

# Cours- Formation LoRa / LoRaWAN



Version du cours : 4.1

Octobre 2019

Sylvain MONTAGNY

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

Université Savoie Mont Blanc

# Présentation du document

Ce cours sur l'internet des Objets et les protocoles LoRa / LoRaWAN est le support d'une formation proposée par l'Université de Savoie Mont Blanc (USMB). Elle est réalisée :

- Aux étudiants du Master ESET (Electronique des Systèmes Embarqués et Télécoms) Site web de la formation : [ [www.master-electronique.com](http://www.master-electronique.com) ]
- Aux entreprises qui en font la demande (1 ou 2 journées) : [sylvain.montagny@univ-smb.fr](mailto:sylvain.montagny@univ-smb.fr)



➔ Ce cours est mis à disposition sur internet et est libre d'utilisation. Toutes personnes souhaitant apporter des modifications / améliorations / corrections peut le faire sur l'adresse mail : [sylvain.montagny@univ-smb.fr](mailto:sylvain.montagny@univ-smb.fr) . L'auteur vous remercie par avance pour votre contribution.

## Sigles et Acronymes

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

# Sommaire

<b>1</b>	<b>LES SYSTEMES EMBARQUES ET L'IOT .....</b>	<b>6</b>
1.1	L'INTERNET DES OBJETS ( INTERNET OF THINGS / IOT ) .....	6
1.1.1	Les systèmes embarqués dans l'IoT .....	6
1.1.2	Différents protocoles dans l'IoT .....	6
1.1.3	Bande de fréquence utilisées.....	7
1.2	MODES DE PARTAGE DU SUPPORT.....	7
1.3	NOTION D'ÉTALEMENT DE SPECTRE .....	8
1.3.1	Transmissions successives.....	8
1.3.2	Transmissions simultanées.....	9
1.3.3	Cas du protocole LoRa.....	9
<b>2</b>	<b>TRANSMISSION RADIO ET PROPAGATION .....</b>	<b>10</b>
2.1	LES UNITES ET DEFINITIONS.....	10
2.2	ETUDE DE LA DOCUMENTATION D'UN COMPOSANT LORA.....	11
<b>3</b>	<b>LA COUCHE PHYSIQUE LORA.....</b>	<b>13</b>
3.1	LA MODULATION LORA .....	13
3.1.1	La forme du symbole (Chirp) .....	13
3.1.2	Durée d'émission d'un symbole .....	15
3.2	CODING RATE.....	16
3.3	UTILISATION DU LORA CALCULATOR .....	17
3.4	TIME ON AIR.....	18
3.5	MISE EN ŒUVRE : LORA EN POINT A POINT.....	19
3.5.1	Utilisation de l'IDE Arduino .....	19
3.5.2	Validation du fonctionnement .....	20
3.5.3	Mise en valeur du Spreading Factor.....	20
<b>4</b>	<b>LE PROTOCOLE LORAWAN .....</b>	<b>21</b>
4.1	STRUCTURE D'UN RESEAU LORAWAN .....	21
4.1.1	Les Devices LoRa .....	21
4.1.2	Les Gateways LoRa.....	21
4.1.3	Le Network Server .....	22
4.1.4	Application Server .....	22
4.1.5	Application / Web Serveur .....	24
4.1.6	Authentification avec le Network Server.....	25
4.1.7	Chiffrement des données vers l'Application Server .....	25
4.1.8	Combinaison de l'authentification et du chiffrement .....	26
4.2	CLASSES DES DEVICES LORAWAN .....	26
4.2.1	Classe A (All) : Minimal power Application .....	26
4.2.2	Classe B (Beacon) : Scheduled Receive Slot.....	27
4.2.3	Classe C (Continuous) : Continuously Listening .....	28
4.2.4	Quelle Gateway pour les flux Downlink ?.....	29
4.3	ACTIVATION DES DEVICES LORA ET SECURITE .....	29
4.3.1	ABP : Activation By Personalization .....	29
4.3.2	OTAA : Over The Air Activation : .....	30
4.3.3	Sécurité .....	31
4.4	LA TRAME LORA / LORAWAN .....	32
4.4.1	Les couches du protocole LoRaWAN .....	32
4.4.2	Couche Application .....	33
4.4.3	Couche LoRa MAC .....	34

4.4.4	Couche physique : Modulation LoRa .....	34
4.4.5	Canaux et bandes de fréquences et Data Rate (DR) .....	35
4.4.6	Trame reçue et transmise par la Gateway LoRaWAN .....	36
4.4.7	Le format JSON .....	37
<b>5</b>	<b>MISE EN ŒUVRE D'UN RESEAU LORAWAN .....</b>	<b>38</b>
5.1	LES DIFFERENTS TYPES DE RESEAUX .....	38
5.1.1	Les réseaux LoRaWAN opérés .....	38
5.1.2	Les réseaux LoRaWAN privés .....	38
5.1.3	Les zones de couvertures .....	38
5.2	THE THINGS NETWORK (TTN) .....	39
5.2.1	Présentation de TTN .....	39
5.2.2	Configuration de la Gateway .....	39
5.2.3	Enregistrement des Gateways, Application et Devices .....	39
5.2.4	Configuration des Devices LoRa .....	40
5.3	MISE EN APPLICATION .....	41
5.4	ANALYSE DES TRAMES ECHANGEES .....	41
5.4.1	Utilisation de la base 64 .....	41
5.4.2	Intérêt et inconvénient de la base 64 .....	43
5.4.3	Uplink : Du Device LoRa au Network Server .....	43
5.4.4	Uplink : Du Network Server à l'Application Server .....	45
5.4.5	Simulation d'Uplink dans l'Application Server .....	46
5.4.6	Downlink : De l'Application Server au Device LoRa .....	46
<b>6</b>	<b>LA RECUPERATION DES DONNEES SUR NOTRE PROPRE APPLICATION .....</b>	<b>47</b>
6.1	RECUPERATION DES DONNEES EN HTTP POST DANS L'APPLICATION .....	47
6.1.1	Présentation du protocole HTTP .....	47
6.1.2	Fonctionnement d'un client et d'un serveur HTTP POST .....	48
6.1.3	Test du serveur HTTP POST .....	49
6.1.4	Test du client HTTP POST .....	49
6.1.5	Récupérer des données sur notre Application avec HTTP POST .....	49
6.1.1	Envoyer des données depuis notre Application avec HTTP POST .....	51
6.2	RECUPERATION DES DONNEES AVEC MQTT DANS L'APPLICATION .....	52
6.2.1	Présentation du protocole MQTT .....	52
6.2.2	Connexion au Broker MQTT .....	53
6.2.3	Qualité de Service au cours d'une même connexion .....	53
6.2.4	Qualité de Service après une reconnexion .....	55
6.2.5	Les Topics du protocole MQTT .....	55
6.2.6	Mise en place d'un Broker MQTT .....	56
6.2.7	Mise en place d'un Publisher et d'un Subscriber MQTT .....	57
6.2.8	Récupérer des données sur notre Application avec MQTT .....	57
6.2.9	Envoyer des données depuis notre Application avec MQTT .....	60
<b>7</b>	<b>CREATION DE NOTRE PROPRE NETWORK ET APPLICATION SERVER .....</b>	<b>63</b>
7.1.1	Objectifs .....	63
7.1.2	Présentation de LoRaServer .....	63
7.1.3	Le « Packet Forwarder » (Gateway) .....	64
7.1.4	LoRa Gateway Bridge .....	66
7.1.5	LoRa Server (Network Server) .....	66
7.1.6	LoRa App Server (Application Server) .....	67
7.1.7	LoRa Geo Server .....	67
7.1.8	Application .....	67
7.2	INSTALLATION DE LORASERVER .....	67

7.2.1	Mise en place de l'environnement .....	67
7.2.2	Installation sur la Raspberry PI .....	68
7.1	CONFIGURATION DE LoRA SERVER POUR L'UPLINK .....	69
7.1.1	Enregistrement d'une nouvelle Organisation.....	69
7.1.2	Enregistrement d'une instance du Network Server.....	69
7.1.3	Enregistrement d'une Application (Sur l'Application Server) .....	70
7.1.4	Enregistrement des Devices LoRa .....	71
7.1.5	Visualisation des trames reçues.....	72
7.2	CONFIGURATION DE LoRASERVER POUR L'INTEGRATION D'APPLICATION.....	74
7.2.1	Récupérer des données sur notre Application avec HTTP POST .....	74
7.2.2	Récupérer des données sur notre Application avec MQTT.....	74
<b>8</b>	<b>CREATION DE NOTRE PROPRE APPLICATION .....</b>	<b>76</b>
8.1	UTILISATION DE NODE-RED .....	76
8.1.1	Présentation .....	76
8.1.2	Mise en place de l'environnement .....	76
8.1.3	Créer une Application spécifique pour TTN.....	77
8.1.4	Création d'un Dashboard .....	79
8.2	PROGRAMMATION EN PHP .....	80
<b>9</b>	<b>VERSIONS DU DOCUMENT .....</b>	<b>82</b>

# 1 Les systèmes embarqués et l'IoT

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

### 1.1.1 Les systèmes embarqués dans l'IoT

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

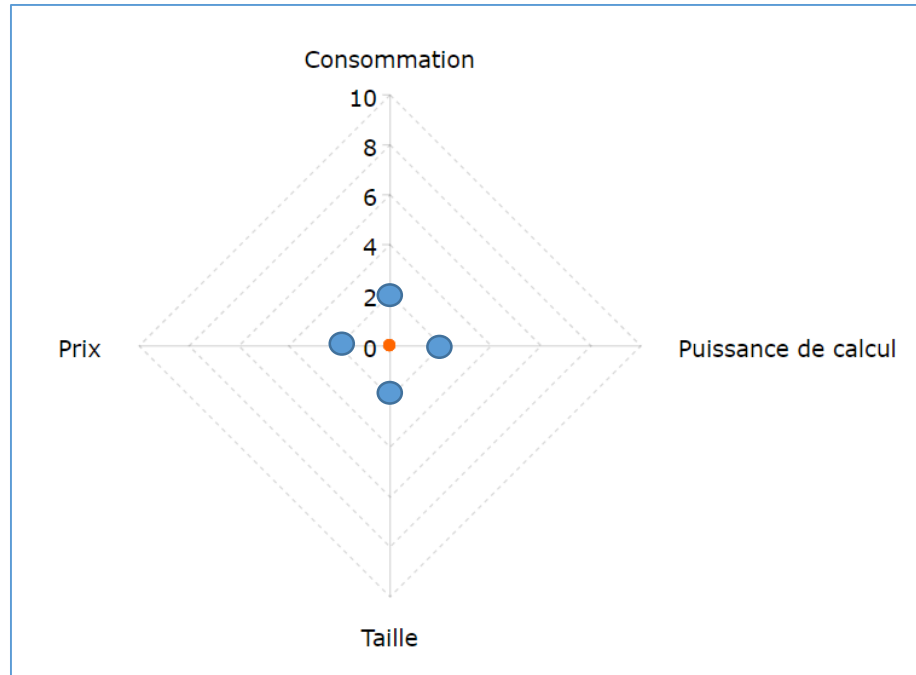


Figure 1 : Consommation, Puissance, Taille et Prix des objets connectés

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

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

### 1.1.2 Différents protocoles dans l'IoT



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

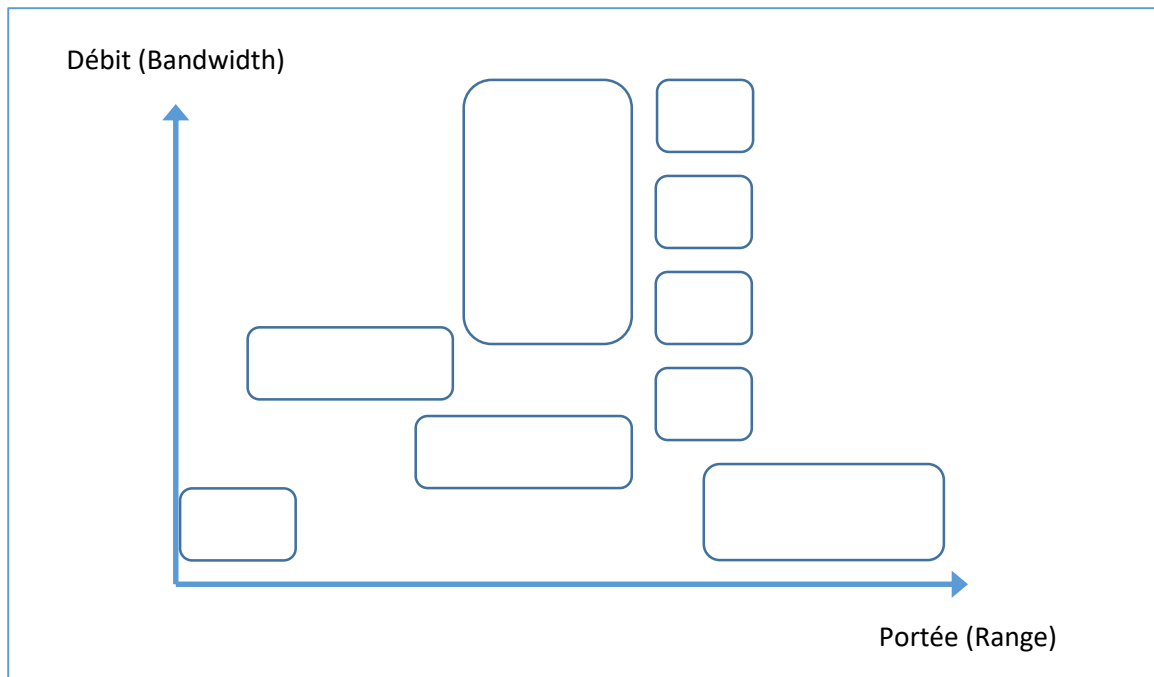


Figure 2 : Protocoles utilisés dans l'IoT en fonction du débit et de la portée

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

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

### 1.1.3 Bande de fréquence utilisées

En Europe les bandes de fréquences libres (sans licence) sont :

13 Mhz (RFID), 169 Mhz, 434 Mhz, 868 Mhz, 2.4Ghz, 5 Ghz et 24 Ghz (Faisceau Hertzien). Parmi ces fréquences, seul le 433 MHz et 868 MHz sont utilisables en LoRa.

## 1.2 Modes de partage du support

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

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

- FDM (Frequency Division Multiplexing) : Ce mode utilise **une partie du spectre** (bande de fréquence), **en permanence**. Exemple : Radio FM.
- TDM (Time Division Multiplexing). Ce mode utilise **tout le spectre** (bande de fréquence), **par intermittence**. Exemple : GSM, on a « 8 time slots » par canal.
- CDMA (Code Division Multiple Access) : Ce mode utilise **tout le spectre, tout le temps**.

Les Devices LoRa sont capables d'émettre sur **un même canal, 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. Plus tard dans le cours, nous expliquerons les différences de la modulation LoRa.

### 1.3 Notion d' "étalement de spectre" utilisée en LoRa

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 1 : 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 contenterons de vérifier leur bon fonctionnement.

#### 1.3.1 Transmissions successives

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 de « l'étalement de spectre », chaque transmission arrive bien à destination avec le message qui avait été transmis. Le premier tableau est déjà rempli en guise d'exemple. La méthode est la suivante :

A l'émission :

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

A la réception :

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

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

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



1'	<b>Message User 2</b>	<b>0</b>	<b>1</b>	<b>0</b>	<b>1</b>
2'	Utilisation code orthogonal User 2	1 1 -1 -1	1 1 -1 -1	1 1 -1 -1	1 1 -1 -1
3'	Symboles transmis User 2 = (1') x (2')	0 0 0 0			
... 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			
6'	<b>Message reçu (<math>\sum (5') / \text{nbr\_bits}</math>)</b>	<b>0</b>			

1''	<b>Message User 3</b>	<b>1</b>	<b>1</b>	<b>0</b>	<b>0</b>
2''	Utilisation code orthogonal User 3	1 -1 -1 1	1 -1 -1 1	1 -1 -1 1	1 -1 -1 1
3''	Symboles transmis User 3 = (1'') x (2'')	1 -1 -1 1			
... 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			
6''	<b>Message reçu (<math>\sum (5'') / \text{nbr\_bits}</math>)</b>	<b>1</b>			

### 1.3.2 Transmissions simultanées

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

Dans l'air :

- On additionne les symboles transmis par tous les User (1, 2, 3) : ligne 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

➡ Valider le fonctionnement pour la réception des messages

1'''	<b><math>\sum</math> des symboles transmis ( 3 + 3' + 3'')</b>	<b>2 -2 0 0</b>			
------	--	-----------------	--	--	--

2'''	Utilisation code orthogonal User 1	1 -1 1 -1	1 -1 1 -1	1 -1 1 -1	1 -1 1 -1
3'''	Décodage (1''') x (2''')	2 2 0 0			
4'''	<b>Message reçu User 1 (<math>\sum (3''') / \text{nbr\_bits}</math>)</b>	<b>1</b>			

2'''	Utilisation code orthogonal User 2	1 1 -1 -1	1 1 -1 -1	1 1 -1 -1	1 1 -1 -1
3'''	Décodage (1''') x (2''')	2 -2 0 0			
4'''	<b>Message reçu User 2 (<math>\sum (3''') / \text{nbr\_bits}</math>)</b>	<b>0</b>			

2'''	Utilisation code orthogonal User 3	1 -1 -1 1	1 -1 -1 1	1 -1 -1 1	1 -1 -1 1
3'''	Décodage (1''') x (2''')	2 2 0 0			
4'''	<b>Message reçu User 3 (<math>\sum (3''') / \text{nbr\_bits}</math>)</b>	<b>1</b>			

### 1.3.3 Cas du protocole LoRa

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

## 2 Transmission radio et propagation

### 2.1 Les unités et définitions

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

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

Tableau 2 : Comparaison entre les gains en dB et en proportion

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



➔ En reprenant la définition des dB remplir le tableau suivant :

Puissance en dBm	Puissance en mW
+ 3 dBm	
- 3 dBm	
+ 10 dBm	
- 10 dBm	

Tableau 3 : Comparaison entre les puissances en dB et en mW



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

- **Sensibilité** : Puissance minimale qu'est capable de détecter un récepteur
- **RSSI** : Received Signal Strength Indication.
- **SNR** : Signal over Noise Ratio



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

- **Le link Budget** : C'est la quantité de puissance que nous pouvons perdre avant d'arriver au récepteur, tout en conservant une transmission fonctionnelle. Autrement dit, c'est la puissance de l'émetteur **moins** la sensibilité du récepteur. Dans l'exemple précédent, le budget que nous avons à disposition est de 93dB.



- ➔ En LoRa, nous avons un Link Budget de 157 dB
- ➔ En LTE (4G), nous avons un Link Budget de 130 dB

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

Le RN2483 est le composant radiofréquence que nous utiliserons dans notre transmission LoRa. Voici quelques une de ses caractéristiques.

KEY PRODUCT FEATURES	
◆	LoRa™ Modem
◆	157 dB maximum link budget
◆	+20 dBm at 100 mW constant RF output vs. V supply
◆	+14 dBm high efficiency PA
◆	Programmable bit rate up to 300 kbps
◆	High sensitivity: down to -137 dBm

Figure 3 : Caractéristiques principales du composant radiofréquence utilisé

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

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

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 4 : Influence du Spreading Factor sur le SNR acceptable

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

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

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

## 3 La modulation LoRa (couche physique)

### 3.1 La modulation LoRa

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

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

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

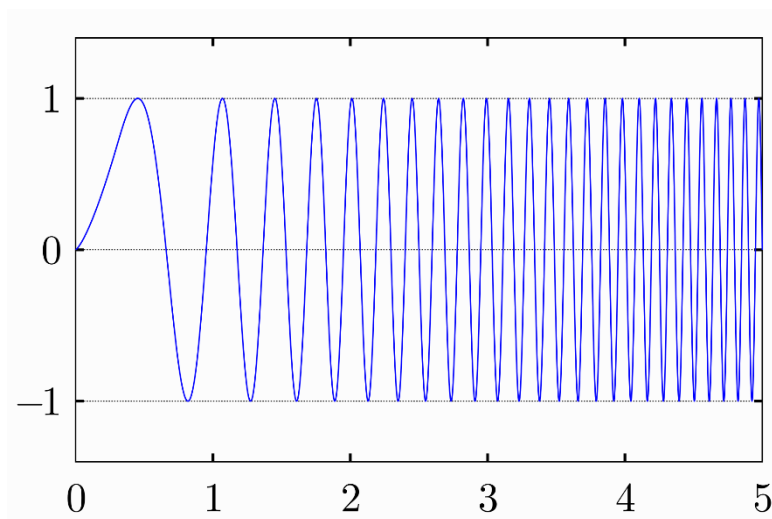


Figure 5 : Symbole de la modulation LoRa (source Wikipédia)

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

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



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

- Fréquence de début :
- Fréquence de fin :

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

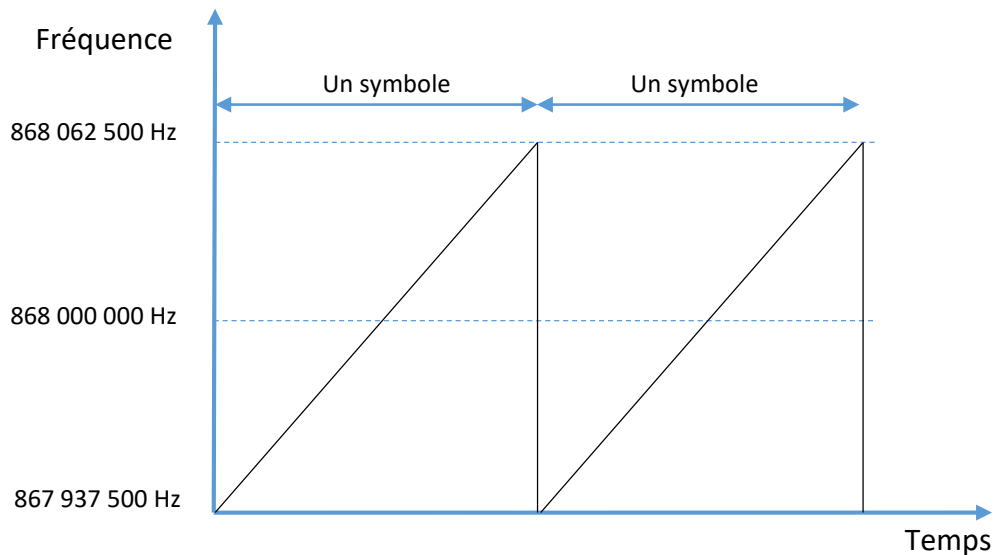


Figure 6 : Forme du symbole de base (CHIRP)

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.

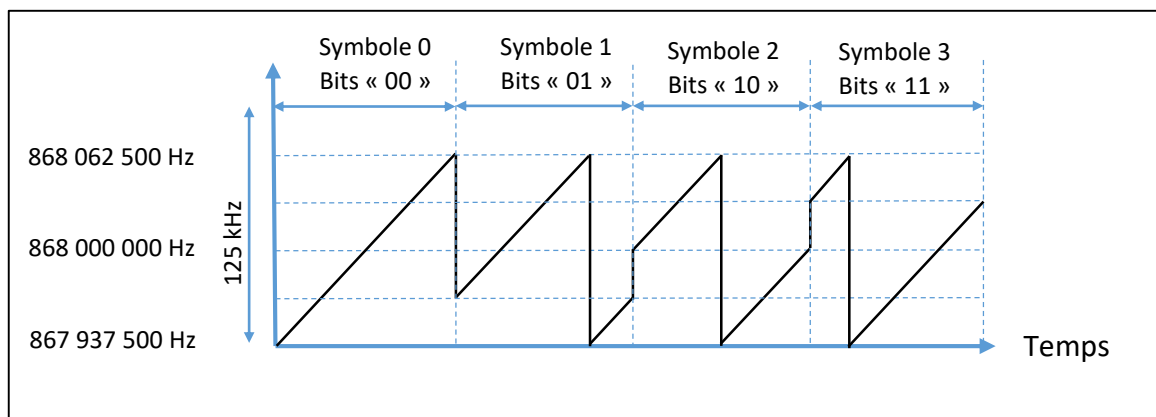


Figure 7 : Symboles émis en Modulation LoRa (Cas théorique en SF2)

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 ( $2^{10}$ ).

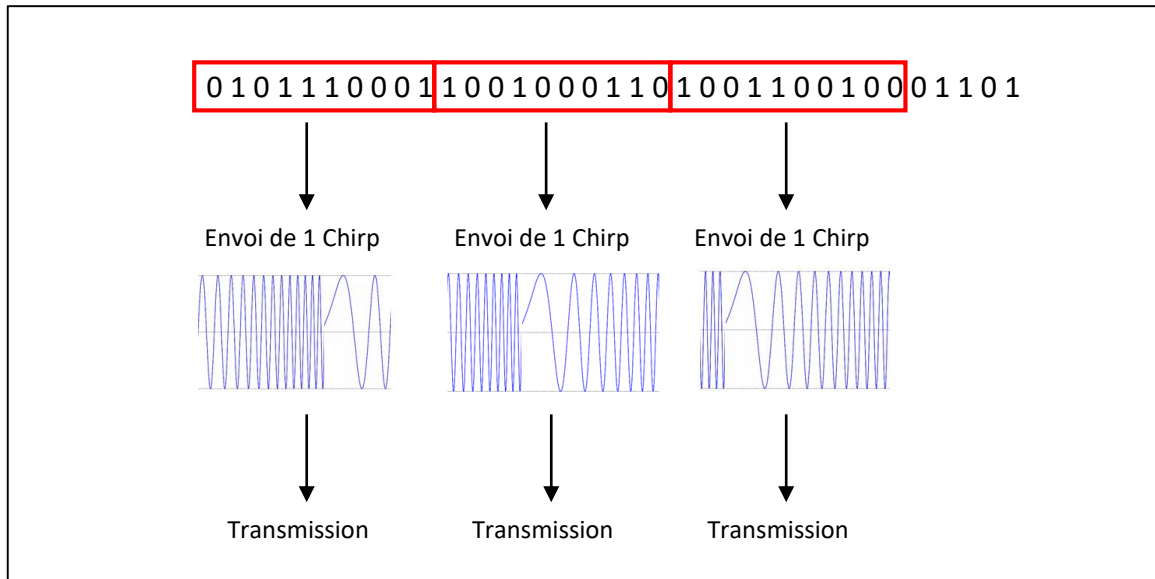


Figure 8 : Emission des Chirp en LoRa

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

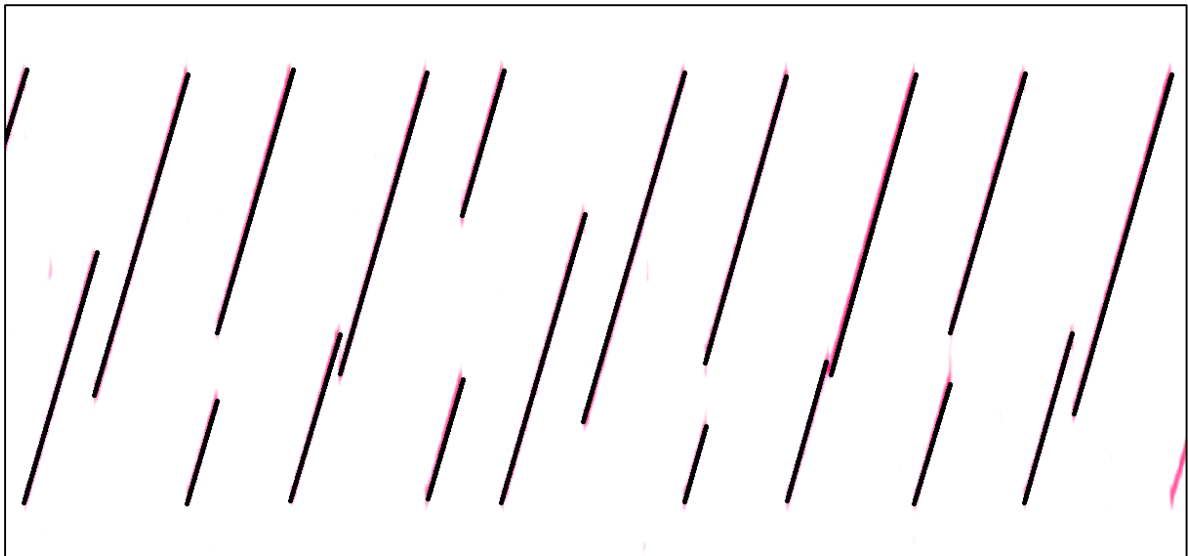


Figure 9 : Visualisation des Chirps LoRa réellement émis pendant une transmission

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

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.

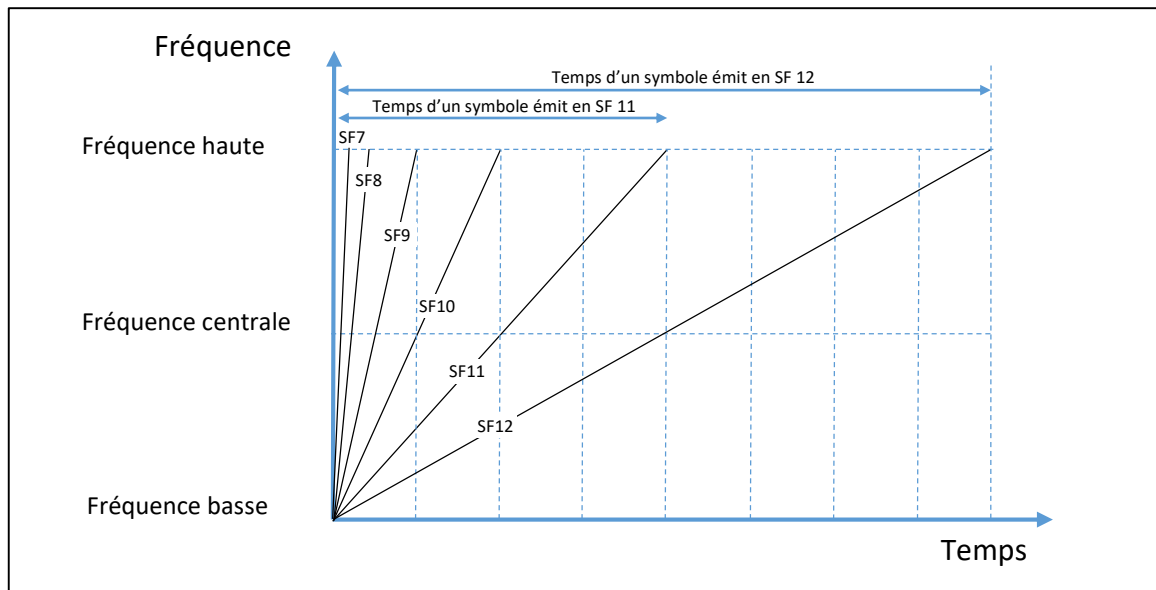


Figure 10 : Temps d'émission d'un symbole (Chirp) en fonction du SF

En revanche, le temps d'émission d'un symbole est inversement proportionnel à la bande passante :  $T_s = \frac{2^{SF}}{Bandwidth}$ . Le débit des symboles est donc de  $\frac{1}{T_s} = F_s = \frac{Bandwidth}{2^{SF}}$ . En toute logique, plus la bande passante est élevée, plus le débit des symboles sera élevé. Comme chaque symbole comprend SF bit, on retrouve alors le débit binaire :

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



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

Exemple : D'après la formule du débit binaire, trouver le débit pour les deux cas suivants :

- **Cas 1 :** Pour SF7 et 125 kHz      >      Débit =
- **Cas 2 :** Pour SF12 et 125 kHz      >      Débit =

### 3.2 Coding Rate

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



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

Figure 11 : Influence du Coding Rate sur le nombre de bits ajoutés

Si on reprend l'exemple précédent avec un CR de 4 / 5, nous avons une augmentation de 1.25 du nombre de bits à transmettre. Redonner la débit en prenant en compte le CR.

- **Cas 1** : Pour SF7 et 125 kHz > Débit =
- **Cas 2** : Pour SF12 et 125 kHz > Débit =

➡ La documentation du composant RN2483 donne les débits en fonction du Spreading Factor, la Bande Passante et le Coding Rate. Vérifier la cohérence du résultat avec votre calcul précédent.

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

Figure 12 : Débit en fonction des paramètres de la transmission LoRa

### 3.3 Utilisation du LoRa Calculator

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

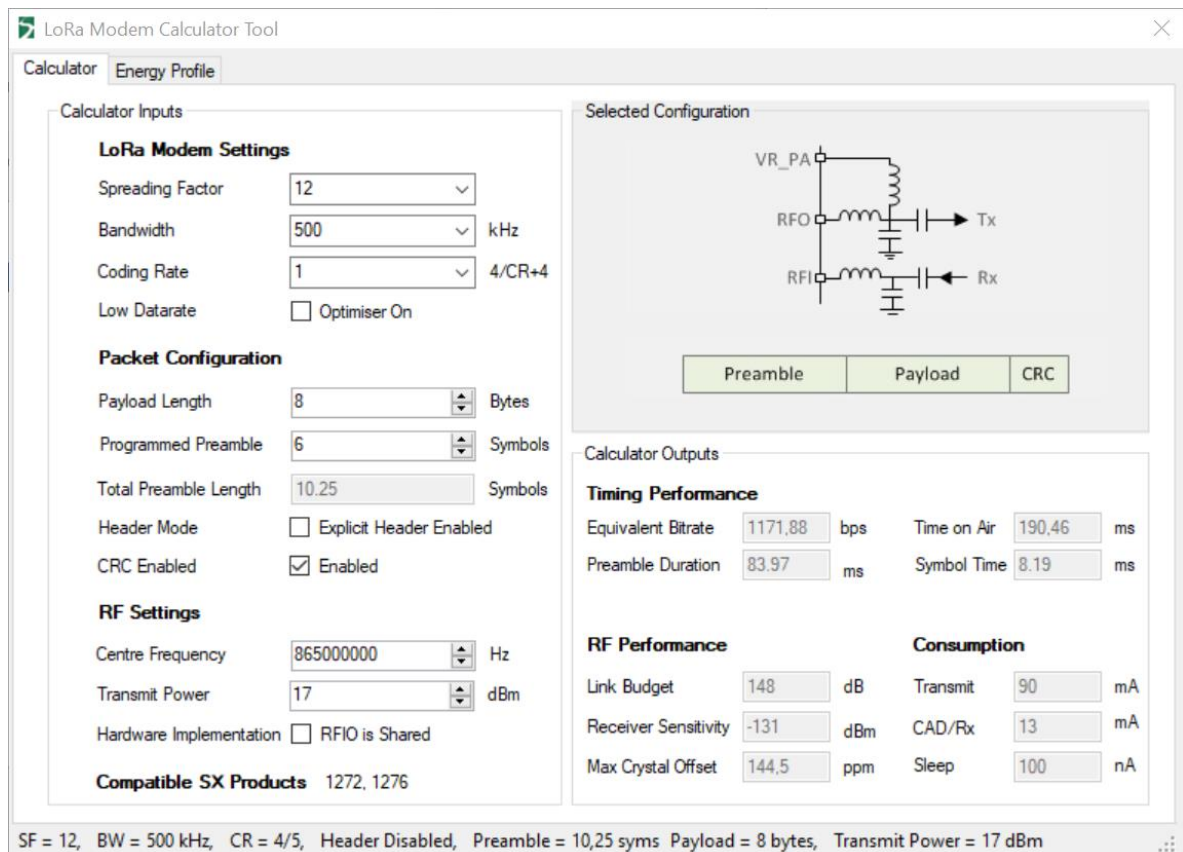


Figure 13 : Le logiciel LoRa Calculator



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

### 3.4 Time On Air

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

➔ En reprenant les exemples précédents, quels sont les débits moyens si on prend en compte le « Time On Air » ?

- Cas 1 : Pour SF7 et 125 khz > Débit =
- Cas 2 : Pour SF12 et 125 kHz > Débit =



➔ Dans le logiciel « LoRa Calculator » (onglet « Energy Profile »), nous pouvons estimer la durée de vie d'une batterie. Nous prenons l'exemple d'un relevé de température dans un bâtiment avec les caractéristiques suivantes :

- SF, CR, Bande Passante, etc : Nous nous plaçons dans le cas 1 des exemples précédents.
- Le relevé de température se fera toutes les 15 mins
- La batterie est constituée de 3 petites piles bâtons [ AAA / LR03 ] 1.2V en série. Soit 3.6 V / 1000 mAh.

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

Ce test sera réalisé avec des modules LoRa RN2483 (Microchip) et une carte microcontrôleur ARDUINO.

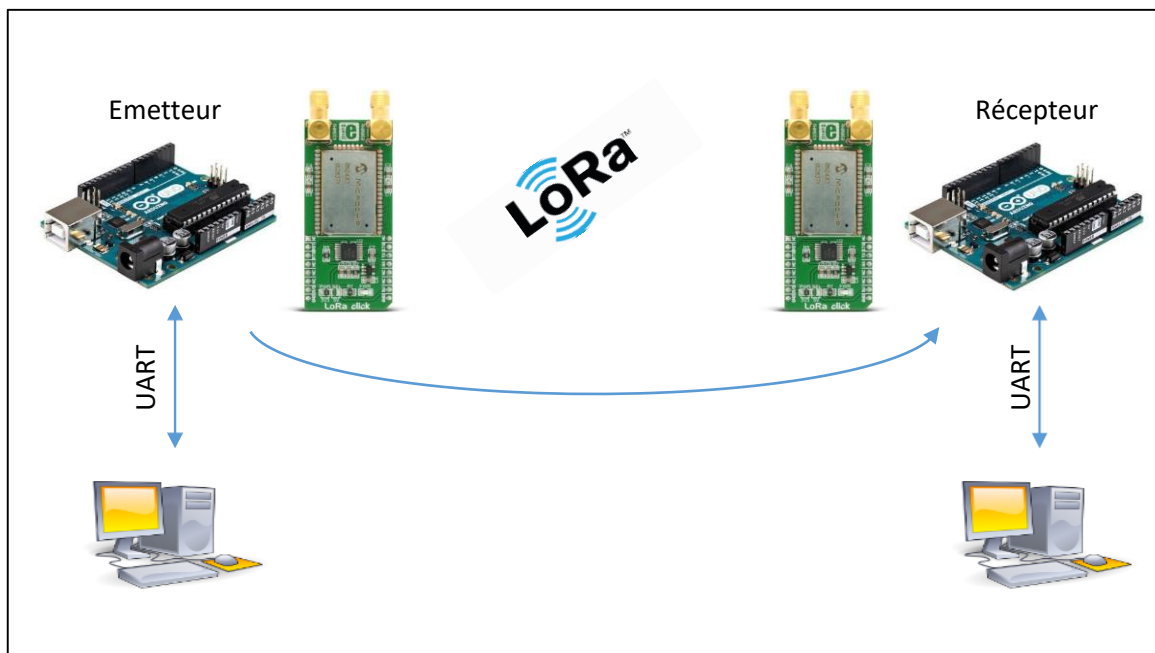


Figure 14 : Communication LoRa en Point à Point

➔ Réaliser le câblage de votre module RN2483 suivant le tableau suivant :

Connecteur Arduino	Click Board RN2483
5V	5V
GND	GND
11	RX
10	TX

Tableau 4 : Câblage de l'ARDUINO et de la click Board RN2483

#### 3.5.1 Utilisation de l'IDE Arduino

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

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

Il y a deux parties dans un sketch :

- Une fonction `setup()` qui sera exécutée qu'une seule fois au démarrage
- Une fonction `loop()` qui sera exécutée en boucle

➔ Ecrire le code suivant :

```

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

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

```

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

### 3.5.2 Validation du fonctionnement

Deux binômes joueront le rôle d'émetteur, tous les autres binômes seront en récepteur.

- ➔ Récupérer les programmes dans Moodle dans le dossier « **RN2483-Arduino-Point2Point** ».
- ➔ Installer la librairie RN2483 : **Arduino IDE > Croquis > Inclure une bibliothèque > Ajouter la bibliothèque .zip.**
- ➔ Répartissez les rôles Emetteurs / Récepteurs et récupérer le programme LoraBlinkerRX ou LoraBlinkerTX en fonction. Les 2 binômes en émission devront modifier le code du sketch Arduino pour transmettre des données spécifiques afin de les différencier. [ `loraSerial.println("radio tx 30");` ]
- ➔ Validez la réception des données émises par les deux binômes qui transmettent des données. Ces données reçues sont écrites à **57600 bauds** sur le moniteur série de l'Arduino.

### 3.5.3 Mise en valeur du Spreading Factor

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

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

## 4 Le protocole LoRaWAN

Le protocole LoRa est le type de modulation utilisé entre deux Devices ou entre un Device et une Gateway. Lorsque nous parlons de l'ensemble de la chaîne de communication (Device, Gateway, Serveur) alors nous parlons de communication LoRaWAN. Nous allons voir l'architecture d'un réseau LoRaWAN, ainsi que les règles de ce protocole.

### 4.1 Structure d'un réseau LoRaWAN

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

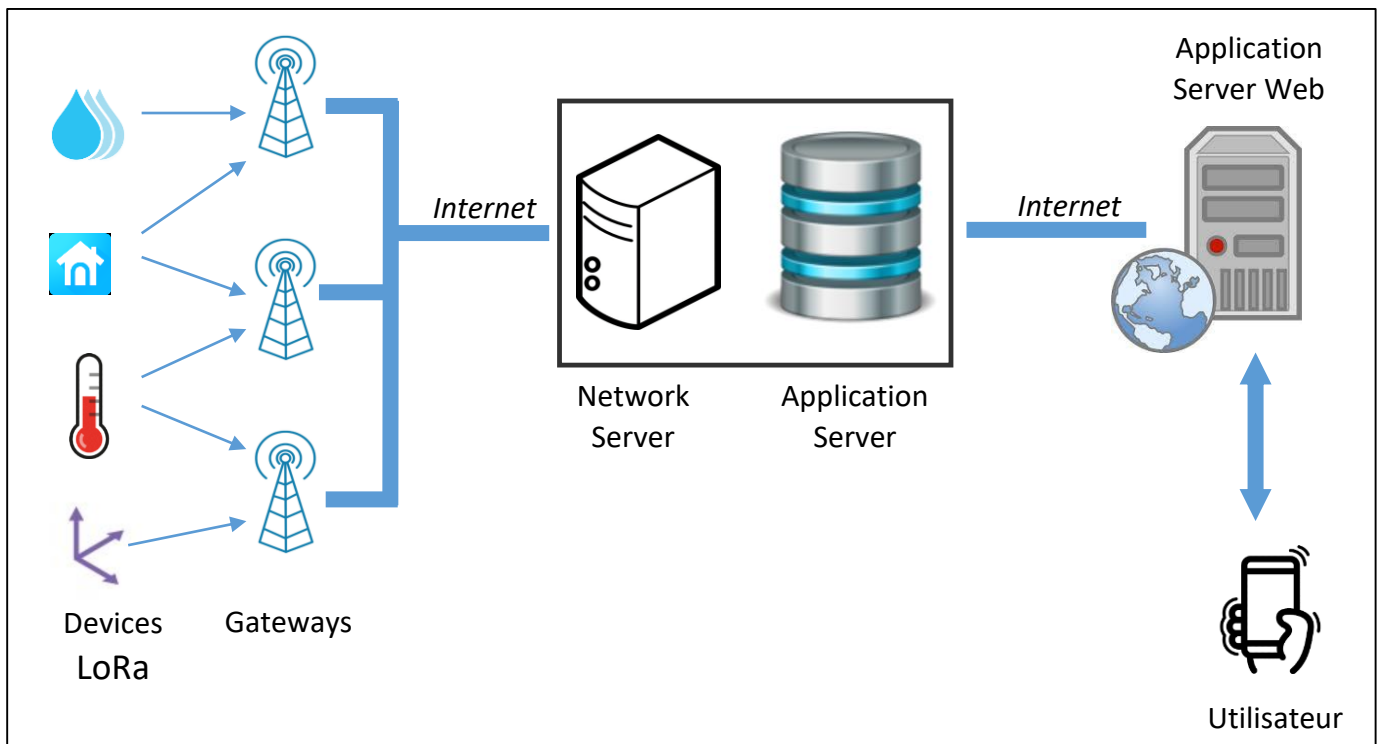


Figure 15 : Structure globale d'un réseau LORAWAN

#### 4.1.1 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.

#### 4.1.2 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.

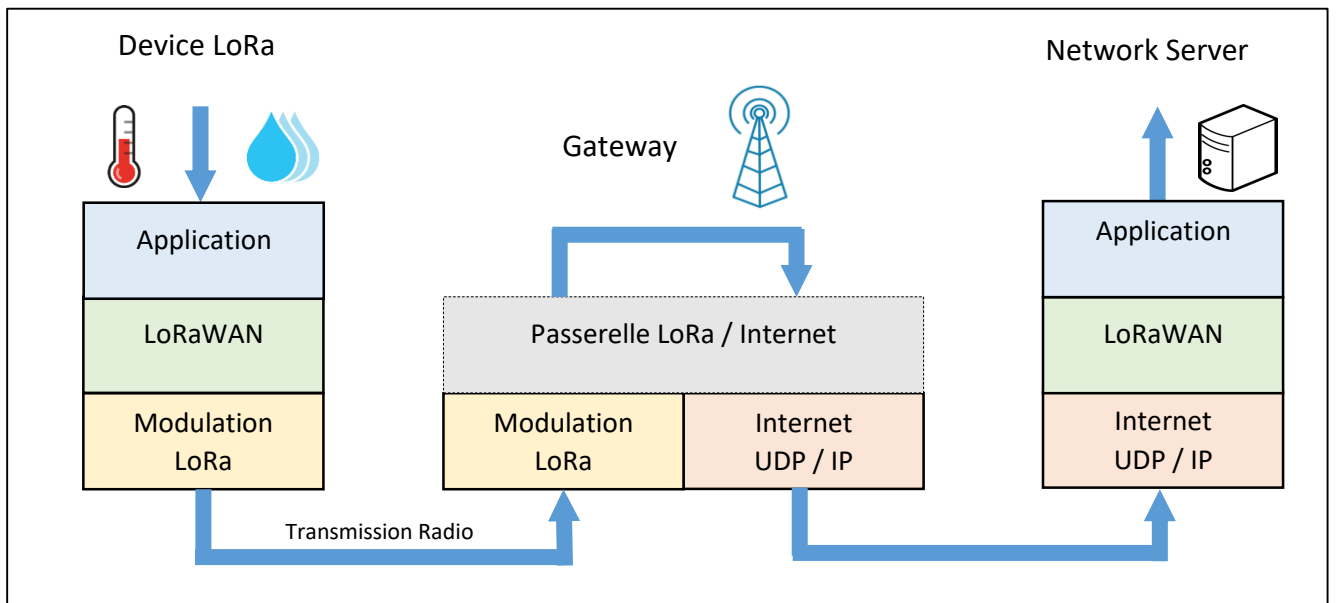


Figure 16 : Le rôle de la Gateway LoRa

Chaque Gateway LoRa possède un identifiant unique (EUI sur 64 bits).

#### 4.1.3 Le Network Server

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

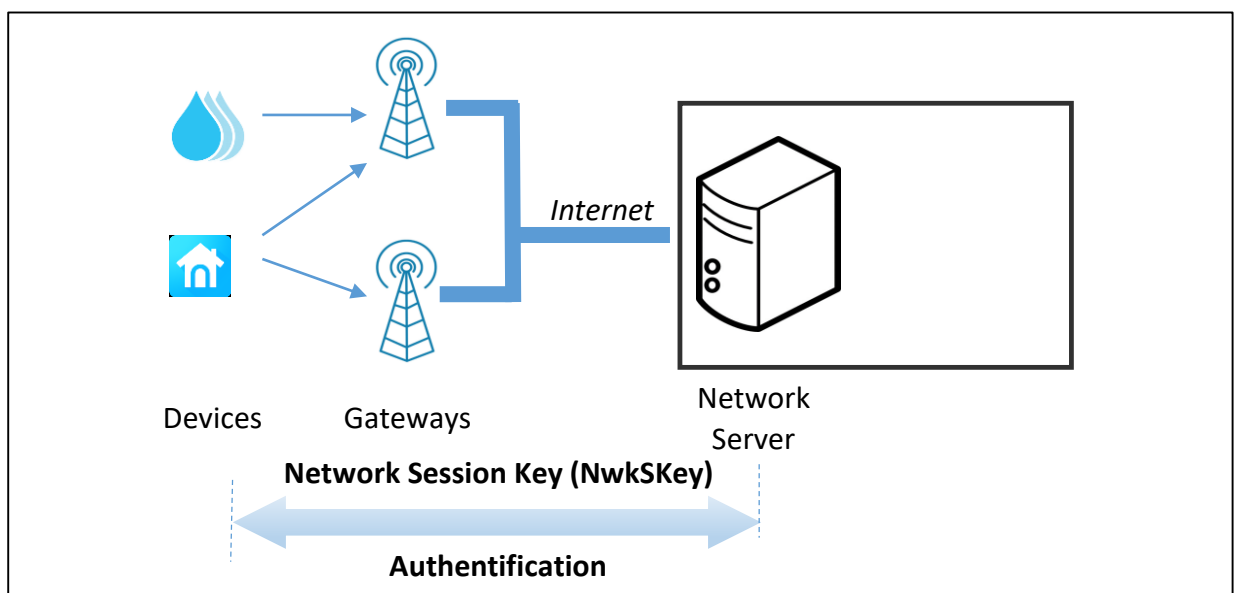
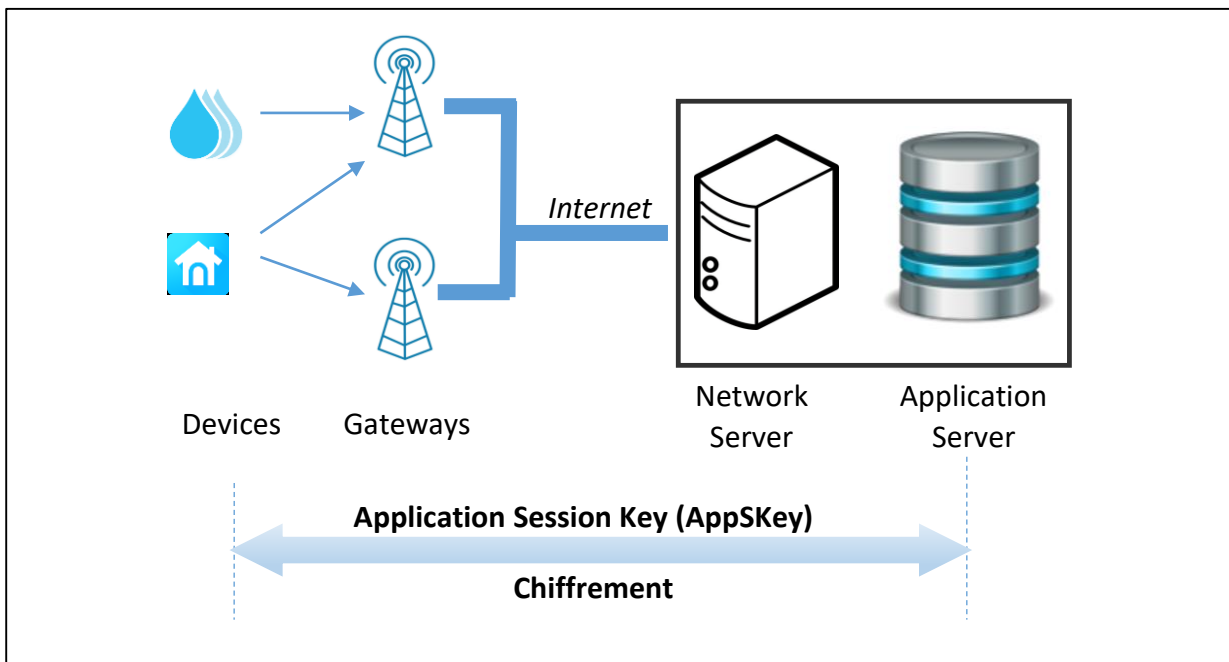


Figure 17 : Authentification entre le Device LoRa et le Network Server

#### 4.1.4 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 grâce à une clé AES 128 bits appelée Application Session Key : **AppSKey**.



*Figure 18: Chiffrement entre le Device LORA et l'Application Server*

Le schéma suivant résume l'utilisation :

- De la Network Session Key (NwkSKey) pour l'authentification entre le Device LoRa et le Network Server
- De l'Application Session Key (AppSKey) pour le chiffrement entre le Device LoRa et l'Application Server.

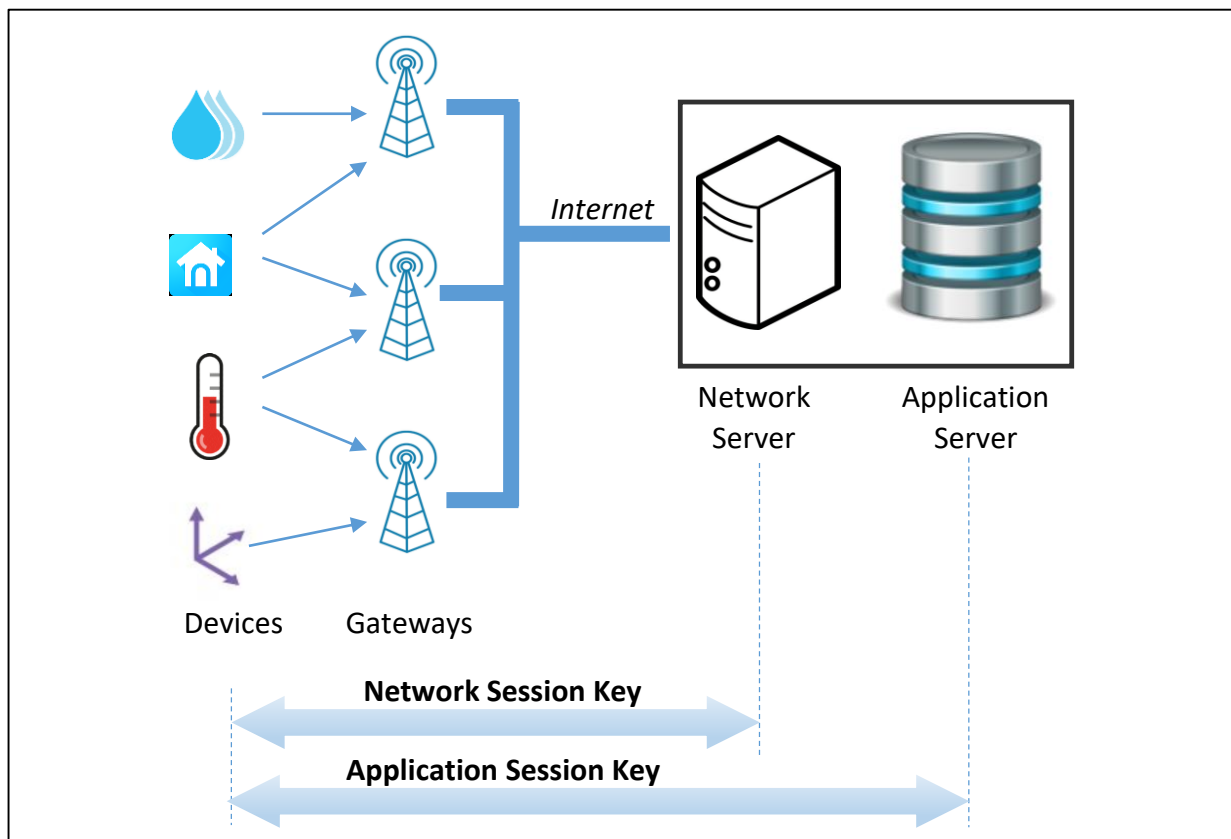


Figure 19 : Authentification et chiffrement en LoRaWAN

#### 4.1.5 Application / Web Serveur

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

- Avec le protocole HTTP
- Avec le protocole MQTT

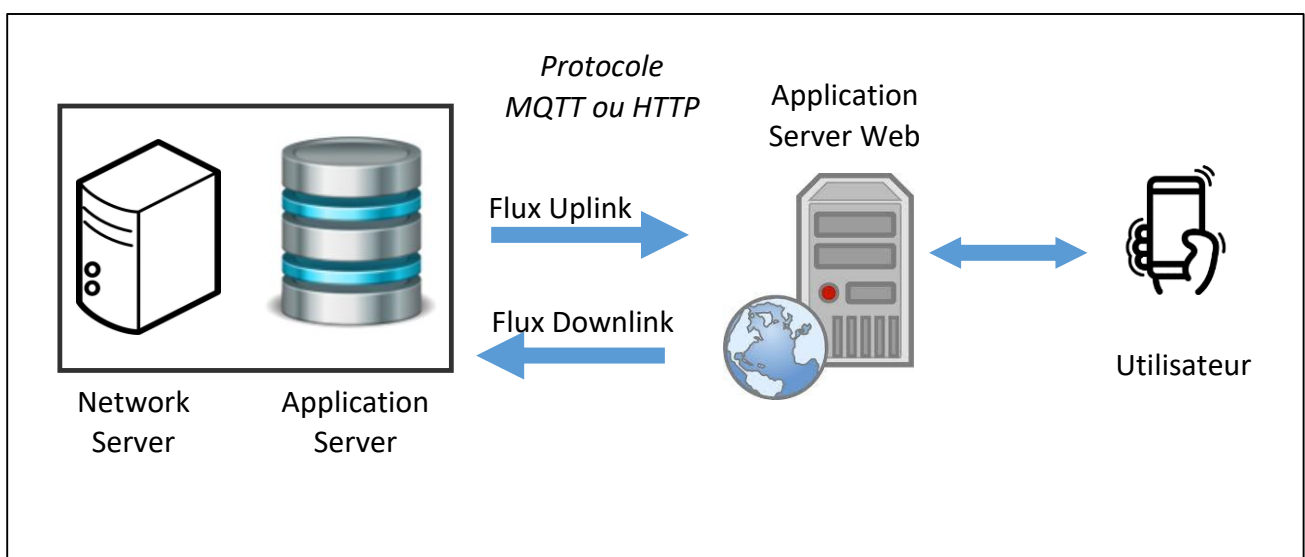


Figure 20 : Application / Web Server



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

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

#### 4.1.6 Authentification avec le Network Server

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

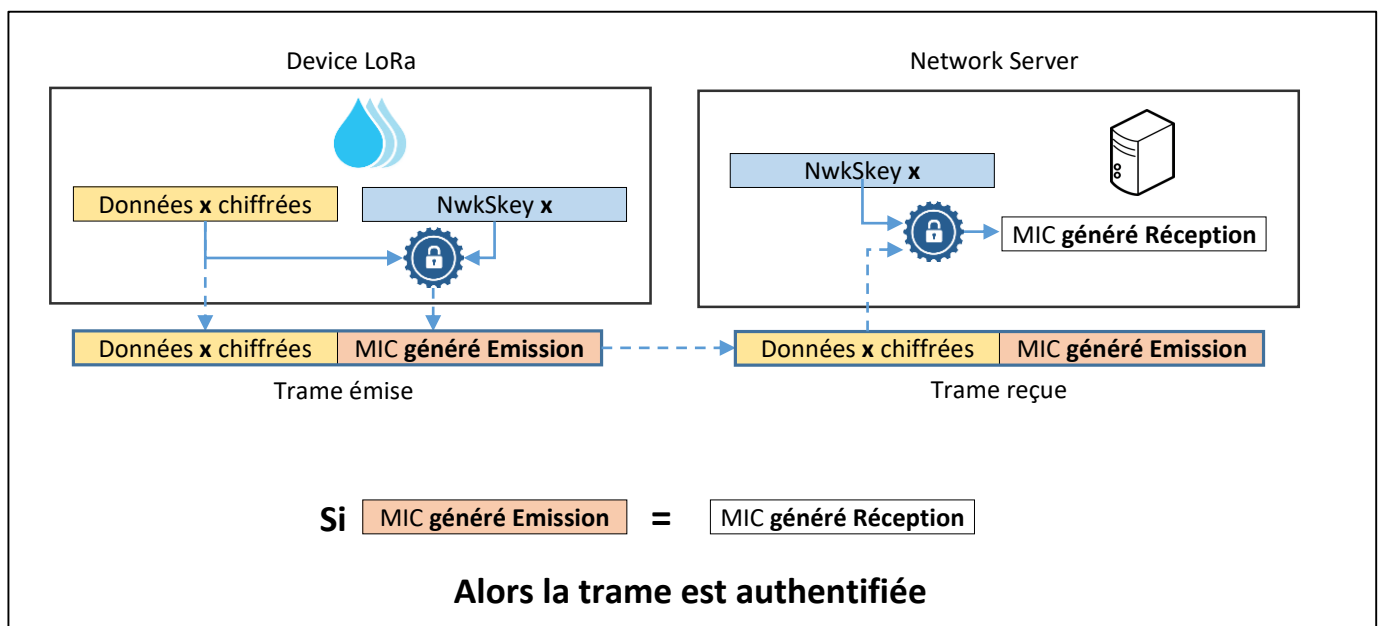


Figure 21 : Authentification d'un Device LoRa par le Network Server

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

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

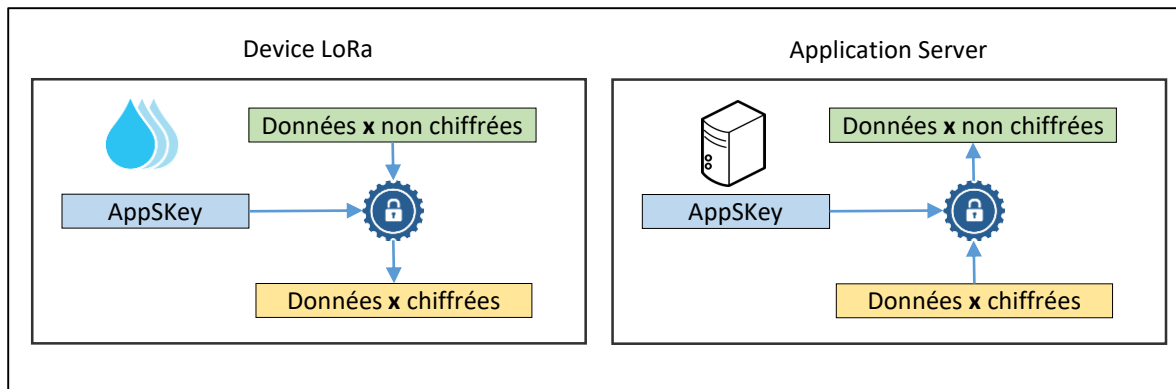


Figure 22 : Chiffrement des données par l'Application Session Key

#### 4.1.8 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.

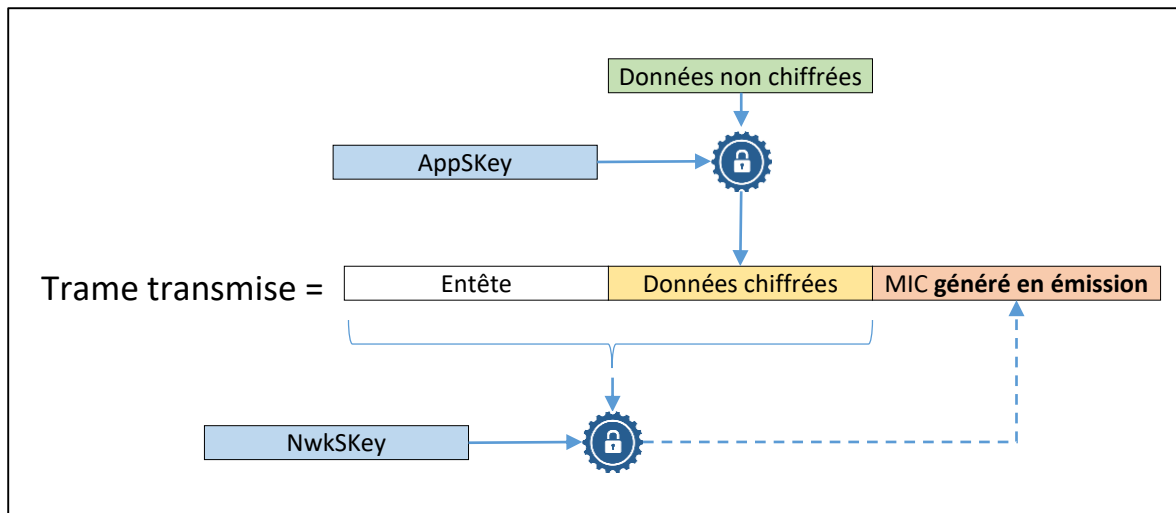


Figure 23 : Chiffrement, puis Authentification

## 4.2 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.

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

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

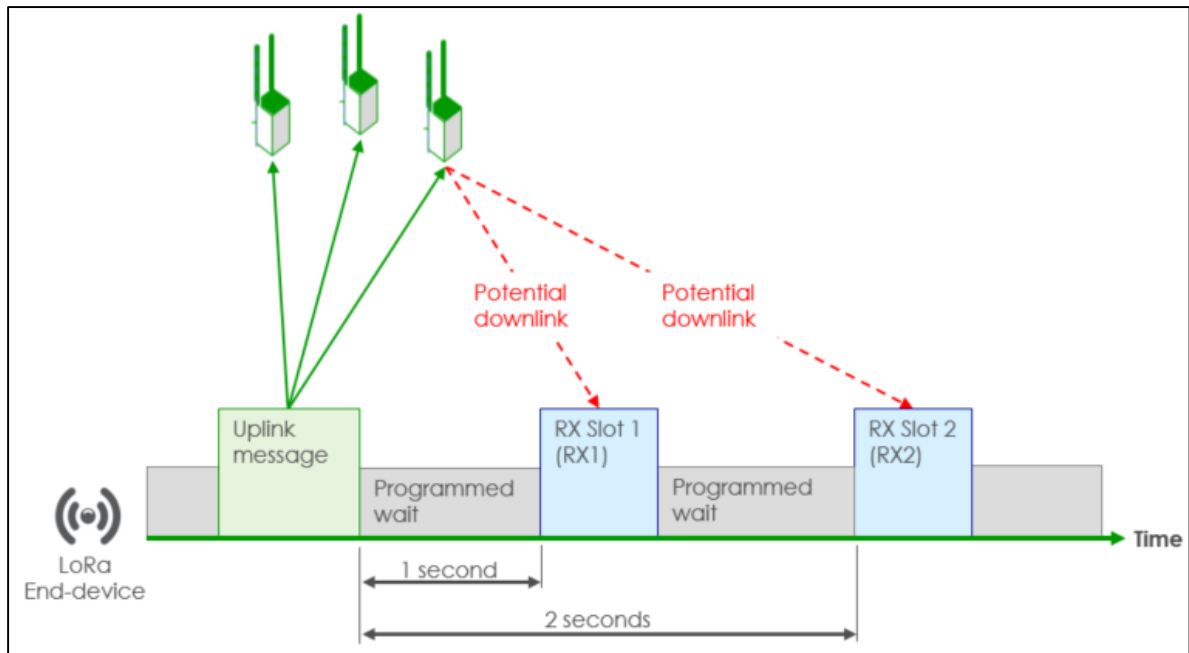


Figure 24 : Slots de réception pour un Device LoRa de classe A.

Source de l'image : <https://zakelijkforum.kpn.com>

La durée des fenêtres doit être au minimum la durée de réception d'un préambule. Un préambule dure  $12.25 T_{\text{symbole}}$  et dépend donc du Data Rate (DR : voir paragraphe 4.4.5 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  $\pm 20 \mu\text{s}$  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  $\pm 20 \mu\text{s}$  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.**

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

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

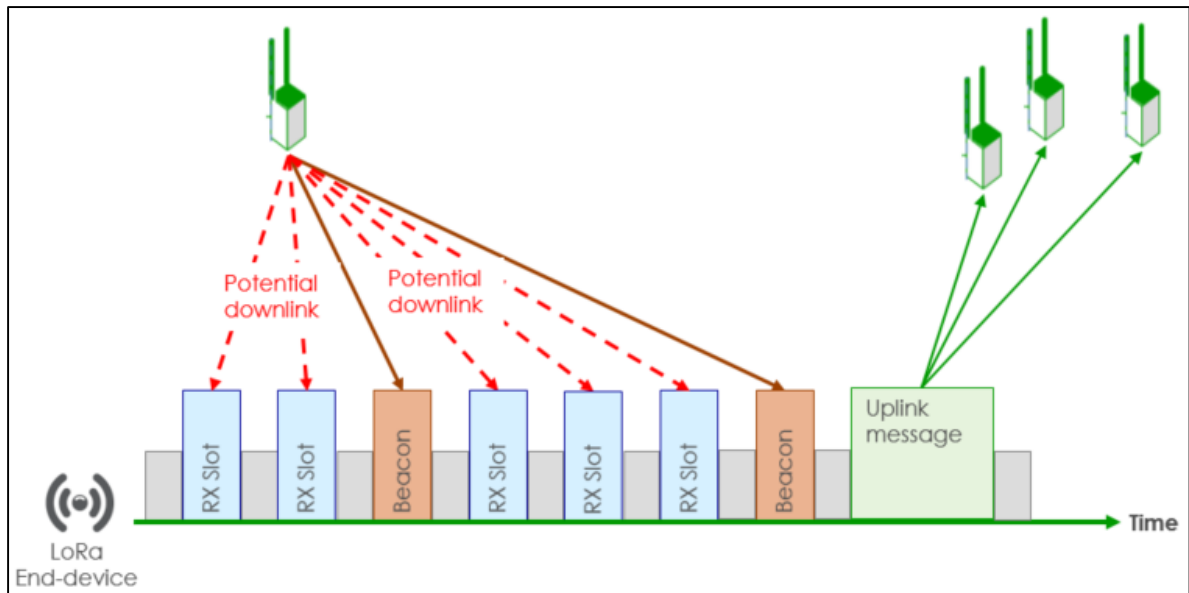


Figure 25 : Slots de réception pour un Device LoRa de classe B

Source de l'image : <https://zakelijforum.kpn.com>



➔ 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.

#### 4.2.3 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.

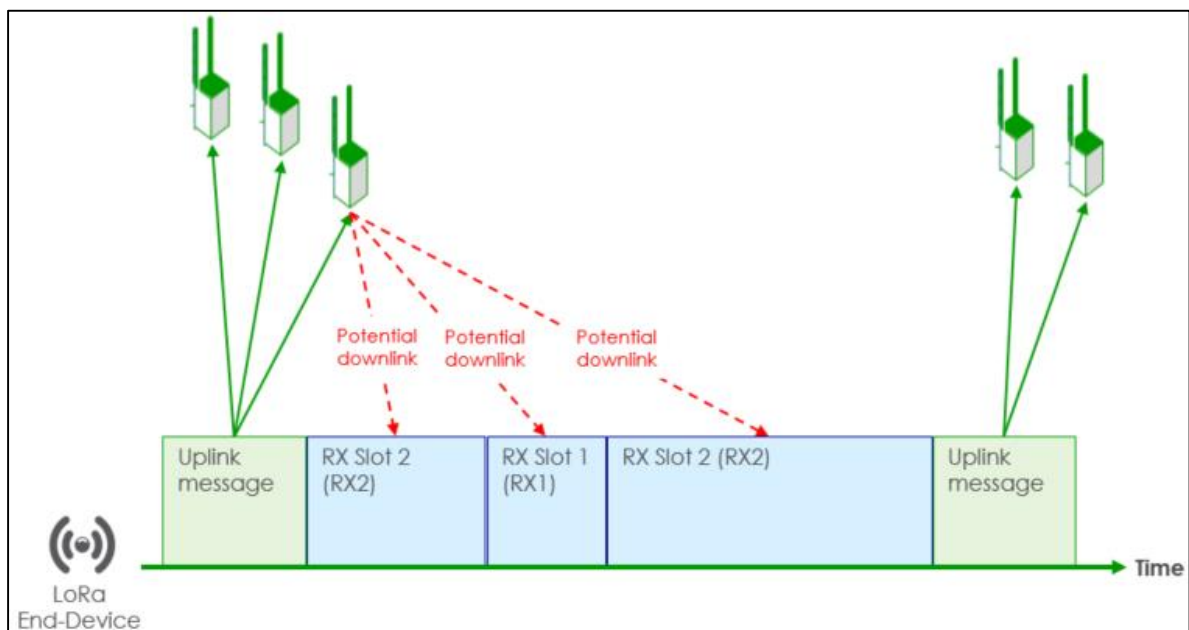


Figure 26 : Slots de réception pour un Device LoRa de classe C

Source de l'image : <https://zakelijforum.kpn.com>



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

#### 4.2.4 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.

### 4.3 Activation des Devices LoRa et sécurité

En LoRaWAN, les trois éléments indispensables pour la communication sont le **DevAddr** pour l'identification du Device, ainsi que deux clés : le **NwkSKey** pour l'authentification et l'**AppSKey** pour le chiffrement. Deux méthodes sont possibles pour fournir ces informations à la fois au Device LoRa et au serveur :

- Activation By Personalization : APB
- Over The Air Activation

#### 4.3.1 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 possède déjà le **DevAddr** et le **NwkSKey**
- L'application serveur possède déjà le **DevAddr** et le **AppSKey**



- ➔ En ABP, toutes les informations nécessaires à la communication sont déjà connues par le Device LoRa et par le Serveur.

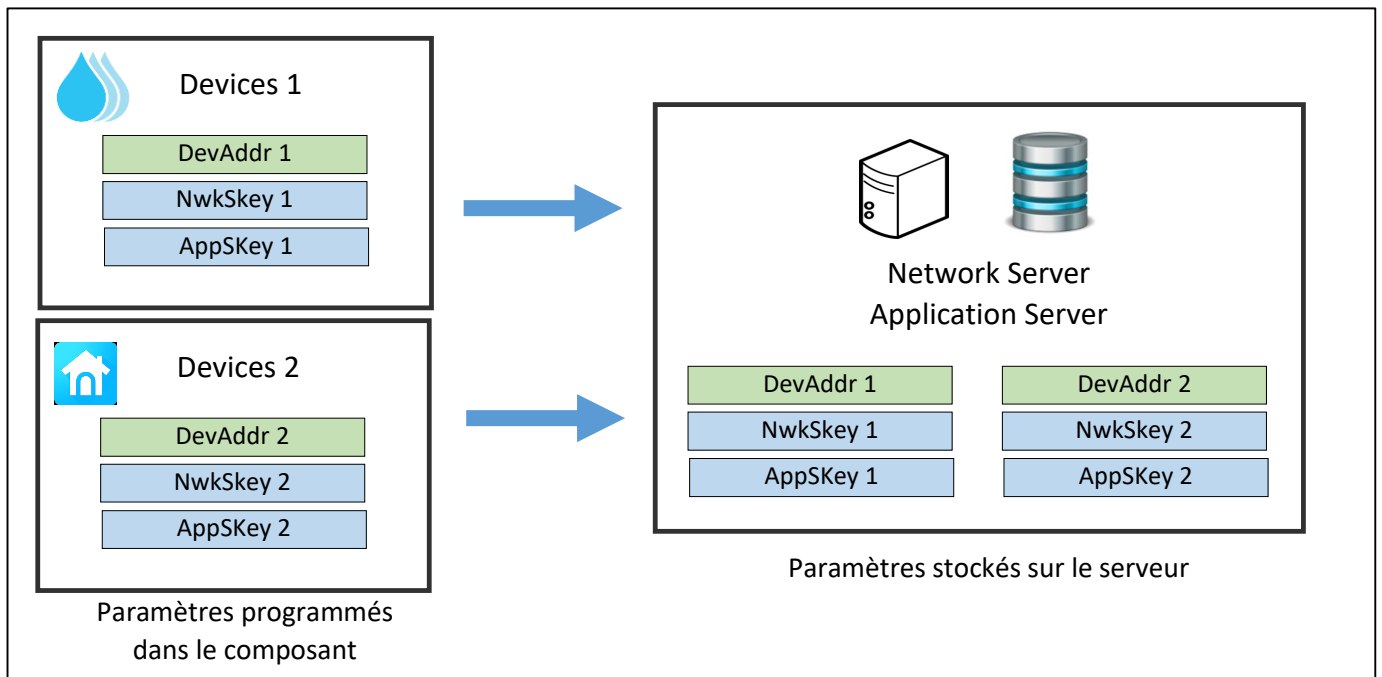


Figure 27 : Paramétrage de DevAddr, NwkSKey et AppSKey

#### 4.3.2 OTAA : Over The Air Activation :

C'est la méthode qu'il faut privilégier car c'est celle qui est la plus sécurisée. Le Device LoRa doit connaître : le **DevEUI**, l'**AppEUI**, et l'**Appkey**. Le Network Server doit lui connaître la même **Appkey**.



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

Grâce à ces informations de départ et une négociation avec le Network Server (Join Request), le Device LoRa et le serveur vont générer les informations essentielles : **DevAddr**, **NwkSKey** et **AppSKey**.

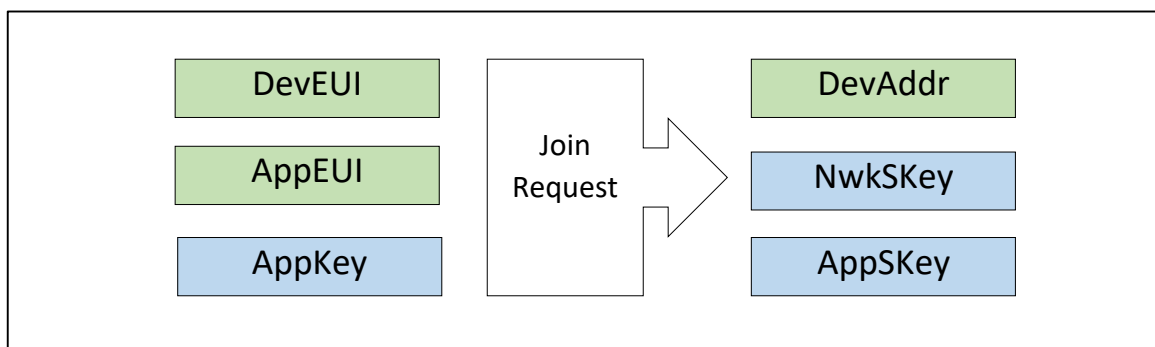


Figure 28 : Obtention du DevAddr, NwkSKey et AppSKey en OTAA

- 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.
- **NwkSKey** : Utilisé pour l'authentification avec le Network Server
- **AppSKey** : Utilisé pour le chiffrement des données

### 4.3.3 Sécurité

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.

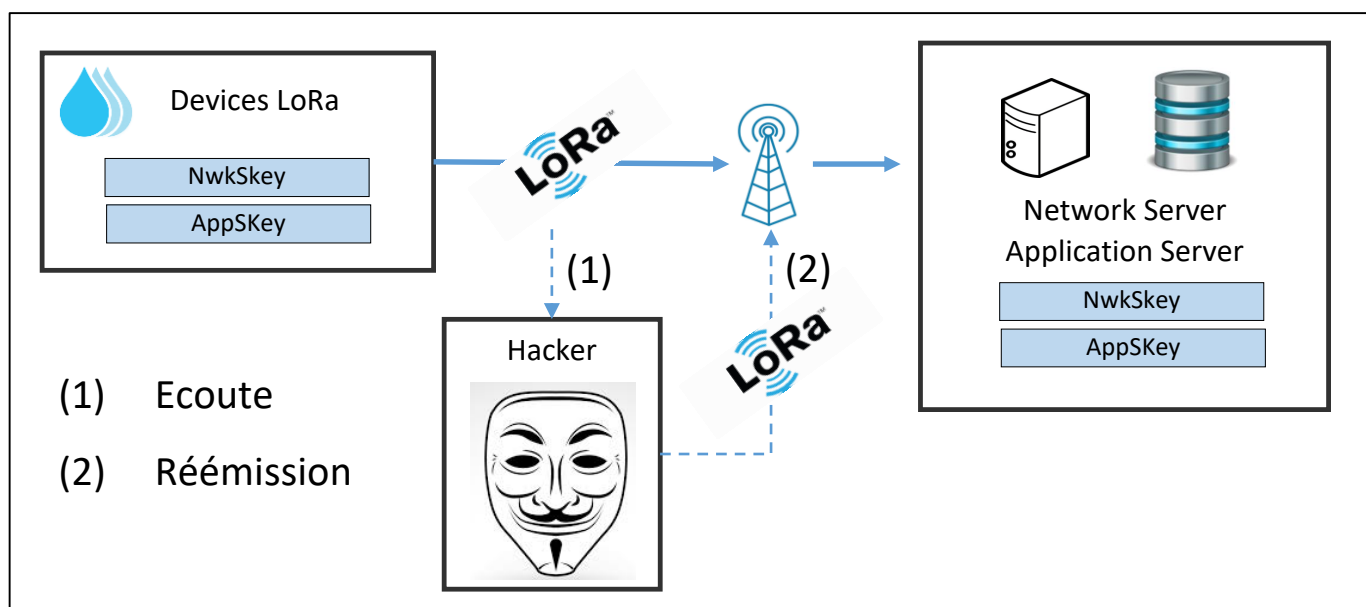


Figure 29 : Attaque par REPLAY

Pour éviter cela, la trame LoRaWAN intègre un champ variable appelé « Frame Counter ». Il s'agit d'un nombre numérotant la trame sur 16 ou 32 bits. L'Application 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. Et comme la trame est chiffrée le Hacker n'a pas moyen de savoir comment modifier un champ de cette trame.

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

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

#### ☐ **Frame Counter Checks**

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

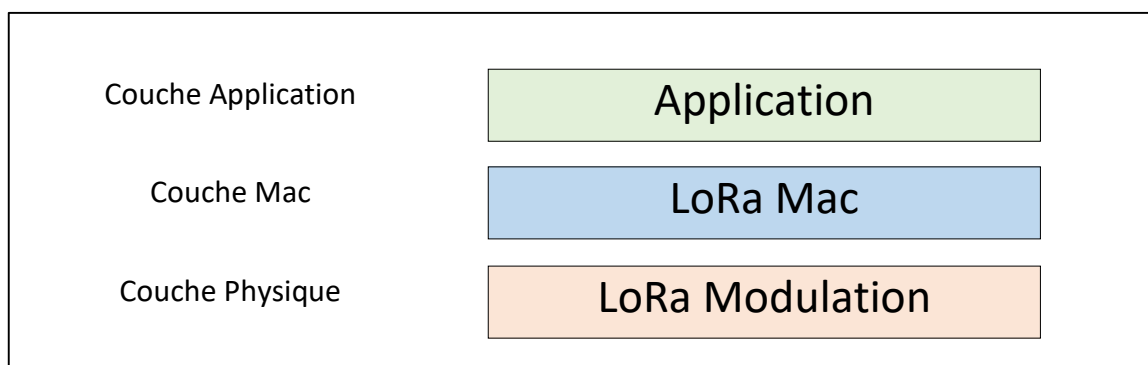
*Figure 30 : Désactivation du "Frame Counter Check" dans TTN*

2. Utiliser l'authentification 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.

## 4.4 La Trame LoRa / LoRaWAN

### 4.4.1 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 31 : Les couches protocolaires du LoRaWAN*

Chaque couche rajoute un service. Lors de l'envoi de la trame, les données utilisateurs sont donc encapsulées dans chaque couche inférieure jusqu'à la transmission. Le détail de l'ensemble de la trame LoRaWAN par couche est décrit sur la figure Figure 32 :



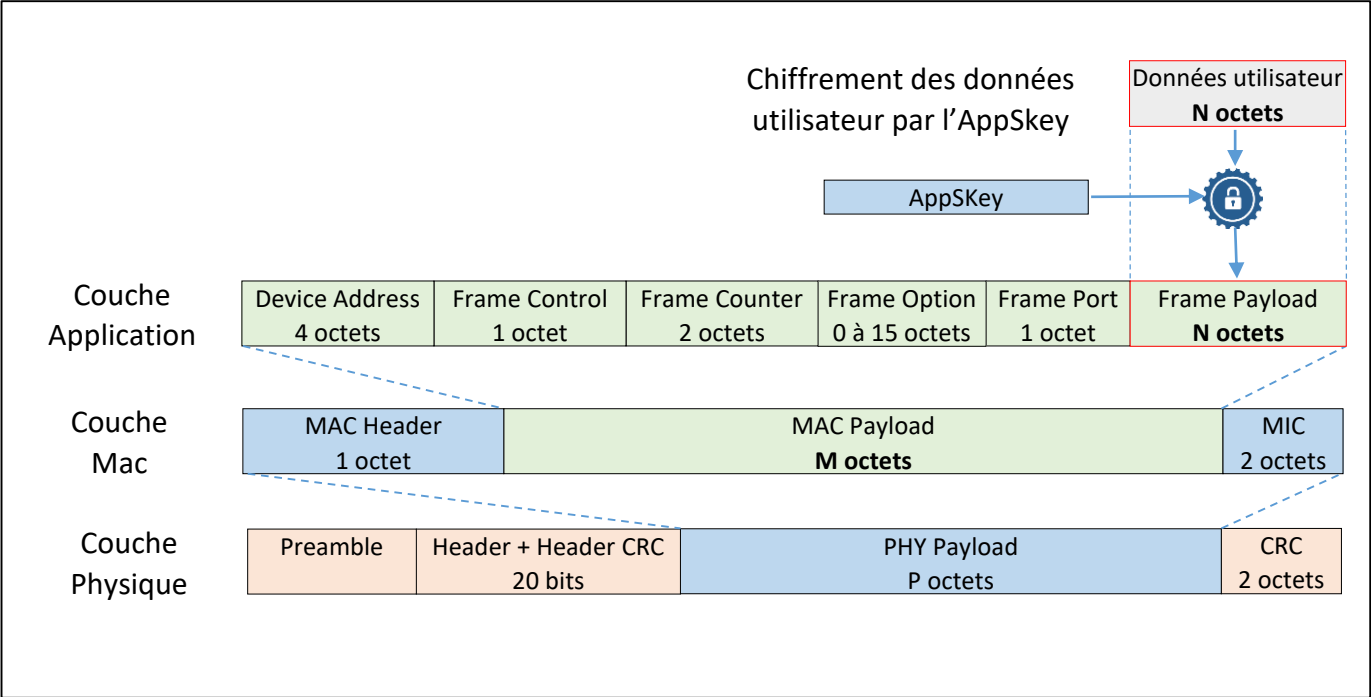


Figure 32 : Trame LORAWAN complète par couche protocolaire

#### 4.4.2 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.

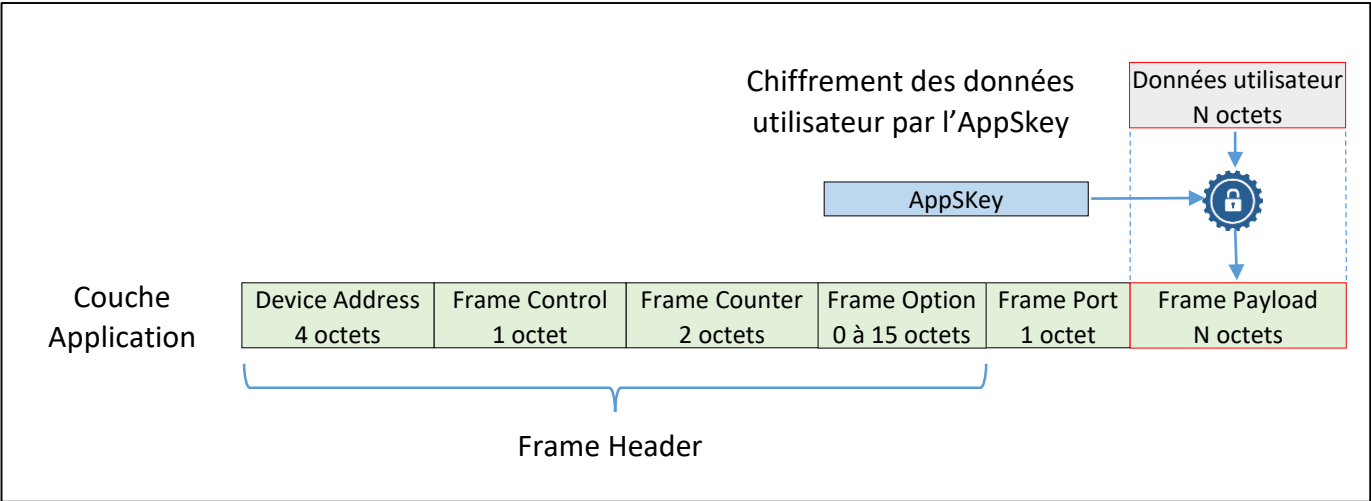


Figure 33 : Trame LORA couche Applicative

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

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

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

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

Tableau 5 : Taille maximum du Frame Payload en fonction du Data Rate

#### 4.4.3 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 4.1.6.

Le protocole LoRa MAC est composé de :

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

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

Tableau 6 : 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.

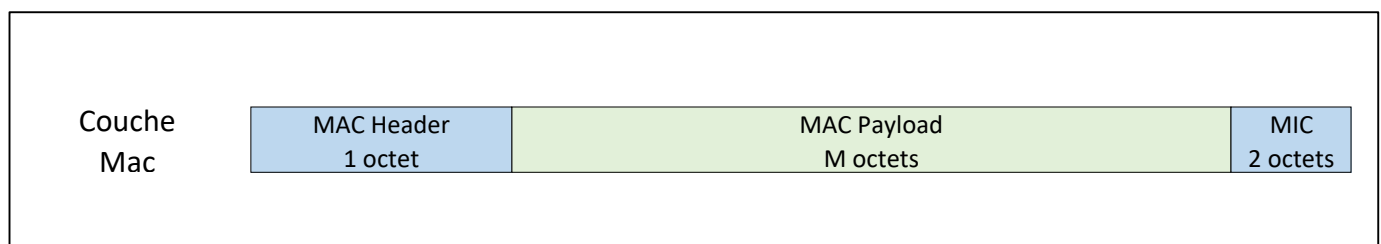


Figure 34 : Trame LORA couche LORA MAC

#### 4.4.4 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 :

Couche Physique	Preamble	Header + Header CRC ( Optionnel ) 20 bits	PHY Payload P octets	CRC 2 octets
-----------------	----------	--	-------------------------	-----------------

Figure 35 : Trame LORA couche Physique

Le Préambule est représenté par 8 symboles + 4.25. Le temps du Préambule est donc de  $12.25 \cdot T_{\text{symbole}}$ .

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

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

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

#### 4.4.5 Canaux et bandes de fréquences et Data Rate (DR)

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 ]. Le LoRaWAN définit les 8 canaux à utiliser pour communiquer avec une Gateway suivant le tableau suivant :

Canaux	Spreading Factor	Bandwidth
868.1 Mhz	De SF7 à SF12	125 kHz
868.3 Mhz	De SF7 à SF12	125 kHz
868.3 Mhz	SF7	250 kHz
868.5 Mhz	De SF7 à SF12	125 kHz
867.1 Mhz	De SF7 à SF12	125 kHz
867.3 Mhz	De SF7 à SF12	125 kHz
867.5 Mhz	De SF7 à SF12	125 kHz
867.7 Mhz	De SF7 à SF12	125 kHz
867.9 Mhz	De SF7 à SF12	125 kHz

Tableau 7 : Canaux, SF et Bandwidth de notre Gateway LoRaWAN



➡ Une Gateway LoRa écoute donc sur 8 canaux simultanément avec les 6 Spreading Factor différents, soit 48 combinaisons possibles.

Comme nous l'avons vu, le Spreading Factor a des conséquences sur le débit de la transmission. Ces débits sont notés DR (Data Rate) et sont normalisés de DR0 à DR6 :

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

Tableau 8 : Dénomination DR en fonction du SF et de la Bande Passante

#### 4.4.6 Trame reçue et transmise par la Gateway LoRaWAN

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.

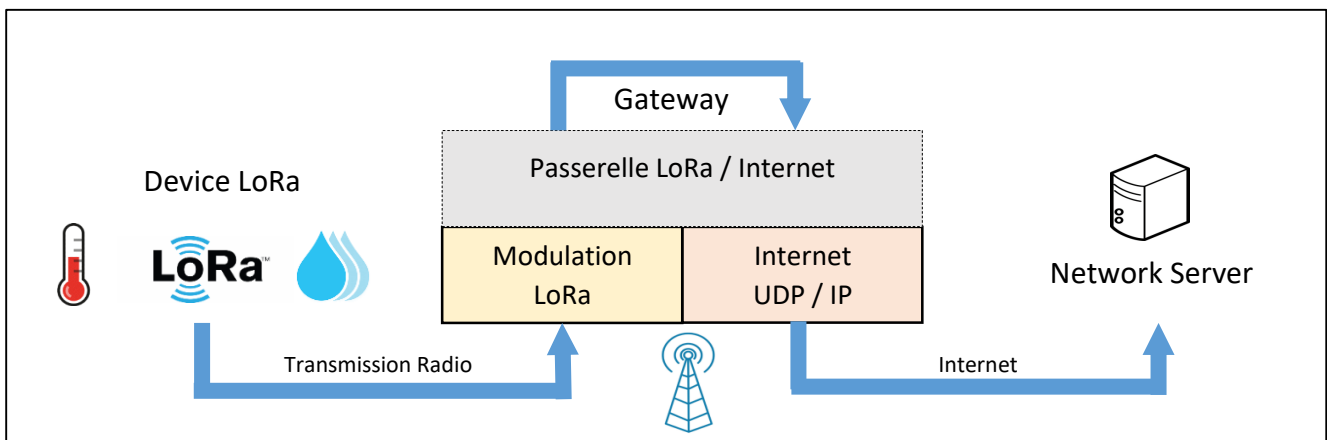


Figure 36 : Rôle de la Gateway LoRa

Coté interface Radio : La Gateway réceptionne une trame LoRaWAN et extrait le PHY Payload. Celui-ci est codé au format ASCII en base 64 (voir le paragraphe 5.4.1). 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 4.4.7). La Gateway a donc bien un rôle de passerelle entre le protocole LoRa d'un côté et un réseau IP de l'autre.

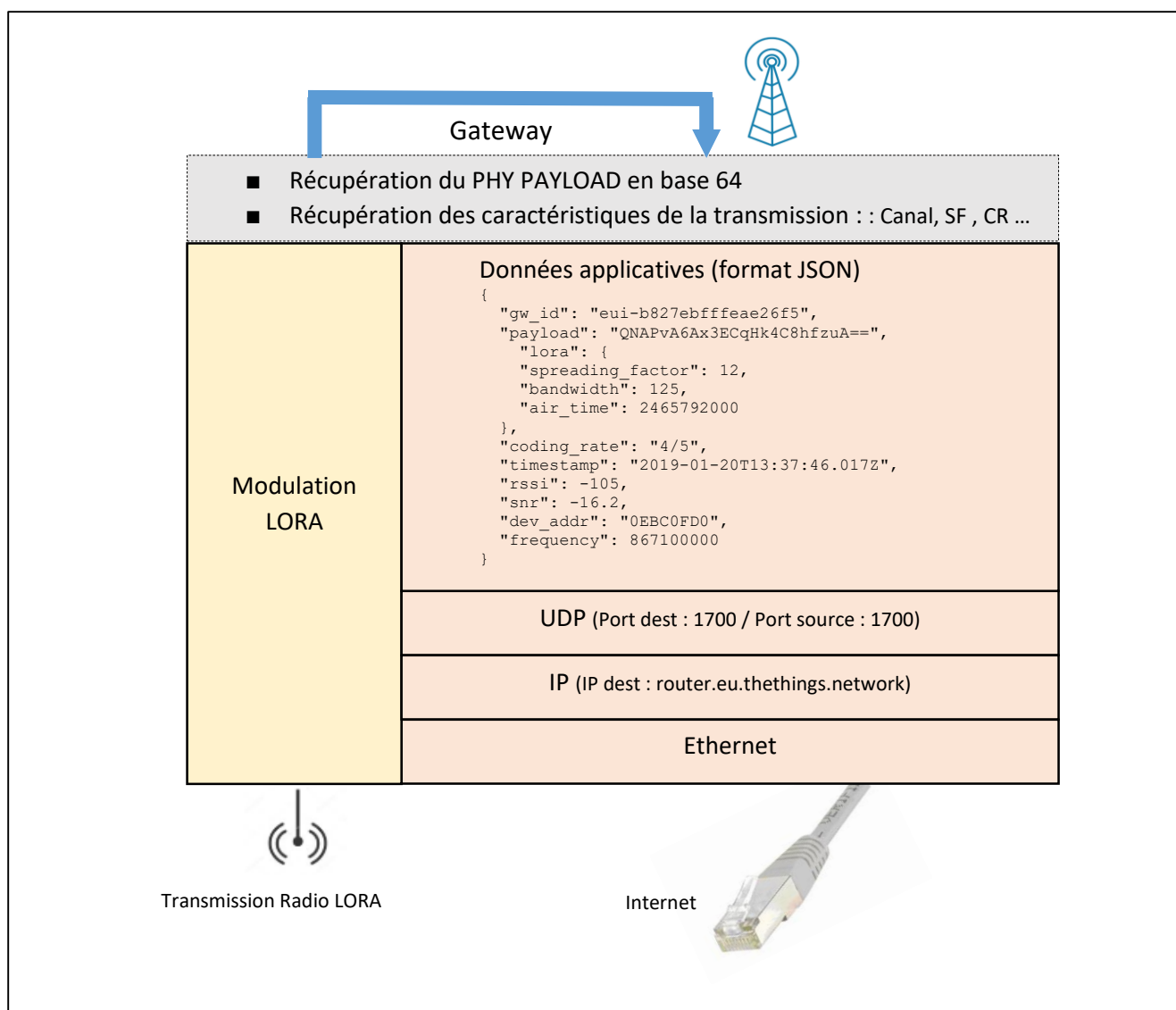


Figure 37 : Passerelle (Gateway) LORAWAN

#### 4.4.7 Le format 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 cet 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 :      "loras": { "spreading\_factor": 12, "air\_time": 2465792000 }
- Un Boolean    exemple :      "service" : true

## 5 Mise en œuvre d'un réseau LoRaWAN

---

### 5.1 Les différents types de réseaux

Les réseaux LoRaWAN peuvent être utilisés suivant deux méthodes :

- En utilisant les réseaux LoRaWAN opérés proposés par les opérateurs de télécoms
- En utilisant votre propre réseau LoRaWAN privé

#### 5.1.1 Les réseaux LoRaWAN opérés

Ce sont des réseaux LoRaWAN proposés par les opérateurs. Par exemple Objenious (filiale de Bouygues) ou encore Orange. Dans le cas de l'utilisation des réseaux opérés, l'utilisateur a juste besoin de s'occuper des Devices LoRa. Les Gateways, le Network Server et l'Application Server sont gérés par l'opérateur.

#### 5.1.2 Les réseaux LoRaWAN privés

Chacun est libre de réaliser son propre réseau privé en implémentant sa propre Gateway pour communiquer avec ses Devices LoRa. L'implémentation du Network Server et de l'Application Server peuvent alors être envisagées de différentes façons.

Network Server et Application serveur privées : C'est le développeur qui les met en place. Dans certaines Gateways, une implémentation de ces deux serveurs est déjà proposée. Sinon il est possible de la mettre en place. Il existe des serveurs (Network et Application) open source, c'est le cas par exemple de LoRa Serveur [ [www.loraserver.io](http://www.loraserver.io) ]



Une autre alternative est : <https://github.com/gotthardp/lorawan-server>

LoRaServer et lorawan-server sont deux applications Open Source.

Network Server et Application Server en ligne : Un certain nombre de Network Server et d'Application Server sont proposés. Ce sont des services payants ou avec des contreparties :

- Lorient [ [www.loriot.io](http://www.loriot.io) ]
- ResIoT [ [www.resiot.io](http://www.resiot.io) ]
- The Things Network [ [www.thethingsnetwork.org](http://www.thethingsnetwork.org) ] : C'est la solution que nous utiliserons.



#### 5.1.3 Les zones de couvertures

Les zones de couvertures des réseaux opérés sont mises à jour par les opérateurs, celle d'objenious est par exemple disponible ici : <https://objenious.com/reseau/>

Pour connaître la zone de couverture de notre Gateway utilisée avec TTN, nous pouvons utiliser l'application TTN Mapper : <https://ttnmapper.org/>. L'idée est de promener son Device LoRa associé à un GPS dans la zone de couverture des Gateway qui nous entourent. À 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 The Things Network (TTN)

### 5.2.1 Présentation de TTN

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

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

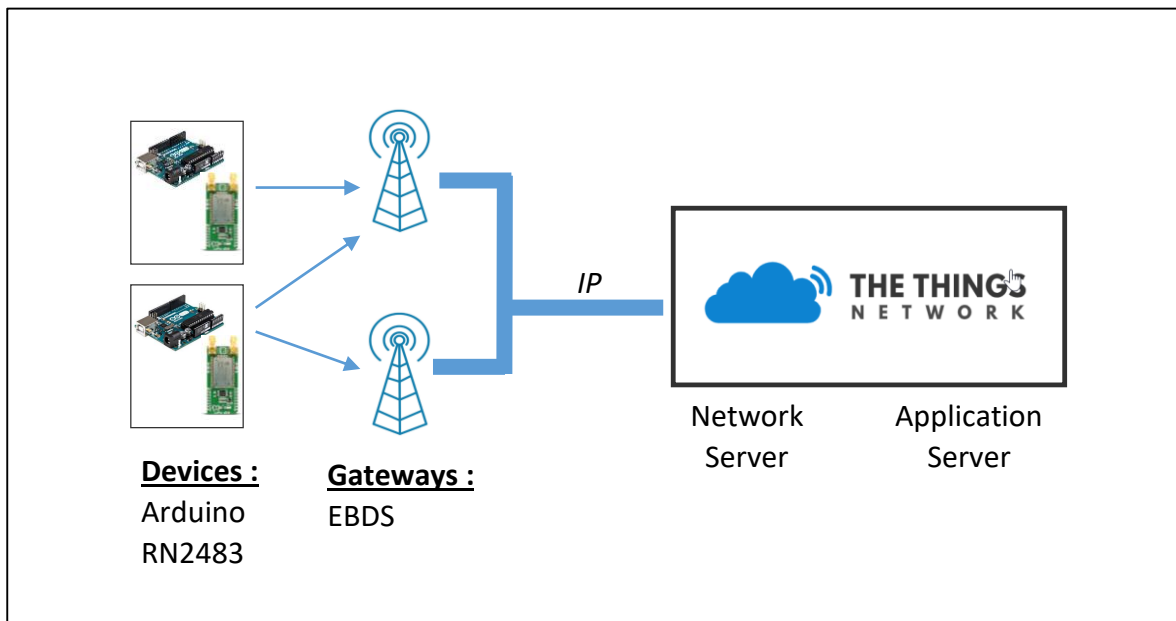


Figure 38 : Schéma de notre application

### 5.2.2 Configuration de la Gateway

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

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

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

### 5.2.3 Enregistrement des Gateways, Application et Devices

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

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

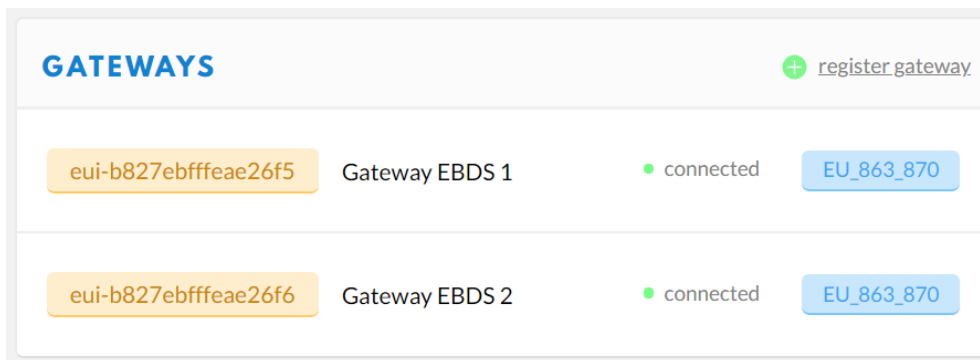


Figure 39 : Enregistrement d'une Gateway LoRa sur TTN

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

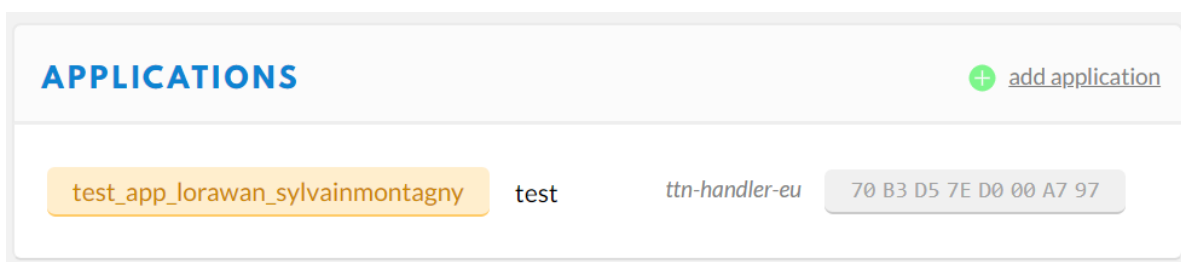


Figure 40 : Enregistrement d'une application sur TTN

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

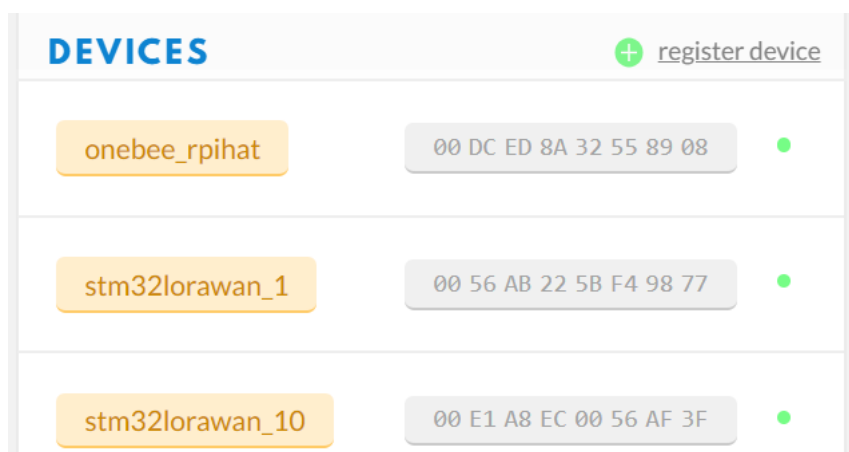


Figure 41 : Enregistrement des Devices LoRa dans une application

## 5.2.4 Configuration des Devices LoRa

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



The screenshot shows the 'Activation Method' section with 'ABP' selected. Below it, the 'Device Address' is set to '26 01 00 8E' (4 bytes). The 'Network Session Key' is 'FB EC 51 00 00 27 00 00 00 00 00 00 00 00 00 00' (16 bytes). The 'App Session Key' is '63 A9 00 00 00 00 00 00 00 00 00 00 00 00 00' (16 bytes).

Figure 42 : Configuration des Devices LoRa en ABP dans TTN

### 5.3 Mise en application

Nous avons dans notre application 10 Devices LoRa d'enregistrés : arduino0 à arduino9. Ils sont tous configurés en ABP. Les configurations des 10 Devices (**DevAddr**, **NetwksKey** et **AppSKey**) sont notés en commentaire dans le code de vous allez récupérer.

- ➔ Récupérer le code Arduino « RN2483-Arduino-LORAWAN.ino » dans Moodle.
- ➔ Modifier votre configuration (**DevAddr**, **NetwksKey** et **AppSKey**) en fonction de votre numéro de binôme.
- ➔ Vérifier que votre Device LoRa arrive bien à transmettre des informations (Uplink) vers la Gateway, puis vers le Network Serveur de TTN (The Things Network).

### 5.4 Analyse des trames échangées

#### 5.4.1 Utilisation de la base 64

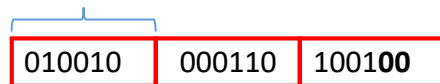
Notre Gateway nous présente le Payload PHY en base64. L'explication de la méthode de représentation en base 64 est fourni au travers d'un exemple : Le code hexadécimal 0x4869 représente nos données binaires que nous souhaitons transmettre en base 64.

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.

Bloc de 6 bits

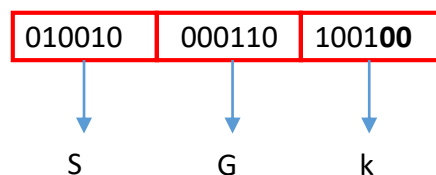


On ajoute 00 pour avoir 6 bits dans le groupe

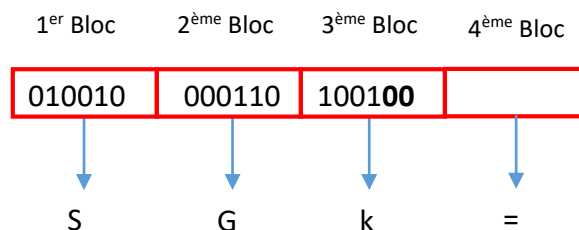
3. S'il manque des blocs pour faire un minimum de 4 blocs, des caractères spéciaux seront ajoutés.
4. Chaque groupe de 6 bits est traduit par le tableau suivant. (Source Wikipédia)

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

Figure 43 : Codage en base 64



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



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

### 5.4.2 Intérêt et inconvénient de la base 64

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

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



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

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

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

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

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

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

time	frequency	mod.	CR	data rate	airtime (ms)	cnt	
▲ 15:00:42	867.3	lora	4/5	SF 7 BW 125	46.3	7	dev addr: 26 01 1AD3   payload size: 14 bytes

Figure 44 : Trame récupérée sur le « Network Server » de TTN

Nous retrouvons bien les valeurs fournies par la Gateway :

- timestamp ( à 1 heure près en fonction du fuseau horaire),

- frequency : 867,3 Mhz
- modulation : Lora
- Coding Rate : 4/5
- data Rate : SF 7 / 125 kHz (DR5)
- air time : 46,3 ms

D'autres informations proviennent de l'analyse du Payload. Le Payload inscrit ici est le **PHY Payload**. Il y a donc une partie chiffrée (**Frame Payload**), mais les entêtes sont en claires (voir paragraphe 4.4.1). 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 45 : PHY Payload présenté dans notre Network Server

Sa taille est bien de 14 octets (en hexadécimal) comme précisé sur la Figure 44

Nous reprenons le format de la trame LoRaWAN vu à la Figure 32. Nous pouvons alors retrouver tous les champs de toute la trame :

PHYPayload = <b>40D31A01260007000FD686EE5074</b>	
PHYPayload = MAC Header[1 octet]   MACPayload[...]   MIC[4 octets]	
MAC Header	= <b>40</b> (Unconfirmed data up)
MACPayload	= <b>D31A01260007000FD6</b>
Message Integrity Code	= <b>86EE5074</b>
MACPayload = Frame Header   Frame Port   FramePayload )	
Frame Header	= <b>D31A0126000700</b>
FPort	= <b>0F</b>
FramePayload	= <b>D6</b>
Frame Header = DevAddr[4]   FCtrl[1]   FCnt[2]   FOpts[0..15]	
DevAddr	= <b>26011AD3</b> (Big Endian)
FCtrl (Frame Control)	= <b>00</b> (No ACK, No ADR)
FCnt (Frame Counter)	= <b>0007</b> (Big Endian)
FOpts (Frame Option)	=

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

# LoRaWAN 1.0.x packet decoder

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

Base64 or hex-encoded packet

Secret NwkSKey (hex-encoded; optional)

Secret AppSKey (hex-encoded; optional)

Figure 46 : LoRaWAN packet decoder

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

Nous reprenons l'exemple de la trame ci-dessus (Figure 45). Pour information, les clés NwkSKey et AppSKey suivantes ont été utilisée :

- NwkSKey : E3D90AFBC36AD479552EFEA2CDA937B9
- AppSKey : F0BC25E9E554B9646F208E1A8E3C7B24

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

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

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

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

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

Figure 47 : 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"
  }
]

```

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

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

Figure 48 : Simulation d'une trame envoyée par un Device LoRa

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

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

Figure 49 : Transmission de données de l' « Application Server » jusqu'au Device LoRa

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

Une demande de confirmation peut être demandée au Device LoRa pour qu'il précise s'il a bien reçu les données. L'option « confirmed » permet donc de spécifier le type de trame dans le MAC Header (voir paragraphe 4.4.3) : « Confirmed Data Down » ou « Unconfirmed Data Down » dans notre cas.

## 6 La récupération des données sur notre propre Application

Nous utiliserons à nouveau le Network Server et Application Server de The Things Network. Jusqu'ici nous nous sommes contentés de vérifier que les données de l'utilisateur (Frame Payload) soient bien arrivées à destination dans l'Application Server. Il faut maintenant les récupérer avec notre propre Application qui aura pour rôle :

- De stocker et traiter les données
- De les mettre à disposition de l'utilisateur (Serveur Web par exemple)
- D'envoyer des commandes de l'utilisateur aux Devices LoRa si besoin (Flux Downlink)

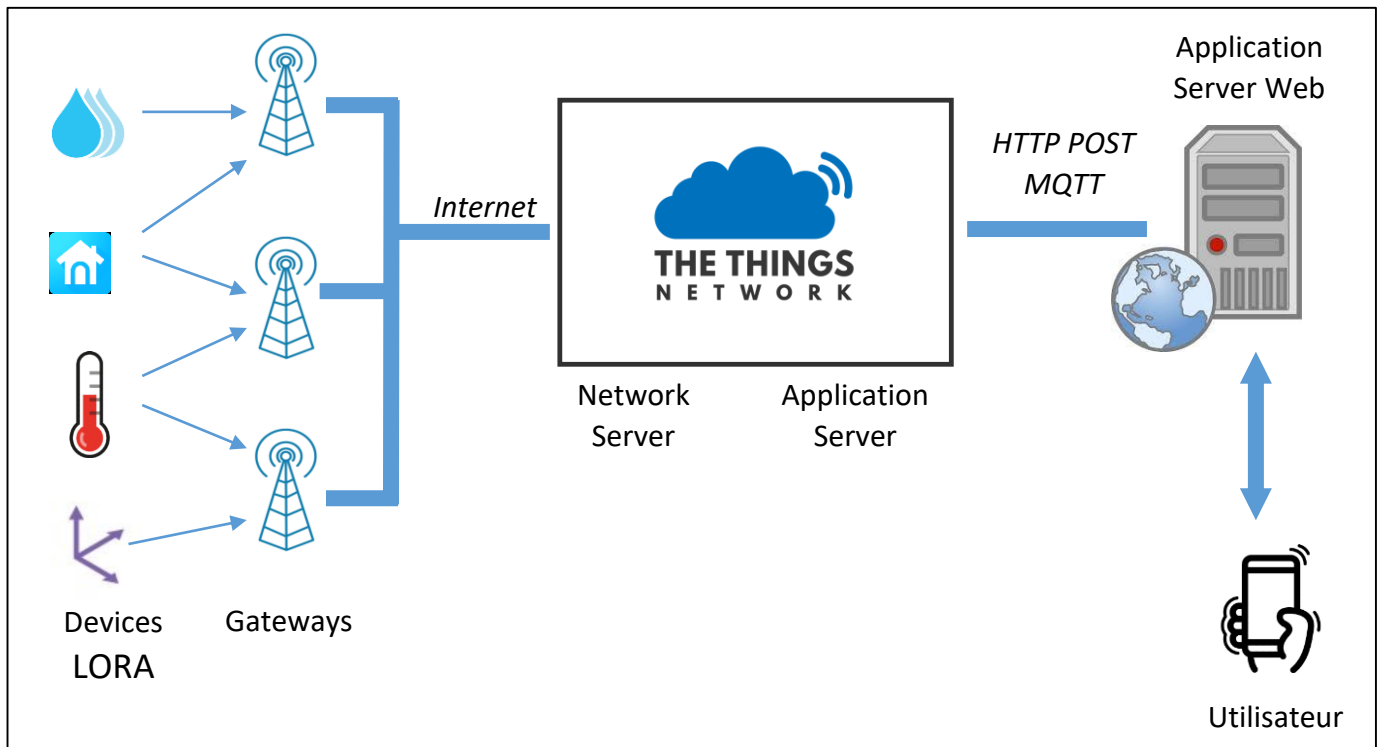


Figure 50 : Structure globale d'un réseau LORAWAN

Nous allons voir deux méthodes pour communiquer entre notre application et TTN :

- Le protocole HTTP POST
- Le protocole MQTT

### 6.1 Récupération des données en HTTP POST dans l'Application

#### 6.1.1 Présentation du protocole HTTP

Dans le monde de l'internet, le protocole HTTP est très utilisé pour le dialogue entre un client et un serveur web. Classiquement, le client (un navigateur web par exemple) fait une requête HTTP GET et le serveur lui retourne le contenu de la page web demandée.

Dans notre cas, la problématique est différente, car le client doit modifier le contenu du serveur avec les données envoyées par le Device LORA. La requête s'appelle HTTP POST. Son nom indique que le client va poster (et non pas récupérer) des données.

L'utilisateur qui souhaite réceptionner des données sur son serveur doit donc concevoir un serveur HTTP capable de traiter des requêtes HTTP POST [ requête notée **(1)** sur la Figure 51 ] fournies par TTN. Dans cette configuration, TTN jouera le rôle de client, et notre application, le rôle de serveur.

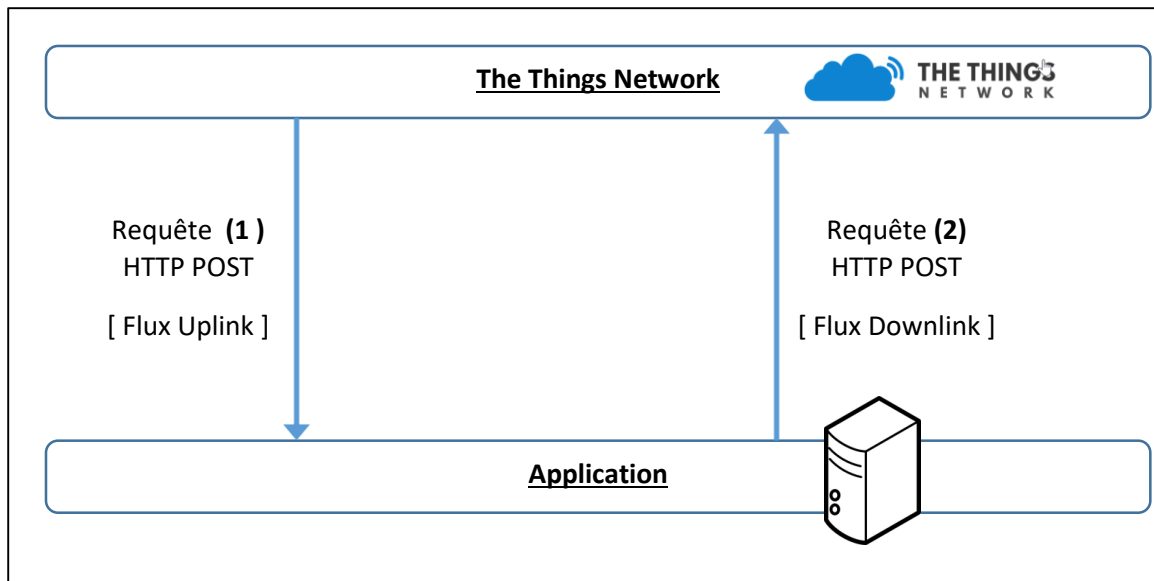


Figure 51 : Communication bidirectionnelle entre TTN et notre Application

A l'inverse l'utilisateur qui souhaite transmettre des données à destination du Device LoRa (Downlink) doit être capable de fournir une requête HTTP POST [ requête notée **(2)** sur la Figure 51 ] vers TTN. Dans cette configuration, TTN jouera le rôle de Serveur, et notre application, le rôle de client.

### 6.1.2 Fonctionnement d'un client et d'un serveur HTTP POST

Afin de bien comprendre le fonctionnement des requêtes HTTP POST, nous allons procéder par étape. Le premier test que nous effectuerons sera decorrélié de TTN. Le principe est le suivant :

- Utilisation d'un serveur HTTP disponible sur le web, gérant les requêtes HTTP POST [ <https://rbaskets.in/> ] ou [ <https://beeceptor.com/> ] par exemple.
- Utilisation d'un client HTTP émettant des requêtes HTTP POST : commande **curl** sous linux ou logiciel **Postman** [ [www.getpostman.com](http://www.getpostman.com) ] sous windows par exemple.

Ce test nous permet de valider le fonctionnement d'un serveur HTTP POST et d'un client HTTP POST. Une fois que nous serons sûrs de leur fonctionnement, nous les utiliserons avec TTN.



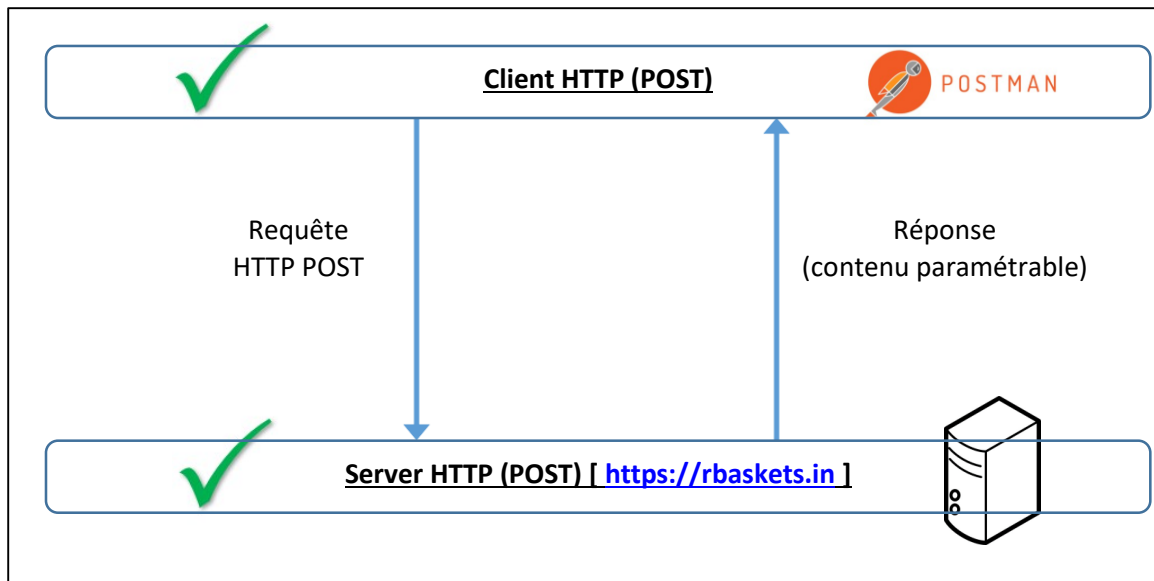


Figure 52 : Validation d'un serveur HTTP POST et d'un client HTTP POST.

### 6.1.3 Test du serveur HTTP POST

Nous utiliserons un serveur web proposé par <https://rbaskets.in/>.

- ➔ Aller sur le site <https://rbaskets.in/> et créer un nouveau « Basket » : Zone où vous pourrez visualiser les requêtes qui ont été émises vers votre serveur.

A partir de maintenant, toutes les requêtes dirigées vers l'adresse créée, seront traitées et s'afficheront. Dans le menu de configuration il est aussi possible de définir le contenu de la réponse qui sera fournie.

- ➔ Définir un format de réponse pour les requêtes HTTP POST avec un body personnalisé.

### 6.1.4 Test du client HTTP POST

On peut réaliser le rôle du client de deux façons, soit en ligne de commande (la commande cURL s'exécute depuis un Terminal sous linux), soit avec le logiciel POSTMAN sous Windows.

- ➔ Commande cURL : `curl -X POST -d "données à envoyer" @IPServeurHTTP`
- ➔ Logiciel Postman : **New > Request > Créer une requête HTTP POST > Send**

Vérifier que les requêtes HTTP POST sont bien récupérées sur votre serveur créé précédemment.

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

Maintenant que nous avons un serveur HTTP (POST) fonctionnel, nous allons configurer TTN pour jouer le rôle du client et pour qu'il envoie des requêtes HTTP POST vers notre Serveur dès qu'une donnée d'un Device LoRa est disponible. Nous gérons donc ici le flux Uplink.

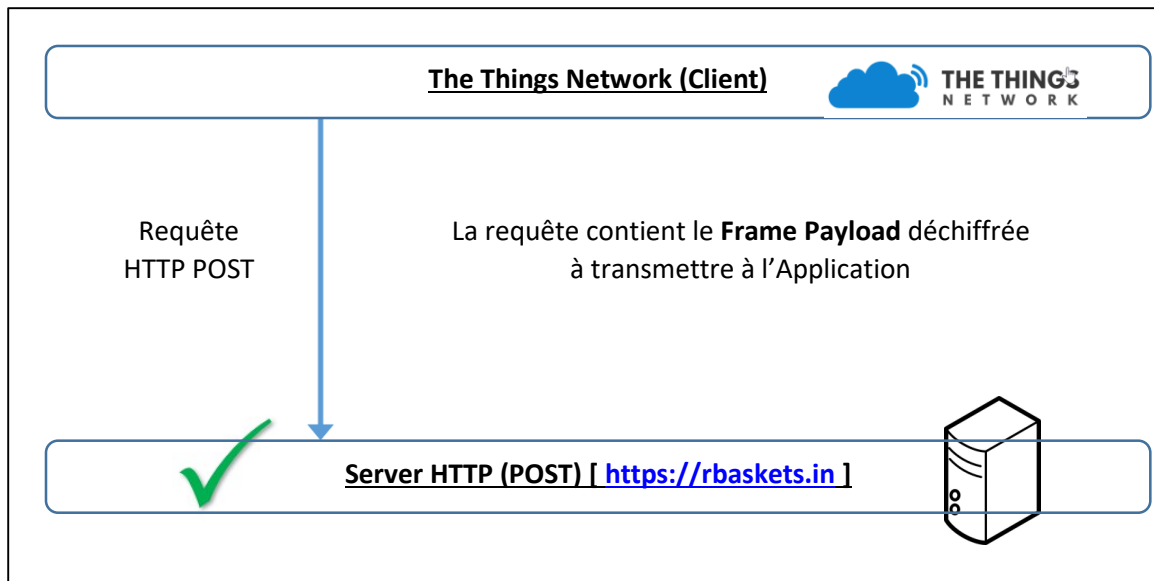


Figure 53 : Récupération des données sur notre Application

Dans TTN, nous intégrons un bloc appelé « HTTP Integration » permettant de jouer le rôle de client, c'est-à-dire d'envoyer des requêtes HTTP POST à destination de notre nouveau Serveur HTTP. L'intégration se fait via la console TTN : **Application > Nom\_de\_votre\_application > Integration**.

The screenshot shows the 'HTTP Integration' configuration in the TTN console. Under the 'URL' section, the endpoint is 'https://rbaskets.in/montagny'. Under the 'Method' section, the HTTP method is 'POST'. Both fields have a green checkmark icon to their right, indicating they are valid.

Figure 54 : Ajout d'un client HTTP POST dans TTN

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

- Envoyer une trame depuis un Device LoRa vers TTN
- Simuler l'envoi d'une trame d'un Device LoRa vers TTN (grâce à l'outil proposé par TTN vu paragraphe 5.4.5)

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 0). Voici quelques compléments d'information :

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

### 6.1.1 Envoyer des données depuis notre Application avec HTTP POST

Lorsque nous avons ajouté le bloc « HTTP Integration » au paragraphe précédent, nous avons aussi implicitement activé la fonctionnalité que TTN soit prêt à recevoir et à traiter les requêtes HTTP POST. La seule information qu'il nous manque est l'adresse du server HTTP vers lequel il faut émettre les requêtes. Si on observe la trame reçue précédemment en provenance de TTN, on s'aperçoit que dans la trame JSON reçue, une URL (`downlink_url`) est spécifiée :

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

C'est cette URL que nous devons utiliser. Nous allons nous placer dans la configuration suivante :

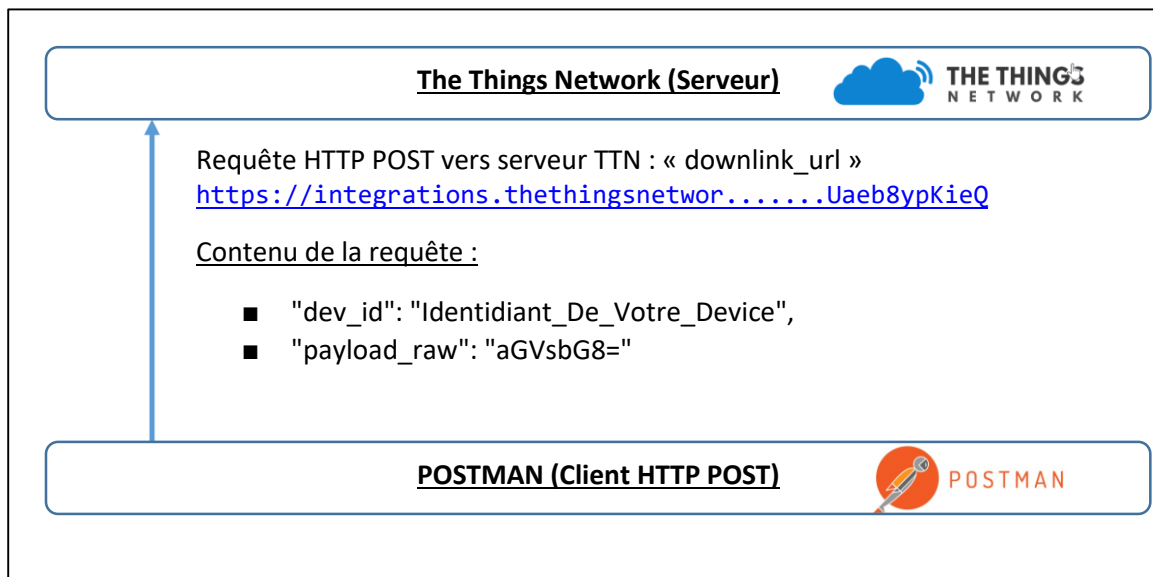


Figure 55 : Envoi de données à partir de notre Application

Comme précédemment, on peut réaliser le rôle du client de deux façons : soit en ligne de commande (la commande `cURL` s'exécute depuis un Terminal sous linux), soit avec le logiciel POSTMAN.

- ➔ **Commande cURL :** `curl -X POST -data '{ "dev_id" : " Identifiant_De_Votre_Device ", "payload_raw" : "AQE=" }' @DuServeurHTTP`
- ➔ **Logiciel Postman :** New > Request > HTTP POST , puis Body > Raw > JSON :

```
{
```

```

    "dev_id": "YourDeviceID",
    "payload_raw": "aGVsbG8="
}

```

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

## 6.2 Récupération des données avec MQTT dans l'Application

### 6.2.1 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).

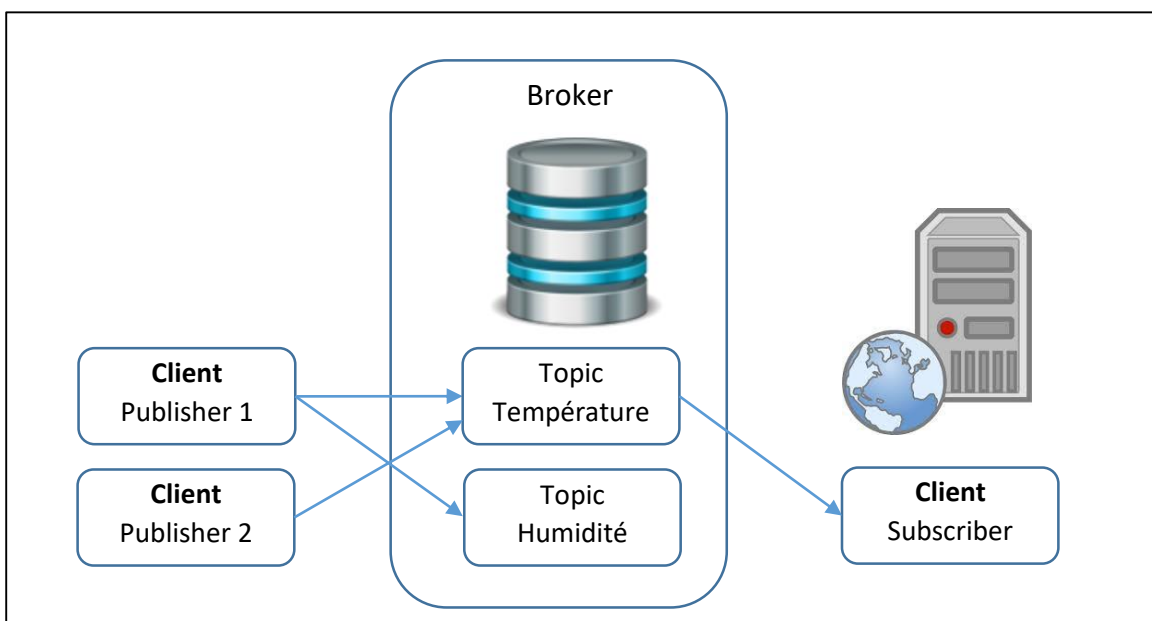


Figure 56 : Modèle Publisher / Subscriber du protocole MQTT

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 :

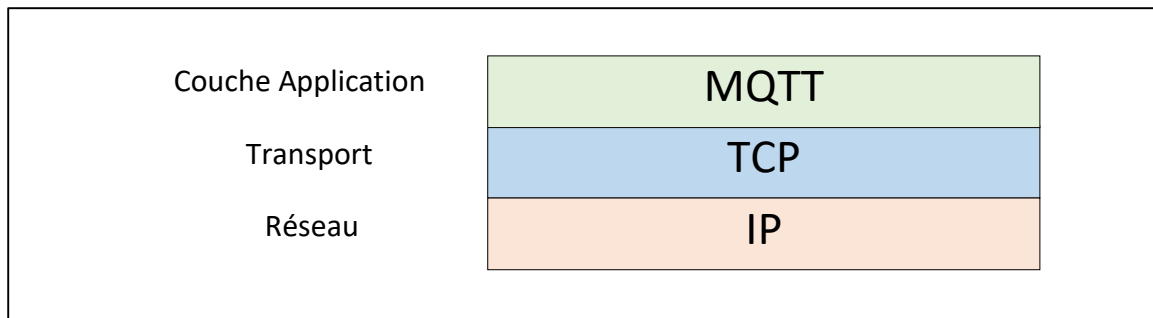


Figure 57 : Protocoles utilisés pour la communication avec MQTT

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 58 : Capture d'une trame MQTT avec Wireshark

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



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

## 6.2.2 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 6.2.4.

## 6.2.3 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. Une parle ici du cas d'une même connexion, c'est-à-dire entre le moment où le Client se connecte, et le moment où :

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

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

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

QoS 1 "At least once" (au moins une fois) : Le deuxième niveau de qualité est "avec acquittement". Le Publisher envoie un message au Broker et attend sa confirmation. De la même façon, le Broker envoie un message à ces Subscriber et attend leurs confirmations. 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 59 montre les trames émises pour chaque niveau de QoS.

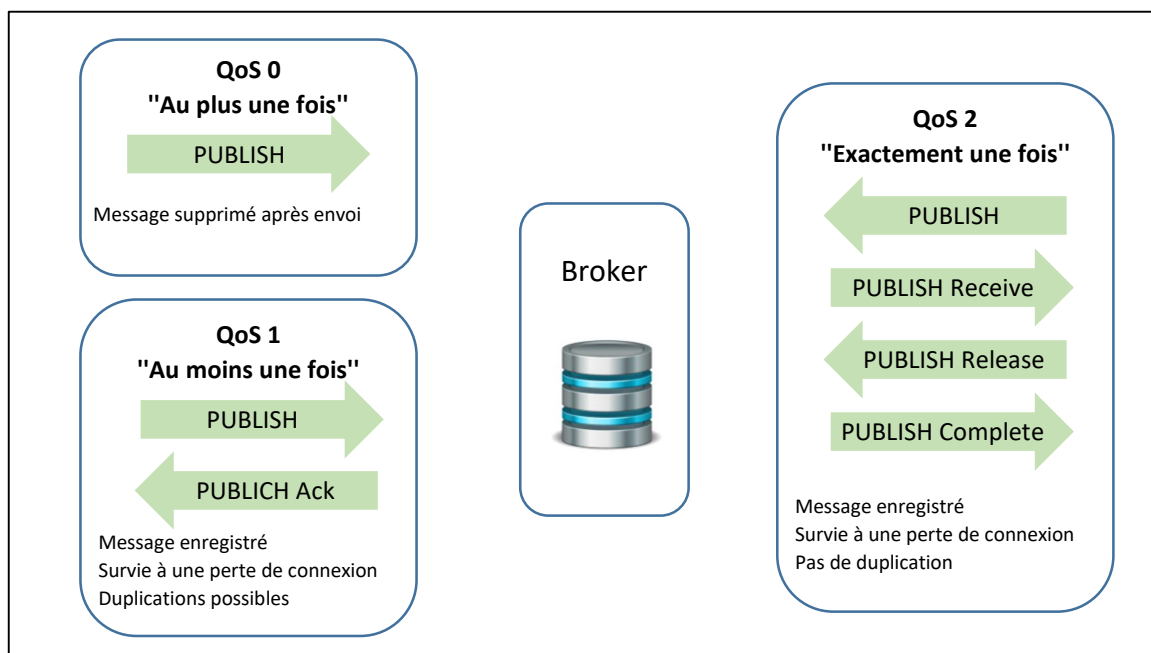


Figure 59 : Qualité de Service en MQTT

La Figure 60 représente les 3 captures de trames sur Wireshark représentant les 3 QoS (1, 2 et 3) que nous venons d'expliquer.

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

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

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

Figure 60 : Capture de trame avec QoS = 0, puis QoS = 1, puis QoS = 2

### 6.2.4 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 réussi à diffuser au Subscriber.

Le Tableau 9 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 9 : Qualité de Service en fonction de la valeur du QoS et du flag `cleanSession`

### 6.2.5 Les Topics du protocole MQTT

Les chaînes décrivant un sujet forment une arborescence en utilisant la barre oblique "/" comme caractère de séparation. La Figure 61 et le Tableau 10 donne un exemple d'organisation de Topic.

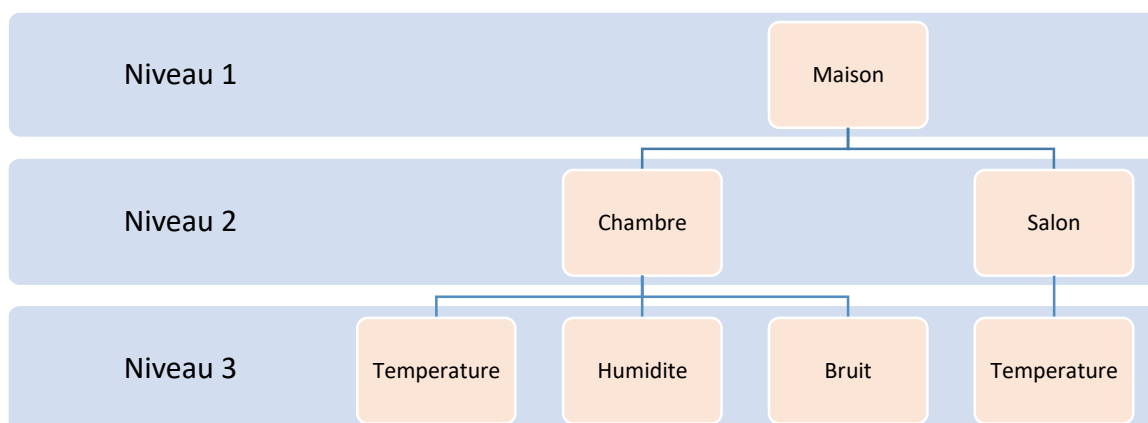


Figure 61 : Exemple de hiérarchie de Topic MQTT

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

Tableau 10 : 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 quelle chaîne de caractères sur le niveau où il est placé.
- Le dièse "#" remplace n'importe quelle chaîne de caractères sur tous les niveaux suivants. Il est obligatoirement placé à la fin.

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

Tableau 11 : Exemple de Topic

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

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

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

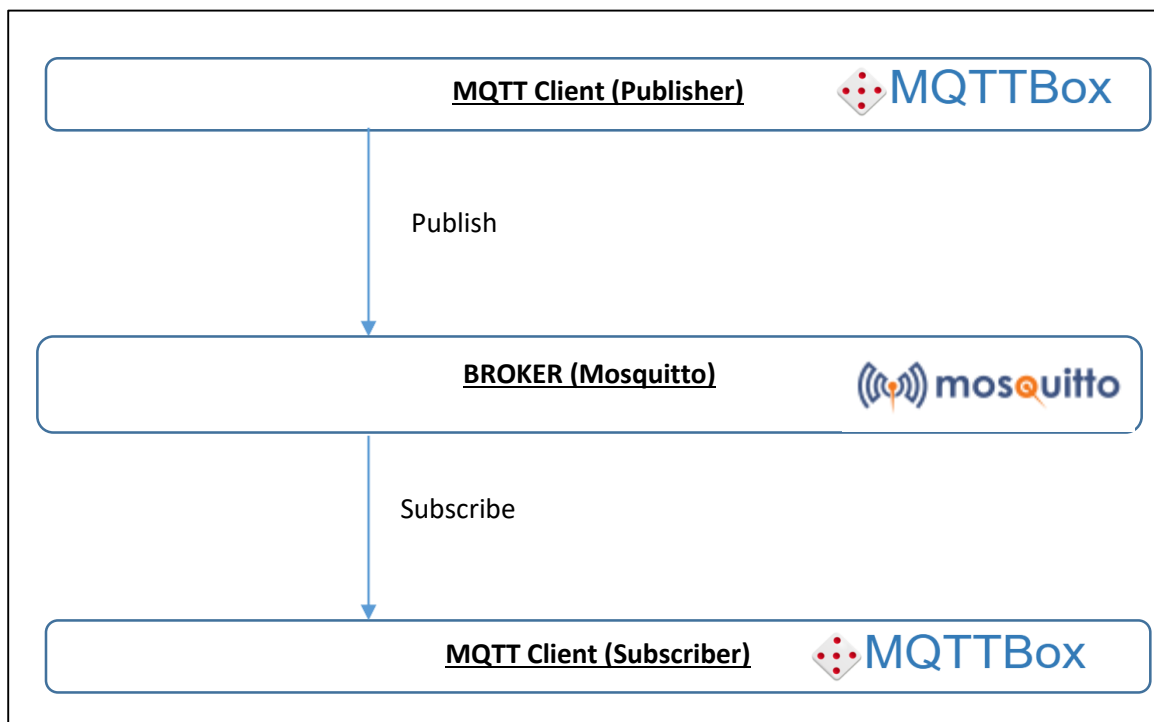


Figure 62 : Test du protocole MQTT



Le Broker MQTT est commun à tout le monde. Nous pouvons soit le réaliser nous-même, soit utiliser un Broker MQTT public de test. Nous utiliserons le Broker de test de Mosquitto disponible sur <https://test.mosquitto.org/>. D'autres sont éventuellement disponibles en cas de problème, par exemple : <http://www.mqtt-dashboard.com/>

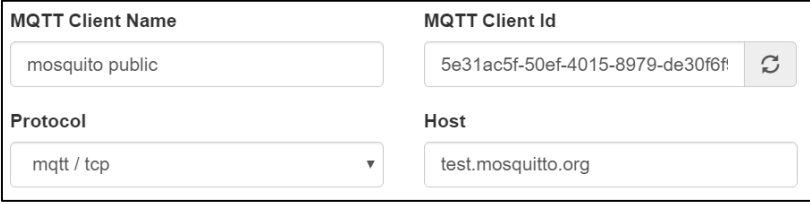
Si nous décidons de le réaliser nous-même à l'aide d'une Raspberry Pi (par exemple). L'installation se ferait en deux lignes de commande :

- `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.

### 6.2.7 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




<b>MQTT Client Name</b> <input type="text" value="mosquito public"/>	<b>MQTT Client Id</b> <input type="text" value="5e31ac5f-50ef-4015-8979-de30f6f"/> 
<b>Protocol</b> <input type="text" value="mqtt / tcp"/>	<b>Host</b> <input type="text" value="test.mosquitto.org"/>

Figure 63 : Configuration d'un Client MQTT dans MQTT Box

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

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

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

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

Nous pouvons donc représenter l'application globale par le schéma suivant :

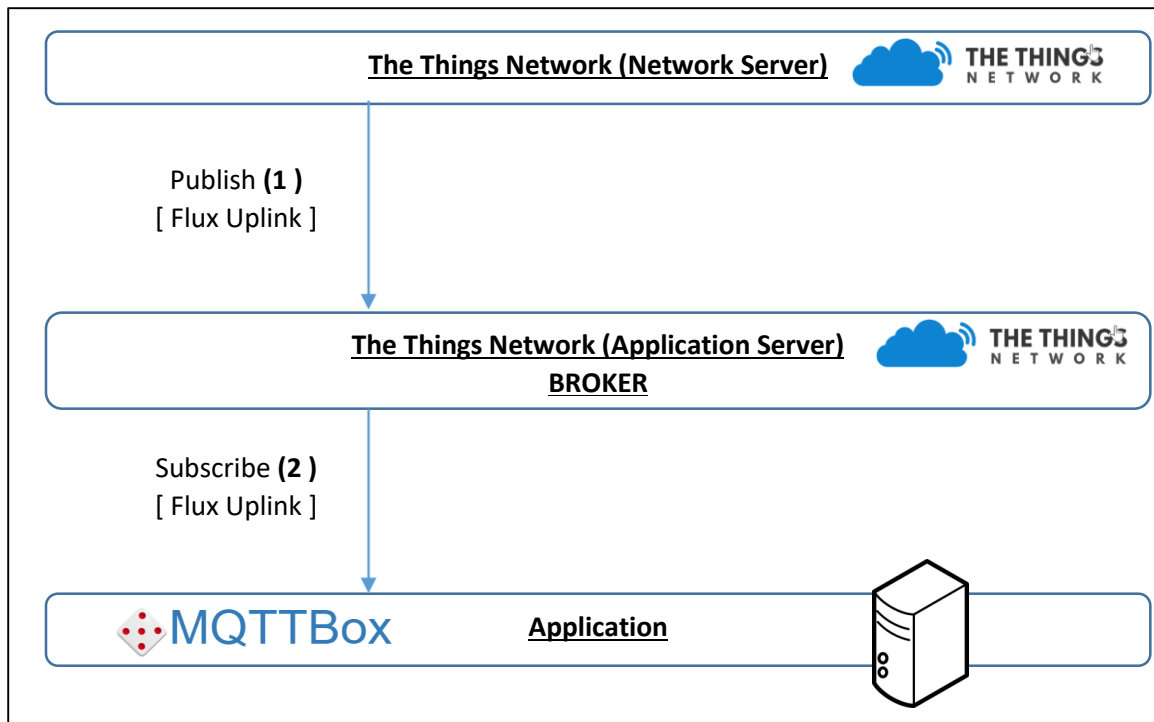


Figure 64 : Publisher et Subscriber dans en MQTT avec TTN

- La publication est représentée par la trame MQTT (1)
- La souscription est représentée par la trame MQTT (2)

Le client Publisher étant réalisé par TTN, il est déjà configuré. Il nous reste à configurer le client Subscriber. 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 soumise à une authentification Username / Password. Dans notre cas le Username correspond au nom de notre application, c'est-à-dire : **seminaire\_lorawan**. Si vous avez choisi une autre application, il faut bien mettre la votre.
- Password : Le Password est nommé « Access Key » par TTN. Elle vous sera donnée par l'enseignant dans le cadre de ce test. Si vous avez enregistré votre propre application, vous trouverez l'Access Key dans : **TTN > Application > Nom\_de\_votre\_application > Overview**.

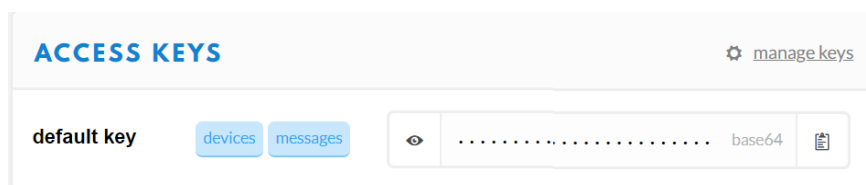


Figure 65 : Access Keys de l'application dans TTN


<b>MQTT Client Name</b> <input type="text" value="MQTT The Things Network"/>	<b>MQTT Client Id</b> <input type="text" value="c029dd54-55e3-4fef-b55e-798fbat"/> 
<b>Protocol</b> <input type="text" value="mqtt / tcp"/> ▼	<b>Host</b> <input type="text" value="eu.thethings.network:1883"/>
<b>Username</b> <input type="text" value="seminaire_lorawan"/>	<b>Password</b> <input type="password" value="....."/>

Figure 66 : 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 12 : 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.

Topic to subscribe
✕

**QoS**  
 ▼

Figure 67 : Configuration du Subscriber

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

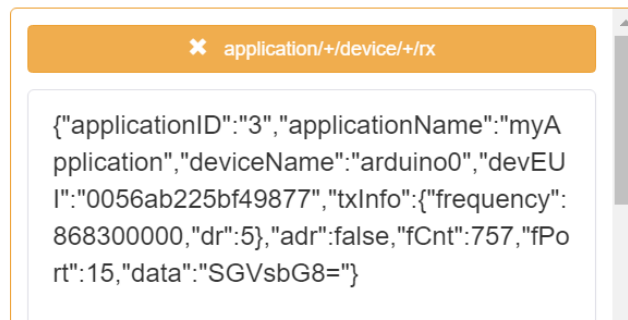


Figure 68 : Trame LoRaWAN reçue sur le Subscriber MQTT

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

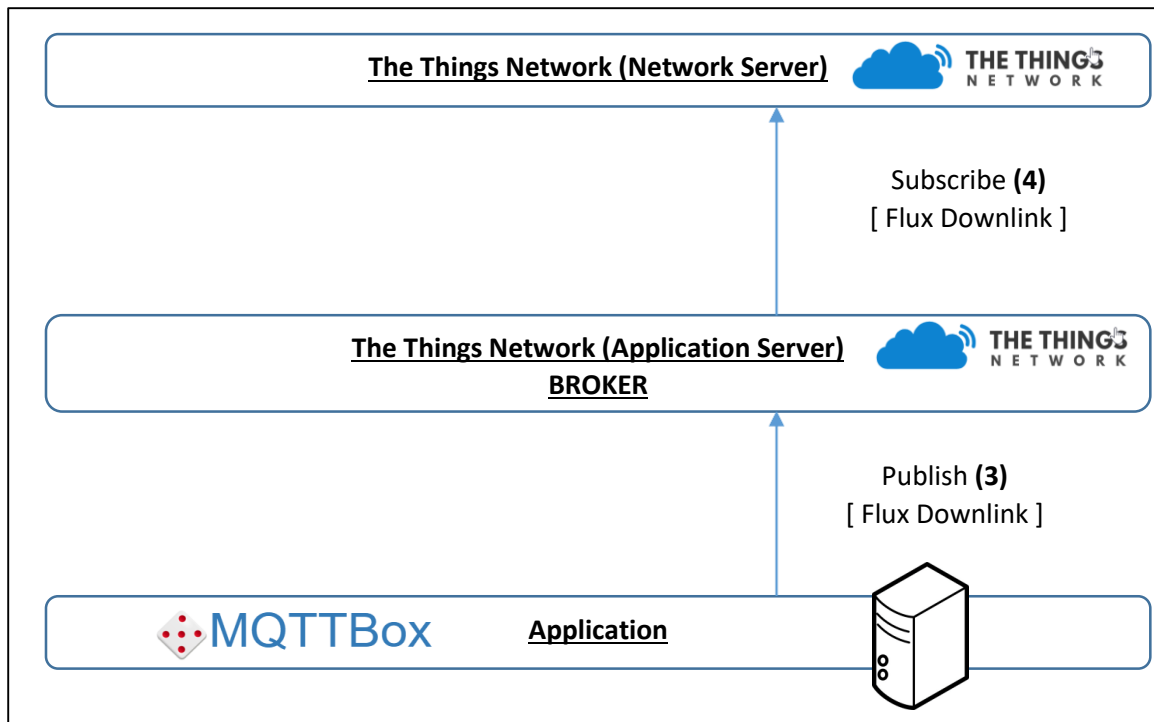
```
{
  "applicationID": "3",
  "applicationName": "myApplication",
  "deviceName": "arduino0",
  "devEUI": "0056xxxxxxxxxx877",
  "txInfo": {
    "frequency": 868500000,
    "dr": 5
  },
  "adr": false,
  "fCnt": 792,
  "fPort": 15,
  "data": "SGVsbG8="
}
```

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

## 6.2.9 Envoyer des données depuis notre Application avec MQTT

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

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



- La publication est représentée par la trame MQTT (3)
- La souscription est représentée par la trame MQTT (4)

Le client Subscriber étant réalisé par TTN, il est déjà configuré. Il nous reste à configurer le client Publisher. Nous utiliserons à nouveau MQTT Box. Le rôle de client dans MQTT Box a été fait au paragraphe 6.2.8. Il nous reste simplement à ajouter le rôle de Publisher. La configuration est la suivante :

- Topic du Subscriber : ***seminaire\_lorawan/devices/arduino0/down*** où *seminaire\_lorawan* est à remplacer par le nom de votre application, et *arduino0* par le nom du Device LoRa vers lequel vous souhaitez envoyer une donnée. (Voir chapitre 6.2.8 pour comprendre les Topics)
- Le Payload doit être au format JSON. L'exemple suivant envoie «hello » :

```
{
  "payload_raw": "aGVsbG8="
}
```

Topic to publish

seminaire\_lorawan/devices/arduino0/down

QoS

0 - Almost Once

Retain ☐

Payload Type

Strings / JSON / XML / Characters

e.g: {'hello':'world'}

Payload

```
{
  "payload_raw": "aGVsbG8="
}
```

Publish

Figure 69 : Configuration du Publisher

Vous devriez voir les données arriver sur votre Device LoRA.

## 7 Création de notre propre Network et Application Server

### 7.1.1 Objectifs

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 destination précise. Il faudra donc les reconfigurer pour qu'elles pointent vers votre nouvelle architecture.

### 7.1.2 Présentation de LoRaServer

LoRaServer [ [www.loraserver.io](http://www.loraserver.io) ] est un projet Open-Source répondant exactement au besoin que nous avons. Nous l'utiliserons sur une cible Raspberry PI. En réalité, il peut souvent être mis en place dans le même système que la Gateway, surtout si celle-ci est elle-même à base de RPI. Les Gateway EBDS que nous possédons possèdent d'ailleurs un Network Server et un Application Server que nous aurions pu utiliser. Néanmoins, pour des raisons pédagogiques, il est intéressant de séparer les systèmes pour bien différencier les problématiques.

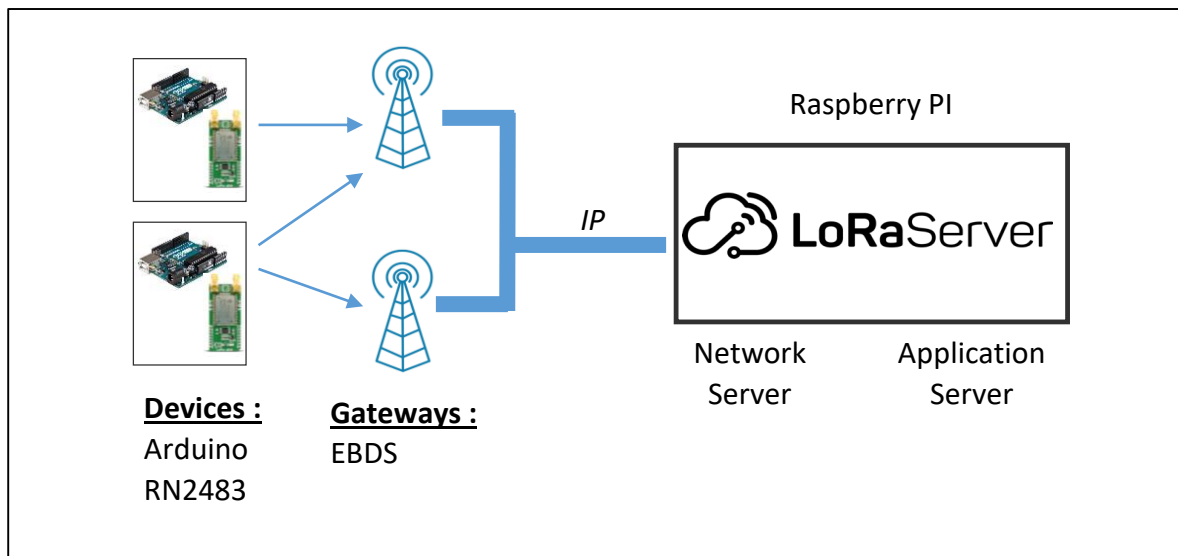


Figure 70 : Architecture globale de notre réseau loRaWAN



➔ **Attention, il y a une ambiguïté dans la dénomination des entités logiciels de LoRaServer. LoRaServer (sans espace entre le mot LoRa et Server) est le nom du projet global. En revanche, LoRa Server (avec un espace) est le nom du service qui joue le rôle du Network Server.**

L'architecture et le fonctionnement de LoRaServer est présentée dans sa documentation par le schéma suivant :

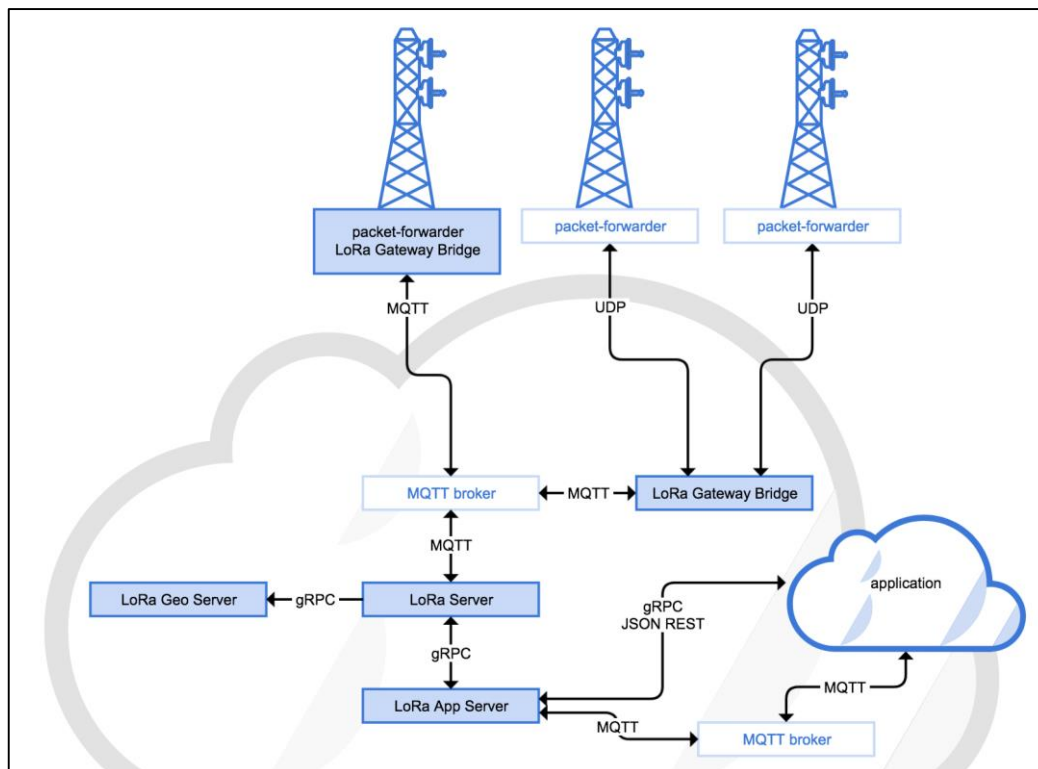


Figure 71 : Architecture détaillée de LoRaServer

On s'attend à voir uniquement un Network Server (LoRa Server) et un Application Server (LoRa App Server) mais ce schéma est plus complexe. Il demande un certain nombre d'explications pour bien comprendre ce que nous allons installer et la façon dont nous l'utiliserons par la suite.

### 7.1.3 Le « Packet Forwarder » (Gateway)

Comme nous l'avons vu au chapitre 4.1.2, les Gateways LoRa sont des passerelles entre la modulation LoRa et un réseau IP. Pour réaliser cette passerelle, un service nommé « UDP Packet Forwarder » a été développé par Semtech : [https://github.com/Lora-net/packet\\_forwarder](https://github.com/Lora-net/packet_forwarder). Dans ce dossier github, un fichier nommé PROTOCOL.TXT explique parfaitement ce protocole applicatif qui travaille au-dessus de UDP.

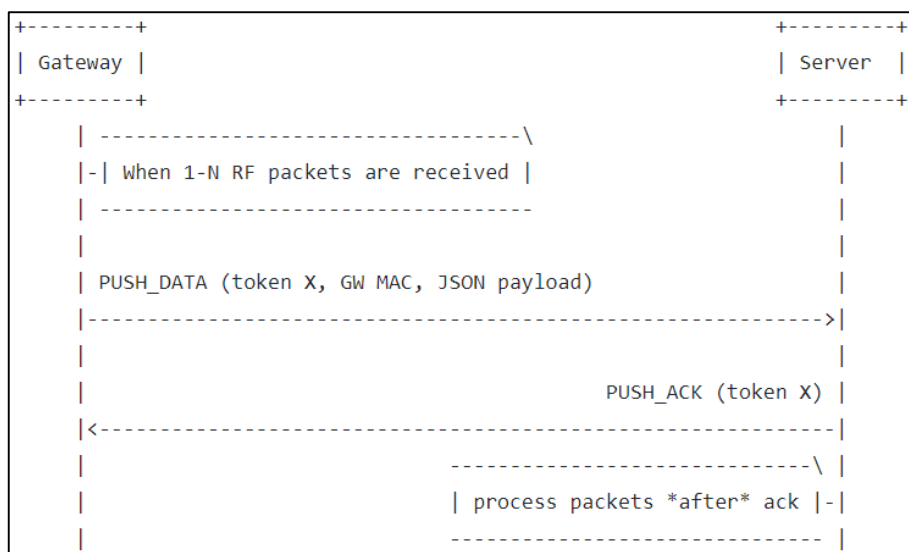


Figure 72 : Protocole Uplink (PUSH\_DATA du Packet Forwarder)



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 :

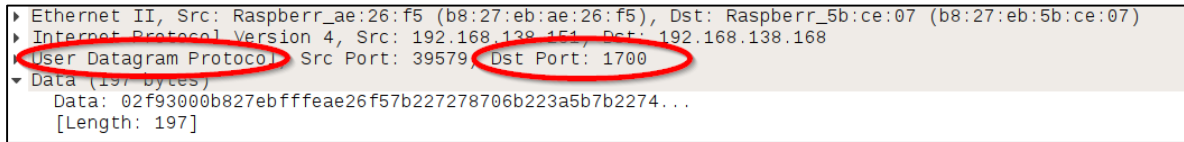


Figure 73 : Trame capturée par Wireshark lors d'un Uplink

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 }

	Champ [1]	Champ [2]	Champ [3]	Champ [4]	
0000	02	f9 30	00	b8 27 eb ff fe ae 26 f5	..0...'....&.{ "rx
0010	70 6b 22 3a 5b 7b 22 74 6d 73 74 22 3a 33 37 35				pk": [{ "tmst": 375
0020	35 30 30 35 38 31 39 2c 22 63 68 61 6e 22 3a 32				5005819, "chan": 2
0030	2c 22 72 66 63 68 22 3a 31 2c 22 66 72 65 71 22				, "rfch": 1, "freq"
0040	3a 38 36 38 2e 35 30 30 30 30 30 2c 22 73 74 61				: 868.500000, "sta
0050	74 22 3a 31 2c 22 6d 6f 64 75 22 3a 22 4c 4f 52				t": 1, "modu": "LOR
0060	41 22 2c 22 64 61 74 72 22 3a 22 53 46 37 42 57				A", "datr": "SF7BW
0070	31 32 35 22 2c 22 63 6f 64 72 22 3a 22 34 2f 35				125", "codr": "4/5
0080	22 2c 22 6c 73 6e 72 22 3a 36 2e 35 2c 22 72 73				", "lsnr": 6.5, "rs
0090	73 69 22 3a 2d 31 2c 22 73 69 7a 65 22 3a 31 38				si": -1, "size": 18
00a0	2c 22 64 61 74 61 22 3a 22 51 4e 4d 61 41 53 59				, "data": "QNMaASY
00b0	41 41 51 41 50 70 79 50 5a 39 35 35 2b 53 6d 59				AAQAPpyPZ955+SmY
00c0	2f 22 7d 5d 7d				/"}]]}

Figure 74 : 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 :

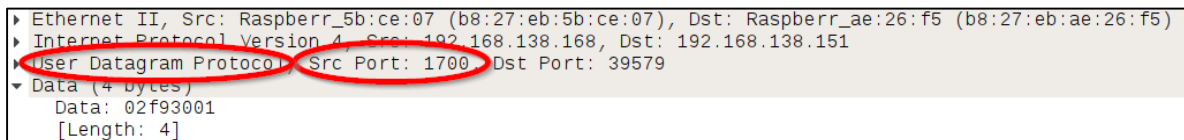


Figure 75 : Trame capturée par Wireshark lors de l'acquittement d'un Uplink

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.

### 7.1.4 LoRa Gateway Bridge

Nous avons décrit au chapitre précédent (7.1.3) le fonctionnement du Packet Forwarder. Le Network Server(Lora Server) que nous allons mettre en place aurait très bien pu s'interfacer directement à ce protocole. Mais cela aurait plusieurs impacts dont un qui est important dans la conception de logiciel : Si le Packet Forwarder change, ou si un autre type de Forwarder est utilisé, alors l'utilisation du LoRa Server devient impossible. Il a donc été choisi de créer une étape intermédiaire. Le Network Server que nous mettrons en plus utilisera le protocole MQTT plutôt que le protocole UDP packet Forwarder.

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

### 7.1.5 LoRa Server (Network Server)

Il s'agit de notre Network Server qui a le même rôle que celui que nous avons étudié au chapitre 4.1.3. Cependant, il s'interfacera au protocole MQTT en recevant les informations à partir du Broker au lieu de recevoir directement les trames en provenance de la Gateway.

### 7.1.6 LoRa App Server (Application Server)

Il s'agit de notre Application Server qui a le même rôle que celui que nous avons étudié au chapitre 4.1.4.

### 7.1.7 LoRa Geo Server

Il s'agit d'un service qui permet d'utiliser certaines propriétés du protocole LoRa pour proposer un service de géolocalisation des Devices LoRa. Nous ne l'utiliserons pas dans notre cas.

### 7.1.8 Application

Il s'agit de notre Application qui a le même rôle que celui que nous avons étudié au chapitre 4.1.5. Elle ne fait pas partie du projet LoRaServer, c'est donc bien toujours à nous de proposer notre propre Application.

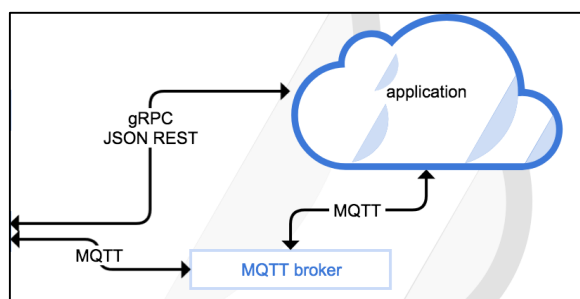


Figure 76 : Les interfaces possibles entre LoRaServer et l'Application utilisateur

Comme nous pouvons le voir sur la figure précédente, deux possibilités sont offertes pour interfacer notre application : JSON REST (incluant HTTP POST) et via un Broker MQTT. Ces deux méthodes ont déjà été traitée au chapitre 6.1 (HTTP POST) et au chapitre 6.2 (MQTT). Nous verrons aussi d'autres possibilités pour personnaliser notre application au chapitre 7.

## 7.2 Installation de LoRaServer

### 7.2.1 Mise en place de l'environnement

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

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

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

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

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

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

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

- `apt-get install tshark` // installation
- `tshark -n -i eth0` // Exemple d'une analyse sur l'interface wlan0

## 7.2.2 Installation sur la Raspberry PI

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

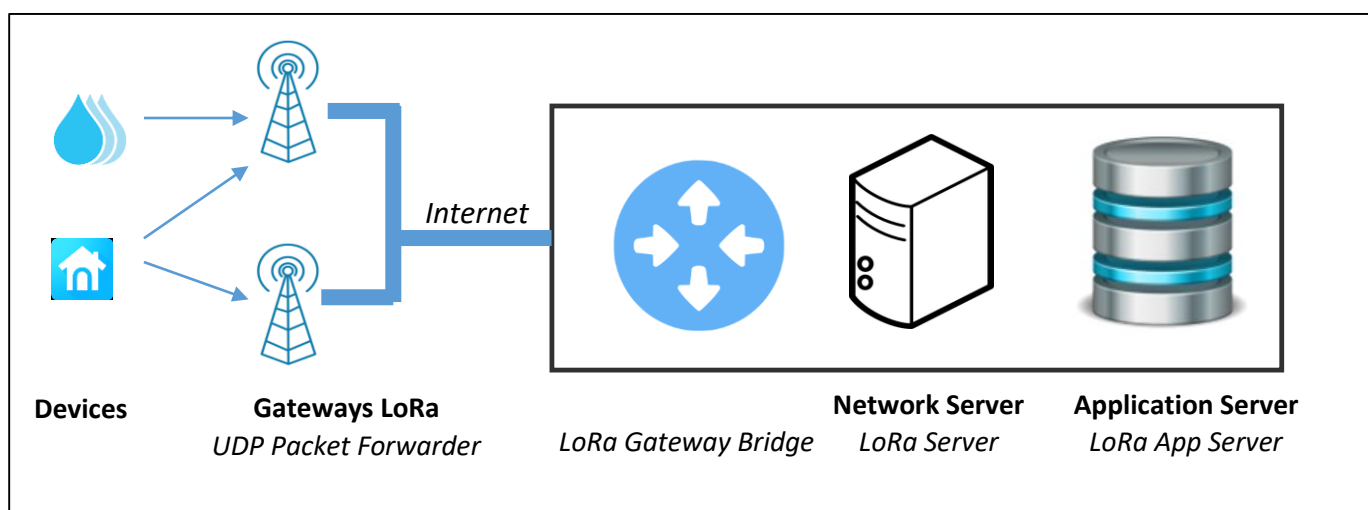


Figure 77 : Architecture globale après installation de LoRaServer

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

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

Les Usernames et Password par défaut sont :

- Username: admin
- Password: admin

L'écran d'accueil sera donc le suivant :

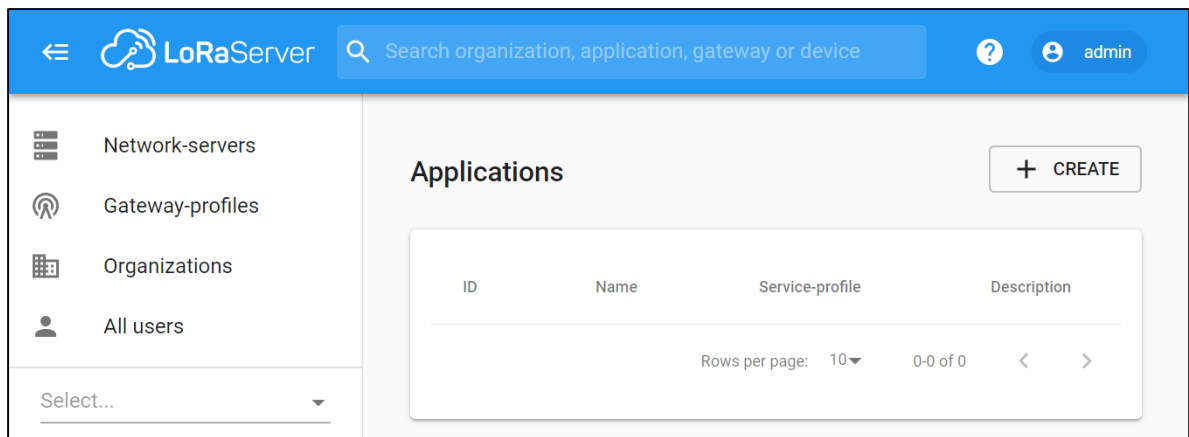


Figure 78 : Ecran d'accueil de LoRaServer après authentification

## 7.3 Configuration de LoRa Server pour l'Uplink

### 7.3.1 Enregistrement d'une nouvelle Organisation

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

Organization name \*

myOrganization

The name may only contain words, numbers and dashes.

Display name \*

myOrganization

**Gateways**

☒ Organization can have gateways

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

[CREATE ORGANIZATION](#)

Figure 79 : Enregistrement d'une nouvelle Organisation

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

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

The screenshot shows a web interface for configuring a Network Server. At the top, there are three tabs: 'GENERAL' (selected), 'GATEWAY DISCOVERY', and 'TLS CERTIFICATES'. Under the 'GENERAL' tab, there are two input fields. The first is labeled 'Network-server name\*' and contains the text 'myNetworkServer'. Below it is a small grey box with the text 'A name to identify the network-server.' The second input field is labeled 'Network-server server\*' and contains the text 'localhost:8000'. Below it is a small grey box with the text 'The 'hostname:port' of the network-server, e.g. 'localhost:8000'.'. At the bottom right of the form is a blue button labeled 'UPDATE NETWORK-SERVER'.

*Figure 80 : Enregistrement d'un nouveau Network Server*

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

- Le nombre de fois par jour que les PING LoRa sont envoyés
- Le canal d'émission
- Le Data Rate (Voir paragraphe 4.4.5)

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

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

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

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

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

On laissera par défaut toutes les autres options.

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

Application name \*

myApplication

The name may only contain words, numbers and dashes.

Application description \*

myApplication

Service-profile \*

myServiceProfile

The service-profile to which this application will be attached. Note that you can't change this value after the application has been created.

Payload codec

None

By defining a payload codec, LoRa App Server can encode and decode the binary device payload for you.

CREATE APPLICATION

Figure 81 : Enregistrement d'une nouvelle Application

### 7.3.4 Enregistrement des Devices LoRa

Il faut tout d'abord faire l'enregistrement d'un Device-profile pour les Devices LoRa que vous souhaitez connecter. Dans notre cas, nous ferons un premier test en spécifiant que nous utilisons seulement le mode d'authentification ABP (voir chapitre 4.3.1).: **Devices profile > Create**

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

Device-profile name \*

myDeviceProfile

A name to identify the device-profile.

LoRaWAN MAC version \*

1.0.0

The LoRaWAN MAC version supported by the device.

LoRaWAN Regional Parameters revision \*

A

Revision of the Regional Parameters specification supported by the device.

Max EIRP \*

20

Maximum EIRP supported by the device.

UPDATE DEVICE-PROFILE

Figure 82 : Enregistrement d'un « Device profile »

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

Device name \*

arduino0

The name may only contain words, numbers and dashes.

Device description \*

arduino0

Device EUI \*

00 56 AB 22 5B F4 98 77

MSB ↺

Device-profile \*

myDeviceProfile

☒ Disable frame-counter validation

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

CREATE DEVICE

Figure 83 : Enregistrement d'un nouveau Device LoRa

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

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

Device address \*

26 01 1a d3

MSB ↺

Network session key (LoRaWAN 1.0) \*

..... 🔒

Application session key (LoRaWAN 1.0) \*

..... 🔒

Uplink frame-counter \*

0

Downlink frame-counter (network) \*

0

(RE)ACTIVATE DEVICE

Figure 84 : Configuration de l'enregistrement du Device LoRa (en ABP)

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

### 7.3.5 Visualisation des trames reçues

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



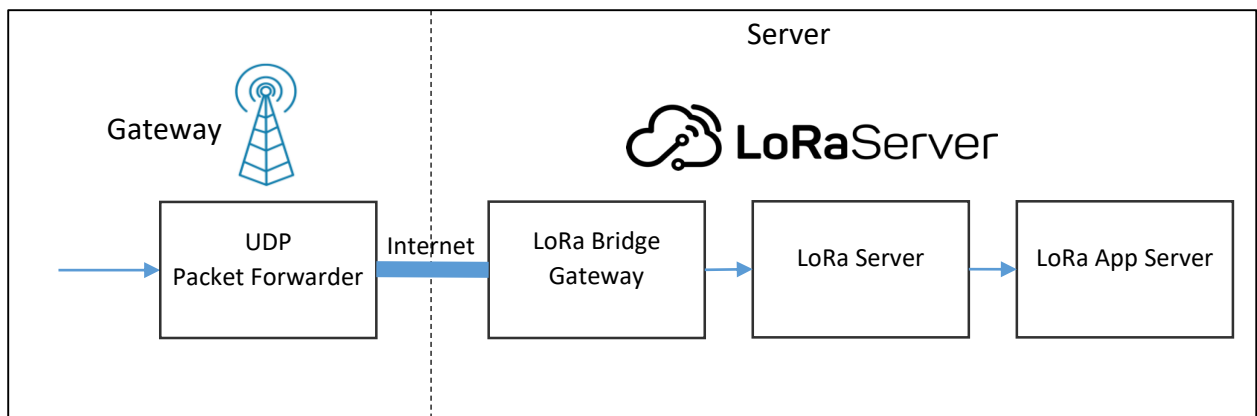


Figure 85 : Résumé du cheminement des trames LoRaWan en Uplink avec LoRaServer

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

CONFIGURATION	KEYS (OTAA)	ACTIVATION	LIVE DEVICE DATA	LIVE LORAWAN FRAMES
<div> <span>?</span> HELP                 <span>  </span> PAUSE                 <span>↓</span> DOWNLOAD                 <span>🗑️</span> CLEAR             </div>				
2:56:16 PM	uplink			▼
2:56:07 PM	uplink			▼
2:55:56 PM	uplink			▼
2:55:46 PM	uplink			▼
2:55:45 PM	uplink			▼

Figure 86 : Réception des trames des Devices LoRa

2:55:46 PM	uplink	^
<pre> adr: false applicationID: "3" applicationName: "myApplication" data: "SGVsbG8=" devEUI: "0056ab225bf49877" deviceName: "arduino0" fCnt: 10262 fPort: 15 ▼ binfo: {} 2 keys   dr: 5   frequency: 868100000                 </pre>		

Figure 87 : Analyse des trames des Devices LoRa

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

CONFIGURATION	KEYS (OTAA)	ACTIVATION	LIVE DEVICE DATA	LIVE LORAWAN FRAMES
<div> <span>?</span> HELP           <span>▶ RESUME</span> <span>⬇ DOWNLOAD</span> <span>🗑 CLEAR</span> </div>				
UPLINK	2:54:34 PM	UnconfirmedDataUp	26011ad3	▼
UPLINK	2:54:24 PM	UnconfirmedDataUp	26011ad3	▼
UPLINK	2:54:13 PM	UnconfirmedDataUp	26011ad3	▼
UPLINK	2:54:03 PM	UnconfirmedDataUp	26011ad3	▼

Figure 88 : Réception des Trames LoRaWAN

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

Nous avons vu dans le chapitre 7.1.8 que l'interface entre LoRaServer et notre Application pouvait être réalisée soit par MQTT, soit par les méthodes REST (HTTP POST). Nous avons déjà implémenté et expliqué ces protocoles dans le cadre de son utilisation avec TTN. Nous allons donc juste configurer LoRaServer pour tester ces deux méthodes.

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

Nous utiliserons exactement la même méthode que celle que nous avons utilisé avec TTN au chapitre 6.1.5. Les différents Clients/Serveurs disponibles pour ces tests sont expliqués au paragraphe 6.1.2.

Dans LoRaServer, ajouter une intégration HTTP : **LoRaServer > Applications > myApplication > Integrations > Create > HTTP Integration** et la configurer comme le montre la Figure 89. Vous remplacerez évidemment le lien par votre propre Endpoints.

Endpoints

Uplink data URL

<https://rbaskets.in/aqsyac8>

Figure 89 : Configuration de l'intégration HTTP POST dans LoRaServer

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

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

Nous allons nous connecter au Broker MQTT de LoRaServer. Pour cela nous utiliserons la même méthode que celle que nous avons utilisé avec TTN au chapitre 6.2.8 . Nous utiliserons MQTTBox, comme nous l'avons déjà fait au paragraphe 6.2.7.

Le Broker de LoRaServer a enregistré les Topics suivant :

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

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

*Tableau 13 : Topics enregistrés dans LoRaServer*

## 8 Création de notre propre Application

### 8.1 Utilisation de Node-RED

#### 8.1.1 Présentation

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

Dans les démonstrations que nous mettrons en œuvre, nous utiliserons une RPI. Par soucis de simplicité, nous utiliserons les dernières versions de Raspbian (système d'exploitation de la RPI) qui intègre d'origine Node-RED. Nous pourrions travailler soit directement sur la RPI à l'aide d'une souris et clavier, mais il est recommandé de travailler à distance avec une connexion SSH.

#### 8.1.2 Mise en place de l'environnement

La mise en place de la RPI avec le système d'exploitation Raspbian a été présentée au chapitre 7.2.1. Vous pouvez reprendre la même installation ou repartir sur une nouvelle.

Il est possible de faire en sorte que Node-RED se lance directement au démarrage grâce à la commande : **sudo systemctl enable nodered.service**

S'il n'est pas actif, lancer Node-RED sur votre RPI : **node-red-start**

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

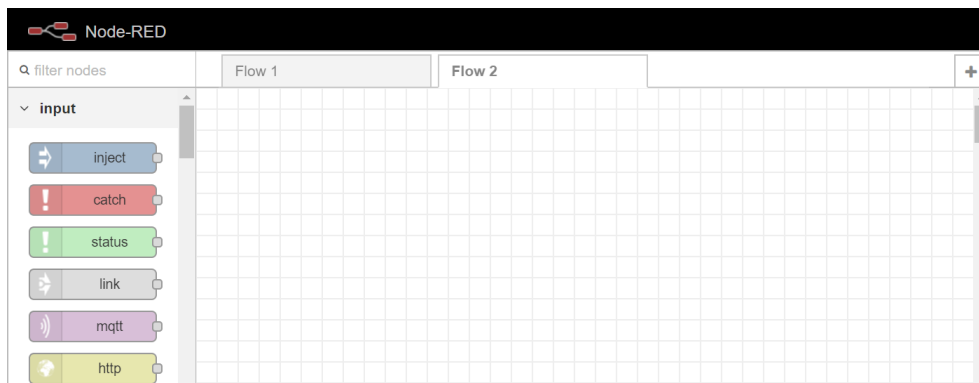


Figure 90 : Page d'accueil de Node-RED

Il est possible de s'interfacer avec TTN avec les Nodes déjà disponibles : grâce au protocole HTTP POST ou MQTT. Néanmoins, nous pouvons avantageusement installer deux bibliothèques spécifiques :

- Une qui facilite l'interfaçage avec TTN
- Une qui facilite la mise en place d'interface graphique

Installation des Nodes TTN :

```
cd $HOME/.node-red
npm install node-red-contrib-ttn
```

Installation de Dashboard (interface graphique)

```
cd $HOME/.node-red
npm install node-red-dashboard
```

Il peut être nécessaire de redémarrer Node-RED pour prendre en compte l'installation de nos deux Nodes.

### 8.1.3 Créer une Application spécifique pour TTN

Nous allons commencer par 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".

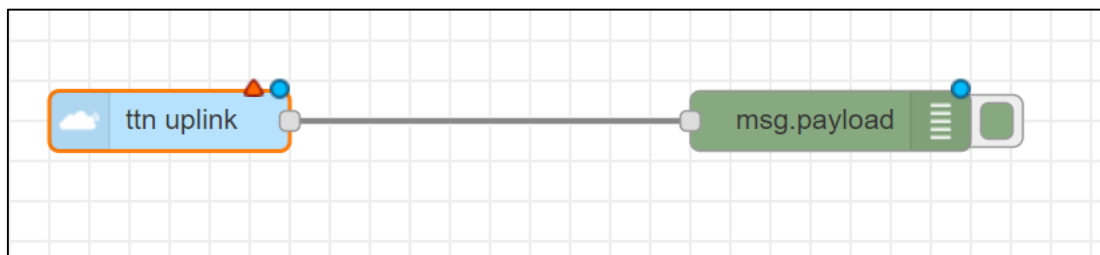


Figure 91 : Visualisation des trames publiées par TTN sur notre client MQTT

Double cliquer sur le Node ttn uplink pour le configurer comme sur la Figure 92 :

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.

L'image montre l'interface de configuration du nœud 'ttn uplink'. Elle contient trois champs : 'Name' avec la valeur 'Reception données TTN', 'App' avec un menu déroulant 'Add new ttn app...' et un bouton 'Add' à droite, et 'Device ID' avec la valeur 'arduino0'.

Figure 92 : Configuration du node "ttn uplink"

Configuration de votre nouvelle application :

Cancel Add

App ID seminaire\_lorawan

Access Key .....

Discovery address discovery.thethingsnetwork.org:1900

Figure 93 : Identifiant et mot de passe de notre application

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

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

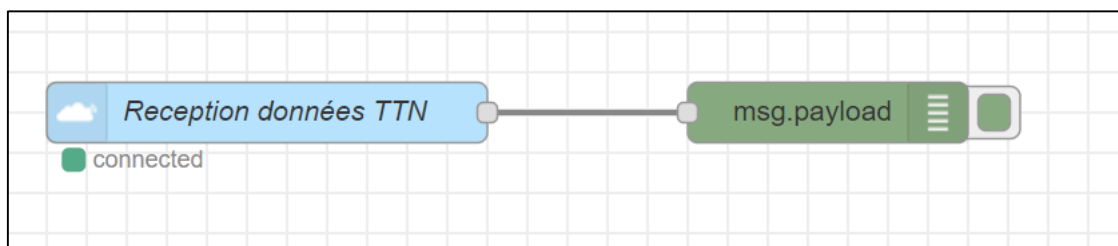


Figure 94 : Utilisation du Node TTN Uplink

Réaliser un test de communication grâce à une simulation d'Uplink comme nous l'avons vu au paragraphe 5.4.5 : **TTN > Application > Nom\_de\_votre\_application > Devices > arduino0**

**SIMULATE UPLINK**

FPort	Payload	
1	31 32	✓ 2 bytes

Figure 95 : Simulation de Uplink dans TTN

Vous devriez voir le résultat dans la fenêtre de Debug. Par exemple pour le test d'envoi de 31 32, correspondant au code ASCII des chiffres '1' et '2', vous devriez avoir la fenêtre de Debug suivante :



Figure 96 : Résultats obtenu dans la fenêtre Debug après la simulation d'Uplink

### 8.1.4 Création d'un Dashboard

Dans la mise en place de l'environnement (chapitre 8.1.2), nous avons installé des Nodes "Dashboard" qui vont nous permettre de réaliser une interface graphique très simple et de la mettre à disposition des utilisateurs via un des graphiques, histogramme, gauges, afficheurs...

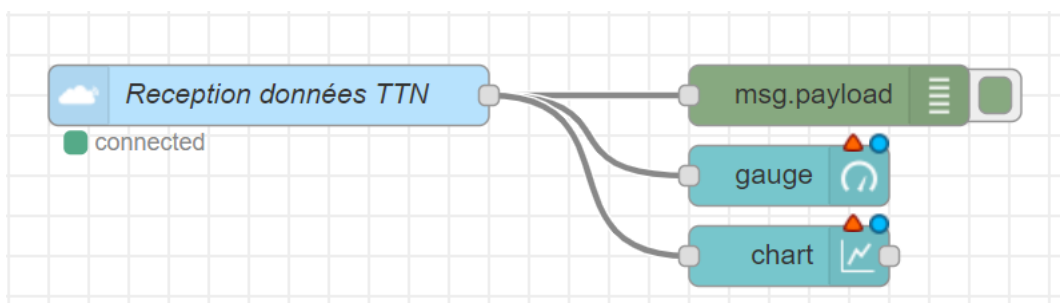


Figure 97 : Implémentation d'une interface graphique dans Node RED

L'interface graphique que nous allons créer sera constituée d'onglet (tab). La mise en place des composant se fait de la façon suivante :

- Chaque élément graphique doit appartenir à un groupe de composant qu'il va falloir créer.
- Chaque groupe de composant doit appartenir à un onglet (tab) qu'il va falloir créer.

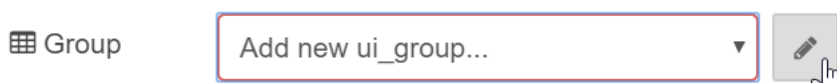


Figure 98 : Création d'un groupe de composant

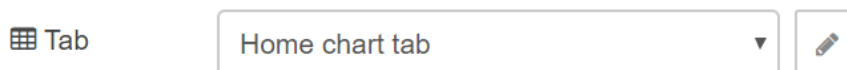


Figure 99 : Création d'un nouvel onglet dans le site web

Cliquer sur "Deploy" pour activer votre application

LURL pour joindre l'interface graphique de votre site web est disponible sur l'adresse [http://@IP\\_Raspeberry\\_PI:1880/ui](http://@IP_Raspeberry_PI:1880/ui) .

Réaliser un test de communication grâce à une simulation d'Uplink comme nous l'avons vu au paragraphe 5.4.5 : TTN > Application > Nom\_de\_votre\_application > Devices > arduino0

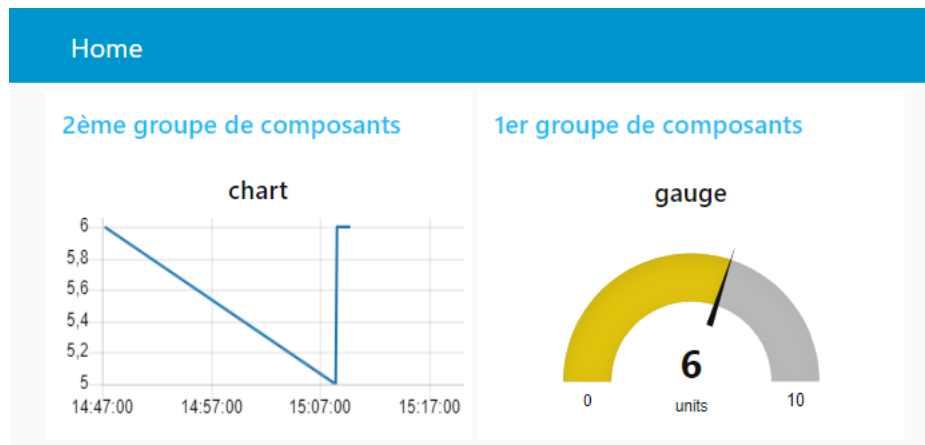


Figure 100 : Interface graphique du site web avec Node-RED

## 8.2 Programmation en PHP

Maintenant que nous avons testé toute l'architecture, nous pouvons mettre en place une application qui jouera deux rôles :

- Un rôle de Serveur HTTP POST récupérant les données en provenance de TTN
- Un rôle de Client HTTP POST qui générera les requêtes vers TTN

Un serveur web apache sur RPI héberge cette application. L'interface graphique du site web est réalisé avec le template CSS proposé par Bootstrap [ <https://getbootstrap.com/> ]

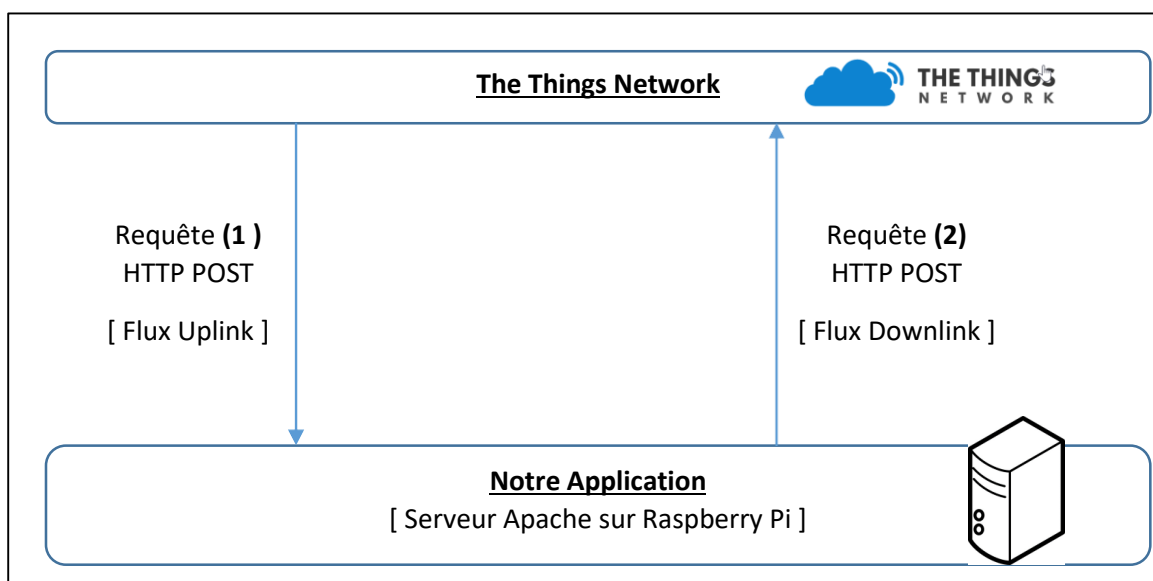


Figure 101 : Architecture de notre application en PHP sur RPI



Nous allons l'utiliser telle quelle. Elle est juste proposée à titre d'exemple L'objectif de cette application est de générer une ouverture / fermeture de ruche à distance dans le cadre de projets étudiants du Master Electronique et Systèmes Embarqués de l'USMB : [ <http://scem-eset.univ-smb.fr/> ]

Supervision des Ruches			
stm32lorawan_1			
<div>OuvrirFermerSupprimer</div>			
#	Date & Heure	Status de la porte	Température
1	11-01-2019 15:27:34	Inconnu	unknown°C

Figure 102 : Application permettant de recevoir et d'émettre des requêtes HTTP POST

- ➡ Configurer TTN pour pointer vers votre nouveau serveur, et tester le flux montant et descendant.

## 9 Versions du document

---

Version 1.0 : 02/02/2019

- **Version initiale.**

Version 2.0 : 23/02/2019

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

Version 3.0 : 8/03/2019

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

Version 4.0 : 26/03/2019

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

Version 4.1 : 14/10/2019

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