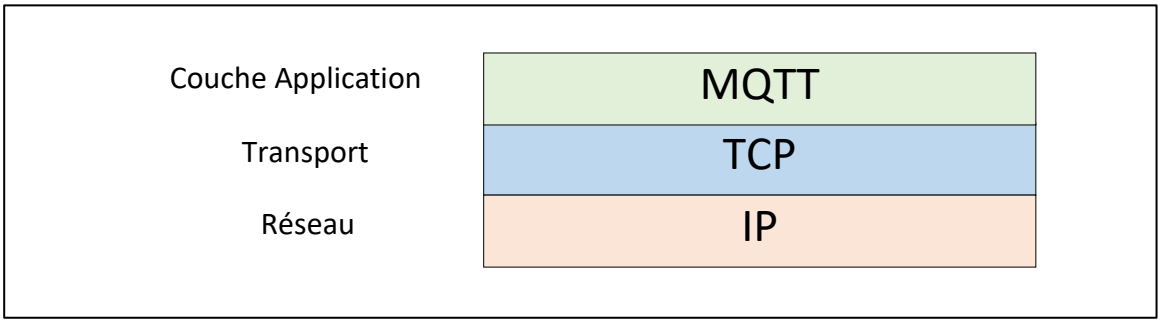


Figure 78 : Protocoles utilisés pour la communication avec MQTT

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

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

Figure 79 : 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 connaitre.**
- **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 ou 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 leur confirmation. Ce mécanisme **garantit** la réception des message 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 ci-dessous montre les trames émises pour chaque niveau de QoS.

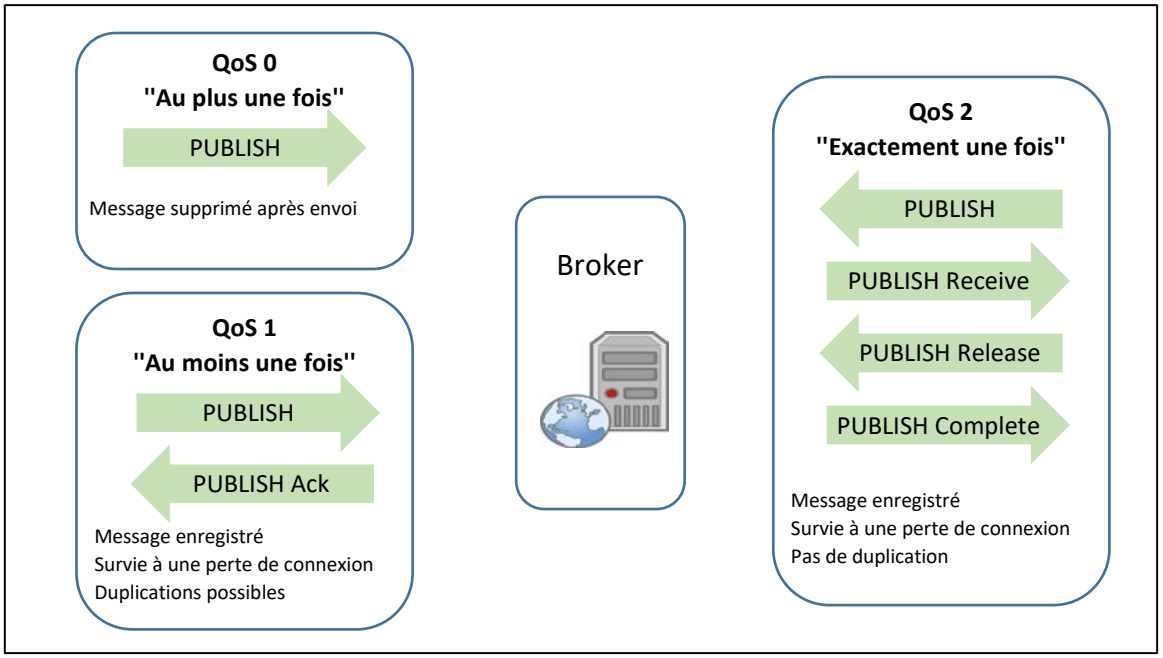


Figure 80 : Qualité de Service en MQTT

La Figure 81 représentes 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 81 : 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 injoignables. 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 réussis 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 82 et le Tableau 16 donne un exemple d'organisation de Topic.

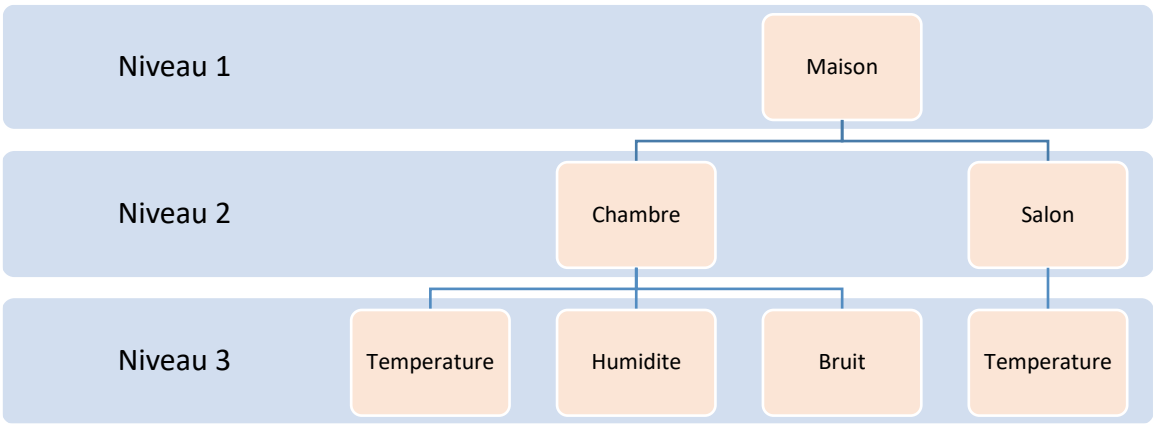


Figure 82 : Exemple de hiérarchie de Topic MQTT

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 chaines de caractères sur le niveau où il est placé.
- Le dièse "#" remplace n'importe quelles chaines 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
- Un client Publisher : MQTT Box sur Windows

- Un client Subscriber : MQTT Box sur Windows

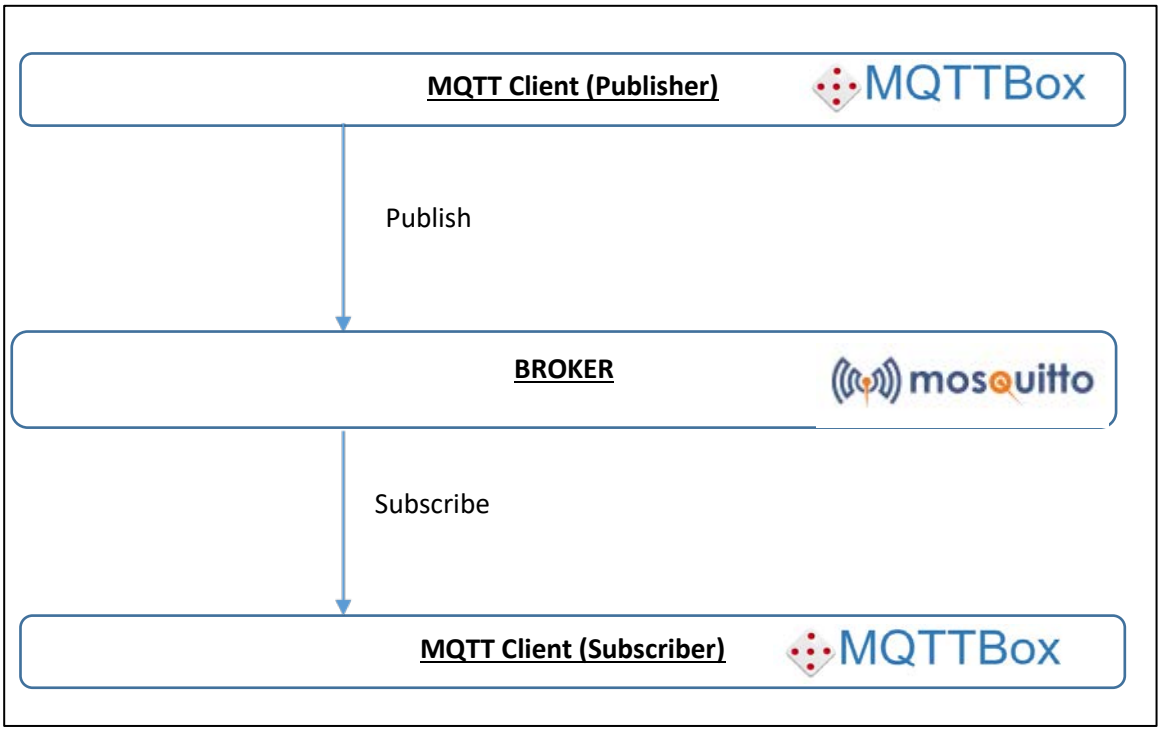


Figure 83 : Test du protocole MQTT

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..

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

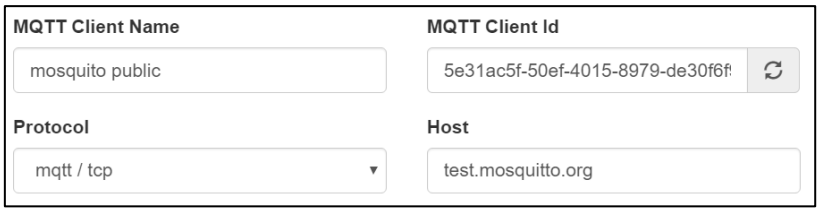


Figure 84 : Configuration d'un Client MQTT dans MQTT Box

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

7.4.8 Récupérer des données en provenance de TTN

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 85. Nos explications feront référence à la trame **(1)**.

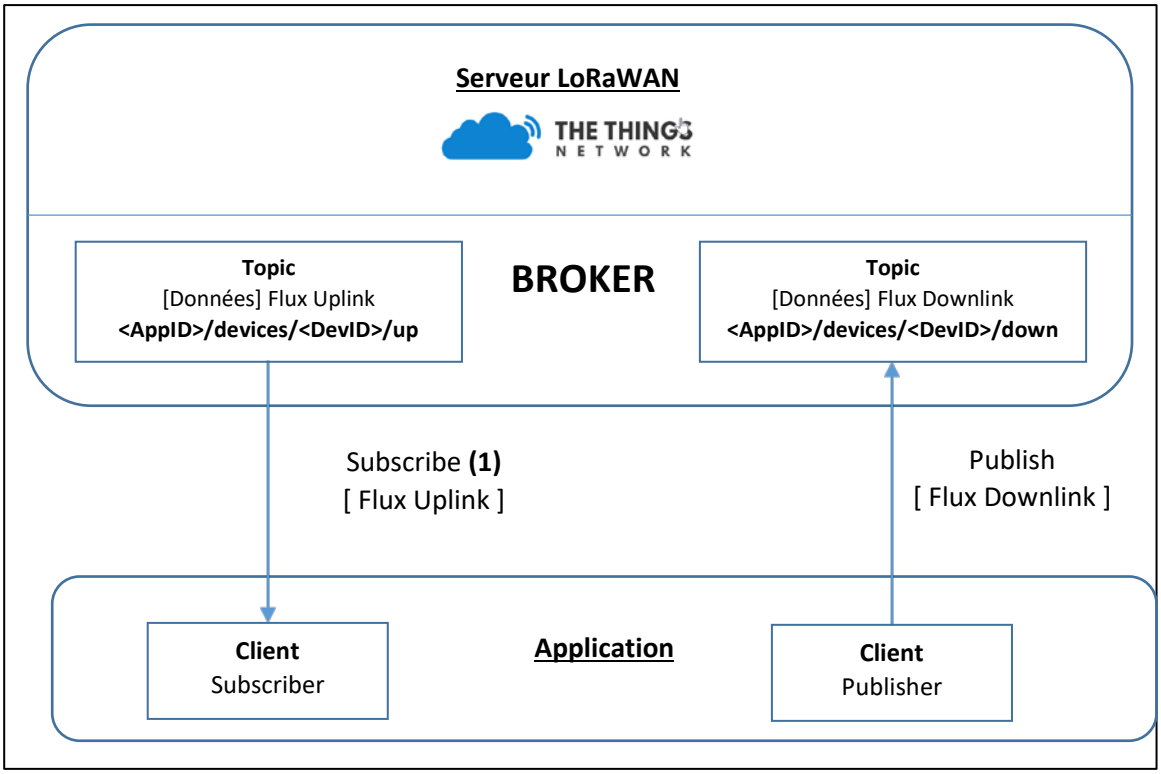


Figure 85 : Mise en place du Broker, des Publishers et des Subscribers

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. Le Username correspond au nom de notre application, c'est-à-dire : **seminaire_lorawan** dans notre cas.
- Password : Le Password est nommé « Access Key » par TTN. Si vous avez enregistré votre propre application, vous trouverez l'Access Key dans : **TTN > Application > Nom_de_votre_application > Overview**.

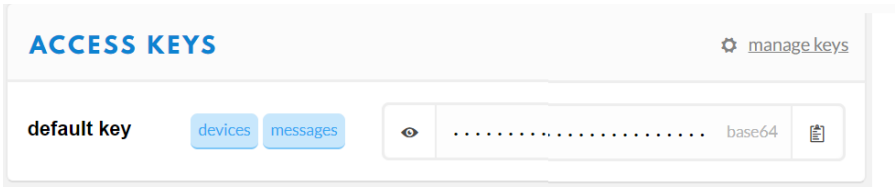


Figure 86 : Access Keys de l'application dans TTN

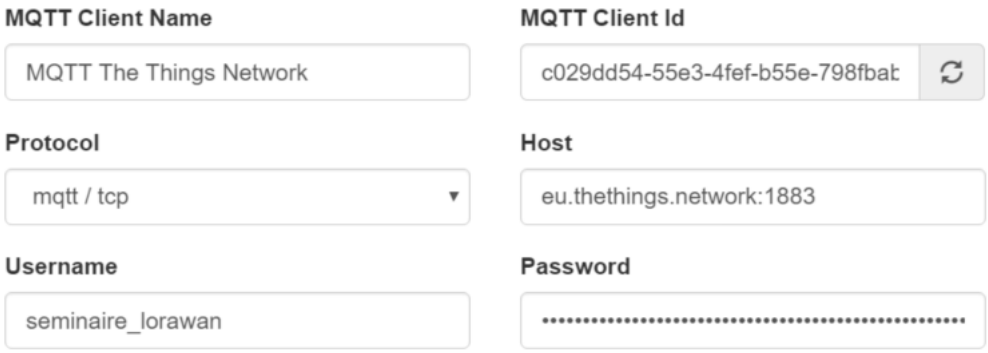


Figure 87 : Configuration du Client dans MQTT Box

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

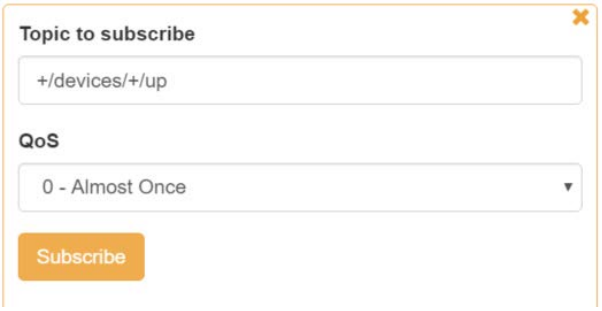


Figure 88 : Configuration du Subscriber

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

Figure 89 : Trame LoRaWAN reçu sur le Subscriber MQTT

Ces données sont écrites en JSON. Nous pouvons les remettre en forme :

```
{
  "applicationID":"3",
  "applicationName":"myApplication",
  "deviceName":"arduino0",
  "devEUI":"0056xxxxxxxx877",
  "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 chaine de caractères "hello" que nous avons émise avec le Device LoRa.

7.4.9 Envoyer des données à destination de TTN

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 85.

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 faite au paragraphe 7.4.8. Il nous reste simplement à ajouter le rôle de Publisher. La configuration est la suivante :

- Topic du Subscriber : **<AppID>/devices/<DevID> /down** où **<AppID>** est à remplacer par le nom de votre application, et **<DevID>** 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
}
```

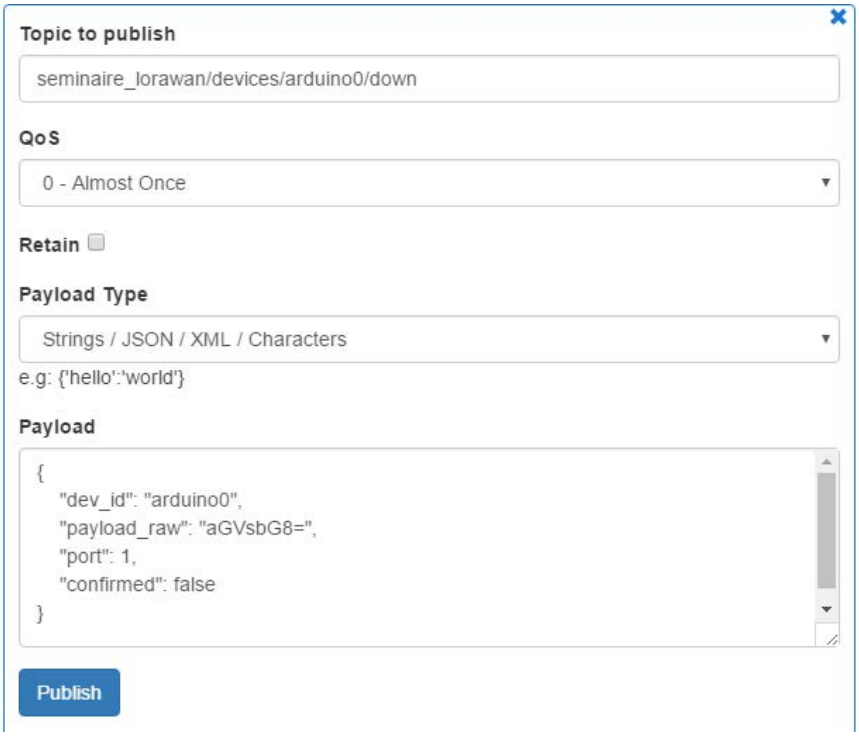


Figure 90 : Configuration du Publisher

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.

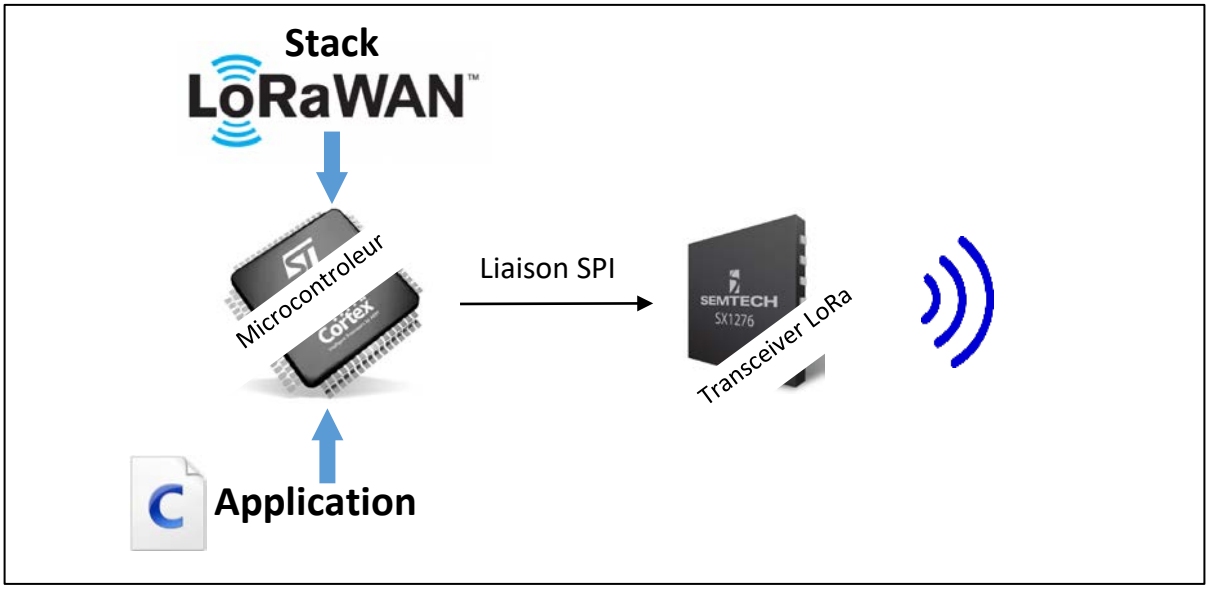


Figure 91 : Device LoRa avec microcontrôleur + Transceiver

Le SX1272 (ou SX1276, ou SX1262) sont des Transceiver 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.

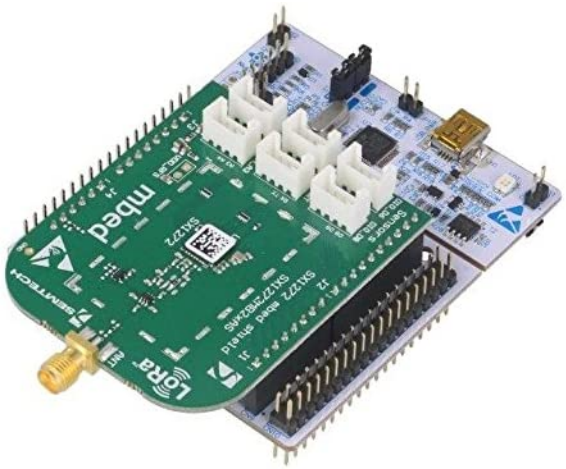


Figure 92 : P-NUCLEO-LRWAN1 package

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 93 : 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).
- Un Transceiver.

Dans ce module, il y a donc plusieurs composants. 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 périphériques à disposition (ADC, I2C, UART, GPIO…) pour mettre en œuvre l'application du Device LoRa.

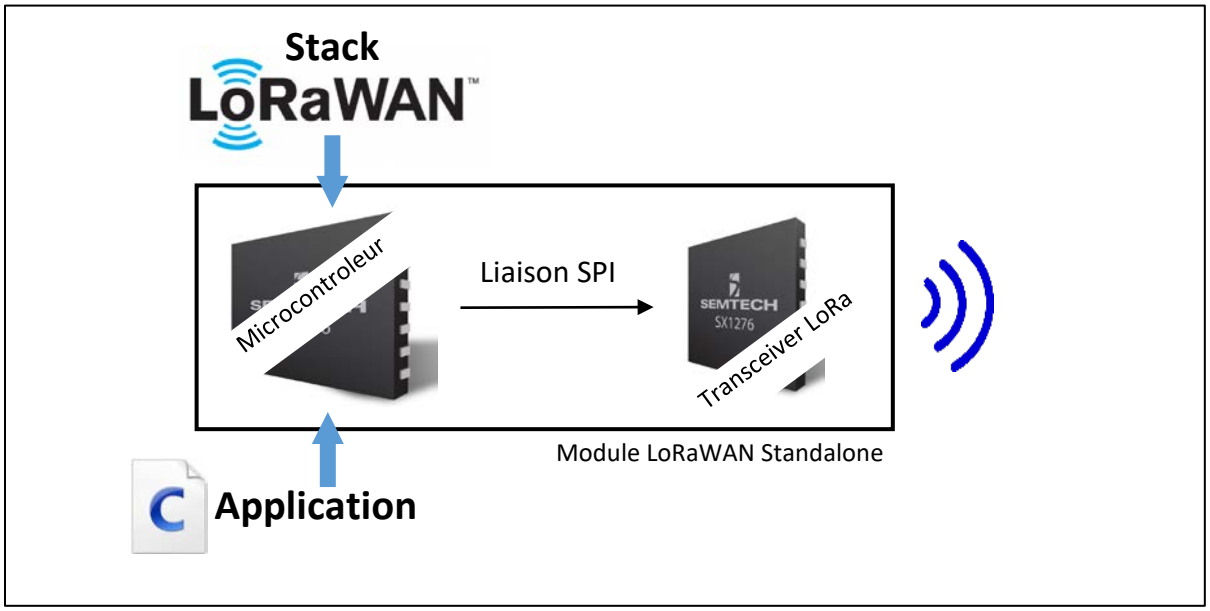


Figure 94 : Module LoRaWAN Standalone

8.2.1 Exemple de module

Le Murata : CMWX1ZZABZ peut fonctionner en Standalone. Il n'aura donc pas besoin d'autres composants pour fonctionner. Seul l'antenne avec adaptation d'impédance est à rajouter.

Figure 95 : 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.

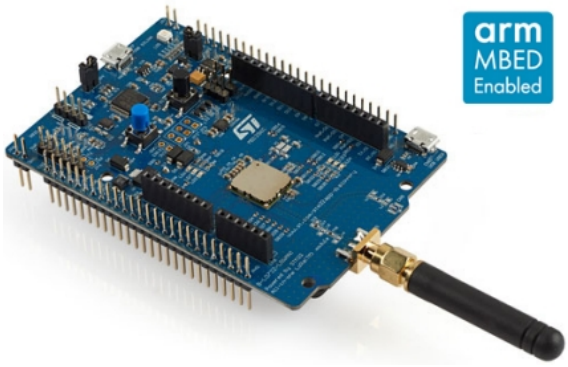


Figure 96 : Carte discovery B-L072Z-LRWAN1 de chez ST

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 97.

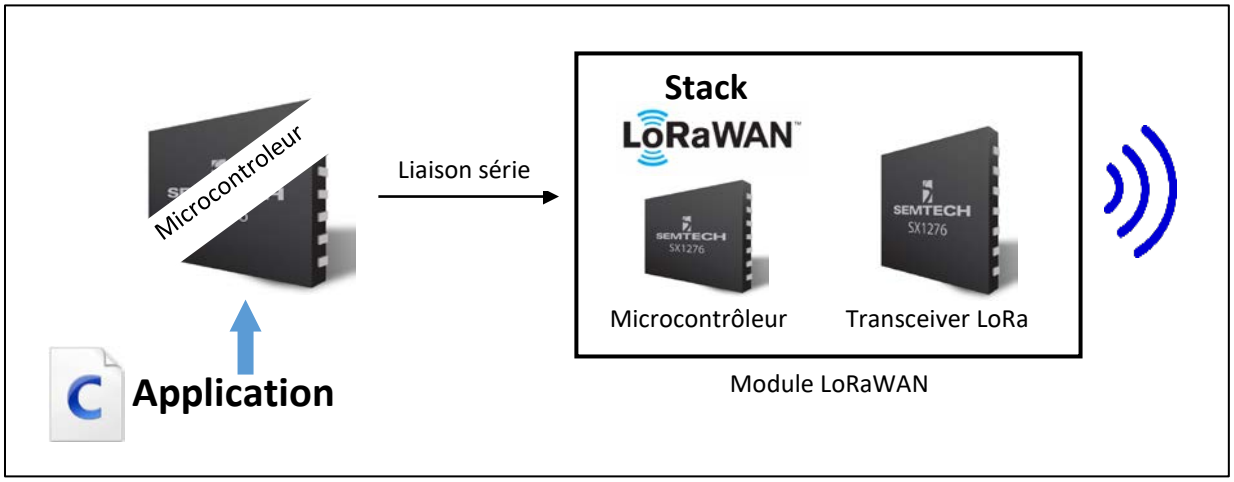


Figure 97 : Device LoRa avec microcontrôleur et module LoRaWAN

Nous pouvons choisir n'importe quel µC 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.

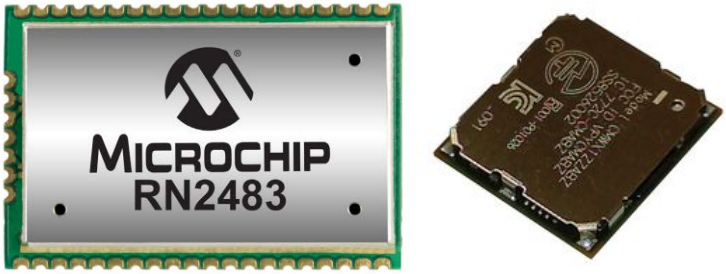


Figure 98 : Module intégrant stack LoRaWAN et Transceiver

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 évidement 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.

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

8.3.1 Exemple de carte de développement

Voici deux exemples de cartes de développement couramment utilisées.

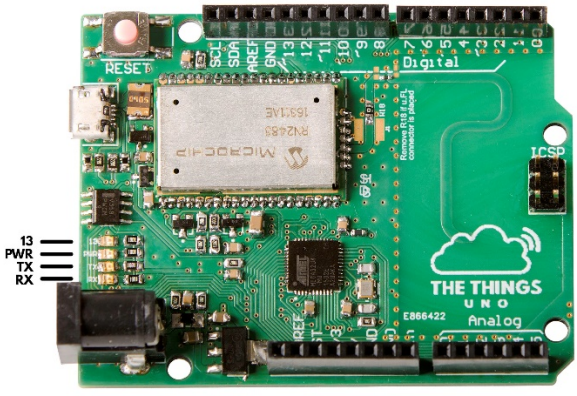


Figure 99 : Microcontrôleur ATMEL + Module RN2483

Figure 100 : 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 coût, de consommation et de surface utilisée.

8.5 Résumé des architectures

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

	Microcontrôleur + Transceiver	**Module LoRa Standalone**	**Microcontrôleur + Module LoRa**	**Microcontrôleur Wireless LoRa**
Encombrement	Moyen	Faible	Elevé	Très faible
Coût	Elevé	Moyen	Très élevé	Faible
Complexité logicielle	Elevée	Elevée	Faible	Elevée

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.
- 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 qui est utilisée lorsqu'on utilise des Arduino :

9 La création de notre propre Network et Application Server

Le réseau LoRaWAN sur lequel nous avons travaillé jusqu'ici était un **réseau dédié hybride** (voir chapitre 5.1.4) : nous utilisions notre propre Gateway mais avec des serveurs LoRaWAN qui ne nous appartenaient pas (TTN). Nous allons maintenant réaliser l'installation de notre propre Network Server et de notre propre Application Server. Cela nous permettra de travailler sur un **réseau LORAWAN privé** (voir chapitre 5.1.2) : ainsi, la Gateway et les serveurs LoRaWAN nous appartiendront.

9.1 Informations préalables

9.1.1 Passage d'un réseau dédié à un réseau privé

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 qui vous appartient. Il faudra donc les reconfigurer pour qu'elles pointent vers les Serveurs LoRaWAN que vous allez mettre en place.

Nous verrons dans ce document l'installation d'un Network Server et d'un Application Server appelé ChirpStack. Dans le cours LoRa - LoRaWAN proposé en vidéo vous verrez aussi l'installation de The Things Stack qui est le serveur LoRaWAN de The Things Network.

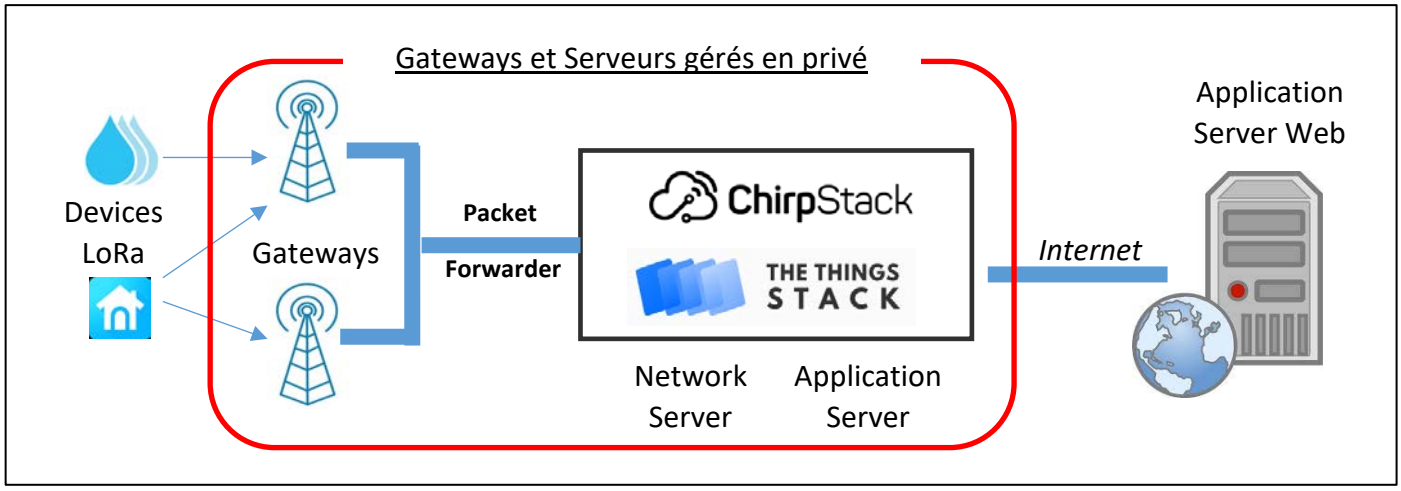


Figure 101 : Infrastructure d'un réseau LoRaWAN privé avec ChirpStack ou The Things Stack

9.1.2 La communication entre Gateways et Serveurs LoRaWAN

Avant de faire l'installation des serveurs LoRaWAN, nous devons bien comprendre l'interaction entre la Gateway et les serveurs représentés sur la Figure 102.

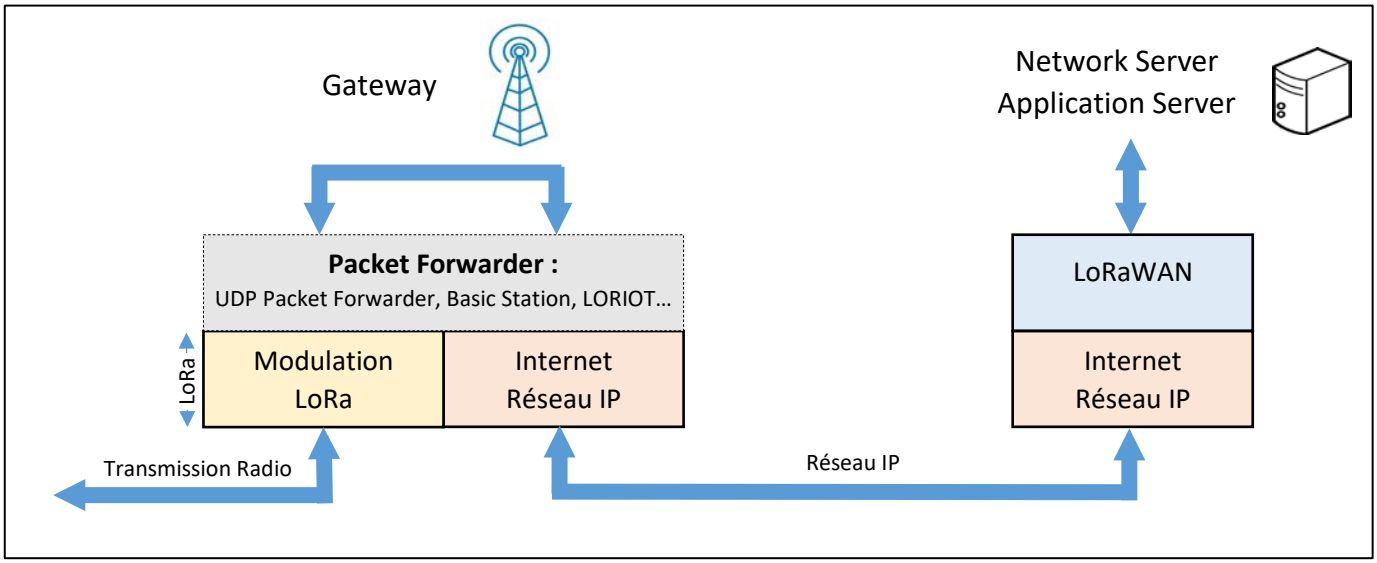


Figure 102 : Gateway et Serveur LoRaWAN

Sur la Gateway, un logiciel appelé Packet Forwarder réalise l'interface entre :

- La modulation radio LoRa.
- Le réseau IP.

En fonction du Packet Forwarder qui sera utilisé, le protocole de communication entre la Gateway et le Serveur LoRaWAN sera différent. Il faut donc obligatoirement vérifier la prise en charge du Packet Forwarder sur la Gateway par le Serveur LoRaWAN. Il existe plusieurs Packet Forwarder disponibles dont deux d'entre eux ont été développés par SEMTECH :

- **UDP Packet Forwarder** : C'est le Packet Forwarder qui est le plus utilisé et tous les Serveurs LoRaWAN le supportent. Il est aussi disponible (à ma connaissance) dans toutes les Gateways. Il est simple et léger mais possède de nombreux défauts notamment du point de vue de la sécurité et du manque de fonctionnalité.
- **Basic Station** : Basic Station est le nouveau Packet Forwarder fourni par SEMTECH. Il offre de nombreuses fonctionnalités supplémentaires : TLS, mise à jour logiciel de la Gateway, synchronisation du temps, gestion des Gateways, etc…

D'autres Packet Forwarder sont aussi disponibles comme par exemple **LORIOT Gateway Software** ou **TTN Packet Forwarder**. Malgré les défauts de **UDP Packet Forwarder**, son utilisation lors d'une phase de développement d'un projet ne pose aucun problème. A titre pédagogique, sa simplicité nous permettra de faire de l'analyse de trames échangées entre la Gateway et le Serveur LoRaWAN. C'est ce que nous verrons au paragraphe 9.1.

9.1.3 Mise en place de l'environnement de travail

Pour installer notre Serveur LoRaWAN, il nous faut une machine connectée à internet et accessible par la Gateway. Cela peut être :

- Votre propre machine (Windows, Linux, MacOS).
- Une Raspberry PI ou équivalent.
- Une machine virtuelle (Virtual Box, VMware, …).
- Un serveur loué chez un fournisseur d'infrastructure (OVH, …).

Si vous utilisez une Raspberry PI, ChirpStack propose une image appelée **chirpstack-gateway-os-full.** Cette image est prête à fonctionner et intègre les services Gateway Bridge, le Network Server et l'Application Server qui se lanceront au démarrage de la RPI. Si vous avez choisi cette méthode d'installation, vous pouvez directement passer à sa configuration que nous allons expliquer au chapitre 9.4.

Dans tous les autres cas, nous utiliserons le système **Docker** et **Docker Compose** pour l'installation de notre serveur. Docker nous permettra de réaliser des installations de façon extrêmement simple, sans avoir à gérer les dépendances et librairies spécifiques à chacun des systèmes d'exploitation (Windows, Linux, MacOS) et des processeurs utilisés (ARM, X86). Il agira comme une couche d'abstraction qui isolera le service installé du reste du système.

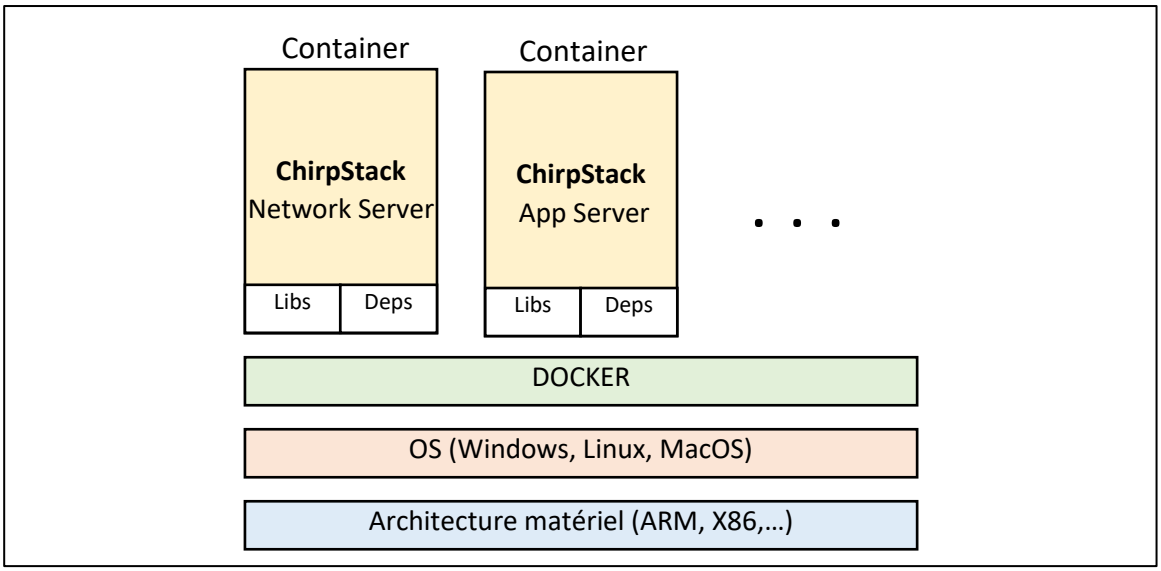


Figure 103 : Utilisation de Docker et Docker Compose pour l'installation des Serveurs LoRaWAN

Nous nous servirons à nouveau de **Docker** et **Docker Compose** plus tard dans le cours lorsque nous créerons notre propre Application, mais pour l'instant nous créerons uniquement les containers nécessaires à ChirpStack.

L'installation de Docker et Docker Compose se réalise de la façon suivante :

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

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 : votre Gateway joue alors le rôle de Gateway ET de Serveur LoRaWAN. 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 présenté dans sa documentation et est simplifié sur la Figure 104 :

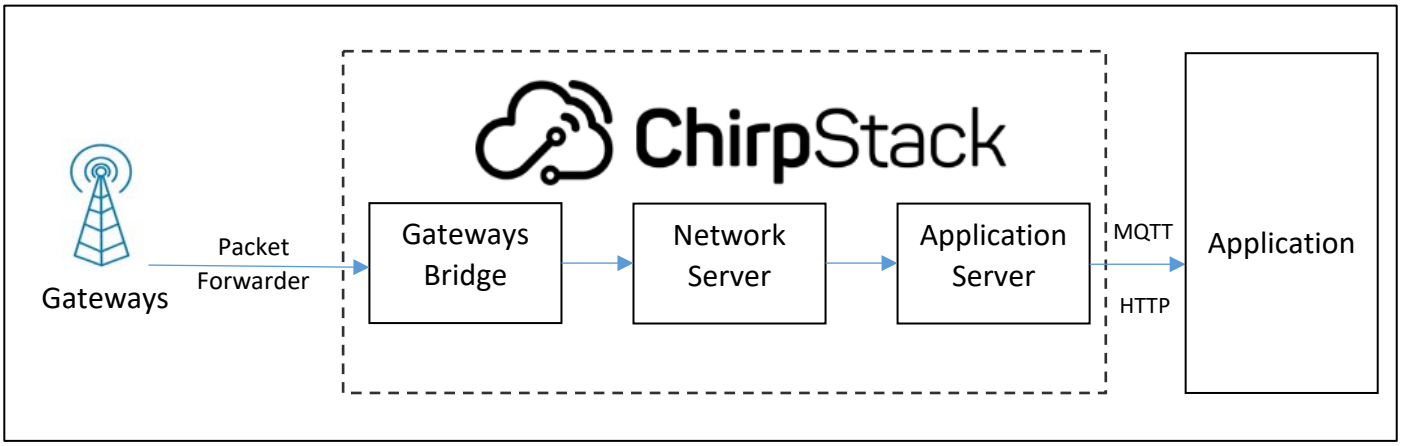


Figure 104 : Architecture détaillée de ChirpStack

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**.

9.2.2 ChirpStack "Gateway Bridge"

Nous avons vu qu'il existe une multitude de protocole permettant la communication entre les Gateways et le Serveur LoRaWAN. La Gateway Bridge est un service qui permet de convertir les différents formats des protocoles Packet Forwarder en un format commun utilisé par le Network Server de ChirpStack.

ChirpStack Gateway Bridge peut s'interfacer avec le **UDP Packet Forwarder** ou **Basic Station**.

9.2.3 ChirpStack "Network Server", "Application Server", Application

Il s'agit de notre Network Server et Application Server qui ont le même rôle que celui que nous avons étudié au chapitre 4.2.3 et 4.2.4.

Comme nous l'avons vu au chapitre 4.2.5, l'Application ne fait pas partie du Serveur LoRaWAN, et ne fait donc pas partie du projet ChirpStack. C'est donc bien toujours à nous de la réaliser comme nous le ferons au chapitre 10. Les protocoles utilisés pour communiquer avec notre Application sont multiples, notamment ceux que nous avons déjà étudiés au chapitre 7.3 (HTTP POST) et au chapitre 7.4 (MQTT).

9.3 Installation de ChirpStack

9.3.1 Installation de ChirpStack

- Télécharger les fichiers d'installation de ChirpStack sur GitHub à l'adresse : https://github.com/brocaar/chirpstack-docker
- Dans le dossier /chirpstack-docker, lancer la commande : **docker-compose up**

L'ensemble des services de chirpstack vont aller se lancer. Parmi eux on reconnaitra :

- Le Gateway Bridge.
- Le Network Server.
- L'Application Server.

La figure représente les services (containeur Docker) générés avec Docker Desktop sous Windows.

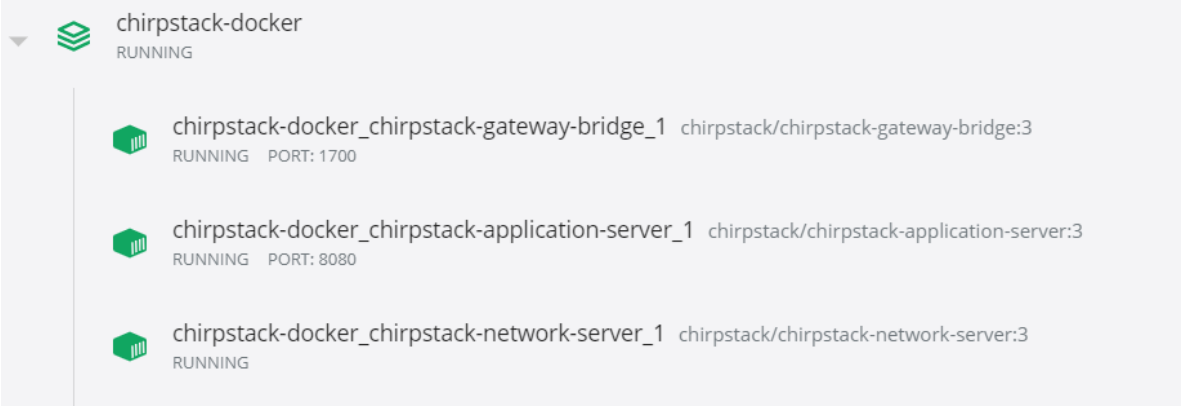


Figure 105 : Containers Docker des 3 services principaux de ChirpStack sous Windows

La figure représente les services (containeur Docker) générés avec Docker Desktop sous Linux avec la commande : **docker ps**

Dans les deux cas, les informations sont similaires et nous montre que les 3 services sont actifs.

```
CONTAINER ID   IMAGE                                  PORTS                    NAMES
ab3af19d89d1   chirpstack/chirpstack-gateway-bridge:3     0.0.0.0:1700->1700/udp   chirpstack-docker_chirpstack-gateway-bridge_1
2ddf064ad1f3   chirpstack/chirpstack-network-server:3                               chirpstack-docker_chirpstack-network-server_1
c210bdb7bef3   chirpstack/chirpstack-application-server:3 0.0.0.0:8080->8080/tcp   chirpstack-docker_chirpstack-application-server_1
```

Figure 106 : Containers Docker des 3 services principaux de ChirpStack sous Linux

Nous pouvons remarquer que deux containers écoutent sur des ports :

- Le service **Gateway Bridge** écoute sur le port 1700/udp. C'est donc vers ce port qu'il faudra que notre Gateway transmette ses informations.

- L'**Application Server** écoute sur le port 8080/tcp. C'est l'accès vers l'interface web pour la configuration de notre Serveur LoRaWAN.

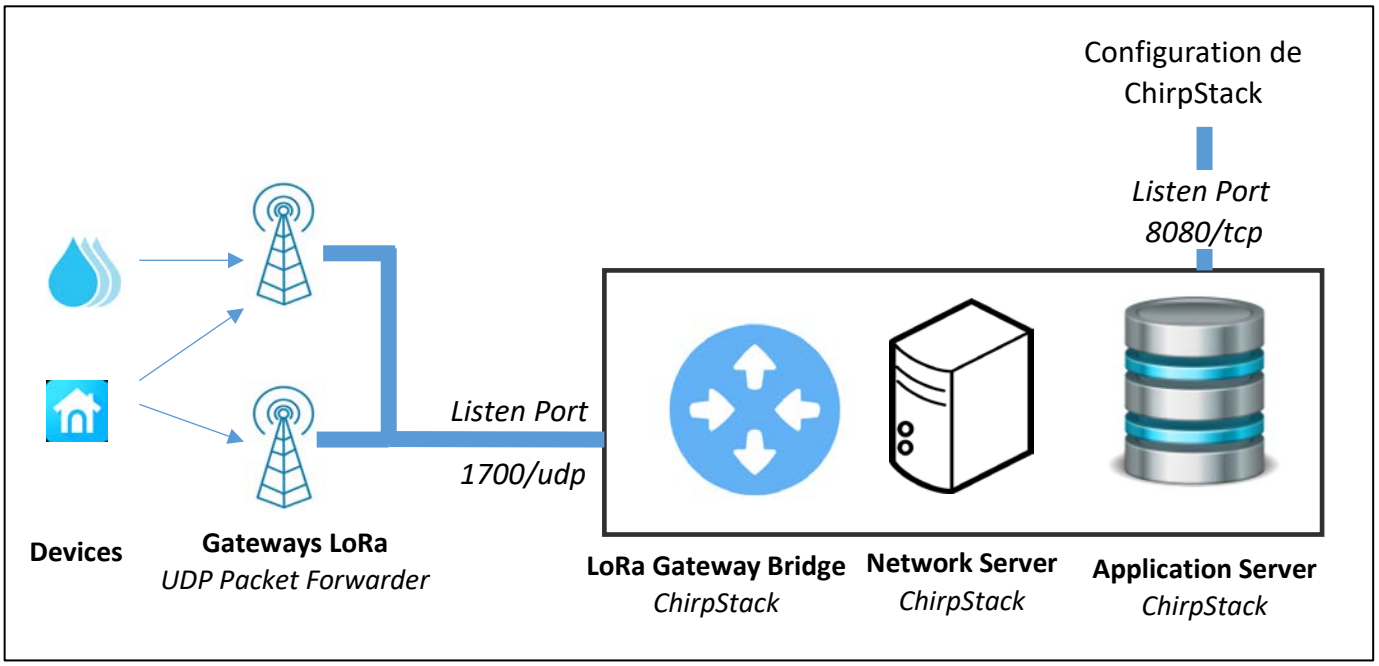


Figure 107 : Architecture globale après installation de ChirpStack

9.4 Configuration de ChirpStack

Après l'installation, la configuration se fait par une interface graphique accessible par le réseau sur le port 8080. Vous devez utiliser un navigateur web et vous connecter sur le serveur ChirpStack sur le port 8080.

- http://localhost:8080 si vous travaillez en local.
- http://@IP_ChirpStack:8080 si vous travaillez à distance.

Les Usernames et Password par défaut sont :

- Username: admin
- Password: admin

9.4.1 Création d'une instance du Network Server

La première étape est de créer une instance du Network Server.: **Network Server > ADD**

La configuration est alors la suivante :

- **Network-server name** : Le nom que vous souhaitez donner à votre instance.
- **Network-server server** : L'adresse IP et le port sur lequel il va écouter. Dans Docker, l'adresse IP est représentée par le nom du container "Network Server" qui a été créé. Sur la Figure 105 et la Figure 106, cela est représenté par le nom "chirpstack-docker_chirpstack-network-server_1". Nous aurons donc la configuration suivante à réaliser.

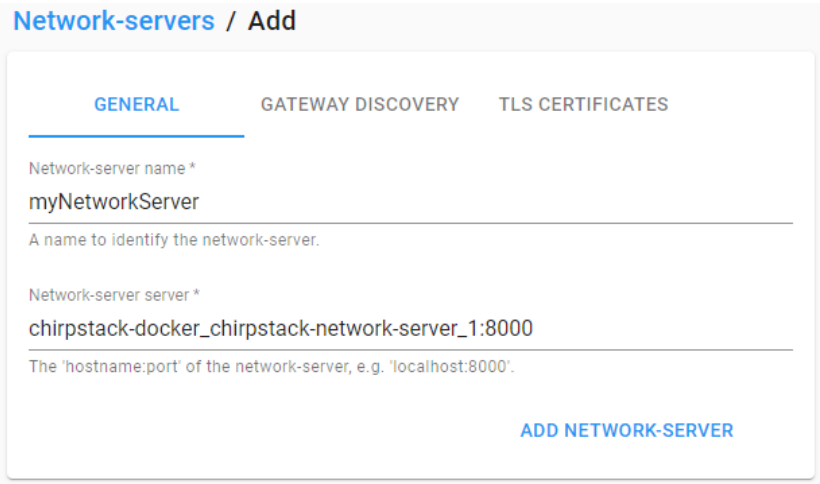


Figure 108 : Enregistrement d'un nouveau Network Server

9.4.2 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 une 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 :

- Application Name : **myApplication**
- Description : Fournir la description de votre choix.
- Service-profile : **myServiceProfile**

9.4.3 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 test en spécifiant que nous utiliserons seulement le mode d'authentification OTAA (voir chapitre 4.4.2).: **Devices profile > Create**

Onglet GENERAL :

- Device-profile name : **myDeviceProfile**
- LoRaWAN MAC version : Version du protocole LoRaWAN de votre Device.
- Max EIRP : Si vous ne connaissez pas la valeur, mettre **16**.
- Uplink interval : Période estimée d'envoi de données. **Cela a peu d'intérêt dans notre test**.

Onglet Join (OTAA/ABP) :

Vous pouvez préciser ici si vous souhaitez travailler en APB ou en OTAA. Cliquer sur "Device supports OTAA" puisque nous faisons l'essai en OTAA.

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

- Device name : **myDevice**
- Device description : Mettre la description de votre Device LoRa.
- Device EUI : Rentrer votre Device EUI.
- Device-profile : **myDeviceProfile**
- Puisque nous travaillons en OTAA, nous n'avons pas besoin de "Disable frame-counter validation" (voir chapitre 4.5.2)

On poursuit la configuration dans **Application > myApplication > Devices > myDevice > KEY(OTAA)**. Il faut alors préciser la clé AppKey.

- Application key : Rentrer votre AppKey.

9.4.4 Enregistrement d'une Gateway

Un Network Server accepte uniquement les données transmises par les Gateways qu'il a enregistrés dans sa base. Dans notre cas, il faudra donc ajouter au moins une Gateway à notre architecture. On ajoute une Gateway dans : **Gateway > Create** avec les informations suivantes :

- Gateway name : **myGateway**
- Device description : Mettre la description de votre Gateway.
- Gateway ID : Rentrer votre identifiant de Gateway.
- Network-server : **myNetworkServer**

9.4.5 Modification de la configuration de notre Gateway

Notre Gateway doit maintenant transmettre les données LoRa à notre serveur ChirpStack que nous venons d'installer. Pour cela, il faudra refaire la configuration de la Gateway comme nous l'avons vu au chapitre 5.2.2. Les informations suivantes devront être renseignées :

- @IP du Network Server : Mettre l'adresse IP du serveur où est installé ChirpStack.
- Port Up d'écoute de ChirpStack: **1700**
- Port Down : **1700**

9.4.6 Visualisation des trames reçues

La configuration minimale de ChirpStack est maintenant terminée. Nous pouvons donc brancher un Device LoRa qui possède les attributs (DevEUI et AppSKey) que nous avons enregistrés dans ChirpStack et le faire émettre.

En allant dans **Application > myApplication > Devices > myDevice > DEVICE DATA**, on peut voir la liste des trames reçues ainsi que les données applicatives déchiffrées (Frame Payload).

Figure 109 : Réception des trames des Devices LoRa

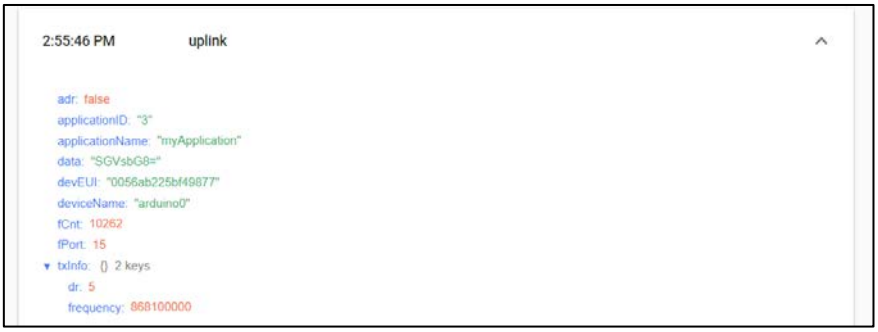


Figure 110 : Analyse des trames des Devices LoRa

Dans l'onglet **Application > myApplication > Devices > myDevice > LORAWAN FRAMES**, 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.

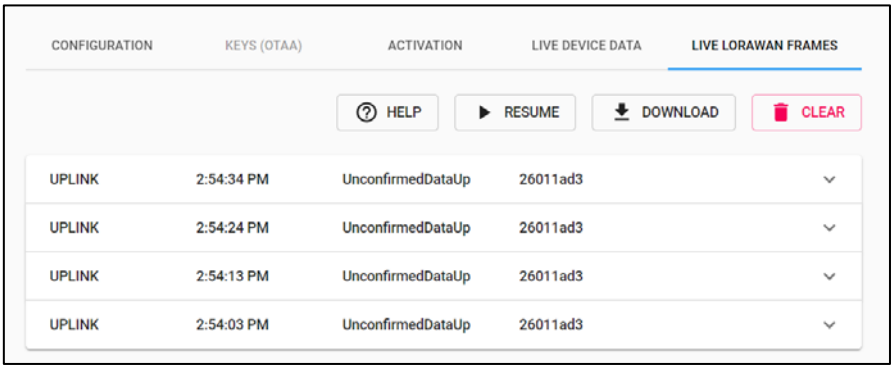


Figure 111 : Réception des Trames LoRaWAN

Enfin, si vos trames n'arrivent pas à votre Application Server, vous pouvez vérifier qu'elles arrivent à votre Gateway : **Gateway > myGateway > LIVE LORAWAN FRAMES**

9.4.7 Récupérer des données avec le protocole HTTP POST

Nous avons vu dans le chapitre 7 sur la récupération des données que les protocoles de communication entre le Serveur LoRaWAN et votre Application pouvaient être MQTT ou HTTP. Nous avons déjà implémenté et expliqué ces protocoles dans le cadre de son utilisation avec TTN. Nous allons donc juste configurer ChirpStack pour tester ces deux méthodes.

Nous utiliserons exactement la même méthode que celle que nous avons utilisé avec TTN au chapitre 7.3. Dans ChirpStack, ajouter une intégration HTTP : **Applications > myApplication > Integrations > Create > HTTP://.** Les informations suivantes devront être renseignées :

- Payload marshaler : Le format utilisé pour transférer les données (JSON est un bon choix).
- EndPoint : L'URL de votre serveur HTTP POST.

9.4.8 Envoyer des données avec le protocole HTTP POST

Cette démonstration est présentée uniquement dans la formation LoRaWAN en vidéo.

9.4.9 Récupérer des données avec le protocole MQTT

Nous utiliserons exactement la même méthode que celle que nous avons utilisée avec TTN au chapitre 7.4.8 . Nous allons nous connecter au Broker MQTT de ChirpStack en utilisant le client MQTTBox, comme nous l'avons déjà fait au paragraphe 7.4.8. Lorsque vous êtes connecté au Broker de ChirpStack, vous pouvez souscrire ou publier sur les Topics référencés dans le Tableau 20 avec la convention suivante :

- **[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]**/command/up

Tableau 20 : Topic à souscrire pour le flux Uplink

L'adresse du Broker MQTT auquel il faut se connecter est : @IP:1883

9.4.1 Envoyer des données avec le protocole HTTP MQTT

Cette démonstration est présentée uniquement dans la formation LoRaWAN en vidéo.

Détail du Topic	Nom du Topic
[Données] Flux Downlink	application/**[applicationID]**/device/**[devEUI]**/command/down

Tableau 21 : Topic à publier pour le flux Downlink

9.1 Analyse du "Packet Forwarder"

9.1.1 Présentation de UDP Packet Forwarder (SEMTECH)

Comme nous l'avons vu au chapitre 9.1.2, les Gateways LoRa et le Serveur LoRaWAN utilisent un protocole spécifique pour communiquer. Nous avons cité plusieurs protocoles disponibles mais le plus utilisé (car le plus simple) est le **UDP Packet Forwarder** qui a été développé par SEMTECH. Vous trouverez toute la documentation de ce protocole dans le dossier GitHub suivant : https://github.com/Lora-net/packet_forwarder . Dans ce dossier, un fichier nommé PROTOCOL.TXT explique parfaitement ce protocole qui travaille au-dessus de UDP.

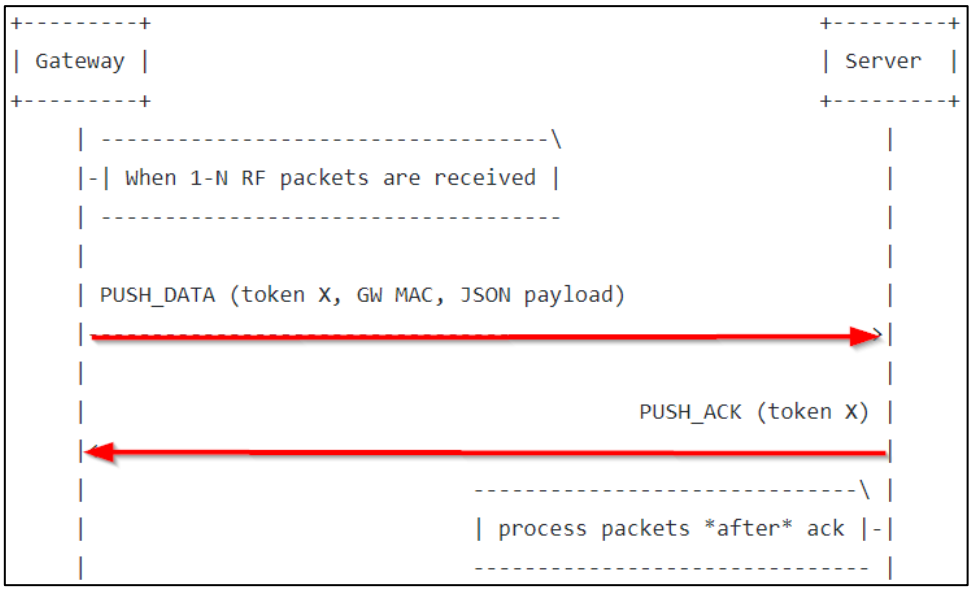


Figure 112 : Protocole Uplink (PUSH_DATA du Packet Forwarder)

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 :

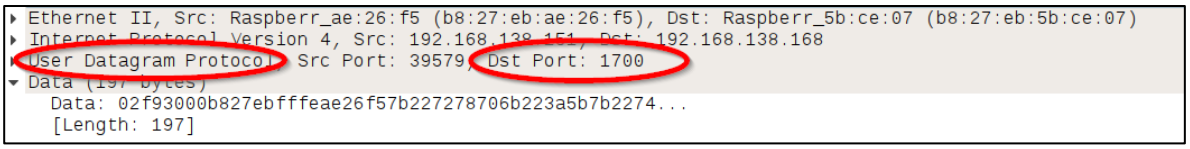


Figure 113 : Trame capturée par Wireshark lors d'un Uplink

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 }

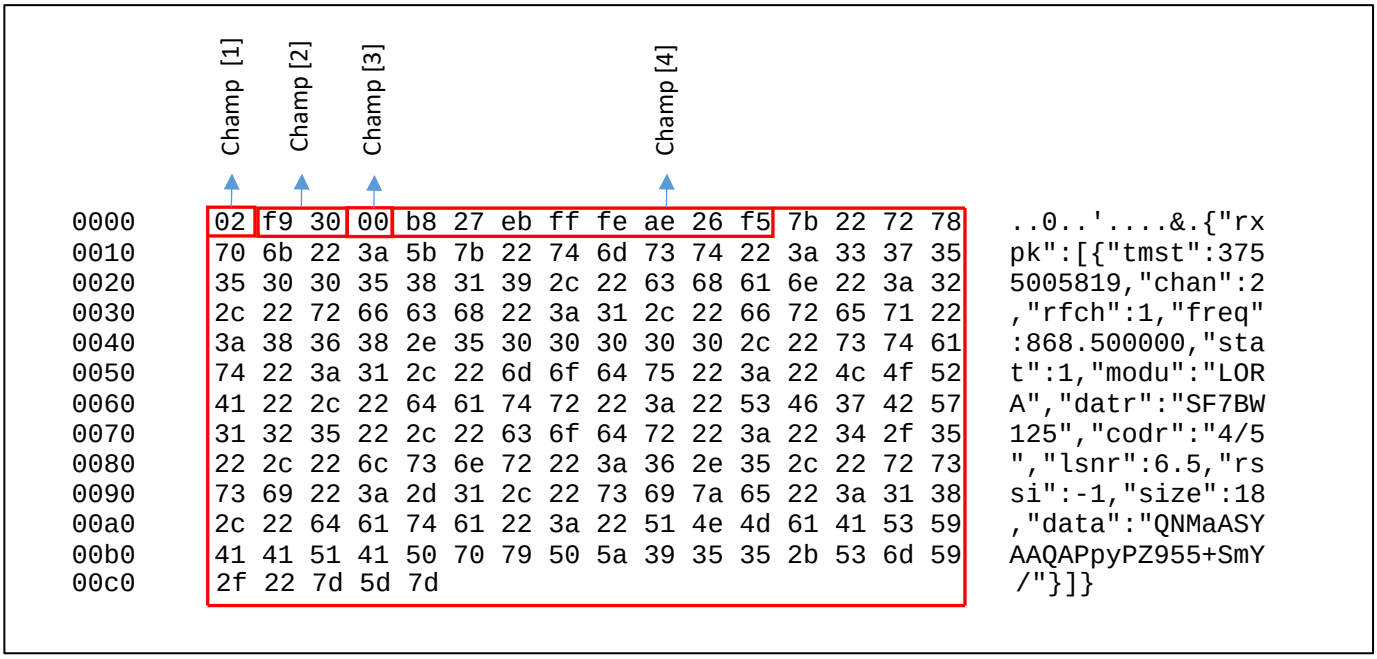


Figure 114 : Analyse du champ Données du protocole « Packet Forwarder »

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":"QNMaASYAAQAPpyPZ955+SmY/"
            }]
}
```

Le champ "data" correspond au PHY Payload.

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

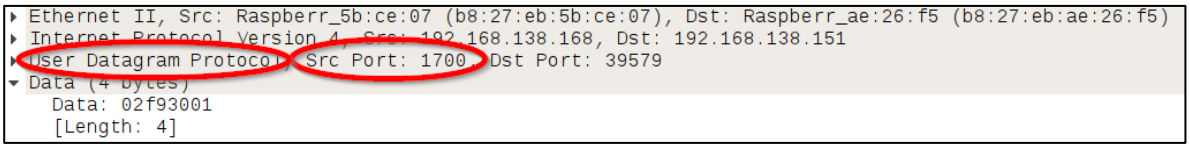


Figure 115 : Trame capturée par Wireshark lors de l'acquittement d'un Uplink

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.

10 La création de notre propre Application

10.1 Choix de l'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 (voir Figure 67). A la fin de ce chapitre, nous aurons implémenté une architecture fonctionnelle complète pour recevoir, stocker, traiter, et afficher ces données sur une page Web. Le Tableau 21 rassemble les différents choix technologiques possibles pour réaliser le prototypage de cette Application Utilisateur. La liste des solutions est très loin d'être exhaustive mais le tableau permet de comprendre les différentes fonctionnalités qui devront être réalisées.

10.1.1 Démonstration avec Node-RED

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 grâce à une interface utilisateur

Dans le sens Downlink :

- Présenter un interface utilisateur (Bouton, champ texte, …).
- Traiter la requête de l'utilisateur.
- Stocker la requête (optionnel)
- Envoyer la requête à l'Application Server de TTN.

C'est pourquoi dans le cadre de ce cours, nous utiliserons Node-RED pour mettre en place ces fonctionnalités. 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).

10.1.2 Autres démonstrations

Dans la formation LoRaWAN en vidéo qui se prête beaucoup mieux à ce genre d'explication, vous retrouverez une structure plus robuste organisée autour des services : Telegraf, InfluxDB, Chronograf et Grafana.

- Gérer la récupération des données > **Telegraf**
- Stocker les données. > **InfluxDB**
- Système de monitoring et Serveur WEB > **Chronograf** ou **Grafana**

Dans les vidéos, la notion de cloud IOT comme **AWS** (Amazon Web Service) ou **Microsoft Azure** sera aussi présentée.

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

Tableau 22 : Choix technologiques disponibles pour la réalisation de l'Application utilisateur

10.2 Mise en place de l'environnement de travail

Pour installer notre Application, il nous faut une machine connectée à internet et accessible par le Serveur LoRaWAN que vous utilisez. Cela peut être :

- Votre propre machine (Windows, Linux, MacOS).
- Une Raspberry PI ou équivalent.
- Une machine virtuelle (Virtual Box, VMware, …).
- Un serveur loué chez un fournisseur d'infrastructure (OVH, …).

L'objectif de ce cours est de simplifier au maximum le processus d'installation de service et de logiciel. Quel que soit le support que vous avez choisi ci-dessus pour installer votre Application, nous allons utiliser **docker** et **docker-compose** pour la mise en place de l'environnement de travail. Dans le chapitre 9 concernant la création de notre propre Network Server et Application Server nous avions déjà rencontré cette problématique. Vous pouvez donc vous y référer pour l'installation des services **docker** et **docker-compose**. Si vous avez déjà installé **docker** et **docker-compose** à cette occasion, votre système est déjà fonctionnel.

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

- Télécharger https://github.com/SylvainMontagny/dashboard-lorawan/
- Placer vous dans le répertoire nodered-dashboard
- Lancer le service Node-RED et InfluxDB : **docker-compose up -d**

Pour vérifier que le service Node-RED est bien actif, vous devez pouvoir faire le test suivant :

- Dans votre navigateur web, le lien http://@IP_Serveur:1880/ doit ouvrir Node-RED [**login** : admin / **Password** : lorawan].

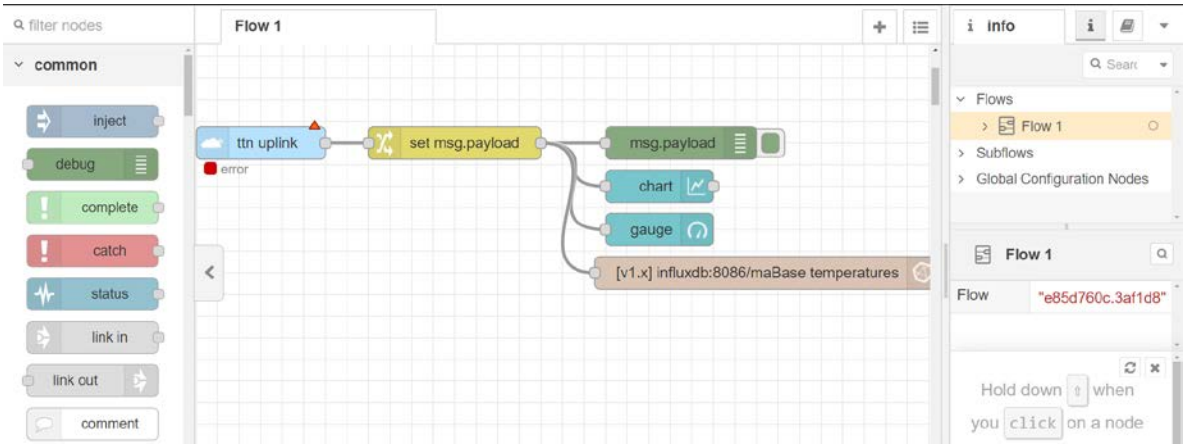


Figure 116 : Page d'accueil de Node-RED

La version de Node-RED que nous vous proposons ici est la version de base à laquelle ont été ajouté les librairies :

- The Things Network
- InfluxDB
- Dashboard

Le service Node-RED doit s'ouvrir avec un flow préconfiguré. Si ce flow n'apparait pas, vous pouvez l'importer facilement : **Menu (en haut à droite) > import > coller le contenu du fichier nodered/flow.json** qui se trouve dans les documents que vous venez de télécharger. Il s'agit de la solution que nous allons mettre en place pas à pas dans les prochains chapitres. Si vous souhaitez partir de zéro et refaire la démarche, alors vous pouvez supprimer ce flow. Si vous souhaitez rapidement faire fonctionner ce flow, il reste uniquement la configuration du node "ttn Uplink" qui est expliquée au paragraphe suivant.

Pour notre cas d'étude, nous nous placerons dans un cas simple qui est l'envoi d'une température sur un octet par un capteur. Nous souhaitons :

- Récupérer la valeur de température (node TTN).
- La stocker dans une BDD (node InfluxDB).
- Afficher sa variation en fonction du temps (node Dashboard).

10.3 Réalisation de l'Application avec Node-RED

10.3.1 Organisation de notre projet

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 :

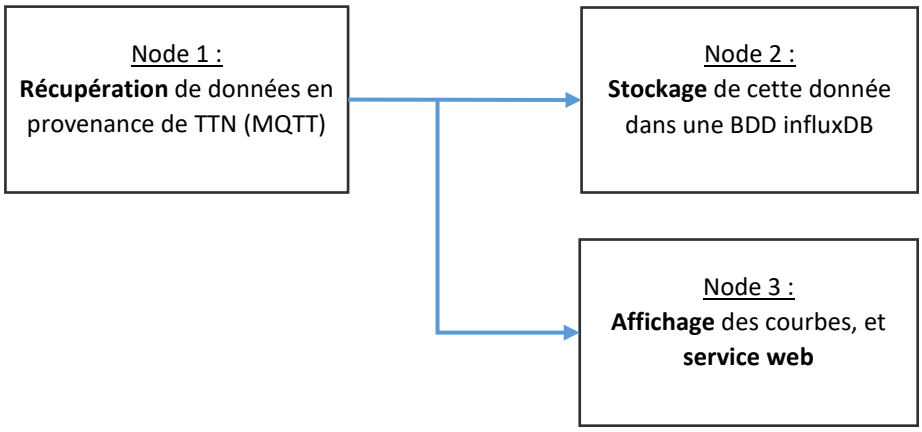


Figure 117 : Les briques du flow node-RED à réaliser

10.3.2 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".

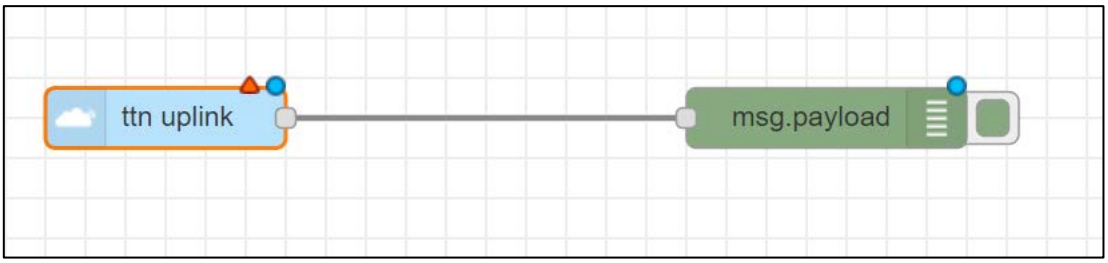


Figure 118 : Utilisation du Node TTN Uplink

Double cliquer sur le Node "ttn uplink" pour le configurer comme sur la Figure 119 :

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 ttn app) pour que Node-RED puisse s'y connecter.

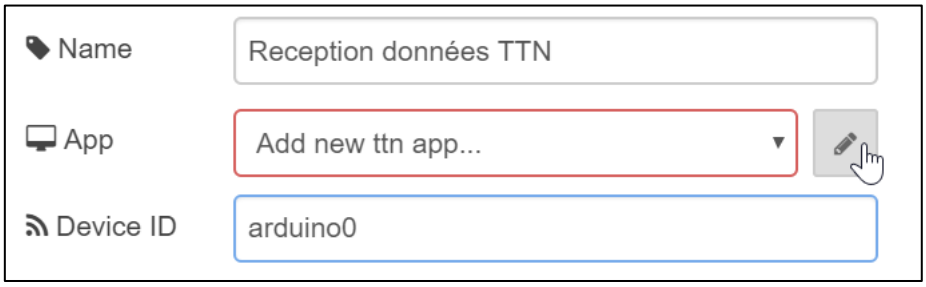


Figure 119 : Configuration du node "ttn uplink"

Configuration de votre nouvelle application :

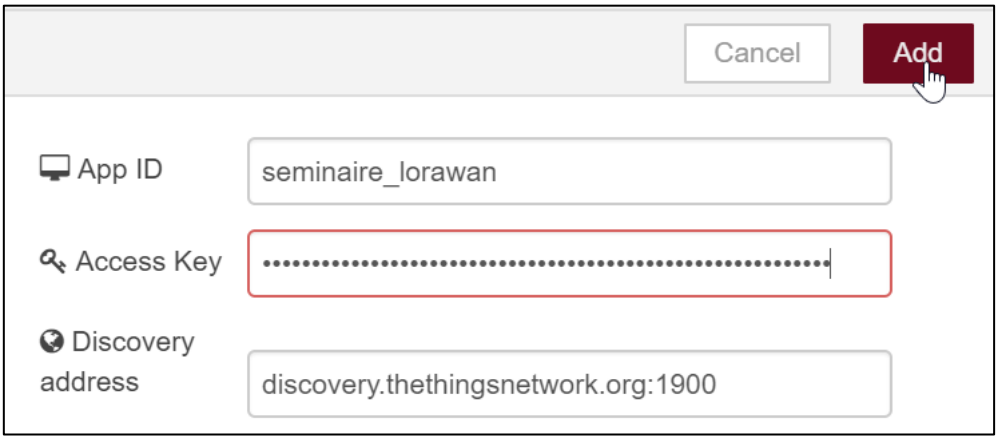


Figure 120 : Identifiant et mot de passe de notre application

- AppID : Nom de votre application. Dans notre cas "seminaire_lorawan".
- Access Key : Comme nous l'avons déjà expliqué plus haut, 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.

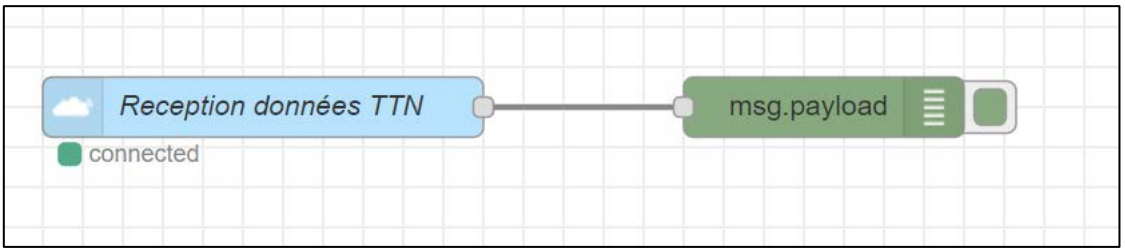


Figure 121 : Visualisation des trames publiées par TTN sur notre client MQTT

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.

10.3.3 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 122 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

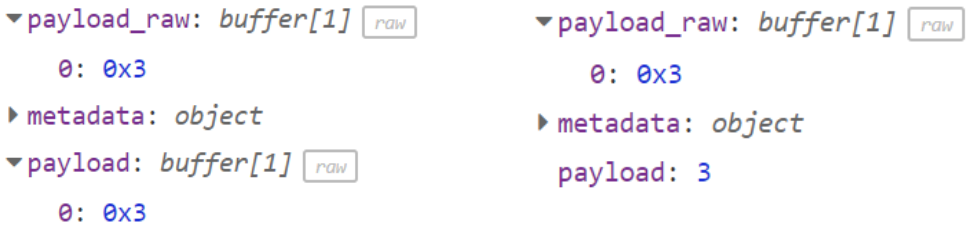


Figure 122 : Format du msg.payload avant (gauche) et après transformation (droite)

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 :

Figure 123 : 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 124 : 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]".

10.3.4 Création 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.

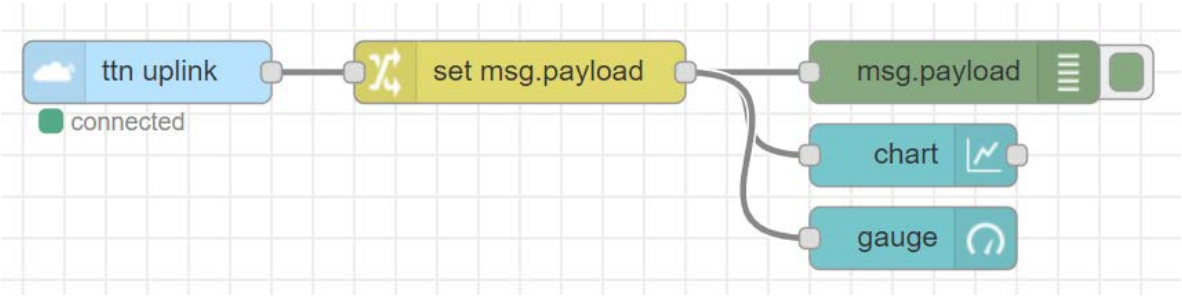


Figure 125 : Implémentation d'une interface graphique dans Node RED

L'URL pour joindre l'interface graphique de votre site web est disponible sur l'adresse http://@IP_Serveur:1880/ui .

Émettez 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.

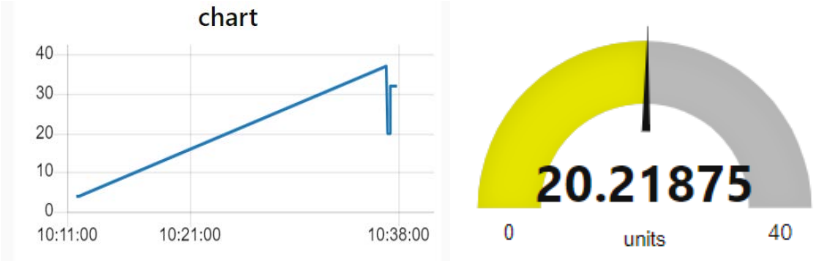


Figure 126 : Interface graphique du site web avec Node-RED

10.3.5 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 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. Dans notre cas, nous définirons un "measurement" appelé "**temperatures**" et les champs (fields) seront appelés "**values**".

La BDD étant 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 127.

La configuration du node est la suivante :

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

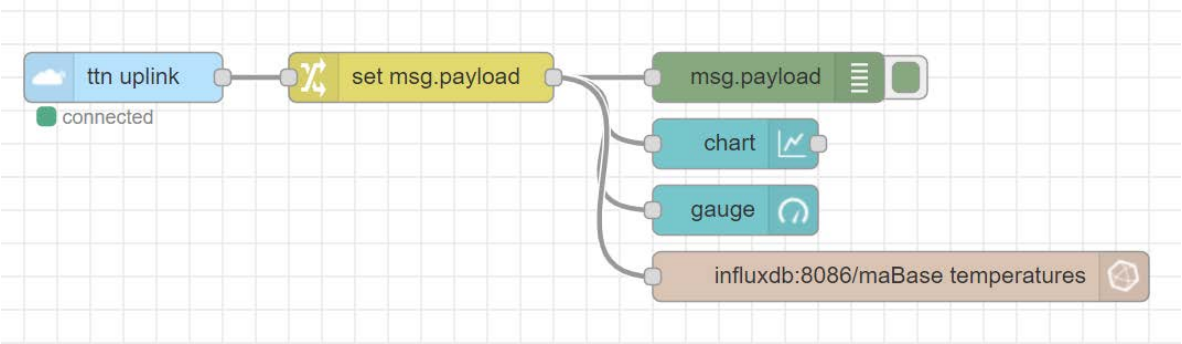


Figure 127 : Enregistrement des températures dans la BDD

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.

11 LoRaWAN avancé

Tous les éléments vus dans ce cours jusqu'à maintenant permettent d'avoir une vision globale du fonctionnement du protocole LoRaWAN. Dans le cadre d'une mise en œuvre simple et fonctionnelle, cela est largement suffisant. Cependant, certaines notions ont été volontairement cachées, reformulées et parfois vulgarisées afin de mettre cette technologie à la portée du plus grand nombre. Ce chapitre est destiné à rentrer plus dans le détail et à apprendre de nouvelles fonctionnalités.

11.1 Le Join Server

11.1.1 Le rôle du Join Server

Nous savons que le protocole LoRaWAN 1.0.X nécessite deux clés pour fonctionner :

- Le NwSKey pour l'authentification.
- L'AppSKey pour le chiffrement.

Le mode d'activation ABP fournit directement ses clés au Device. Cela simplifie le processus au détriment de la sécurité globale. Le mode d'activation recommandé est donc OTAA afin qu'un nouveau jeu de clés NwSKey et AppSKey soit généré à chaque nouvelle session grâce au Join-Request émit par le Device. **Mais qui exactement gère ce Join-Request?** Jusqu'à maintenant, nous avions considéré que cette phase d'activation était traitée par le Network Server. En réalité, c'est une entité spécifique qui s'appelle le Join Server qui s'en charge. Le Join Server est directement connecté à la fois au Network Server et à l'Application Server et possède un identifiant unique sur 8 octets (EUI) appelé le JoinEUI. Lors d'un Join-Request, le JoinEUI doit être émis pour spécifier quel Join Server sera utilisé pour traiter la phase d'activation.

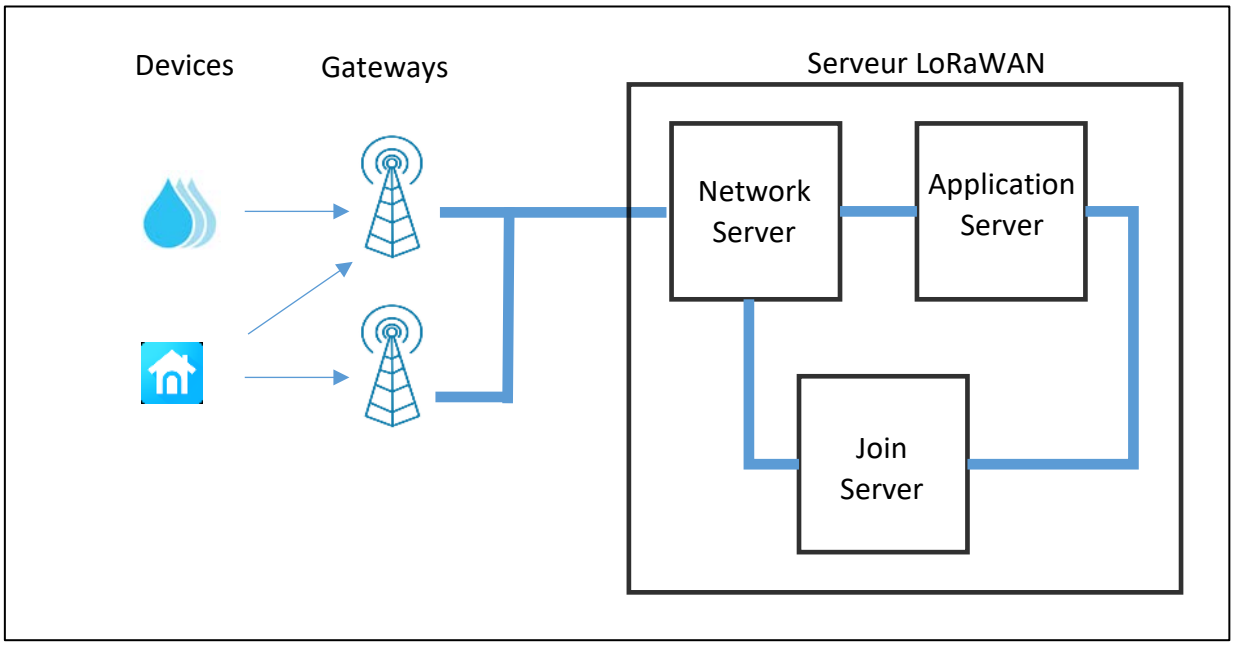


Figure 128 : Join Server

Depuis la version du protocole LoRaWAN 1.0.4, le JoinEUI Remplace l'AppEUI. Lorsque vous configurer un Device LoRa il faut donc connaitre la version du protocole qui est géré par la stack LoRaWAN. De plus, lorsque vous renseignez les informations sur le Serveur LoRaWAN, il faut préciser la version du protocole utilisé. Sur la Figure 129, nous avons représenté sur la première

ligne l'information AppEUI à rentrer dans le cas de l'utilisation d'un protocole 1.0.0 à 1.0.3 et sur la deuxième ligne, le JoinEUI pour les versions supérieures.

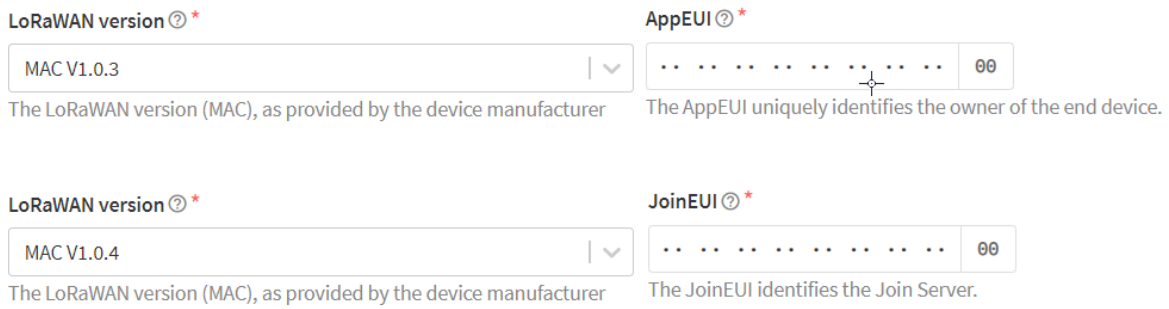


Figure 129 : AppEUI ou JoinEUI en fonction de la version du protocole

11.1.2 La procédure d'activation en OTAA

Dans un chapitre précédent, la Figure 37 représentait le déroulement de procédure de Join-Request. Nous pouvons donc faire quelques précisions sur cette procédure en utilisant le Join Server. La trame Join-Request est composé d'un JoinEUI, DevEUI et d'un DevNonce comme le montre la Figure 130.

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

Figure 130 : Constitution d'un Join-Request

Le DevNonce est une valeur importante pour le Join-Request car il permet de protéger des attaques par REPLAY exactement comme nous l'avons vu pour le Frame Counter au chapitre 4.5.2 : Un DevNonce ne peut pas être utilisé deux fois pour un Device. Le nombre de Join-Request maximum est donc de 65536, ce qui ne devrait bien sur jamais arriver car dans un fonctionnement normal, un Device LoRa générera une demande d'activation (Join-Request) rarement. La spécification LoRaWAN propose une gestion différente du DevNonce en fonction de la norme LoRaWAN avec laquelle on travaille :

- **Jusqu'à LoRaWAN 1.0.3**, le Serveur LoRaWAN accepte le Join-Request uniquement si le DevNonce n'a jamais été utilisé. Donc en LoRaWAN 1.0.3, le Device LoRa tire aléatoirement un DevNonce parmi les 65536 disponible. Si le Device n'a pas sauvegardé les valeurs utilisées précédemment pour le DevNonce, cela n'est pas très grave car au Join-Request suivant il va tirer une nouvelle valeur au hasard et aura encore beaucoup de chances de tomber sur une valeur qui n'a jamais été utilisée.
- **A partir du LoRaWAN 1.0.4**, le Serveur LoRaWAN accepte le Join-Request uniquement si la valeur utilisée est supérieure à celle utilisée précédemment. Donc en LoRaWAN 1.0.4, le Device Lora va utiliser un compteur qui s'incrémente pour le DevNonce. Si le Device reboot, il doit donc impérativement conserver la dernière valeur du DevNonce dans une mémoire non volatile. Dans le cas contraire, le Join-Request sera refusé.

Lorsque le Join-Request est accepté, le Serveur LoRaWAN retourne un Join-Accept avec les informations représentées Figure 131. Parmi celles-ci se trouvent le DevAddr, des informations de configuration du réseau, ainsi que tout ce que le Device a besoin pour générer de son côté le NwkSKey et l'AppSKey.

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

Figure 131 : Format d'une trame Join-Accept

Le Join Server a pour rôle de :

- Stocker en toute sécurité la clé AppKey.
- Répondre au Join-Request émis par les Devices autorisés.
- Stocker et transmettre en toute sécurité le NwSKey généré au Network Server.
- Stocker et transmettre en toute sécurité l'AppSKey généré à l'Application Server.

On peut se demander l'intérêt d'une telle structure, car la gestion des clés est alors simplement déportée sur un autre serveur. Pour répondre à cette question, il faut reprendre et améliorer la Figure 128. En effet, le Join Server n'est pas destiné à être un service interne et enfermé dans le Serveur LoRaWAN et connecté à un unique Network Server et Application Server.

- Un Network Server peut être connecté à plusieurs Join Server.
- Un Join Server peut être connecté à plusieurs Network Server.

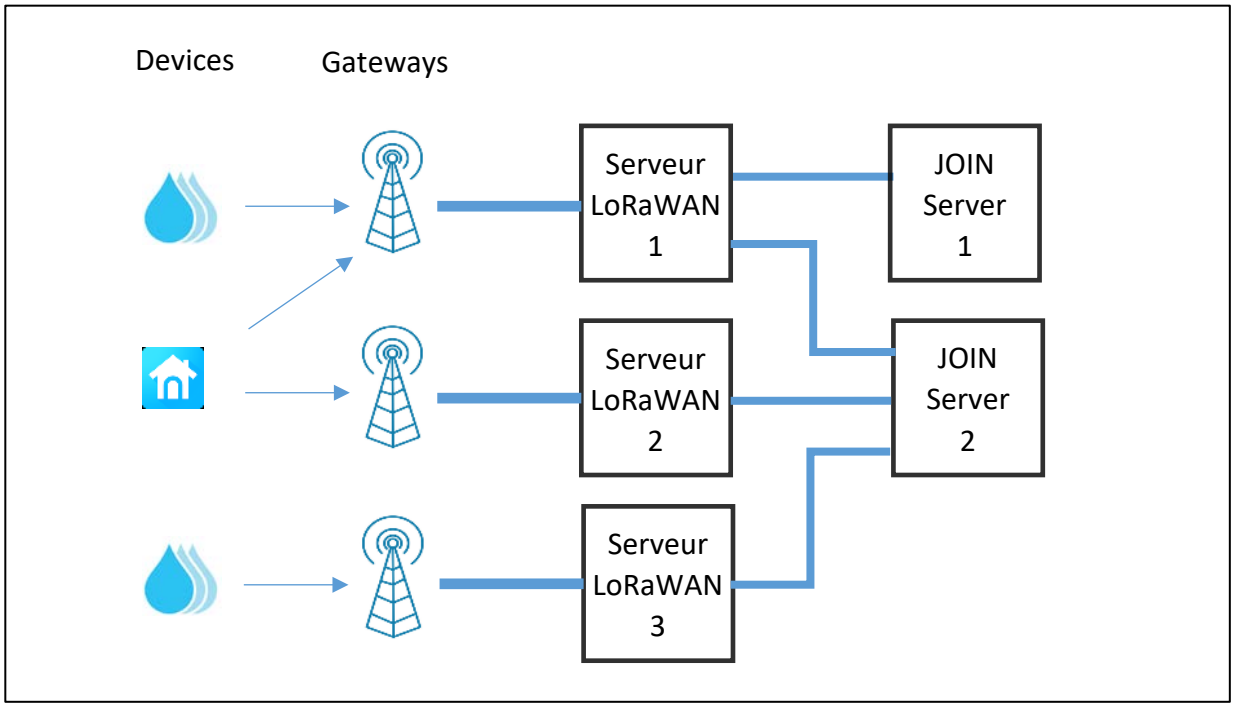


Figure 132 : Interconnexion des Join Server avec les Serveurs LoRaWAN

Une telle structure sera donc très avantageuse d'un point de vue de la sécurité et de l'interopérabilité des Devices au sein de différent réseaux.

Avant qu'un réseau LoRaWAN puisse fonctionner, un certains nombres d'échanges doivent être réalisés entre le concepteur du Device (Device Maker), l'utilisateur final (End User) et le Join Server comme le montre la Figure 133.

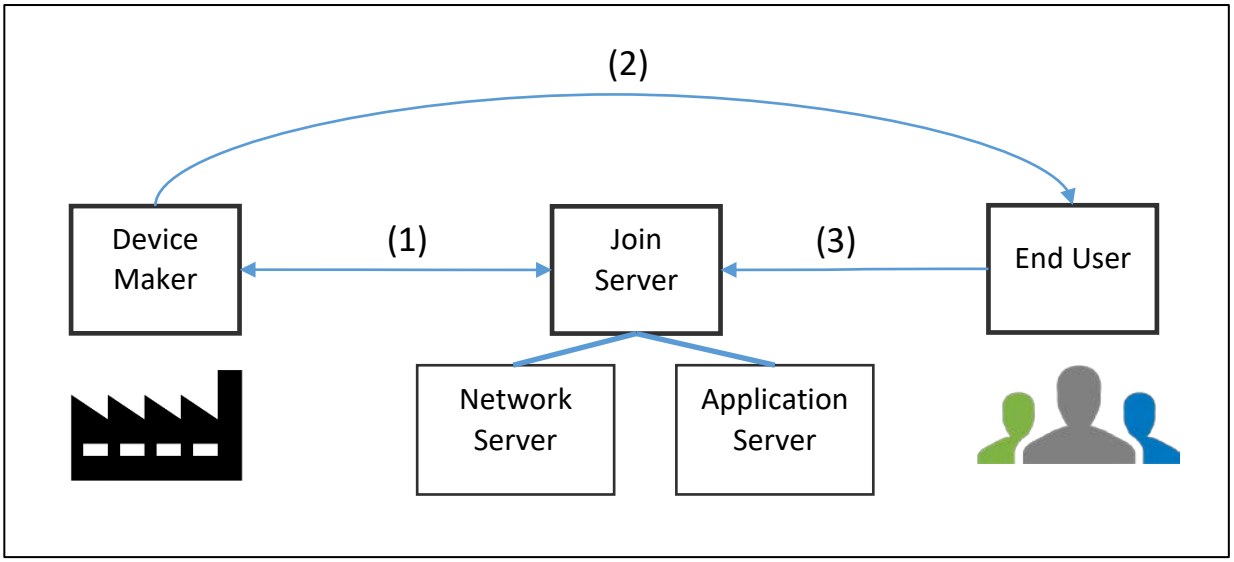


Figure 133 : Echanges entre le Device Maker, l'utilisateur final et le Join Server

1. Lors de l'industrialisation, le Device Maker va provisionner les Devices avec les clés (AppKey en LoRaWAN 1.0.x) ainsi que le JoinEUI et le DevEUI. Le DevEUI et l'AppKey doivent aussi être stockés sur le Join Server. Il faut donc qu'ils se mettent d'accord d'un moyen sécurisé pour que les clés ne soient pas exposées. Cette échange de clés est décrit par le symbole (1) sur la Figure 133. Il faudra ensuite que ces clés sont enregistrées de façon très sécurisée sur les deux entités.
2. Une fois le Device construit, il est vendu à l'utilisateur final. Cette action est décrite par le symbole (2) sur la Figure 133.
3. Enfin, l'utilisateur final doit prouver la propriété du Device en le réclamant sur le Join Server. Cette action permettra de définir les Network Server et Application Server lié au Join Server et autorisera la création des clés de Sessions (NwSKey et AppSKey) lors de l'activation du Device (Join Request). Cette action est décrite par le symbole (3) sur la Figure 133 .

Les points (1) et (3) vont être décrits dans les prochains chapitres.

11.1.3 La sécurisation des clés

La sécurité du protocole LoRaWAN est très dépendante de la façon dont sont stockées les clés. Il s'agit, bien évidement des Session Key (NwkSKey et AppSKey), mais aussi de la (ou des) "root key" (AppKey en LoRaWAN 1.0.x).

Les challenges à relever sont les suivants :

- **Comment place-t-on les même clés "root keys" dans le Device et dans le Join Server sans les exposer?** Cette opération est délicate car la programmation du Device LoRa est faite chez un industriel qui n'est pas forcément quelqu'un de confiance, donc vous ne pouvez pas nécessairement lui fournir la liste des clés à intégrer dans vos composants sans précaution.
- **Comment éviter l'accès aux clés (NwSKey, AppSKey et AppKey) malgré les tentatives d'intrusion matériel ?** L'accès physique aux mémoires stockant les clés doit être le plus limité possible. Un détection d'intrusion doit amener à l'autodestruction des clés.
- **Comment utiliser les clés sans laisser d'indice permettant d'aider à les retrouver?** Les calculs réalisée pendant le chiffrement ne doivent pas laisser de trace, comme par exemple

des variations de tension ou des appels de courant qui pourraient donner des indices sur les valeurs utilisées dans les calculs.

Pour sécuriser le stockage de ces clés et pour réaliser toutes les opérations de cryptographie, des modules matériels spécifiques à la fois coté Device et coté Serveur sont donc nécessaires.

- On parle de SE (**S**ecure **E**lement) coté Device. Ce sont des tout petits composants proches du microcontrôleur. Ils sont même parfois intégrés au microcontrôleur.

Par exemple le composant ATECC608B-TNGACT possède toutes ces caractéristiques, et est en plus déjà provisionné, c'est-à-dire qu'il contient déjà les "root keys" pour le serveur LoRaWAN ACTILITY.

- On parle de HSM (**H**ardware **S**ecurity **M**odule) coté serveur. Ils ont les mêmes caractéristiques, mais avec plus de puissance et de fonctionnalités.

11.1.4 Réclamation du Device

Lorsque le Device est fabriqué, il est provisionné avec ses clés. Ces mêmes clés sont stockées sur un Join Server qui pour l'instant n'est pas autorisé à répondre au Join-Request du Device puisque qu'aucune Application client n'est configurée dans l'Application Server. Il faut donc que le client enregistre son Device, puis prouve sa propriété grâce à la procédure de réclamation (Device Claiming). La preuve de propriété du Device peut être faite grâce à un QR Code qui a été fourni par le vendeur, ou par les informations du Device (DevEUI) associées à un code que nous trouvons dans le Device lui-même comme le montre la Figure 134.

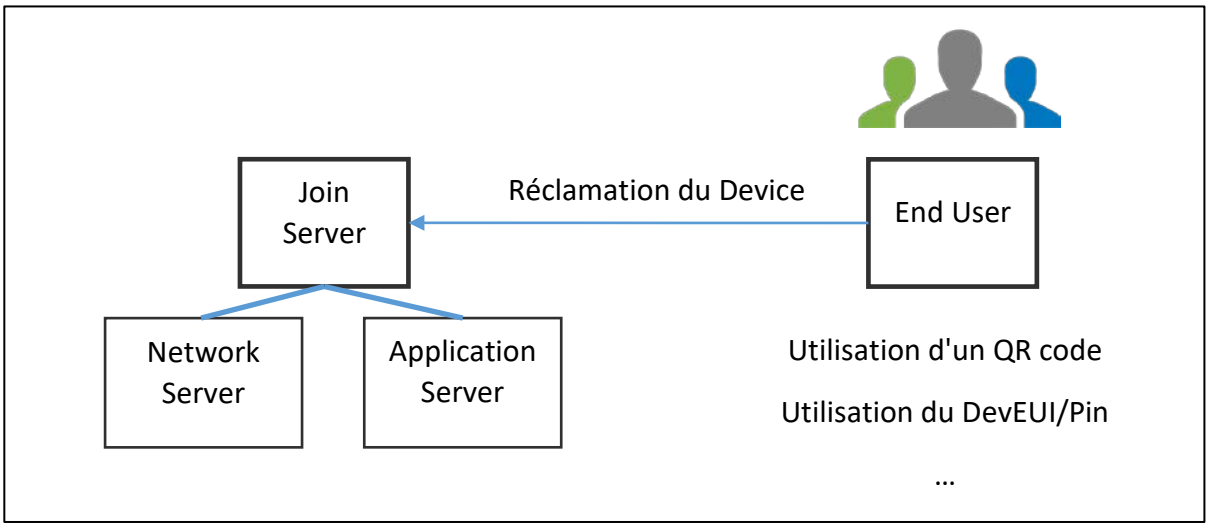


Figure 134 : Réclamation d'un Device auprès d'un Join Server

Lorsque la réclamation est réussie, la procédure d'activation en OTAA grâce au Join-Request se déroulera exactement comme nous l'avons déjà vu jusqu'à maintenant et les clés de sessions seront alors distribuées au Network Server (NwkSKey) et Application Server (AppSKey) définis.

Si une nouvelle procédure de réclamation est réalisée, par exemple par un nouveau propriétaire, alors le Device devra refaire un Join-Request, car la réclamation d'un Device n'engendre pas le transfert des clés de sessions entre 2 applications. Il transfère uniquement la propriété du Device.

Versions du document

Version 1.0 : Février 2019

Version initiale.

Version 2.0 : Février 2019

Ajout : Paragraphe : Les systèmes embarqués dans l'IoT.
Complément : Amélioration de l'exercice étalement de spectre.
Correction : Câblage Arduino.
Ajout : Fonctionnement de l'Arduino et ajout de librairies.
Complément : Couche physique LoRa. Ajout de 4 figures pour l'explication du Chirp.
Ajout : Acronymes utilisés au début du cours.
Complément : dB et dBm.
Ajout : Utilisation du logiciel LoRa Calculator

Version 3.0 : Mars 2019

Ajout : Figure sur le rôle de la Gateway LoRa
Ajout : Paragraphe sur l'authentification avec le Network Server
Ajout : Paragraphe sur le chiffrement avec l'Application Server
Ajout : Sommaire en début de cours
Ajout : 2 schémas sur ABP et OTAA
Retrait : Explication du détail du Join-Request en OTAA : Réalisation d'un schéma plus explicite (TODO).
Ajout : Schéma sur l'attaque par REPLAY
Ajout : Paragraphe des différents réseaux LoRaWAN existants : Opérés et Privés
Ajout : Schémas des trames LoRaWAN : Physique / LoRa MAC / Application
Complément : Information sur les Classes de Devices
Ajout : Paragraphe sur la Gateway utilisée pour le Downlink
Ajout : Paragraphe sur le JSON
Complément : Information sur le codage en Base64
Ajout : Information sur le décodage des trames LoRaWAN
Modification : Explications sur l'envoi et la récupération des données en HTTP/MQTT

Version 4.0 : Mars 2019

Ajout : Chapitre sur la "Création de notre propre Application"
Ajout : Chapitre sur la "Création de notre propre Network et Application Server"
Ajout : Figure sur la visualisation des Chirps LoRa par SDR (Radio Logicielle)
Complément : Information sur MQTT
Complément : Information sur les Topics de TTN en MQTT

Version 4.1 : Octobre 2019

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

Version 4.2 : Février 2020

Modification : Passage de l'arduino + RN2483 au module The Things UNO (Léonardo).
Complément : Des renvois sur les paraphes concernés ont été ajoutés.
Complément : Ajout du schéma de connexion en OTAA. Join-Request / Join-Accept.
Modification : Changement de serveur HTTP POST : https://rbaskets.in.
Ajout : Les champs JSON permettent 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 : Exemple sur le CDMA
Modification : Figure 22 : Le rôle de la Gateway LoRa, la représentation en couche fait apparaitre LoRaMAC pour être conforme à la norme et éviter les ambiguïtés.
Modification : Figure 64 : 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). TODO.
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
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 LoRaWAN sur UDEMY

Version 7 : Septembre 2020

Udemy : Ajout de Quizz en ligne sur chacun des fins de chapitre.
Udemy : Création des vidéos 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

Version 8 : Octobre 2020

Udemy : Ajout des vidéos sur la création de notre propre Network Server et Application Server.
Modification : Mise à jour du chapitre sur la configuration de la Gateway
Modification : Mise à jour complète de la création de notre propre Network Server et Application Server. Passage de LoRaServer à ChirpStack
Modification : Les énoncés des manipulations sont dans un document "élève" à part.
Modification : Les corrections des exemples et exercices sont systématiquement fournies.
Correction : Très nombreuses corrections (orthographe) grâce à l'œil attentif d'Albert.
Ajout : Liste des acronymes complétés.

Version 9 : Février 2021

Information : Le flux Downlink en HTTP POST et en MQTT sur Chirpstack est seulement traité en vidéo sur UDEMY.
Modification : Refonte complète du chapitre sur la création de notre propre Application
Ajout : Incitation à la collaboration et signalement des erreurs.
Udemy : Ajout du chapitre sur l'installation et la configuration de "The Things Stack"
Udemy : Réparation de la vidéo de l'adaptive Data Rate
Modification : Le site thethingsstack.io devient thethingsindustries.com/docs
Ajout : Présentation de la formation LoRa LoRaWAN en ligne sur 2 jours

Version 10 : Avril 2021

Ajout : Figure résumant le rôle de l'Application Utilisateur
Ajout : Annexes pour résumer toutes les configurations HTTP – MQTT des différents Serveurs.
Ajout : Annexes pour résumer les adresses et ports des Serveurs LoRaWAN courants.
Modification : Maximum Payload size est précisé sans répéteur (242 octets ou lieu de 222)
Modification: Réorganisation du github pour le Dashboard de l'application.
Udemy : Ajout de 9 vidéos du chapitre "Création de votre propre Application".
Ajout : Paragraphe sur la LoRa Alliance.
Ajout : Paragraphe sur les versions du protocole LoRaWAN.
Modification : Attaque par Replay. En OTAA, le DevNonce doit être conservé en mémoire.
Correction : Le MIC est sur 4 octets et non sur 2. Correction de 2 figures + vidéo Udemy.
Modification : Explication de la Base 64 et de son intérêt.
Modification : Le cours est le plus possible indépendant de TTN (jusqu'au chapitre 7)
Ajout : Nouveau chapitre sur le Join Server et la sécurité.

Toutes remarques, modifications, améliorations, corrections peuvent être proposées par email : sylvain.montagny@univ-smb.fr ou par le formulaire suivant : https://cutt.ly/erreurs.

Annexes

Connexion Gateway -> Serveur LoRaWAN

With Semtech UDP Packet Forwarder

TTN v2

- **Network Server (TTN) :** router.eu.thethings.network
- **Port Up :** 1700
- **Port Down :** 1700

TTN v3

- **Network Server (TTN) :** eu1.cloud.thethings.network
- **Port Up :** 1700
- **Port Down :** 1700

LORIOT

- **Network Server (loriot) :** eu1.loriot.io
- **Port Up :** 1780
- **Port Down :** 1780

LoRa Cloud

- **MQTT Broker :** ssl://eu868.mqtt.loracloud.com:8883 // A configurer dans chirpstack-gateway-bridge

Downlink en HTTP POST

Pour TTN v2 :
[https://www.thethingsnetwork.org/docs/applications/http/index.html]

➡ **POSTMAM > New > Request > Save Request**

- 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.

```
"downlink_url":"https://integrations.thethingsnetwork.org/ttn-eu/.....................8ypjj7ZnL3KieQ"
```

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

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

Pour The Things Stack ou TTN v3 :
[https://www.thethingsindustries.com/docs/integrations/mqtt/]

➡ **POSTMAM > Import > Raw**

```
curl
https://thethings.example.com/api/v3/as/applications/app1/webhooks/wh1/devices/dev1/down/push \
  -X POST \
  -H 'Authorization: Bearer NNSXS.VEEBURF3KR77ZR..' \
```

Les modifications suivantes doivent être apportées à la requête précédente :

- ***https://thethings.example.com*** : Mettre l'adresse http://xxxxx.master-stic.fr ou https://eu1.cloud.thethings.network
- ***app1*** : Mettre le nom de l'application
- ***wh1*** : Mettre le nom du Web Hook
- ***dev1*** : Mettre le nom du Device
- ***NNSXS.VEEBURF3KR77ZR..*** : Mettre l'API Key (**Console TTN > API Key > Add API Key**)

➡ Rajouter le champs JSON : POSTMAN > **Body > Pretty > JSON :**

```
{"downlinks":[{"frm_payload":"aGVsbG8=","f_port":15,"priority":"NORMAL"}]}
```

Pour ChirpStack ou LoRaCloud :
[https://www.chirpstack.io/application-server/api/http-examples/]

POSTMAM > Import > Raw

```
curl -X POST --header 'Content-Type: application/json' --header
'Accept:     application/json'      --header      'Grpc-Metadata-
Authorization: Bearer <API TOKEN>' -d '{ \
   "deviceQueueItem": { \
     "confirmed": false, \
     "data": "aGVsbG8=", \
     "fPort": 10 \
   } \
 }' 'http://localhost:8080/api/devices/0101010101010101/queue'
```

Les modifications suivantes doivent être apportées à la requête précédente :

- ***<API TOKEN>*** : ChirpStack > API Key > Create API Key
- ***http://localhost*** : Mettre l'adresse http://xxxxx.master-stic.fr ou https://ns.loracloud.com
- ***0101010101010101*** : Mettre le DevEUI

Pour LORIOT :
[https://docs.loriot.io/display/LNS/Send+Downlinks+via+API]

POSTMAM > Import > Raw

```
curl --location --request POST 'https://eu1.loriot.io/1/rest' \
--header 'Content-Type: application/json' \
--header 'Authorization: Bearer <API TOKEN>' \
--header 'Content-Type: text/plain' \
--data-raw '{
"cmd": "tx",
"EUI": "BE7A000000000009",
"port": 33,
"confirmed": false,
"data": "0ffff0",
"appid": "BE01000D"
}'
```

Les modifications suivantes doivent être apportées à la requête précédente :

- ***<API TOKEN>*** : Application> Access Token > Generate authentication token
- **"EUI" :** Mettre le DevEUI
- ***"appid"*** : *Application ID*

Uplink en MQTT

Pour TTN v2 :

- **Adresse du Broker** : eu.thethings.network:1883
- **Username** : Le nom de votre application
- **Password** : Access Key de votre application : TTN > Application > Overview > Access Key
- **Topic** :

Détail du Topic	Nom du Topic
Flux Uplink	**<AppID>**/devices/**<DevID>**/up
Tout le Flux Uplink	**+**/devices/**+**/up

<AppID> correspond au nom de votre Application.
<DevID> correspond au nom de votre Device LoRa.
Le symbole '+' permet de souscrire à tous les Devices de toutes les applications.

Pour TTN v3 :

- **Adresse du Broker** : eu1.cloud.thethings.network:1883
- **Username** : Le_nom_de_votre_application@ttn
- **Password** : Application > Integrations > MQTT > Generate new API Key
- **Topic** :

Détail du Topic	Nom du Topic
Flux Uplink	v3/**{application id}**@**{tenant id}**/devices/**{device id}**/up
Tout le Flux Uplink	v3/**+**/devices/**+**/up

{application id} correspond au nom de votre application.
{tenant id} correspond à "ttn".
{device id} correspond au nom de votre Device LoRa.
Le symbole '+' permet de souscrire à tous les Devices de toutes les applications.

Pour ChirpStack :
[Documentation : https://www.chirpstack.io/application-server/integrations/mqtt/]

- **Adresse du Broker** : xxxxxx.master-stic.fr:1883
- **Username** : Rien par défaut.
- **Password** : Rien par défaut.
- **Topic** :

Détail du Topic	Nom du Topic
Flux Uplink	application/**[applicationID]**/device/**[devEUI]**/event/up
Tout le Flux Uplink	application/**+**/device/**+**/event/up

[applicationID] correspond à l'ID (numéro !) de votre Application.
[devEUI] correspond au DevEUI (lettre en minuscule !) de votre Device.

Le symbole '+' permet de souscrire à tous les Devices de toutes les applications.

Pour The Things Stack :

[Documentation : https://www.thethingsindustries.com/docs/integrations/mqtt/]

- **Adresse du Broker** : xxxxxx.master-stic.fr:1883
- **Username** : Le nom de votre application
- **Password** : Application > Integrations > MQTT > Generate new API Key
- **Topic** :

Détail du Topic	Nom du Topic
Flux Uplink	v3/**{application id}**@**{tenant id}**/devices/**{device id}**/up
Tout le Flux Uplink	v3/**+**/devices/**+**/up

{application id} correspond au nom de votre application.
{tenant id} correspond à "ttn".
{device id} correspond au nom de votre Device LoRa.
Le symbole '+' permet de souscrire à tous les Devices de toutes les applications.

Downlink en MQTT

Pour TTN v2 :

- **Adresse du Broker** : eu.thethings.network:1883
- **Topic** :

Détail du Topic	Nom du Topic
Flux Downlink	**<AppID>**/devices/**<DevID>**/down

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

- **Payload JSON** :

```
{
    "dev_id": "<DevID>",
    "payload_raw": "aGVsbG8=",
    "port": 1
}
```

Pour TTN v3 :

- **Adresse du Broker** : eu1.cloud.thethings.network:1883
- **Username** : Le_nom_de_votre_application@ttn
- **Password** : Application > Integrations > MQTT > Generate new API Key
- **Topic** :

Détail du Topic	Nom du Topic
Flux Downlink	v3/**{application id}**@**{tenant id}**/devices/**{device id}**/down/replace

{application id} correspond au nom de votre application.
{tenant id} correspond à "ttn".
{device id} correspond au nom de votre Device LoRa.

- **Payload JSON** :

```
{
  "downlinks": [{
    "f_port": 15,
    "frm_payload": "aGVsbG8=",
    "priority": "NORMAL"
  }]
}
```

Pour ChirpStack :
[Documentation : https://www.chirpstack.io/application-server/integrations/mqtt/]

- **Adresse du Broker** : xxxxxx.master-stic.fr:1883
- **Username** : Rien par défaut.
- **Password** : Rien par défaut.

- **Topic** :

Détail du Topic	Nom du Topic
Flux Downlink	application/**[applicationID]**/device/**[devEUI]**/command/down

[applicationID] correspond à l'ID (numéro !) de votre Application.
[devEUI] correspond au DevEUI (lettre en minuscule !) de votre Device.

- **Payload JSON** :

```
{
    "confirmed": false,
    "fPort": 10,
    "data": "aGVsbG8="
}
```

Pour The Things Stack :
[Documentation : https://www.thethingsindustries.com/docs/integrations/mqtt/]

- **Adresse du Broker** : xxxxxx.master-stic.fr:1883
- **Username** : Le nom de votre application
- **Password** : Application > Integrations > MQTT > Generate new API Key
- **Topic** :

Détail du Topic	Nom du Topic
Flux Downlink	v3/**{application id}**/devices/**{device id}**/down/replace

{application id} correspond au nom de votre application.
{device id} correspond au nom de votre Device LoRa.

- **Payload JSON** :

```
{
  "downlinks": [{
    "f_port": 15,
    "frm_payload": "aGVsbG8="
  }]
}
```