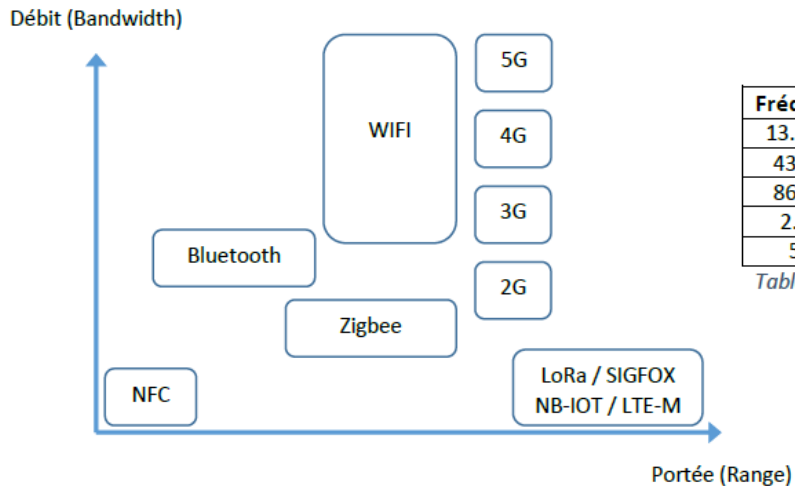


# Discussion 1 Comment fonctionne l'architecture Lorawan

## Réseaux sans fil

Les protocoles utilisés dans l'IoT en fonction du débit et de la portée :



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

Tableau 1 : Bandes de fréquence libres et utilisations

## Lora/Lorawan

### Classes des Devices LoRaWan

**Classe A** : Minimal power Application. (All) (**celui choisi standard**)

Les Devices peut transmettre (Uplink) à la Gateway sans vérification de la disponibilité du récepteur.

Cette transmission est suivie de 2 fenêtres de réception très courtes.

La gateway peut alors transmettre pendant le RX Slot1 et RX Slot 2 mais pas les deux.

Le device ne peut pas recevoir s'il n'a pas émis il n'est donc pas joignable facilement.

**Classe B** : Scheduled Receive Slot. (Beacon)

Même chose que classe A mais d'autres fenêtres de réception sont programmées à des périodes précises. Afin de synchroniser les fenêtres de réception du device, la gateway doit transmettre des balises (Beacons) régulièrement.

Le device est joignable régulièrement sans qu'il soit nécessairement obligé d'émettre. Mais il consomme plus.

**Classe C** : Continuous Listening (Continuous)

Le device ont des fenêtres de réception constamment ouvertes entre 2 uplinks.

Le device est joignable en permanence (énergivore)

Pour le Downlink : 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.

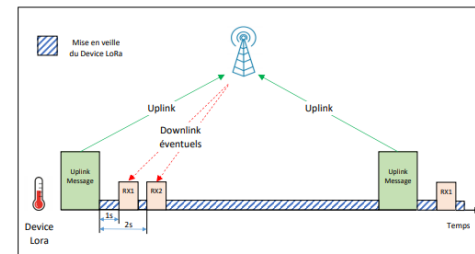


Figure 30 : Slots de réception pour un Device LoRa de classe A.

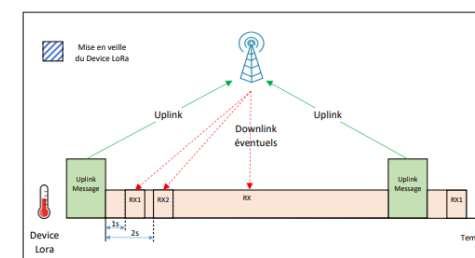


Figure 32 : Slots de réception pour un Device LoRa de classe C

LoRa a une puissance d'émission maximum sur la bande 868 Mhz est de 14dBm (25mW).

Au final, ce qui compte, c'est surtout la différence entre la puissance PE et la sensibilité du récepteur. C'est ce qu'on appelle le **Link Budget**, En LoRa, nous avons un Link Budget d'environ 157 dB, En LTE (4G), nous avons un Link Budget d'environ 130 dB.

## Différence entre les trames

**Lora** c'est le type de modulation permettant d'envoyer des données entre un émetteur et un récepteur

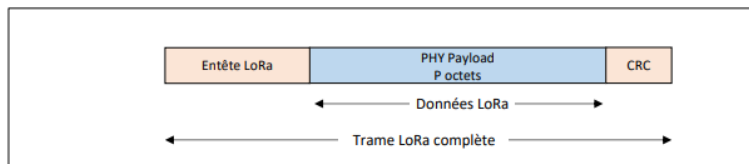


Figure 17 : Trame LoRa

**Lorawan** (avec un service supplémentaire) Architecture du réseau et format de trame.

On parle de l'ensemble de la chaîne de communication alors nous parlons de protocole LoRaWAN

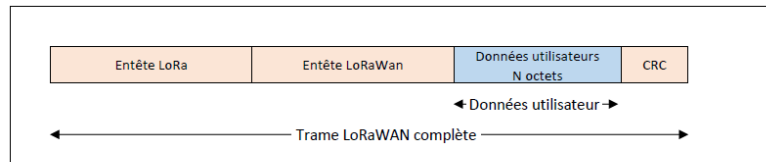


Figure 19 : Trame LoRaWAN

Le duty-cycle impose qu'un device Lora ne transmette pas plus de 1% du temps. 1% donc pour 1 temps je ne dois plus émettre pendant 99%. Cas Time on air de 46.3ms le device doit émettre pendant  $99 \times 46.3 = 4.58$  secondes.

## Architecture d'un réseau lorawan

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

Cet identifiant est utile pour enregistrer et activer une Gateway sur un Network Server

### 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**. (pour l'authentification)

### 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

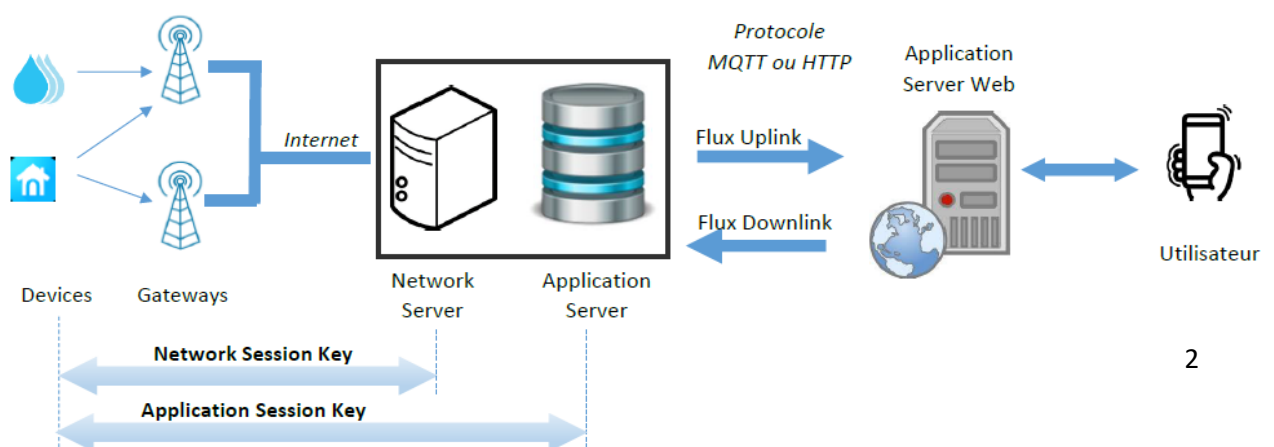
Grâce à une clé AES 128 bits appelée Application Session Key : **AppSKey**. Contrairement au **NwkSKey (Network Session Key)**, il s'agit bien ici d'un chiffrement

### Application Utilisateur

Cette application doit récupérer les données de l'Application Server. Dans ce cadre, cela sera fait de deux façons : avec le protocole HTTP et MQTT

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

**Le flux habituel dans l'IoT est le flux Uplink (flux montant), c'est-à-dire l'émission de données des objets vers le serveur. Comme nous l'expliquerons au paragraphe 4.3, en LoRaWAN il est aussi possible de transférer des données aux Device LoRa par un flux Downlink (flux descendant).**



### Les 3 éléments indispensables pour la communication LoRaWAN :

Node :

DevAddr : identifie le device

AppSKey : clé de chiffrement

NwSKey : clé authentification

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

### 2 méthodes sont possibles pour fournir ces informations à la fois au device Lora et au serveur :

Activation By Personalization (ABP) ou Over The Air Activation (OTAA)

#### **ABP (la plus simple des méthodes)**

Le device possède déjà DevAddr, AppSKey, NwSKey

Le Network Server et application serveur possède déjà DevAddr, NwSKey, AppSKey.

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

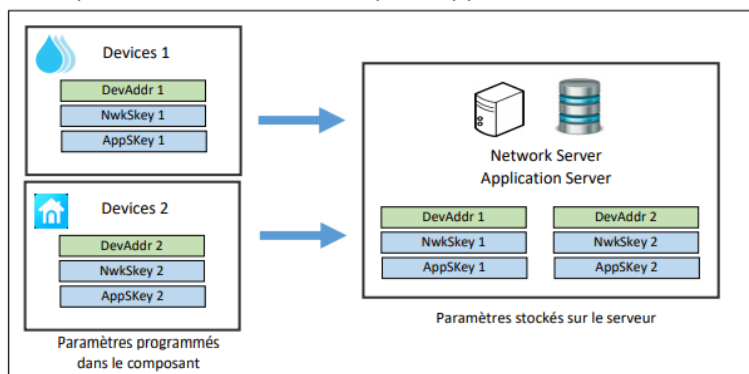


Figure 34 : Paramétrage de DevAddr, NwkSkey et AppSkey en ABP

#### **OTAA (celui choisi standard)**

Le DevAddr, AppSKey, NwSKey vont être générés lors d'une procédure de négociation (Join Request) dès lors que le device se connectera pour la première fois au serveur.

Au préalable le device Lora doit connaître le **DevEUI, AppEUI, AppKey**.

Le network server doit lui connaître le DevEUI, AppEUI et AppKey.

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

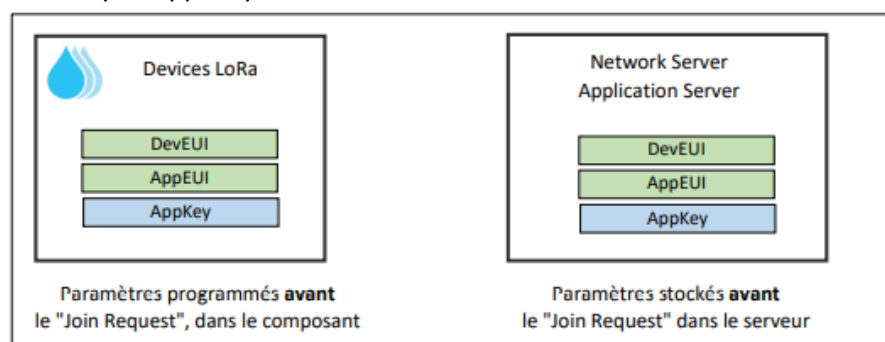


Figure 35 : Paramétrage du Device LoRa et du Serveur avant le Join Request (OTAA)

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

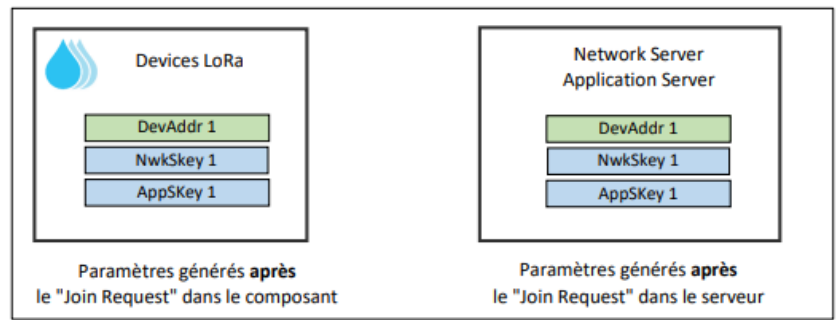


Figure 36 : Paramétrage du Device LoRa et du Serveur après le Join Request (OTAA)

Informations possédées par le Device LoRa avant le Join Request :

DevEUI : Unique identifier pour le device Lora (équivalent à une @MAC sur Ethernet) certain device Lora ont déjà un DevEUI fourni en usine.

AppEUI : Unique identifier pour l'application server.

AppKey : AES 128 clé utilisée pour générer le MIC (Message Code Integrity) lors de la join Request il est partagé avec le Network server.

Informations possédées par le Device LoRa après le Join Request :

NwkSKey : Utilisé pour l'authentification avec le Network Server.

AppSKey : Utilisé pour le chiffrement des données.

DevAddr : Identifiant sur 32 bits unique au sein d'un réseau LoRa.

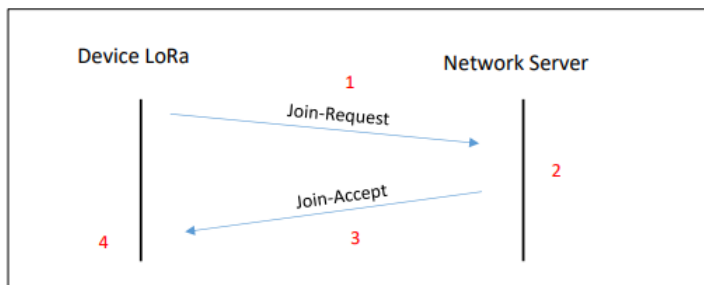


Figure 37 : Join Request – Join Accept en OTAA

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

Sécurité ABP ou OTAA sont équivalentes.

Le point faible sont les clés stockées en permanence dans le Device Lora.

ABP (NwkSKey et AppSKey) et OTAA (AppKey)

Elles devront être stockées dans des mémoires sécurisées

Protection contre l'attaque par Replay

Les informations transmises sont chiffrées en AES 128bits l'attaque pas man-middle

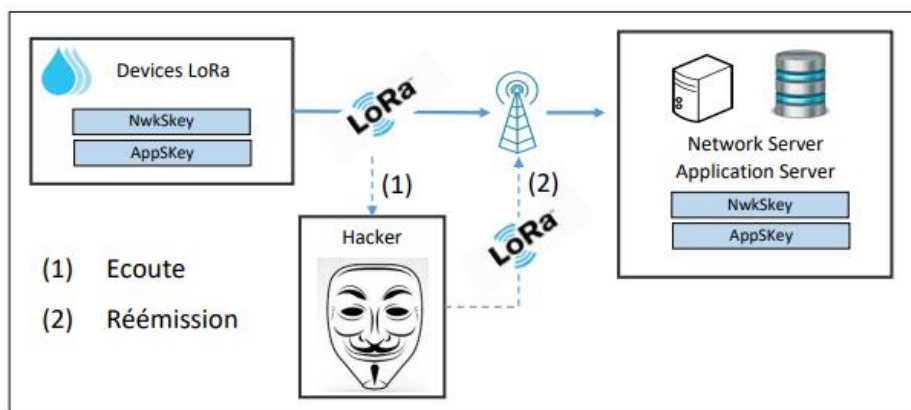


Figure 39 : Attaque par REPLAY

Même si les données sont chiffrées les données sont transportées et elles sont bien comprises par l'Application Server. Pour éviter la trame Lorawan intègre un champ variable appelée Frame Counter. Nombre sur 2 octets numérotant la trame. Et le server accepte que si la Frame Counter est supérieure à la précédente.

Et si le hack modifie le Frame Counter au hasard, l'authentification échouera car le calcul du MIC avec le Network Session Key ne sera plus valide.

Mais quand le device est remis à zéro il faut que le frame côté server est aussi à zéro Frame counter check

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

L'attribution du devAddr

Nécessaire pour la communication Lora. Généré par le serveur, le devAddr ne sont pas uniques c'est l'association DevAddr / NwkSKey qui permet d'identifier le device parmi les milliards de systèmes Lorawan existants. Et auront un préfixe comme 0x2601.

Pour OTAA le device réémette un join request au server.

Pour ABP reprogrammer le firmware du device.

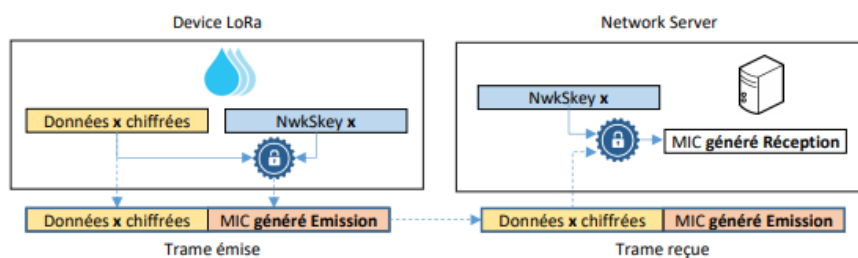
	ABP	OTAA
Sécurité globale	Stockage en mémoire sécurisée : <b>NwSKey</b> et <b>AppSKey</b>	Stockage en mémoire sécurisée : <b>AppKey</b>
Gestion du Frame Counter	Sauvegarde en mémoire non volatile obligatoire. Possibilité de désactiver la vérification du compteur en risquant des attaques par replay.	Prise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

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

### Authentification avec le Network Server

Le NwSKey (Network Session Key) sert d'authentification entre le Device LoRa et le Server. Un champ MIC (Message Integrity Control) est rajouté à la trame (calculé en fonction des données transmises et le NwSKey et le même calcul est effectué à la réception).

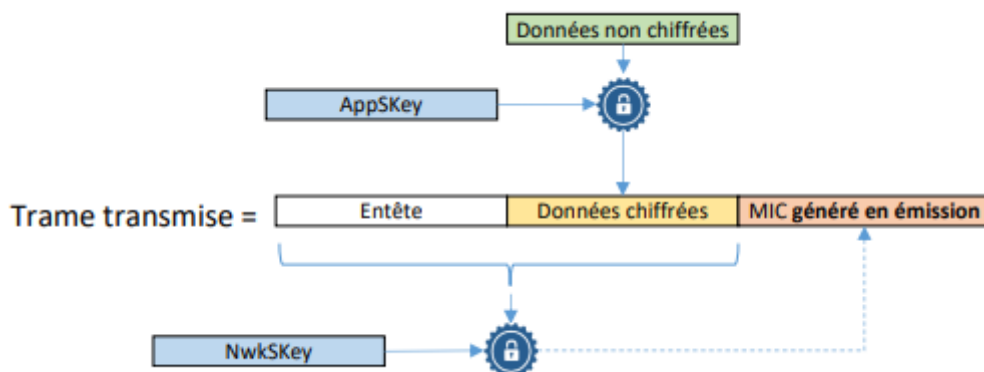
Dans la figure on parle de données x chiffrées. Ces données ont au préalable été chiffrées par l'AppSKey avant de passer le processus d'authentification décrit ici.



Si **MIC généré Emission** = **MIC généré Réception**

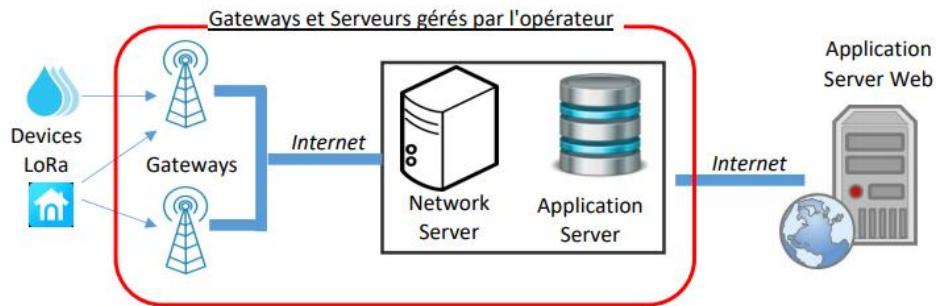
**Alors la trame est authentifiée**

Ce qui donne le chiffrement par authentification



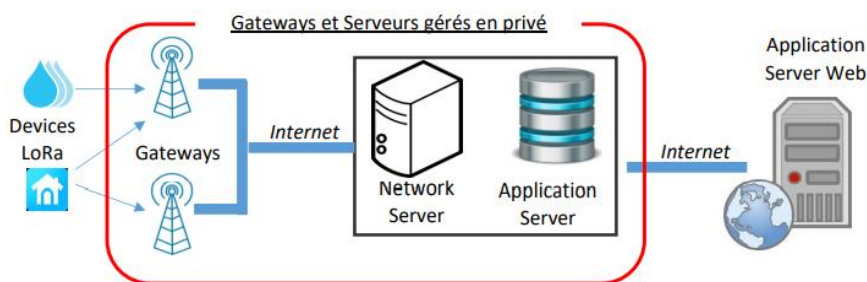
## Les réseaux et serveurs LoRaWAN

### Les réseaux LoRaWAN opérés



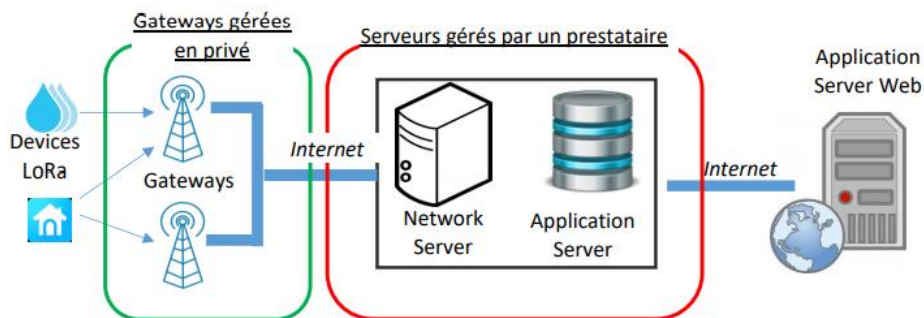
Avec les abonnements orange, bouygues, illimité en uplink (duty\_cycle). Prix par message de 5cts  
Abonnement 2€/mois, Bouygues 144 msg/jour uplink et 6 msg downlink pour 20€ TTC/capteur / an

### Les réseaux LoRaWAN privés (celui choisi standard avec VPN)



Il faut implémenter son propre réseau de Gateways et son infrastructure. La mise en place d'un network server et Application server private. Dans certaines gateways une implémentation de ces 2 serveurs est proposée. Il existe des serveurs (network et application) open source chirpstack et thethingsstack

### Alternative Le réseau LoRaWAN dédié (hybride)



Les zones de couverture Pour connaître la zone de couverture de notre Gateway utilisée avec TTN, nous pouvons utiliser l'application TTN Mapper : <https://ttnmapper.org/>.

les couches du protocole LoRaWAN

Les couches protocolaires supplémentaires se rajoutent pour le lorawan avec une communication en point à point. Chaque couche rajoute un service.

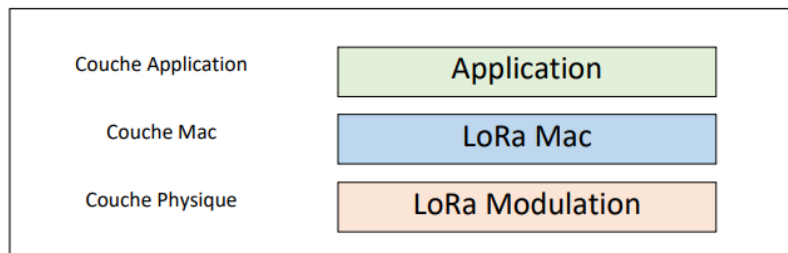
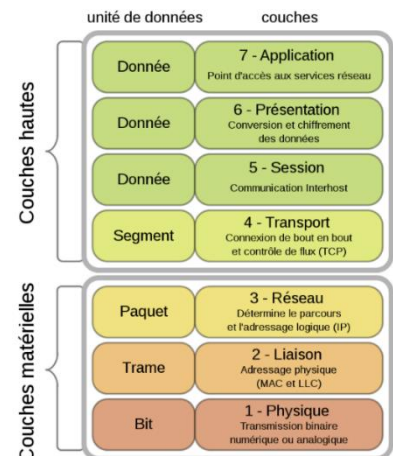
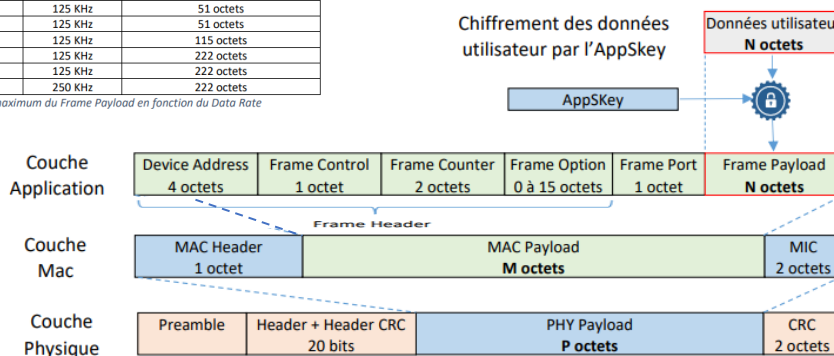


Figure 58 : Les couches protocolaires du LoRaWAN

Lors d'un envoi de la trame les données utilisateurs sont encapsulées dans chaque couche inférieur jusqu'à la transmission

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

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



Le protocole LoRa de la couche MAC composé de :

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

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

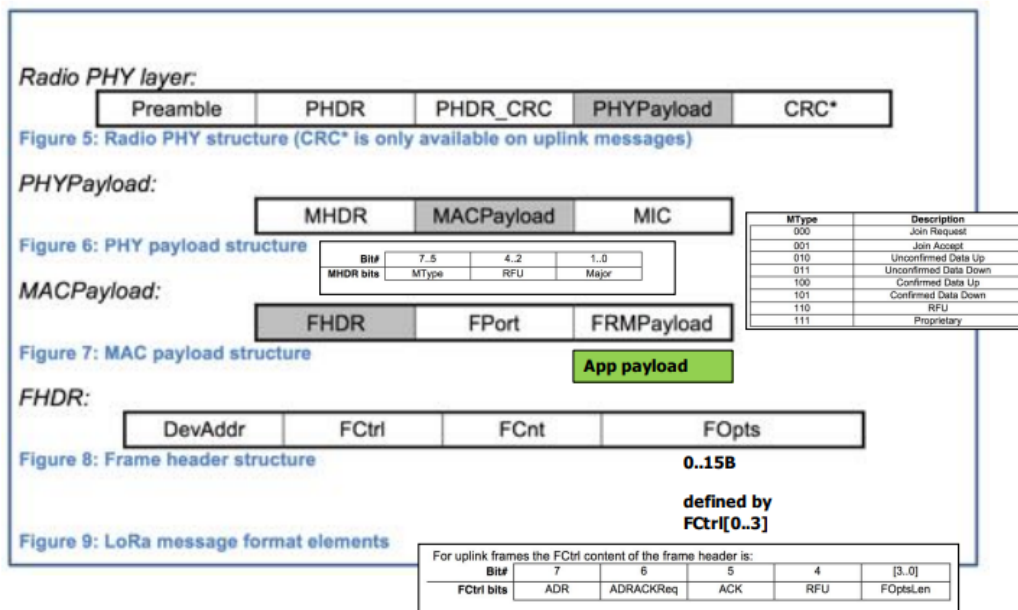
Tableau 14 : Les types de messages transmis en LoRaWAN

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

La couche 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 transmettra simplement au bout d'un temps aléatoire.

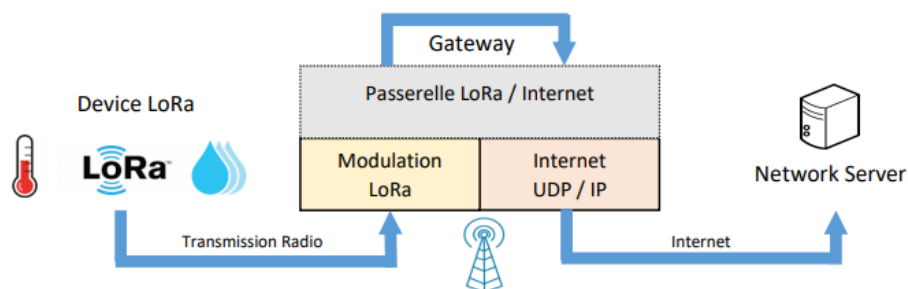
La couche physique est représentée par l'émission de la trame. Le préambule est représenté par 8 symboles + 4.25. Le temps du Préambule est donc de 12.25 Tsymbole, le Header optionnel seulement présent dans le mode de transmission par défaut (explicite). Il est transmis avec un Coding Rate de 4/8. Il 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. PHY Payload contient toutes les informations de la couche Lora MAC, CRC sert à la détection d'erreur de la trame LoRa.





La Gateway : du LoRa à la trame IP.

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



Côté interface Radio : La Gateway réceptionne une trame Lorawan et extrait le PHY Payload. Elle va coder ce PHY Payload en format ASCII en base 64. La Gateway extrait aussi toutes les informations utiles sur les caractéristiques de la réception qui a eu lieu : SF, Bandwith, RSSI, Time On Air ...

Côté 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 au format JSON. La Gateway a donc bien un rôle de passerelle entre le protocole LoRa d'un côté et un réseau IP de l'autre.

## Analyse des trames IP

Le format JSON

Les données applicatives sont formatées en JSON, les valeurs

Un string exemple : "coding\_rate" : "4/5"

Un nombre exemple : "spreading\_factor" : 12

Un objet exemple : "lora": { "spreading\_factor": 12, "air\_time": 2465792000 }

Un booléen exemple : "service" : true

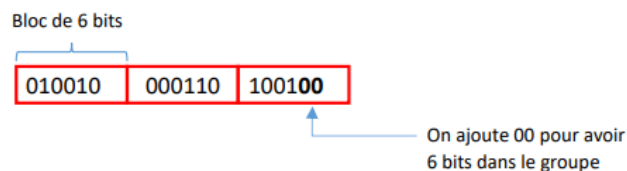
Utilisation de la base 64

Dans les données applicatives, un champ important est à notre disposition : Payload.

Le Payload correspond aux données transmises dans la trame LoRa. Elles sont brutes et non déchiffrées.

Notre Gateway nous présente le Payload PHY en base 64. L'explication de la méthode exemple : le code hexa 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 en base 64 est SGk=
2. On regroupe en bloc de 6 bits et doit être un multiple de 4 (minimum 4 blocs). S'il manque on rajoute des zéros



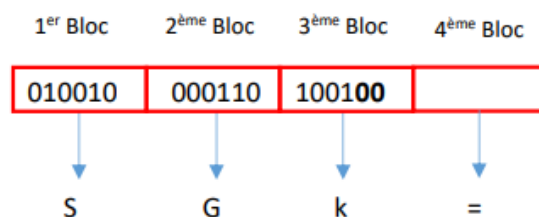
3. S'il manque des blocs pour faire un minimum de 4 blocs, des caractères spéciaux seront ajoutés.
4. Chaque groupe de 6 bits est traduit par le tableau suivant

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 65 : Codage en base 64



5. Si un bloc de 6 bits manque (multiple de 4) on rajoute des =



#### Intérêt et inconvénient de la base 64

C'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 tableau ci-dessus. Mais cette restriction a un inconvénient nous ne pouvons coder que 6 bits ( $2^6 = 64$ ) au lieu de 8. Donc moins efficace de façon générale.

## Exemple pour The Things Network

### EXEMPLE POUR THE THINGS NETWORKS

#### Uplink : du Device LoRa au Network Server

Le Network Server TTN reçoit les trames IP en provenance de la Gateway. Un certain nombre d'information peuvent être retrouvés avec cette trame (DevAddr, SFn Bandwidth ...) mais les données applicatives sont chiffrées avec l'AppSKey. Donc sans l'AppSKey il n'est pas possible de comprendre la totalité du message reçu.

Trame IP

```
{
  "gw_id": "eui-b827ebfffeae26f6",
  "payload": "QNMASyABwAP1obuUHQ=",
  "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 :

On retrouve les valeurs fournies par la gateway :

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

Figure 66 : Trame récupérée sur le « Network Server » de TTN

D'autres information proviennent de l'analyse du Payload. Le Payload inscrit ici est le PHY Payload.

Il y a donc une partie chiffrée (Frame Payload) mais les entêtes sont en clairs.

Ce PHY Payload est QNMASyABwAP1obuUHQ=

Lorsqu'il est exprimé en hexadécimal au lieu de la base 64 il vaut 40D31A01260007000FD686EE5074

Comme le montre le Network Serer de TTN

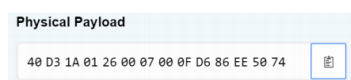


Figure 67 : PHY Payload présenté dans notre Network Server

Sa taille est bien de 14 octets (rn hexa) comme dit la fig66

Reprenons le format de la trame LoRaWAN fig59 nous pouvons retrouver tous les champs de toute la trame :

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

Decodeur de trame lorawan : [LoRaWAN 1.0.x packet decoder \(runkit.sh\)](#)

Uplink : du Network Server à l'Application Server. Pour informations les clés

NwkSKey : E3D90AFBC36AD479552EFEA2CDA937B9

AppSKey : F0BC25E9E554B9646F208E1A8E3C7B24

Le Network server à décoder l'ensemble de la trame. Si le MIC est correct (authentification de 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 chap précédent)

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

D6 est le contenu chiffré. Lorsqu'il est déchiffré avec l'AppSKey on trouve 01/ avec le decodeur en ligne. L'Application Server recevra les données chiffrées seulement si le Device LoRa est enregistré. On peut vérifier que Application Server de TTN nous retourne le payload (Frame Payload) de 01

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

Figure 69 : 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": [
    {
      "gw_id": "eui-b827ebfffeae26f6",
      "timestamp": 2447508531,
      "time": "",
      "channel": 4,
      "rssi": -82,
      "snr": 9,
      "latitude": 45.63647,
      "longitude": 5.8721523,
      "location_source": "registry"
    }
  ]
}
```

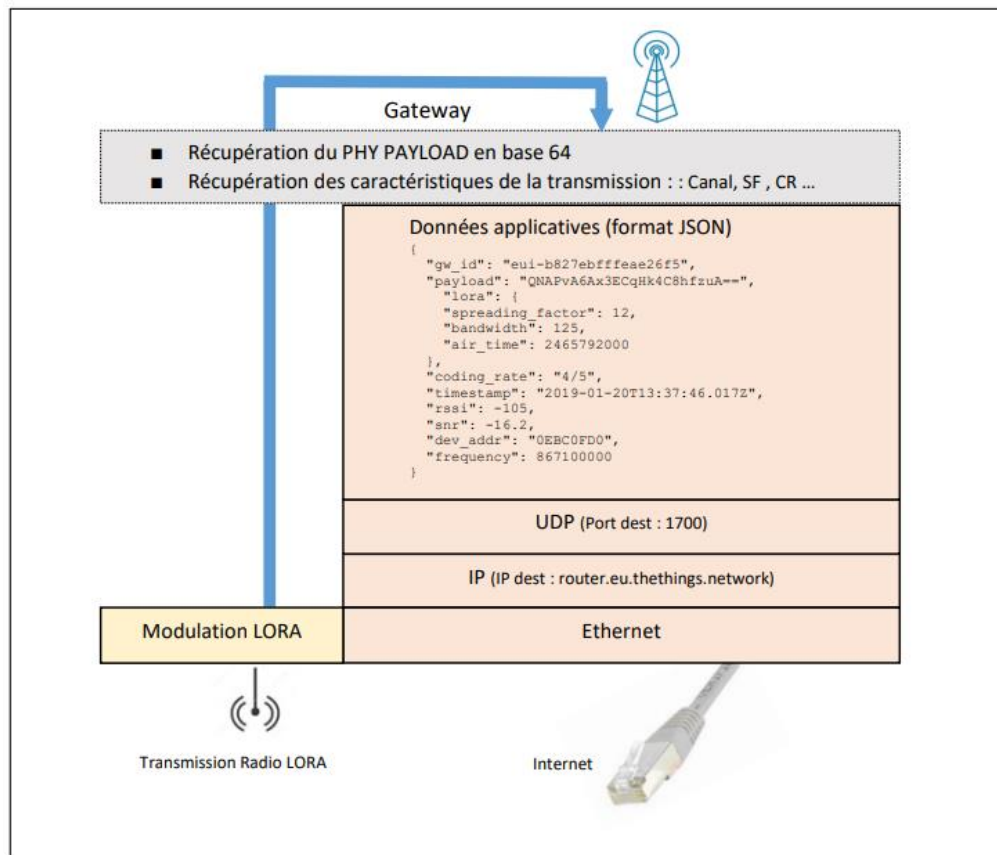
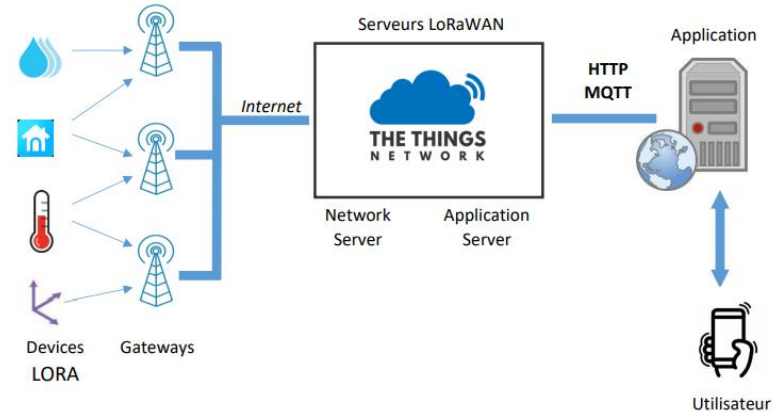


Figure 64 : Passerelle (Gateway) LORAWAN

## Récupération des données et rendus de nos données

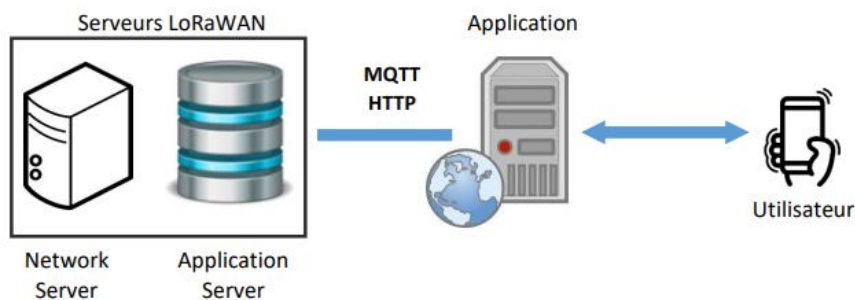
L'application doit :

- . Recevoir des données des serveurs LoRaWAN
- . Transmettre des données à destination des devices LoRaWAN
- . Faire la gestion du serveur lorawan : ajouter/supprimer une application ajouter/supprimer un device LoRa ...



<b>uplink</b> <b>La récupération des données.</b> Le stockage et traitement. La mise en forme (graphiques, tableaux...). L'envoi à l'utilisateur.	<b>Downlink</b> Présenter une interface utilisateur (Bouton, champ texte, ...). Traiter les commandes de l'utilisateur. <b>Envoyer ces commandes aux Serveur LoRaWAN</b>
---	---

Communiquer avec notre application nous avons 2 méthodes le protocole HTTP et MQTT



Récupération des données avec le protocole HTTP (GET)

Le http fait appel à un transfert d'information entre client et serveur. Il faut désigner qui est le client et le serveur.

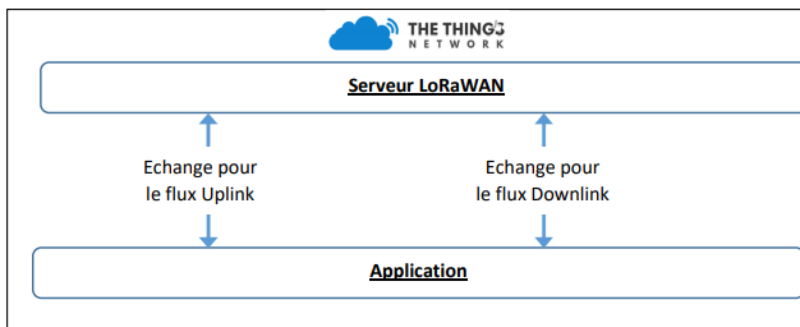


Figure 73 : Communication entre le serveur LoRaWAN et notre Application avec HTTP

Uplink : on cherche à récupérer sur notre application les données du serveur LoRaWAN.

La première idée est de faire une requête depuis notre application à LoRaWAN, pour qu'il fournisse ces données puisque quand on fait une requête http GET à un serveur web il vous retourne le contenu de la page HTML qu'il contient. Ici c'est donc LoRaWAN qui a le rôle de serveur qui retourne les données LoRa et le client notre Application.

Downlink : on cherche à récupérer sur le serveur LoRaWAN les données que l'utilisateur souhaite envoyer au Device LoRa. Donc le serveur LoRaWAN fasse des requêtes http GET à notre application pour qu'elle fournisse ces données lors de la réponse donc dans ce cas le LoRaWAN est le client et notre Application joue le rôle de serveur.

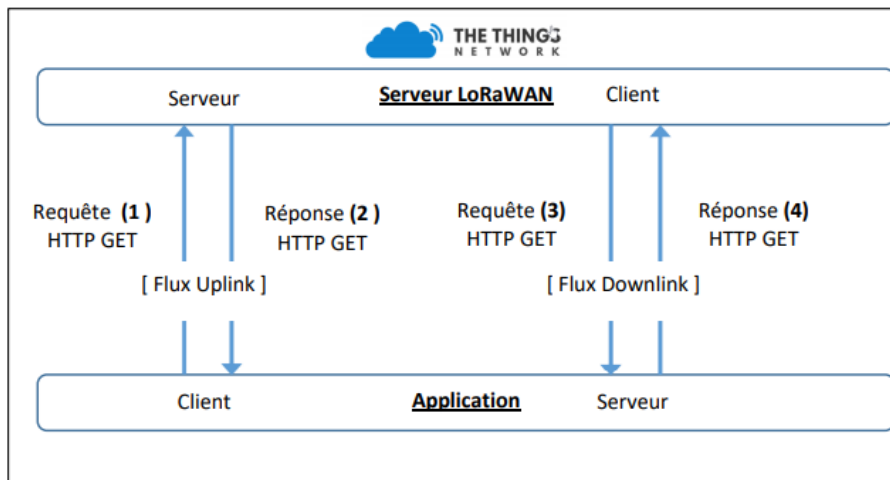


Figure 74 : Flux Uplink et Downlink en HTTP GET

Installation des services http pour le flux Uplink (trame 1 & 2)

Nous devons mettre un serveur http sur nos serveurs LoRaWAN et un client http sur notre Application.

Sur le serveur lorawan la console TTN>Application>Nom de l'Application>Intégration>Data Storage.

Et prendre connaissance de l'API disponible pour récupérer les données.

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

Figure 76 : API REST disponible sur le serveur HTTP

Pour le test utiliser les commandes directement sur la page de la documentation de l'API ou la méthode plus générique avec le logiciel POSTMAN qui permet de générer toutes sortes de requête http il suffit de se référer à la doc pour choisir les bons formats.

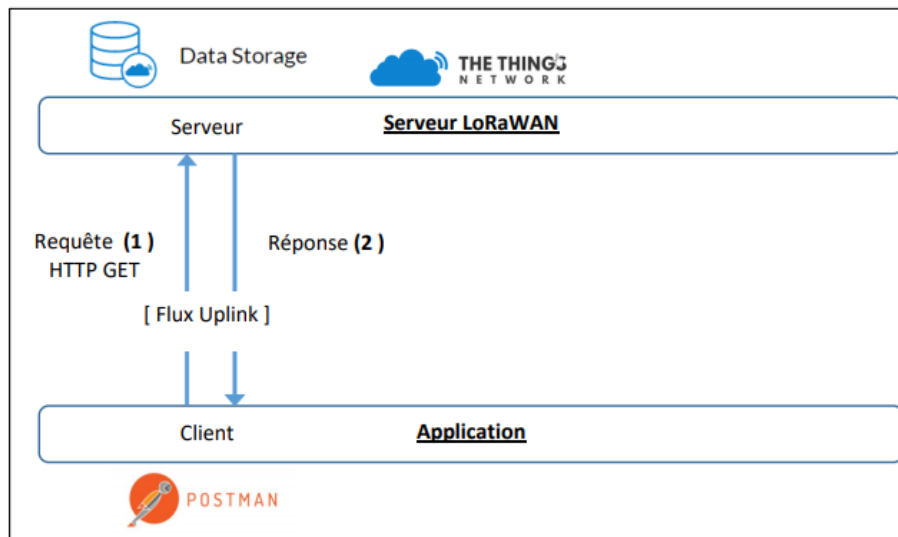


Figure 77 : Flux Uplink et Downlink en HTTP GET

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

- ➔ **POSTMAN > New > Request > Save Request**
- ➔ Créer la requête suivante :

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

- ➔ Envoyer la requête : **Send**

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

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

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

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

#### Inconvénient

C'est la difficulté d'installer un client http générant des requête GET sur TTN ou autre sur le serveur Lorawan en downlink.

Nous passons notre temps à demander des données qui n'existent potentiellement pas. En effet nous faisons des requêtes sans savoir si elles sont vraiment arrivées. Si un capteur émet de façon irrégulière des valeurs nous devons faire des requêtes périodiques avec des réponses vides. Et en downlink la même chose.

On aimerait avoir une réception rapide sur notre Application et donc des demandes très fréquentes au serveur et une charge réseau.

La solution réorganiser les clients, les serveurs et requêtes pour optimiser on peut avec la méthode POST.

Protocole HTTP POST



On va plutôt faire la demande mais que le serveur LoRaWAN fournit elle-même à l'Application pour poster les données POST(1) et la réponse (2) est un simple acquittement et ne contient pas de données. Dont TTN lorawan joue le rôle de client et notre application le rôle de serveur.

Inversement en downlink l'utilisateur qui souhaite transmettre des données à destination du Device doit fournir une requête http POST (3) vers TTN qui joue le rôle de serveur et notre application le rôle de client.

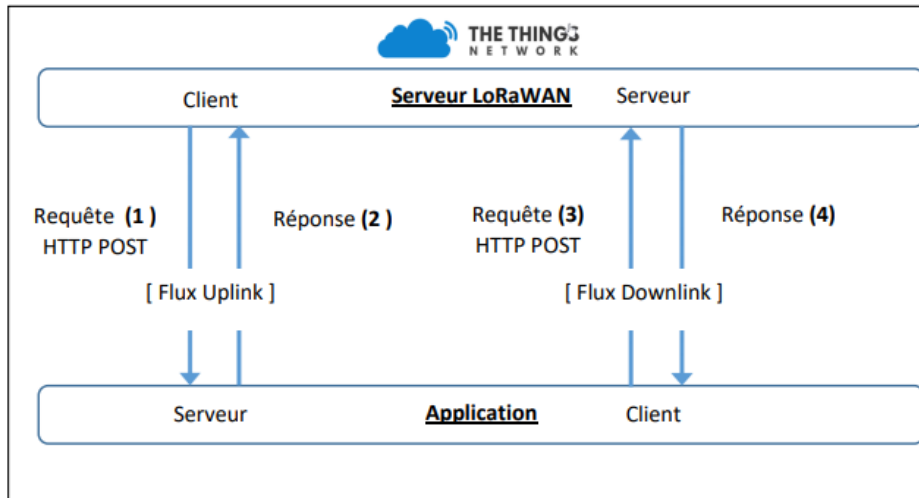


Figure 78 : Flux Uplink et Downlink en HTTP POST

Installation du serveur http POST rbaskets.in

Pour le client TTN a intégrer

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

mon-serveur-http-post

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

default key devices messages

**URL**  
The URL of the endpoint

https://rbaskets.in/9nrvi2s

**Method**  
The HTTP method to use

POST

Figure 79 : Ajout d'un client HTTP POST dans TTN

Pour valider le fonctionnement on peut

Envoyer une trame depuis un Device Lora vers TTN

Ou

Simuler l'envoi d'une trame d'un Device Lora vers TTN (avec l'outil proposé par TTN)

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"
}
```

Installation des services http pour le flux Downlink

Les trames (3) et (4), nous devons mettre en place un client http sur notre Application et un serveur http sur notre serveur LoRaWAN (TTN)

Sur TTN le serveur existe déjà précédemment.

Pour le client POSTMAN et on génère une requête http POST

➡ **POSTMAN > New > Request > Save Request**

➡ Créer la requête suivante :

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

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

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

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

➡ Envoyer la requête : **Send**

Envoyer du texte (payload\_raw en base 64 aGVsbG8= qui correspond à hello. Utiliser les encodeur/décodageur en ligne.

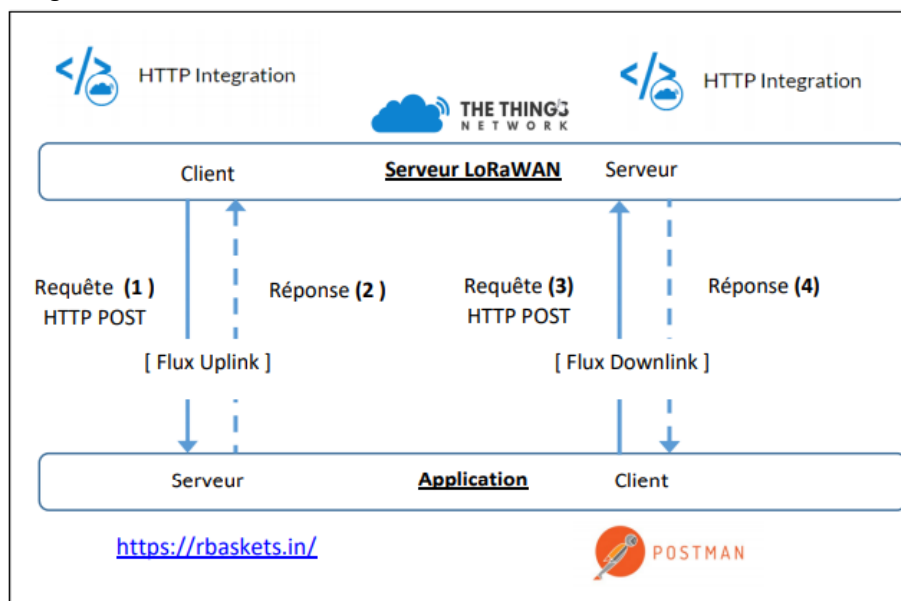


Figure 80 : Flux Uplink et Downlink en HTTP POST

## Récupération des données avec le protocole MQTT (Transport de télémétrie Message Queuing)

Communication par message asynchrone

Envoyer et recevoir les données à travers le réseau Ethernet

On avait les protocoles JMS (java message service) et AMQP

**Evènementiel** (par topic) (publish/subscribe)



MQTT est un protocole léger qui permet de s'abonner à des flux de données. Basé sur Publisher/Subscriber. Donc pas de requête, la donnée sera transmise au Subscriber dès lors que celle-ci a été reçue dans le Broker. (Serveur central)

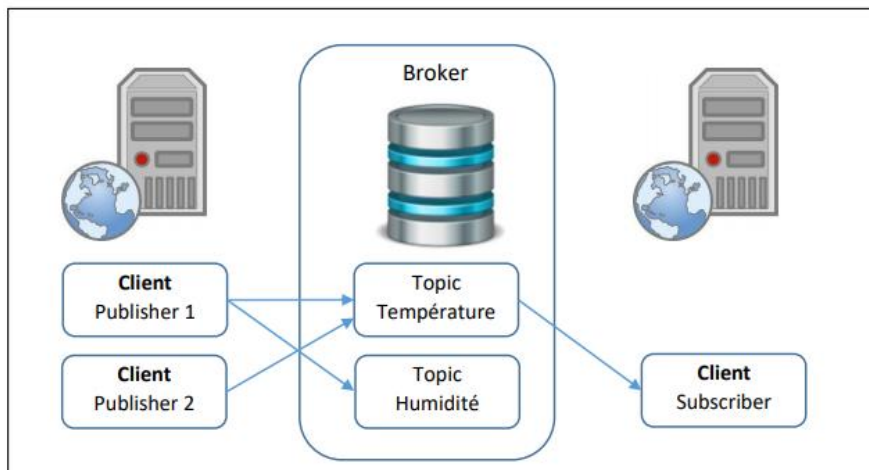


Figure 81 : Modèle Publisher / Subscriber du protocole MQTT

MQTT repose sur TCP l'encapsulation est donc la suivante

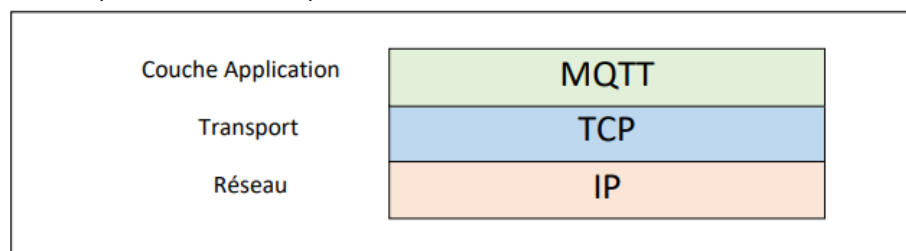


Figure 82 : Protocoles utilisés pour la communication avec MQTT

Vérification 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 83 : 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 subscribers n'ont pas besoin de se connaître et ne sont pas obligés de s'exécuter en même temps.

## Connexion au Broker MQTT

On s'intéresse principalement aux options de connexion qui permettront de gérer la Qualité de Service (QoS).

Pour se connecter un client MQTT envoie 2 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.

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

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

Soit il se déconnecte explicitement (Close connexion)

Soit il n'a rien émis, ni fait signe de vie pendant le temps KeepAlive.

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

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

QoS 1 "At least once" (au moins une fois) : Le deuxième niveau de qualité est "avec acquittement". Le Publisher envoie un message au Broker et attend sa confirmation. De la même façon, le Broker envoie un message à ces Subscribers et attend leur confirmation. Ce mécanisme garantit la réception des messages MQTT. Cependant, si les acquittements n'arrivent pas en temps voulu, ou s'ils se perdent, la réémission du message d'origine peut engendrer une duplication du message. Il peut donc être reçu plusieurs fois.

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

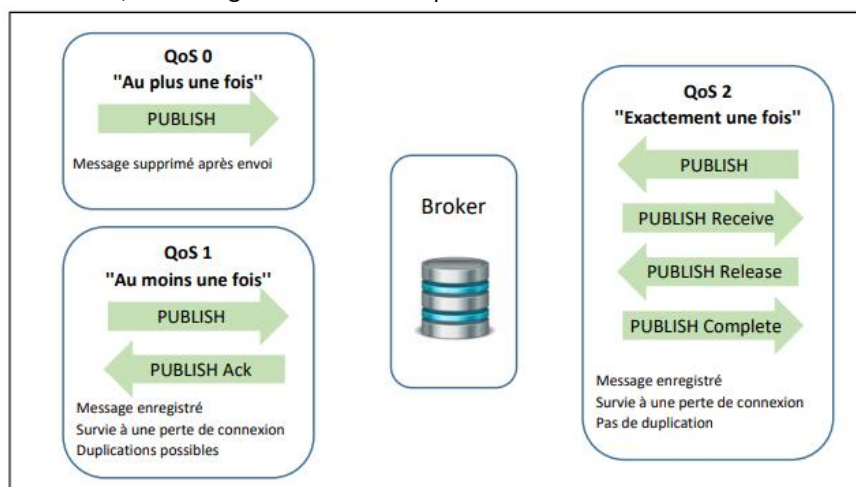


Figure 84 : Qualité de Service en MQTT

## Les trames émises

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

## Qualité de service après une reconnexion

Si les subscribers sont 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 doit être activée à l'ouverture de connexion grâce au flag `cleanSession = 0`. La connexion sera persistante le Broker enregistre tous les messages qu'il n'a pas réussis à diffuser au Subscriber.

Résumé de l'effet du flag `cleanSession` et du QoS

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

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

## Les Topics du protocole MQTT

### Organisation des Topics

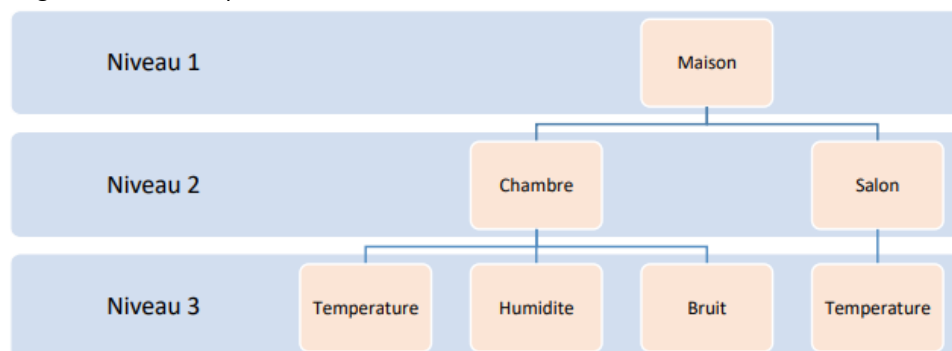


Figure 86 : Exemple de hiérarchie de Topic MQTT

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

Tableau 16 : Exemple de Topic

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

Le `+` remplace n'importe quelles chaînes de caractères sur le niveau où il est placé

Le `#` remplace n'importe quelles chaînes de caractères sur tous les niveaux suivants il est obligatoirement placé à la fin.

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

Tableau 17 : Exemple de Topic

## MISE EN PLACE D'UN BROKER MQTT

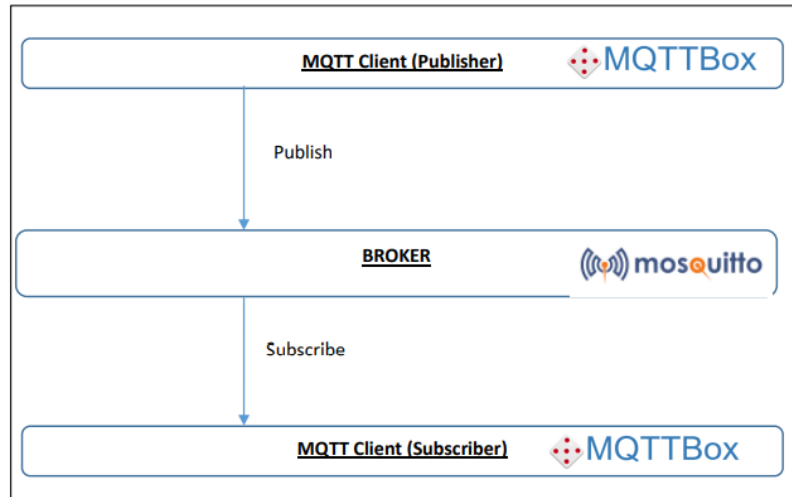


Figure 87 : Test du protocole MQTT

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

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

1. `apt-get install mosquitto` // Installation
2. `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.

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

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 88 : Configuration d'un Client MQTT dans MQTT Box

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

## RECUPERER LES DONNEES EN PROVENANCE DE TTN

En uplink TTN joue le rôle du Broker

Notre Application le rôle de Subscriber

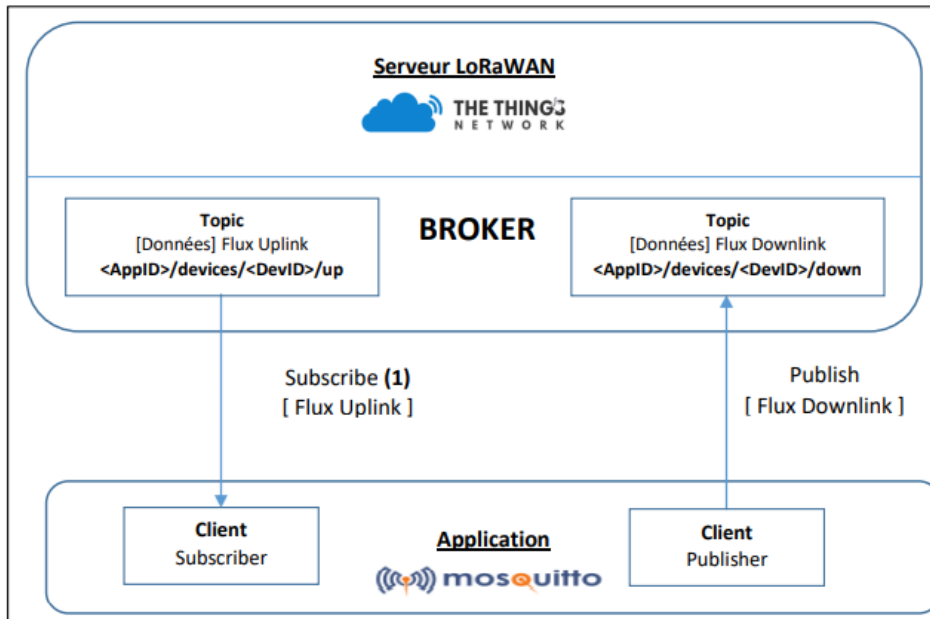


Figure 89 : Mise en place du Broker, des Publishers et des Subscribers

TTN intègre le Broker et le client nous utilisons MQTTBox mais avec les configurations suivantes

Protocol : mqtt/tcp

Host @IP du Broker celui de TTN adresse : eu.thethings.network

Username : connexion vers le Broker MQTT nom de l'application.

Password : nommé Access Key par TTN TTN>Application>Nom\_de\_votre\_application>Overview

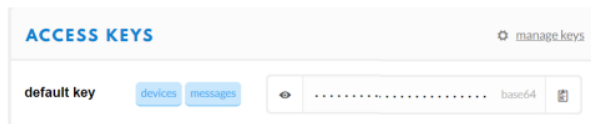


Figure 90 : Access Keys de l'application dans TTN

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

Figure 91 : Configuration du Client dans MQTT Box

Le client peut se connecter au Broker il reste à définir le fait que le client sera Subscriber et il recevra les informations dépendant du Topic auquel nous souscrivons

Topic disponible sur le broker TTN

AppID correspond au nom de votre Application

DevID correspond au nom de votre Device LoRa



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

Tableau 18 : Topics enregistrés dans TTN

Topic : +/devices/+/up on souscrit à tous les Devices LoRa de toutes nos applications pour le flux Uplink

Figure 92 : Configuration du Subscriber

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

Figure 93 : Trame LoRaWAN reçue sur le Subscriber MQTT

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

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

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



## Envoyer des données à destination de TTN

Downlink dans ce cas

L'application est le Publisher

TTN joue le rôle de Broker

Donc la trame (2)

Il reste à configurer le client Subscriber

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

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

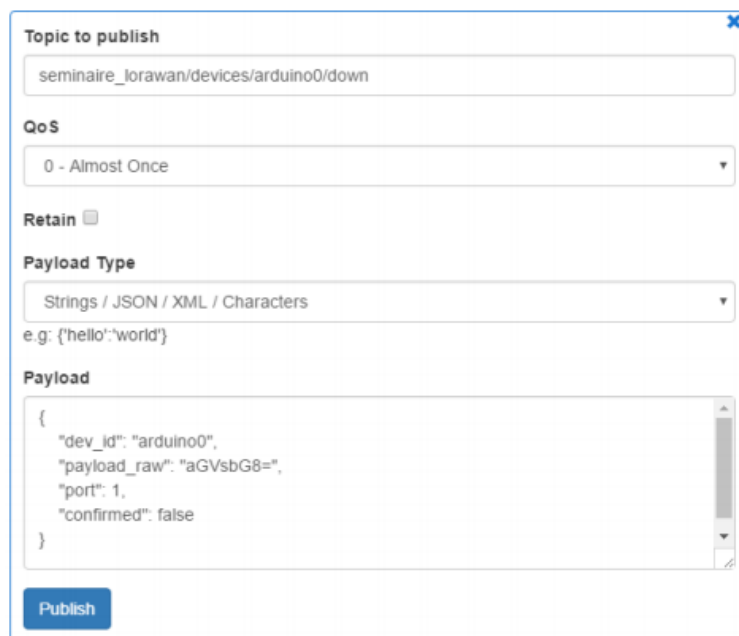


Figure 94 : Configuration du Publisher

Vous devriez voir les données arriver sur votre Device LoRA.

La création de notre propre device Lora il est indispensable de posséder :

Un espace pour stocker le Firmware de l'application utilisateur

Une pile de protocole Lorawan

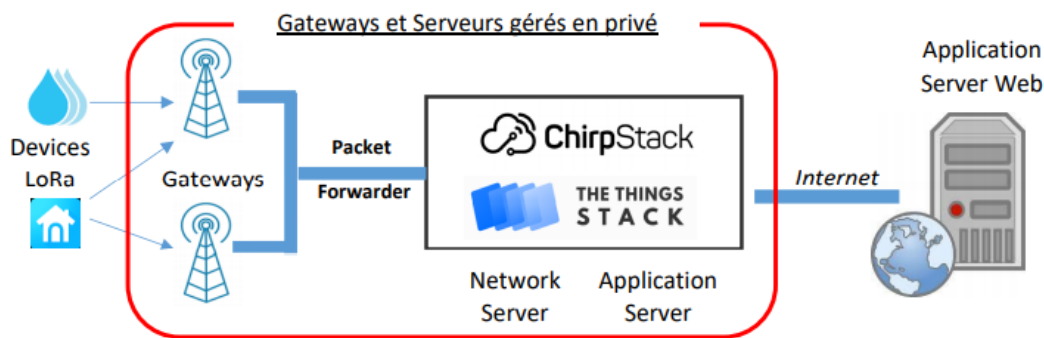
Une interface radion Transceiver LoRa

A partir de là, on peut imaginer plusieurs Architectures

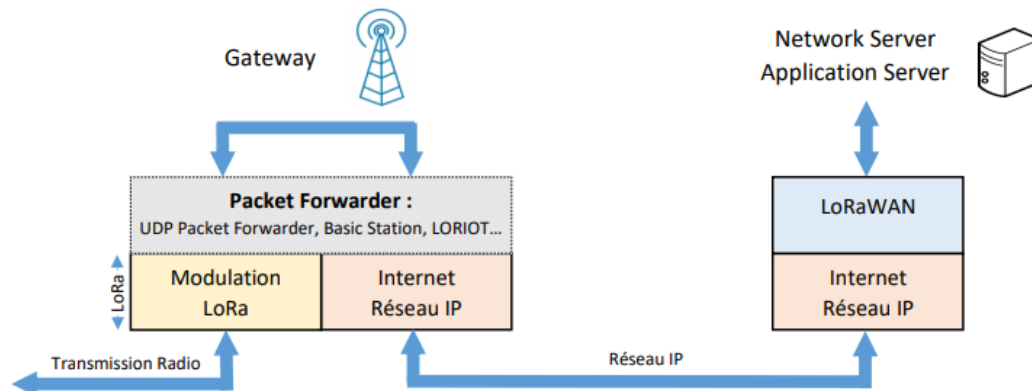
## Créer notre network et Application serveur (réseau interne)

Cela permet de travailler sur un réseau LORAWAN privé.

Installation d'un Network Server (The Things stack server lorawan de the things Network) et d'une Application server (chirpstack.io)



Communication entre Gateways et Serveur LORAWAN



Sur la Gateway un logiciel appelé Packet Forwarder réalise l'interface entre la modulation radio Lora et le réseau IP.

En fonction du Packet Forwarder utilisé le protocole de communication entre la Gateway et le serveur lorawan sera différent.

Il faut donc vérifier la prise en charge du Packet Forwarder sur la Gateway par le serveur LORAWAN.

Il existe plusieurs packet Forwarder disponibles dont 2 développés par SENTECH :

**UDP Packet Forwarder** : c'est le packet le plus utilisé et tous les serveurs lorawan le supportent. Il est disponible dans toutes les gateways, simple, léger mais de nombreux défaut au niveau de la sécurité et du manque de fonctionnalité.

**Basic Station** : il offre des fonctionnalités TLS, mise à jour logiciel gateway, synchronisation du temps, gestion des gateways ...

D'autres comme LORIIOT Gateway Software ou TTN Packet Forwarder.

Mise en place de l'environnement de travail

Pour installer le serveur lorawan une machine connectée à internet (pc, raspberry PI, vmware, serveur ovh)

Pour un raspberry une image appelée chirpstack-gateway-osfull elle est prête et intègre les services gateway bridge, network server et application server.

Dans les autres cas on peut utiliser Docker et Docker compose

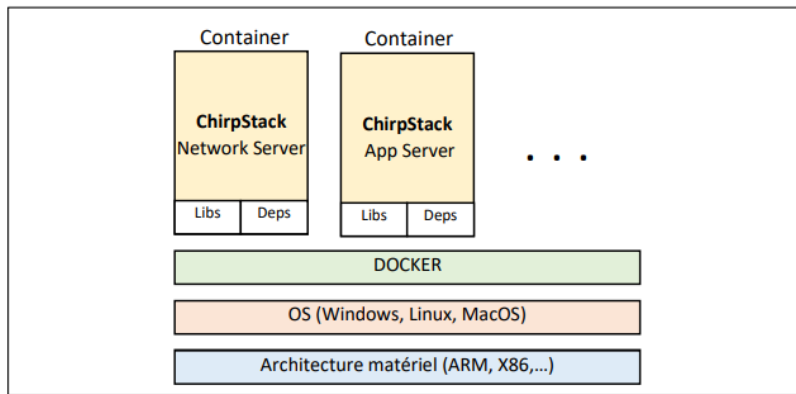
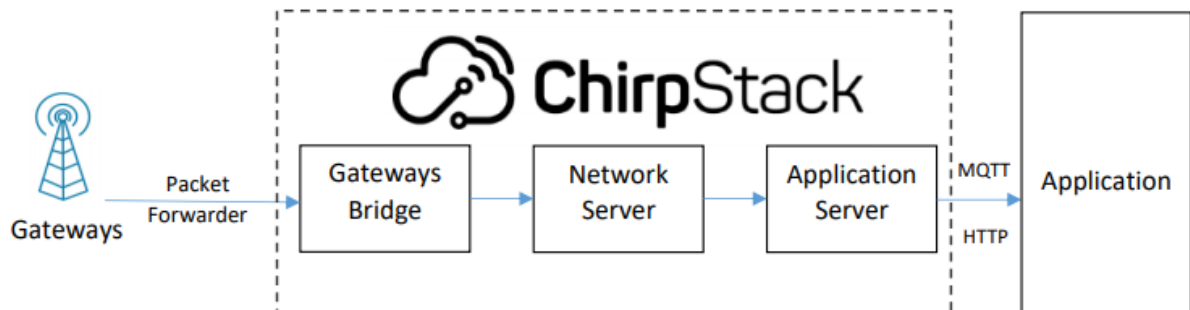


Figure 107 : Utilisation de Docker et Docker Compose pour l'installation des Serveurs LoRaWAN

### Présentation de Chirpstack

Pour une phase de prototypage Chirpstack peut être mis en place dans le même système que la Gateway.

### Architecture de Chirpstack



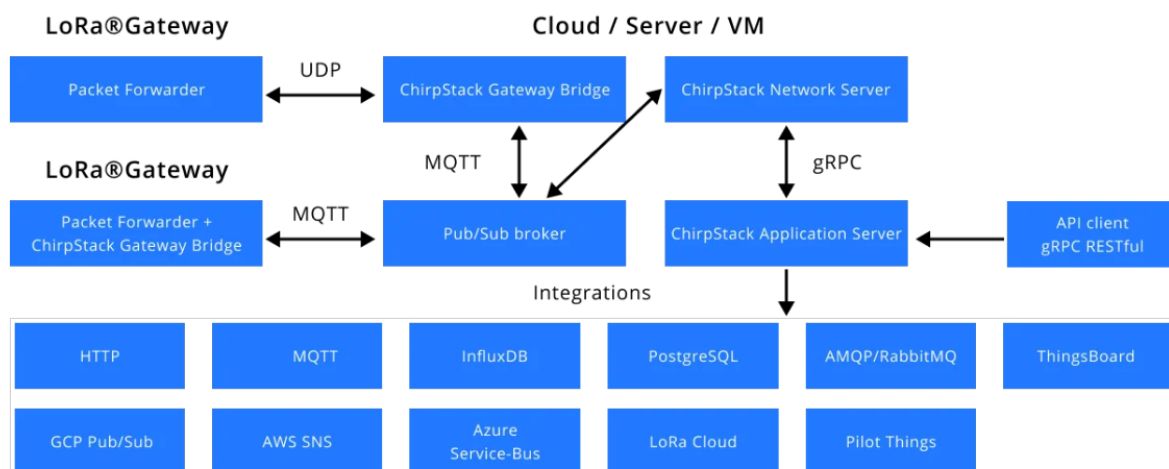
Gateway Bridge : il existe une multitude de protocole permettant la communication entre les Gateways et le Serveur LORAWAN. C'est une service qui permet de convertir les différents formats des protocoles Packet Forwarder en un format commun par le Network server de chirpstack. Il peut s'interfacer avec le UDP Packet Forwarder ou Basic station.

Network-Server, Application-Server : partie réseau et décodage des trames, chiffrement, authentification.

### Installation de Chirpstack

[GitHub - brocaar/chirpstack-docker: Setup ChirpStack using Docker Compose](https://github.com/brocaar/chirpstack-docker)

Dans le dossier /chirpstack-docker, lancer la commande : docker-compose up



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



Figure 109 : Containers Docker des 3 services principaux de ChirpStack sous Windows

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

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

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

Figure 110 : Containers Docker des 3 services principaux de ChirpStack sous Linux

Nous pouvons remarquer que deux containers écoutent sur des ports :

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

Application server écoute sur le port 8080/tcp c'est l'accès vers l'interface web pour la configuration de notre serveur lorawan

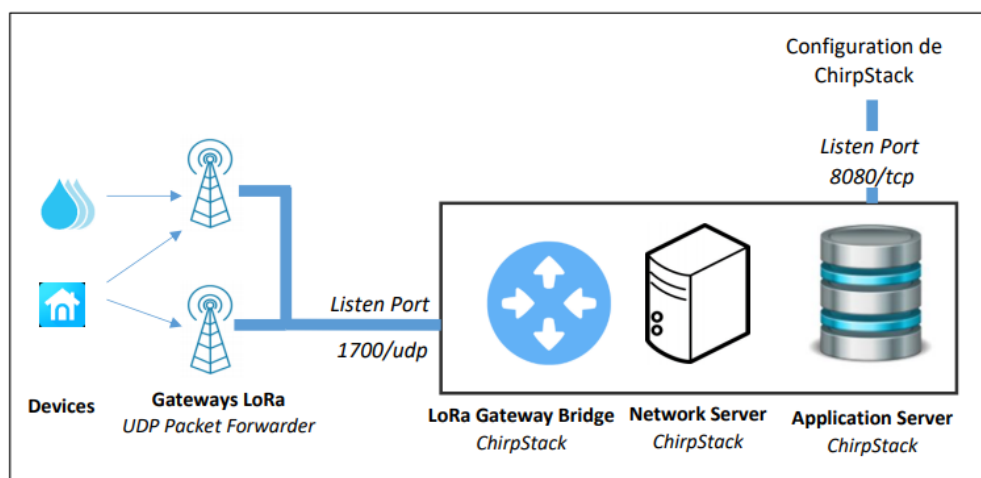


Figure 111 : Architecture globale après installation de ChirpStack

Configuration :

- <http://localhost:8080> si vous travaillez en local.
- [http://@IP\\_ChirpStack:8080](http://@IP_ChirpStack:8080) si vous travaillez à distance.

Les Usernames et Password par défaut sont :

- Username: admin
- Password: admin

#### 9.4.1 Création d'une instance du Network Server

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

La configuration est alors la suivante :

- **Network-server name** : Le nom que vous souhaitez donner à votre instance.
- **Network-server server** : L'adresse IP et le port sur lequel il va écouter. Dans Docker, l'adresse IP est représentée par le nom du container "Network Server" qui a été créé. Sur la Figure 109 et la Figure 110, cela est représenté par le nom "chirpstack-docker\_chirpstack-network-server\_1". Nous aurons donc la configuration suivante à réaliser.

*Figure 112 : Enregistrement d'un nouveau Network Server*

Enregistrement d'une Application (sur l'application server)

Il faut tout d'abord faire un enregistrement d'un "service-profile". Un "service profile" permet de faire la connexion entre le Network Server et une organisation. Service-profiles > Create.

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

On laissera par défaut toutes les autres options.

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

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

Enregistrement des Devices Lora

Il faut tout d'abord faire l'enregistrement d'un Device-profile. pour les Devices LoRa que vous souhaitez connecter.

Dans notre cas, nous ferons un test en spécifiant que nous utiliserons seulement le mode d'authentification OTAA (voir chapitre 4.4.2).: Devices profile > Create

Onglet GENERAL :

- Device-profile name : myDeviceProfile
- LoRaWAN MAC version : Si vous ne connaissez pas votre version de stack LoRa, mettez 1.0.0. Les versions étant rétro-compatibles, vous aurez toutes les chances que cela fonctionne.
- Max EIRP : Si vous ne connaissez pas la valeur, mettre 0
- Uplink interval : Période estimée d'envoi de données. Cela a peu d'intérêt dans notre test.

Onglet JOIN (OTAA/ABP) :

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

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

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

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

- Application key : Rentrer votre AppKey.

Enregistrement d'une Gateway :

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

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

Modification de la configuration de notre Gateway

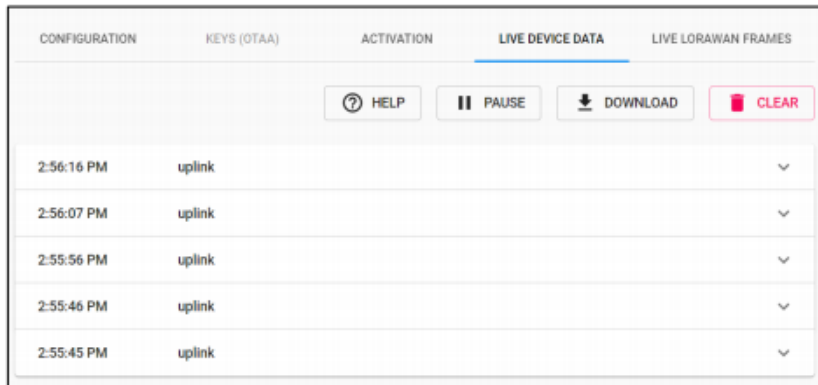
Notre Gateway doit maintenant transmettre les données LoRa à notre serveur ChirpStack que nous venons d'installer. Pour cela, il faudra refaire la configuration de la Gateway comme nous l'avons vu au chapitre 5.2.2. Les informations suivantes devront être renseignées :

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

## Visualisation des trames reçues

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

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



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	▼

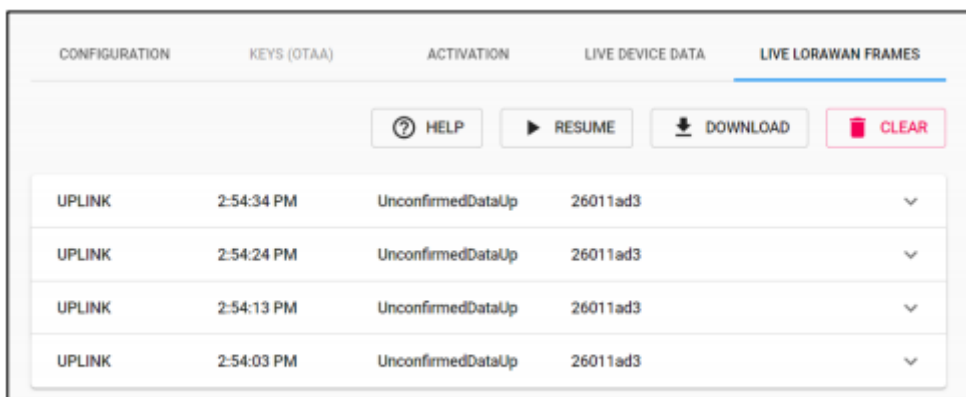
Figure 113 : Réception des trames des Devices LoRa



2:55:46 PM	uplink	▲
<pre>adr: false applicationID: "3" applicationName: "myApplication" data: "SGVsbG8=" devEUI: "0056ab225b48377" deviceName: "arduino0" fCnt: 10262 fPort: 15 ▼ binfo: {} 2 keys   dr: 5   frequency: 868100000</pre>		

Figure 114 : Analyse des trames des Devices LoRa

Dans l'onglet Application > myApplication > Devices > myDevice > LORAWAN FRAMES, nous pouvons voir les trames LORAWAN, et si besoin, étudier le contenu de ce qu'elles contiennent. En revanche, les données applicatives sont ici chiffrées.



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 115 : Réception des Trames LoRaWAN

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

Récupérer des données avec le protocole HTTP POST

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

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

Récupérer des données avec le protocole MQTT

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

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

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

Tableau 20 : Topics enregistrés dans le Broker de ChirpStack



## Analyse du "Packet Forwarder"

### Présentation de UDP Packet Forwarder (SEMTECH)

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

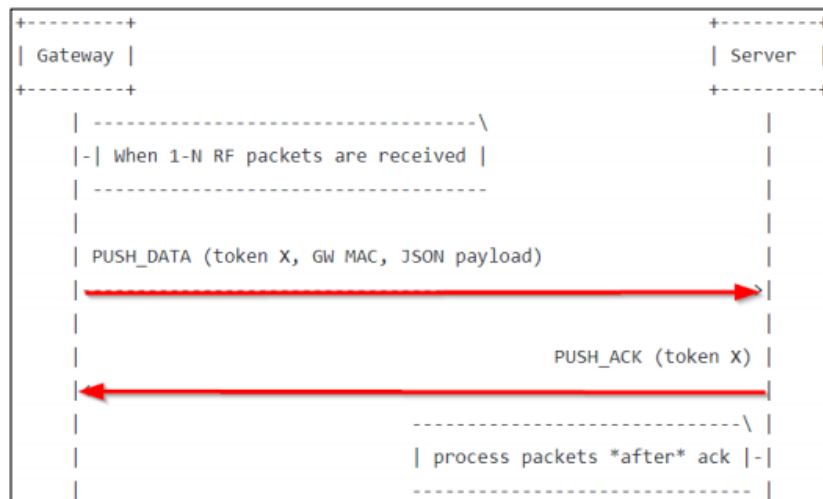


Figure 116 : Protocole Uplink (PUSH\_DATA du Packet Forwarder)

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 :

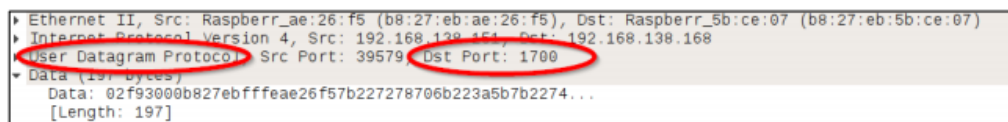


Figure 117 : Trame capturée par Wireshark lors d'un Uplink

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

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

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

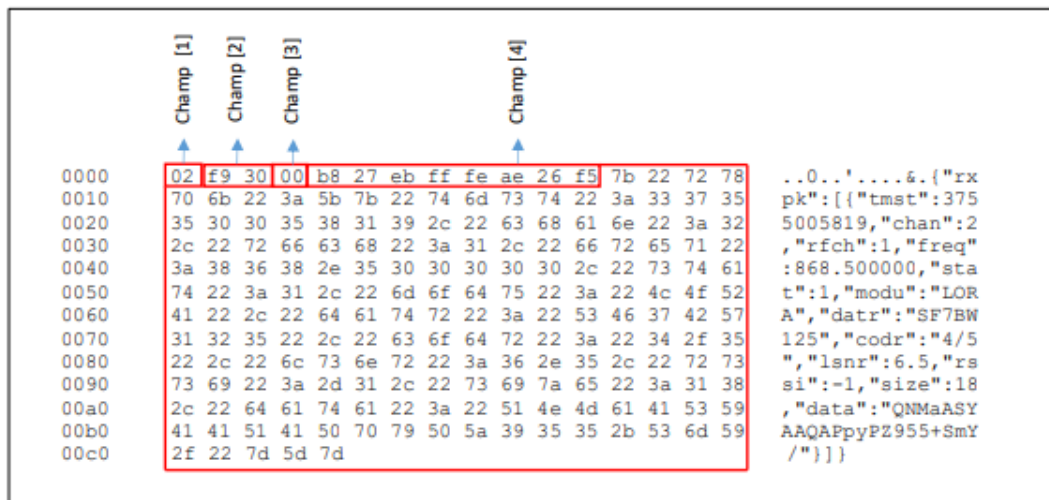


Figure 118 : Analyse du champ Données du protocole « Packet Forwarder »

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 :

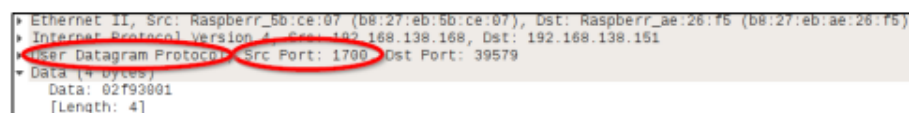


Figure 119 : Trame capturée par Wireshark lors de l'acquittement d'un Uplink

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.

## Création de l'application avec Node-RED

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

- De gérer la récupération des données.
- De stocker et traiter les données.
- De les mettre en forme par un système de monitoring (graphiques, tableaux, ...).
- De les mettre à disposition de l'utilisateur avec un Serveur Web. Dans le sens Downlink :
- Présenter un interface utilisateur (Bouton, champ texte, ...).
- Traiter les commandes de l'utilisateur.
- Envoyer ces commandes à l'Application Server de TTN.

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

vous retrouverez une structure plus robuste organisée autour des services : Telegraf, InfluxDB, Chronograf et Grafana.

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

Mais aussi en cloud IOT AWS ou Microsoft Azure
















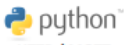
Service à rendre	Solutions génériques	Choix technologiques possibles			
Service web : Mise à disposition de contenu WEB	Serveur WEB Interface utilisateur		 Grafana	 kibana	 
Gestion de l'affichage des données : Courbes, histogrammes, tableaux, jauges, etc...	Solution de monitoring complète, librairies pour Dashboard...		 chronograf		  HIGHCHARTS
Sauvegarde des données	Base de données Fichier texte		 influxdb	 elasticsearch	  
Récupération des données	Endpoint HTTP Subscriber ou Publisher MQTT		 telegraf	 logstash	 python HTTP / MQTT

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

Mise en place de l'environnement de travail

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

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

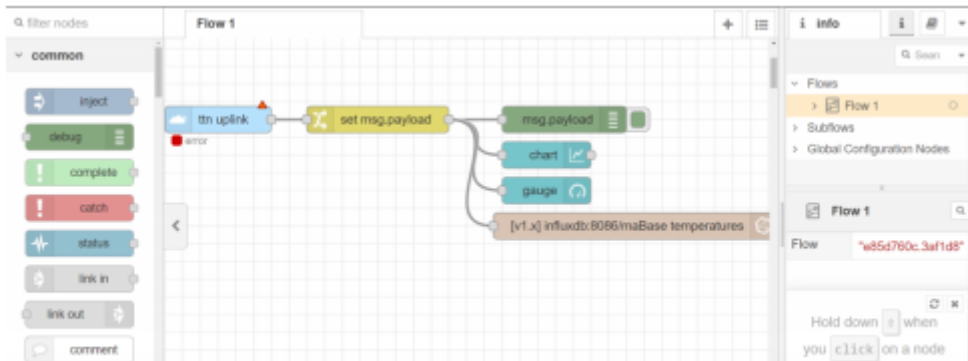


Figure 120 : Page d'accueil de Node-RED

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

Décompresser l'archive : `unzip master.zip`

Placer vous dans le répertoire nodered : `cd nodered`

Lancer le service Node-RED : `docker-compose up -d`

C'est la version de nodeRed avec les librairie the things network, influxdb, dashboard.

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

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

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

Réaliser l'application nodeRed

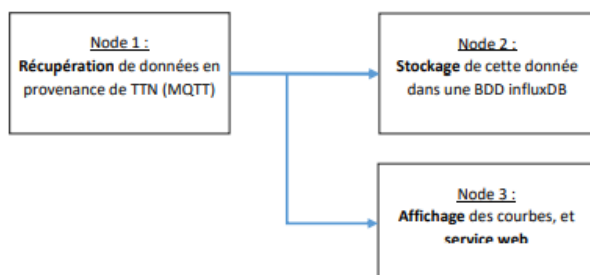


Figure 121 : Les briques du flow node-RED à réaliser

## Récupération des données avec le node TTN

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

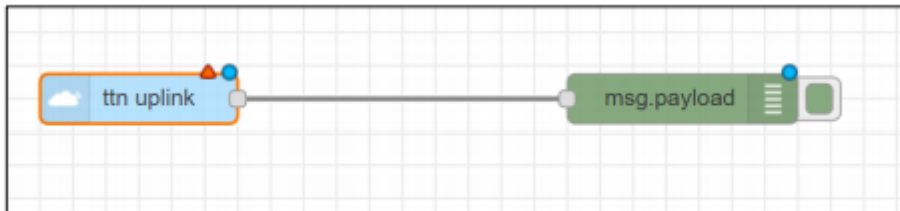


Figure 122 : Utilisation du Node TTN Uplink

Double cliquer sur le Node "ttn uplink" pour le configurer comme sur la Figure 124 : 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.

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

Figure 123 : Configuration du node "ttn uplink"

Configurer

Cette image montre la fenêtre de configuration de l'application TTN. Elle contient les champs suivants : "App ID" avec la valeur "seminaire\_lorawan", "Access Key" avec des points de suspension, et "Discovery address" avec la valeur "discovery.thethingsnetwork.org:1900". Des boutons "Cancel" et "Add" sont situés en haut à droite.

Figure 124 : Identifiant et mot de passe de notre application

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

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

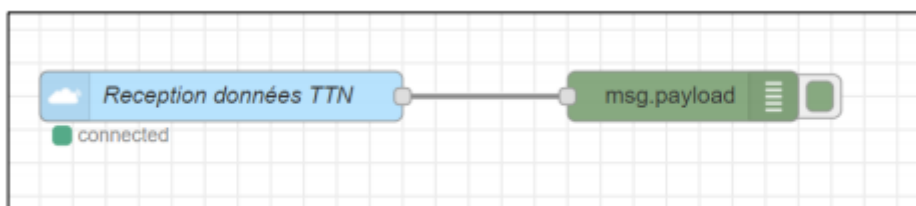


Figure 125 : Visualisation des trames publiées par TTN sur notre client MQTT

#### Modification du format du payload

Pour écrire dans la BDD ou envoyé sur le dashboard il est nécessaire d'avoir un msg.payload représentant un nombre alors que le msg.payload reçu par TTN est un buffer(ou autre si vous utiliser le Decoder de TTN) la fig127 montre le format du msg.payload fourni par le code TTN uplink et le format msg.payload que nous souhaitons avoir



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

<pre> ▼ payload_raw: buffer[1] raw   0: 0x3 ▶ metadata: object ▼ payload: buffer[1] raw   0: 0x3         </pre>	<pre> ▼ payload_raw: buffer[1] raw   0: 0x3 ▶ metadata: object   payload: 3         </pre>
---	--

Figure 126 : Format du msg.payload avant (gauche) et après transformation (droite)

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

<pre> ▼ payload_raw: buffer[1] raw   0: 0x3 ▶ metadata: object ▼ payload: buffer[1] raw   0: 0x3         </pre>	<pre> ▼ payload_raw: buffer[1] raw   0: 0x3 ▶ metadata: object   payload: 3         </pre>
---	--

Figure 127 : Transformation du format de msg.payload

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





Figure 128 : Modification du msg.payload par un node "change"

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



### 10.3.4 Création d'un Dashboard

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



Figure 129 : Implémentation d'une interface graphique dans Node RED

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

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

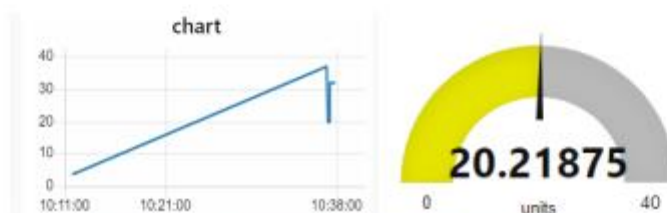


Figure 130 : Interface graphique du site web avec Node-RED

Stockage des informations dans une base de données

Nous utiliserons la base de données InfluxDB. Une librairie de node-RED fournit un node permettant de s'interfacer très facilement à InfluxDB. Nous utiliserons principalement le node "influxdb out" pour écrire des données dans la BDD. La documentation des nodes InfluxDB est fournie par le lien suivant : [node-red-contrib-influxdb \(node\) - Node-RED \(nodered.org\)](https://nodered.org/docs/contrib/influxdb)

Elle montre bien que si le msg.payload est un nombre, alors sa valeur sera enregistrée. Les écritures dans la BDD se font grâce au protocole LINE dont la syntaxe est très bien expliquée ici

[InfluxDB line protocol tutorial | InfluxDB OSS 1.8 Documentation \(influxdata.com\)](https://docs.influxdata.com/influxdb/v1.8/write_protocols/line_protocol_tutorial/)

Dans notre cas, nous définirons un "measurement" appelé "temperatures" et les champs (fields) seront appelés "values". La BDD étant déjà installée d'après la mise en place de l'environnement de

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

docker exec -it influxdb /bin/bash

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

- Se connecter à la BDD influxDB [ influx ]
- Voir les bases présentes [ SHOW DATABASES ]
- Utiliser une base de donnée existante [ USE maBase ]
- Enregistrer une valeur dans un "measurement" appelée "temperatures" [ INSERT ]
- Voir les "measurements" présents [ SHOW MEASUREMENTS ]
- Lister les valeurs présentes dans le "measurement" temperatures [ SELECT ]

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

Nous allons maintenant placer les données reçues par TTN dans notre BDD. Il suffit de les donner à un "node influxdb out" comme le montre la Figure 132.

La configuration du node est la suivante :

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

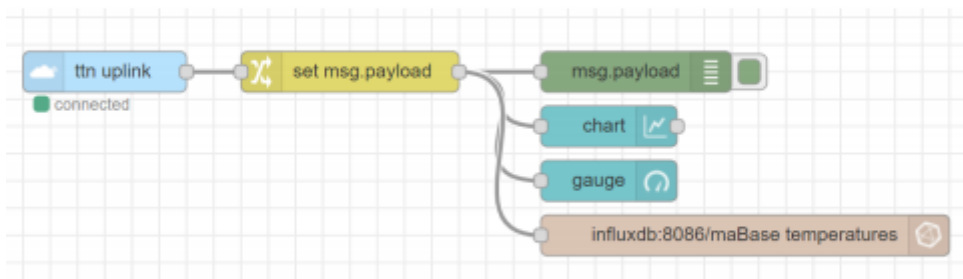


Figure 131 : Enregistrement des températures dans la BDD

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



## Architecture Microcontrôleur + Transceiver

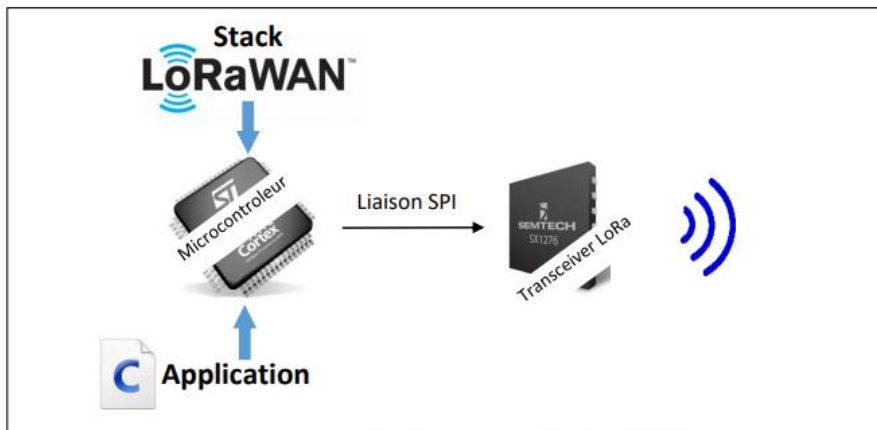


Figure 95 : Device LoRa avec microcontrôleur + Transceiver

Le microcontrôleur gère à la fois la pile LoRaWAN et le Firmware de l'application. Cela nécessite d'avoir une pile de protocole LoRaWAN (stack loRaWAN) à disposition.

Le SX1272 (1279 ou 1262) sont des transceiver Lora de chez SEMTECH.

Ils gèrent la partie physique du protocole : Modulation, détection de préambule ...

La gestion complète du protocole LoRaWAN est réalisée par une pile logicielle qui est implémenté dans le microcontrôleur/

Les différentes stack loRaWAN sont présentée plus bas. Les transceivers sont pilotés en SPI

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

Carte Nucleo loRaWAN STM32L073 (cortex M0+) avec un transceiver SX1272



Figure 96 : P-NUCLEO-LRWAN1 package

Après le prototypage on peut passer sur les cartes P-NUCLEO et réaliser son PCB mais le plus facile utiliser les cartes modules breakout qui intègre en plus déjà des composants

## Module standalone

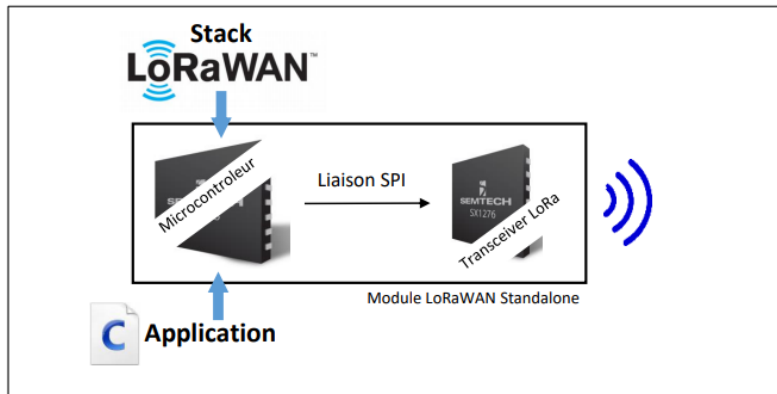


Figure 98 : Module LoRaWAN Standalone

Tout intégré plus simple mais on a toujours la proximité du firmware applicatif et la stack lorawan à gérer.  
On a aussi du ADC I2C UART GPIO pour mettre en œuvre l'application du Device Lora.

## Glossaires

<b>ABP</b>	<b>Activation By Personalization</b>	
<b>ADR</b>	<b>Adaptive Data Rate</b>	
<b>AppEUI</b>	<b>Application Extended Unique Identifier</b>	
<b>AppKey</b>	<b>Application Key</b>	
<b>AppSKey</b>	<b>Application Session Key</b>	
<b>BDD</b>	<b>Base de Données</b>	
<b>BW</b>	<b>BandWidth</b>	
<b>CDMA</b>	<b>Code Division Multiple Access</b>	
<b>CHIRP</b>	<b>Compressed High Intensity Radar Pulse</b>	
<b>CR</b>	<b>Coding Rate</b>	
<b>CRC</b>	<b>Check Redundancy Cycle</b>	
<b>DevAddr</b>	<b>Device Address</b>	
<b>DevEUI</b>	<b>Device Extended Unique Identifier</b>	
<b>FDM</b>	<b>Frequency Division Multiplexing</b>	
<b>HTTP</b>	<b>HyperText Transfer Protocol</b>	
<b>IoT</b>	<b>Internet Of Things</b>	
<b>JSON</b>	<b>JavaScript Object Notation</b>	
<b>JoinEUI</b>	<b>Join Extended Unique Identifier</b>	
<b>LoRa</b>	<b>Long Range</b>	
<b>LoRaWAN</b>	<b>Long Rang Wide Area Network</b>	
<b>LPWAN</b>	<b>Low Power Wide Area Network.</b>	
<b>LTE-M</b>	<b>Long Term Evolution Cat M1</b>	
<b>MIC</b>	<b>Message Integrity Control</b>	
<b>MQTT</b>	<b>Message Queuing Telemetry Transport</b>	
<b>NB-IoT</b>	<b>NarrowBand Internet of Things</b>	
<b>NwkSKey</b>	<b>Network Session Key</b>	
<b>OTAA</b>	<b>Over The Air Activation</b>	
<b>QoS</b>	<b>Quality of Service</b>	
<b>RSSI</b>	<b>Received Signal Strength Indication.</b>	
<b>SDR</b>	<b>Software Digital Radio</b>	
<b>SF</b>	<b>Spreading Factor</b>	
<b>SNR</b>	<b>Signal Over Noise Ratio</b>	
<b>TDM</b>	<b>Time Division Multiplexing</b>	
<b>TOA</b>	<b>Time One Air</b>	
<b>TTN</b>	<b>The Things Network</b>	

