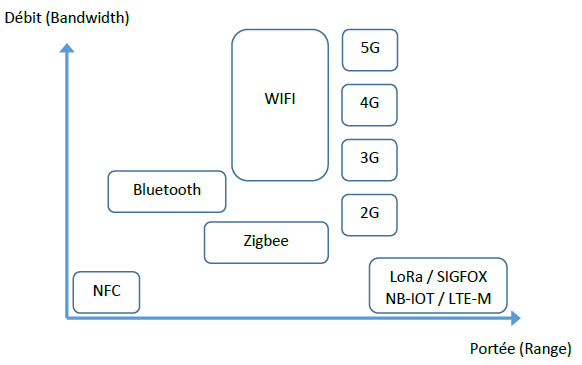
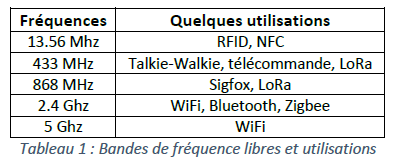
Discussion 1 Comment fonctionne l’architecture Lorawan

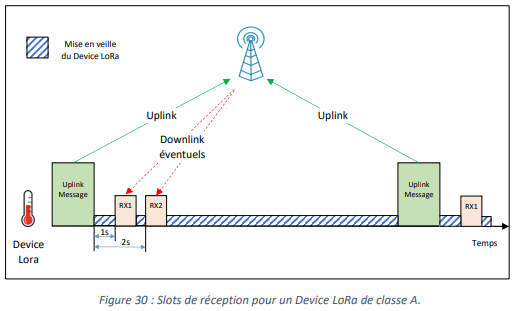
## Réseaux sans fil

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



## Lora/Lorawan

Classes des Devices LoRaWan

**Classe A** : Minimal power Application. (AII) **(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.

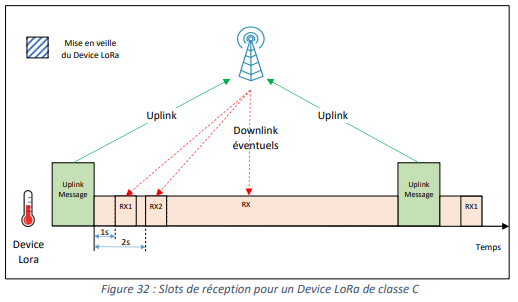
La gateway peut alors transmettre pendant le RX Slot1 et RX Slot 2 mais pas les deux.

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)

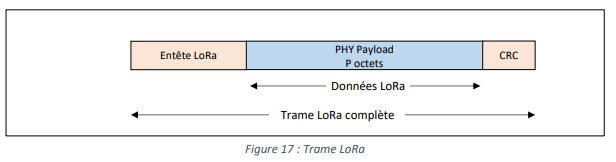
Pour le Downlink : La Gateway utilisée pour le Downlink est celle qui a reçu le dernier message du Device LoRa. Un message ne pourra jamais arriver à destination d’un Device LoRa si celui-ci n’a jamais transmis, quel que soit sa classe A, B ou C.

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.**

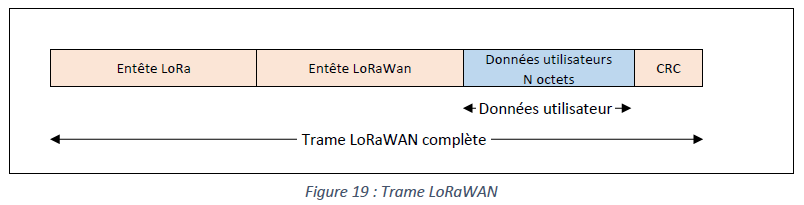
Différence entre les trames

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



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

On parle de l’ensemble de la chaine de communication alors nous parlons de protocole 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 aire de 46.3ms le device doit émettre pendant 99\*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 **N**etwork **S**ession **K**ey **: 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

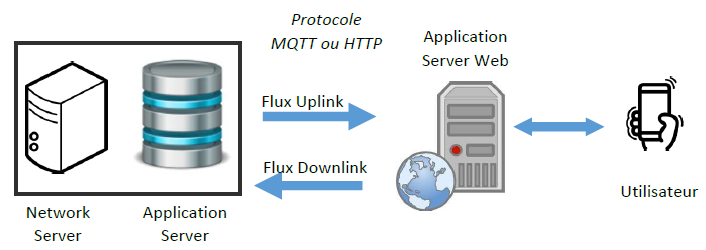
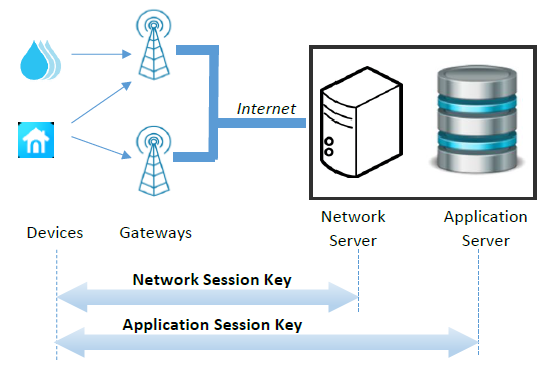
Grâce à une clé AES 128 bits appelée Application Session Key : **AppSKey**. Contrairement au **NwkSKey (N**etwork **S**ession **K**ey), il s'agit bien ici d'un chiffrement

**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 :

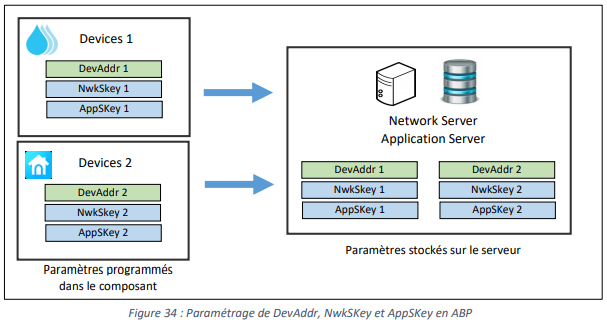
Activation By Personalization (ABP) ou Over The Air Activation (OTAA)

**ABP (la plus simple des méthodes)**

Le device possède déjà DevAddr, AppSKey, NwSKey

Le Network Server et application serveur possède déjà DevAddr, NwkSKey, 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



**OTAA (celui choisi standard)**

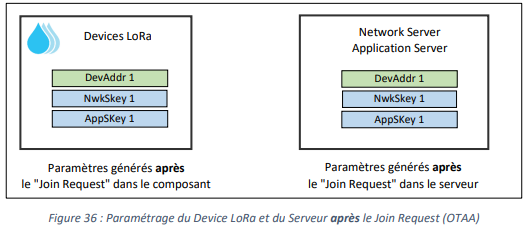
Le DevAddr, AppSKey, NwSKey vont être générés lors d’une procédure de négociation (join Request) dès lors que le device se connectera pour la première fois au serveur.

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

Le network server doit lui connaitre 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.



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.

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.

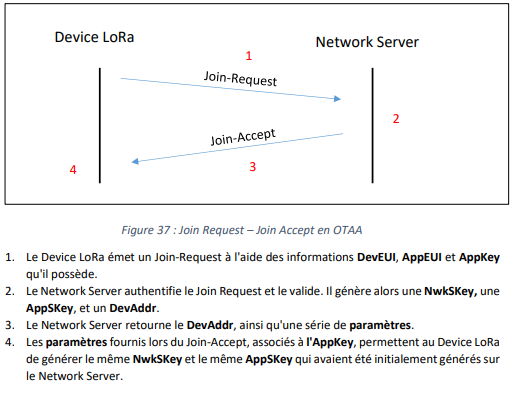
AppKey : AES 128 clé utilisée pour générer le MIC (Message Code Integrity) lors de la join Request il est partagé avec le Network server.

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.



Sécurité ABP ou OTAA sont équivalentes.

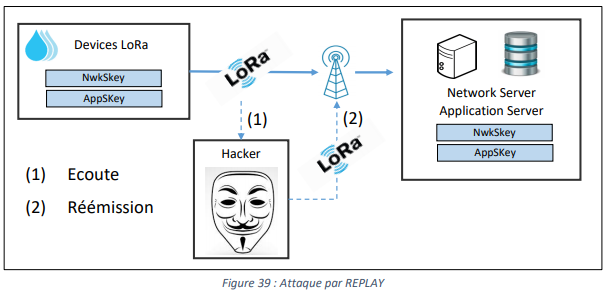
Le point faible sont les clés stockées en permanence dans le Device Lora.

APB (NwSKey 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



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 hasar,d 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

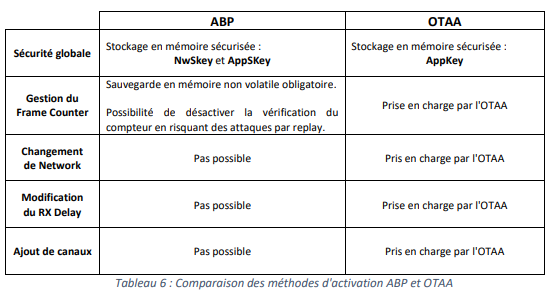
1. En effet, à chaque ouverture de session en OTAA, le Frame Counter est réinitialisé coté serveur
2. Utiliser l’activation par OTAA au lieu de ABP. En effet, à chaque ouverture de session en OTAA, le Frame Counter est réinitialisé coté serveur
3. Conserver la valeur du Frame Counter dans une mémoire 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 / NwSKey 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.

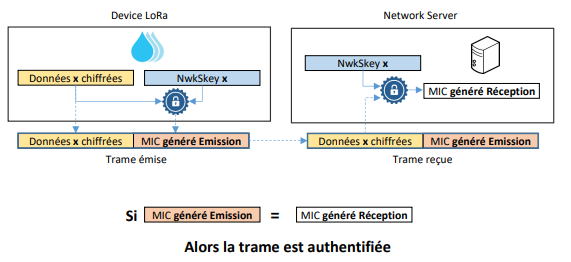
Pour ABP reprogrammer le firmware du device.



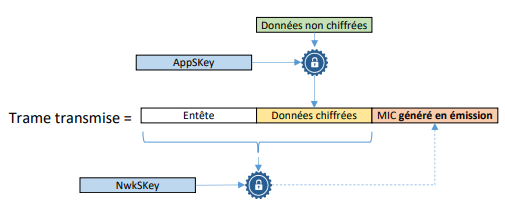
Authentification avec le Network Server

Le NwkSKey (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 NwkSkey 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.

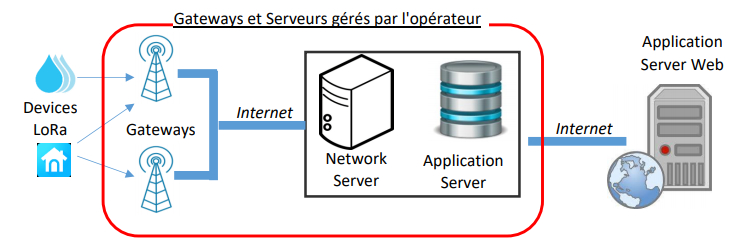


Ce qui donne le chiffrement par authentification



Les réseaux et serveurs LoRaWAN

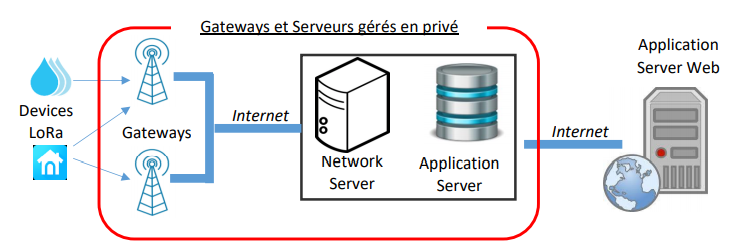
Les réseaux Lorawan opérés



Avec les abonnement s 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

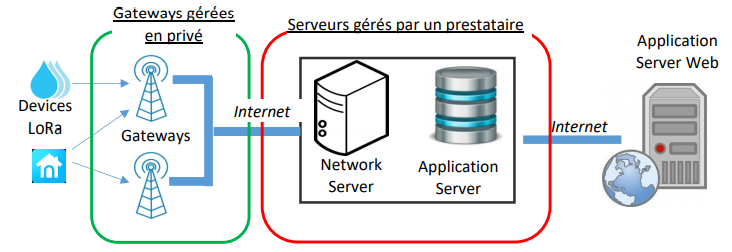
Les réseaux Lorawan privés **(celui choisi standard avec VPN)**



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) oopen source chirpstack et thethingsstack

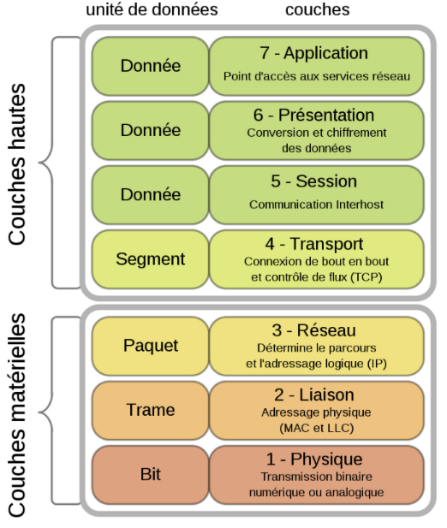
Alternative Le réseau Lorawan dédié (hybride)

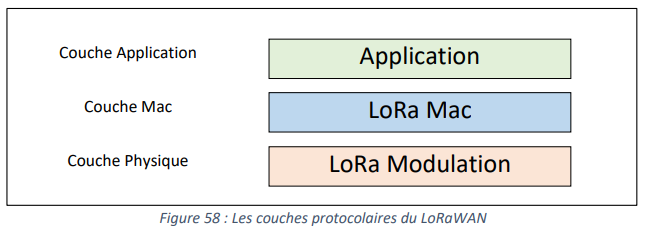




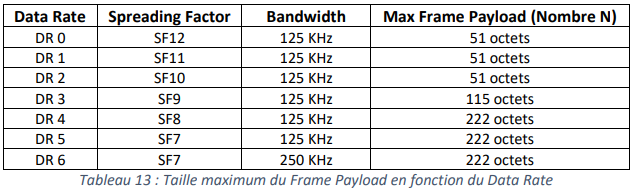
Les zones de couverture Pour connaitre la zone de couverture de notre Gateway utilisée avec TTN, nous pouvons utiliser l’application TTN Mapper : https://ttnmapper.org/ .

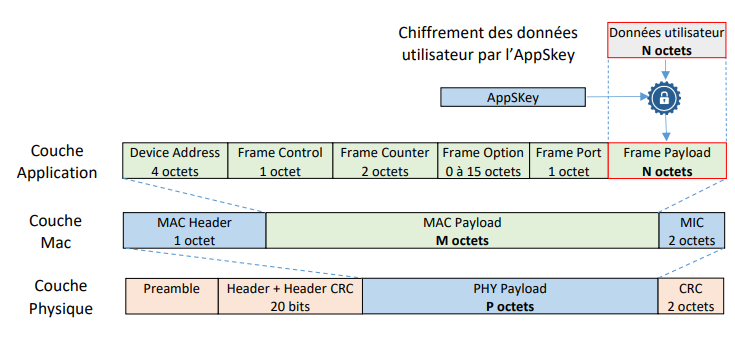
les couches du protocole LoRaWAN

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



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

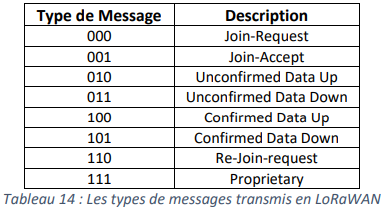






Le protocole LoRa de la couche MAC composé de :

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



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

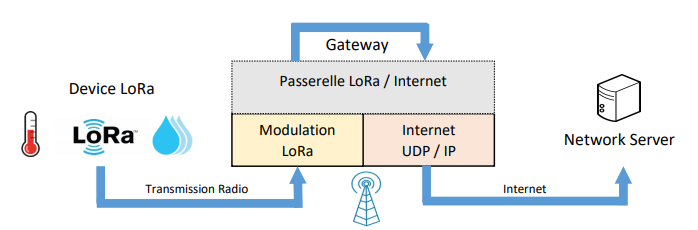
La couche modulation LoRa, le choix du moment d’émission des Devices LoRa se fait de façon simple. Lorsqu’un équipement doit émettre, il le fait sans contrôle et ne cherche pas à savoir si le canal est libre. Si le paquet a été perdu, il le transmettra simplement au bout d’un temps aléatoire.

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.

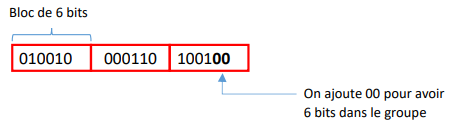
Le Payload correspond aux données transmises dans la trame LoRa. Elles sont brutes et non déchiffrées.

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 an base 64.

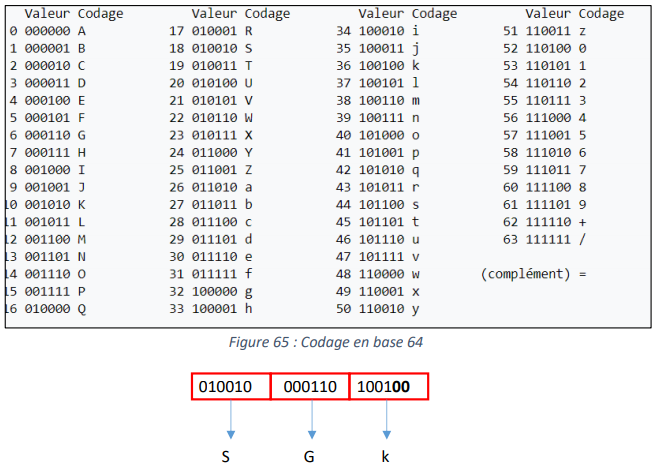
1. On écrit les données à transmettre en binaire

0x4869 : 0100 1000 0110 1001 en base 64 est SGk=

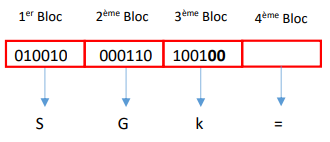
1. On regroupe en bloc de 6 bits et doit être un multiple de 4 (minimum 4 blocs). S’il manque on rajoute des zéros



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



1. 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 (2puissance 6 = 64) au lieu de 8. Donc moins efficace de façon générale.

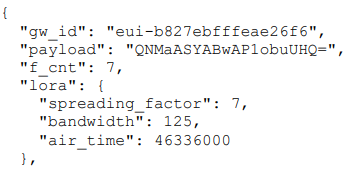
## Exemple pour The Things Network

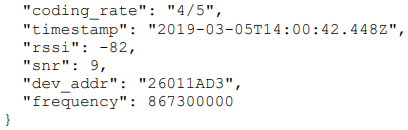
**EXEMPLE POUR THE THINGS NETWORKS**

**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.

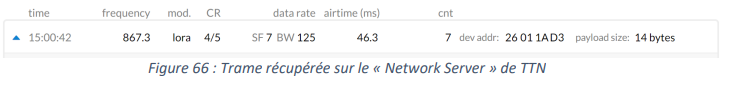
Trame IP





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

On retrouve les valeurs fournies par la gateway :



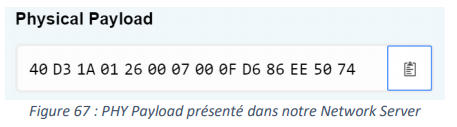
D’autres information provennent de l’analyse du Payload. Le Payload inscrit ici est le PHY Payload.

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

Ce PHY Payload est QNMaASYABwAP1obuUHQ=

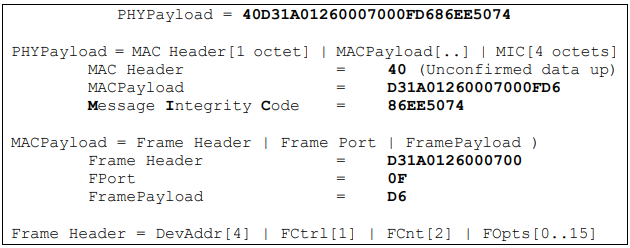
Lorsqu’il est exprimé en hexadécimal au lieu de la base 64 il vaut 40D31A01260007000FD686EE5074

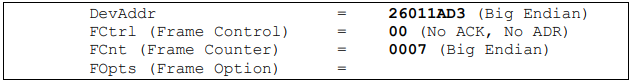
Comme le montre le Network Server de TTN



Sa taille et 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 :





Decodeur de trame lorawan : [LoRaWAN 1.0.x packet decoder (runkit.sh)](https://lorawan-packet-decoder-0ta6puiniaut.runkit.sh/)

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

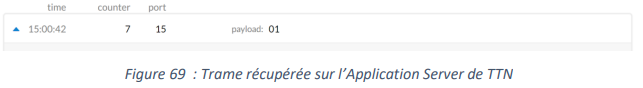


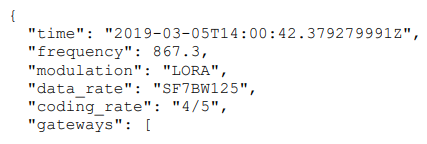
Le Network server à décoder l’ensemble de la trame. Si le MIC est correct (authentification de trame par le NwkSKey) alors le Network Server va passer le contenu du message chiffré (Frame Payload) à l’Application Server, dans notre cas le Frame Payload est (d’après le décodage effectué au chap précédent)

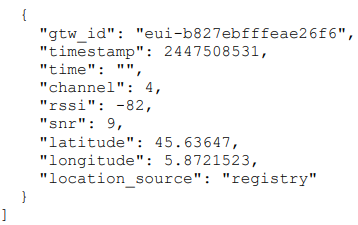


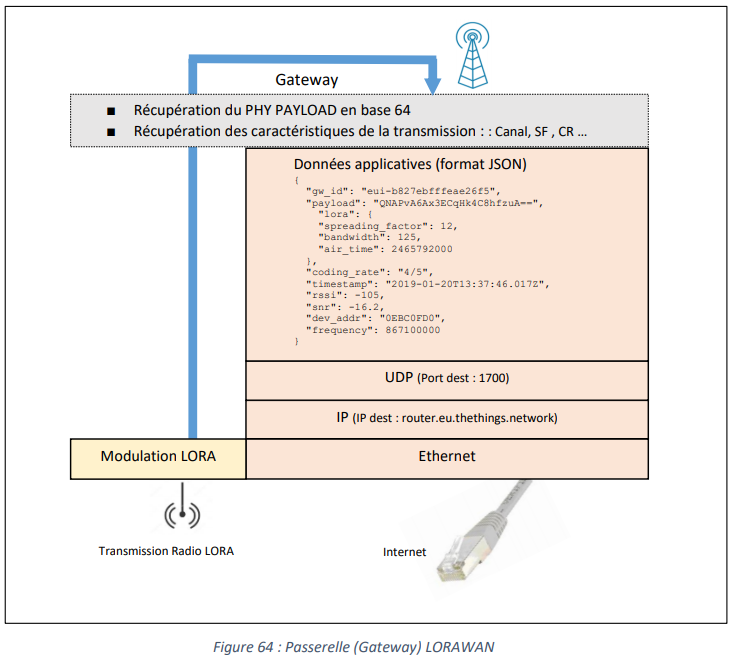
D6 est le contenu chiffré. Lorsqu’il est déchiffré avec l’AppSKey on trouve 01/ avec le décodeur en ligne.

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









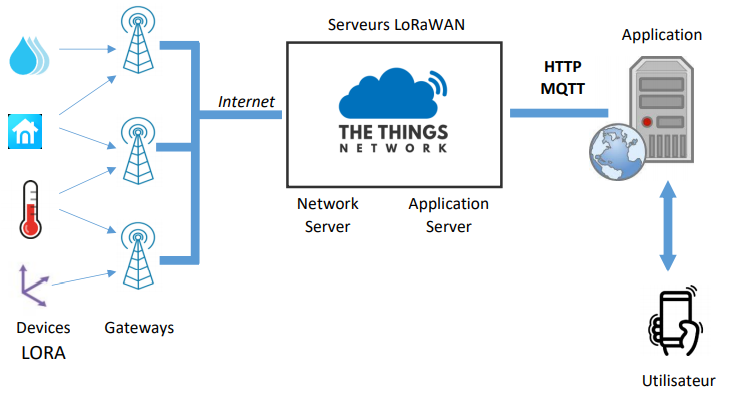
## Récupération des données et rendus de nos données

L’application doit :

. Recevoir des données des serveurs LoRaWAN

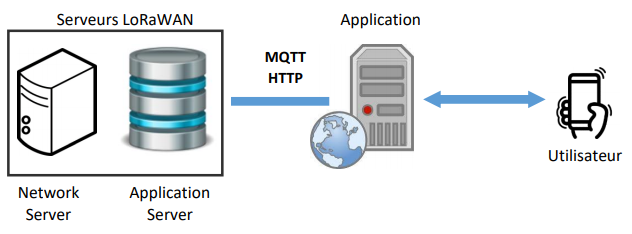
. Transmettre des données à destination des devices LoRaWAN

. Faire la gestion du serveur lorawan : ajouter/supprimer une application ajouter/supprimer un device LoRa …



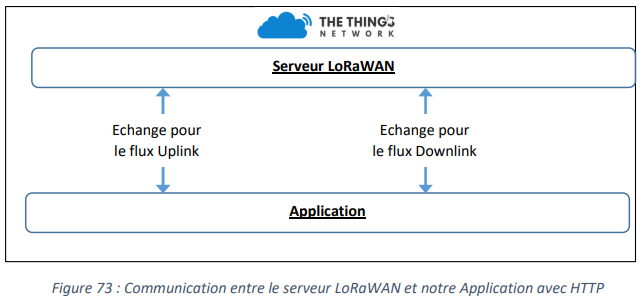
|  |  |
| --- | --- |
| uplink  **La récupération des données.**  Le stockage et traitement.  La mise en forme (graphiques, tableaux…).  L’envoi à l’utilisateur. | Downlink  Présenter une interface utilisateur (Bouton, champ texte, …).  Traiter les commandes de l'utilisateur.  **Envoyer ces commandes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

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



Récupération des données avec le protocole HTTP (GET)

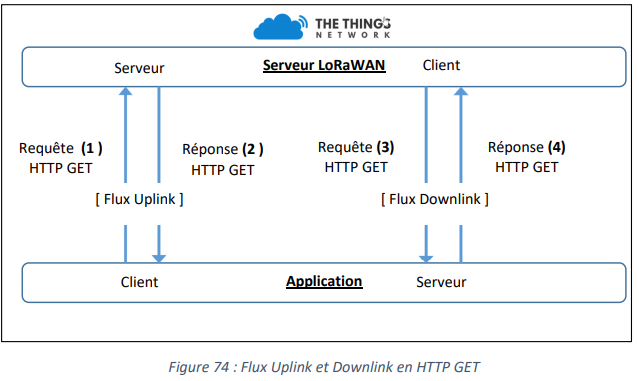
Le http fait appel à un transfert d’information entre client et serveur. Il faut désigner qui est le client et le serveur.



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.

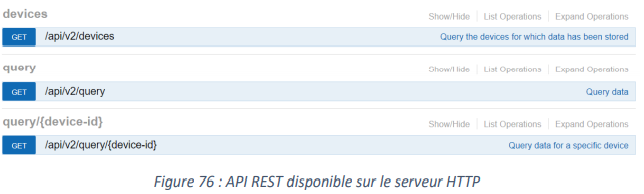


Installation des services http pour le flux Uplink (trame 1 & 2)

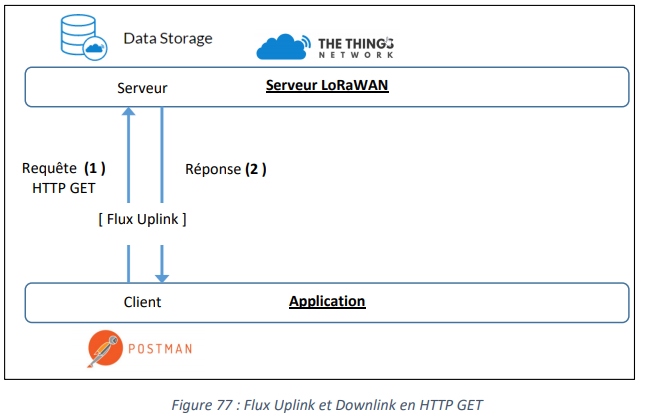
Nous devons mettre un serveur http sur nos serveurs LoRaWAN et un client http sur notre Application.

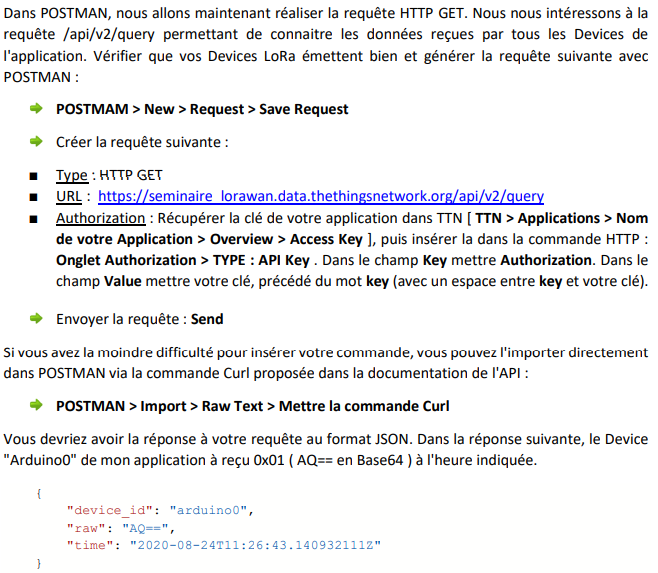
Sur le serveur lorawan la console TTN>Application>Nom de l’Application>Intégration>Data Storage.

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



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.





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 devrons faire des requêtes périodiques avec des réponses vides. Et en downlink la même chose.

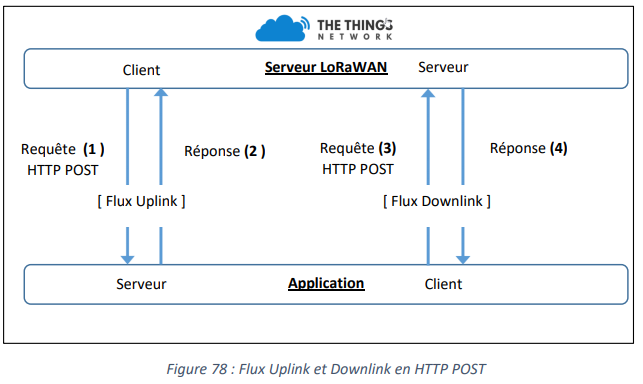
On aimerait avoir une réception rapide sur notre Application et donc des demandes très fréquentes au serveur et une charge réseau.

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

Protocole HTTP POST

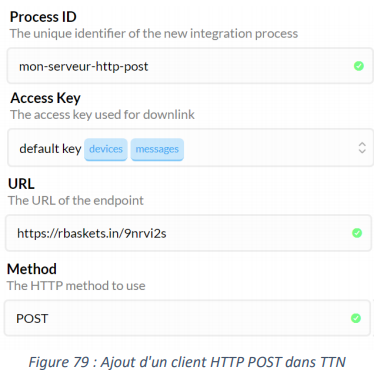
On va plutôt faire la demande mais que le serveur LoRaWAN fournit elle-même à l’Application pour poster les données POST(1) et la réponse (2) est un simple acquittement et ne contient pas de données. Dont TTN lorawan joue le rôle de client et notre application le rôle de serveur.

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.



Installation du serveur http POST rbaskets.in

Pour le client TTN a intégrer



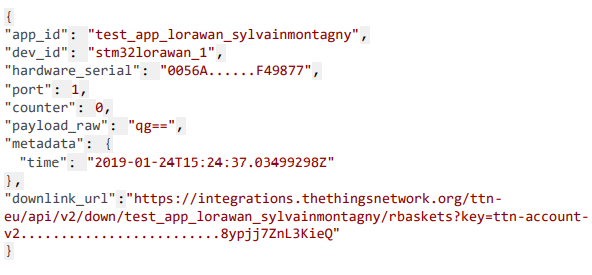
Pour valider le fonctionnement on peut

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



Installation des services http pour le flux Downlink

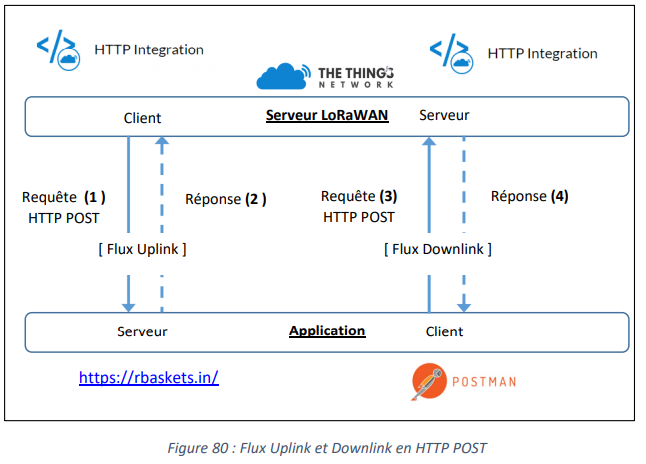
Les trames (3) et (4), nous devons mettre en place un client http sur notre Application et un serveur http sur notre serveur LoRaWAN (TTN)

Sur TTN le serveur existe déjà précédemment.

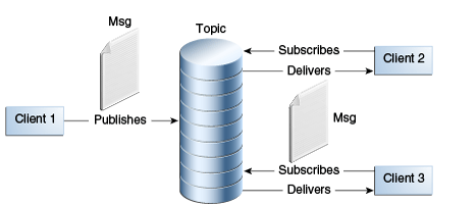
Pour le client POSTMAN et on génère une requête http POST



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



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

Communication par message asynchrone

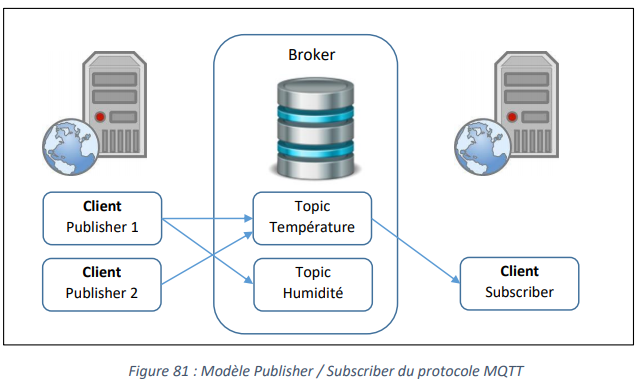
Envoyer et recevoir les données à travers le réseau Ethernet

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

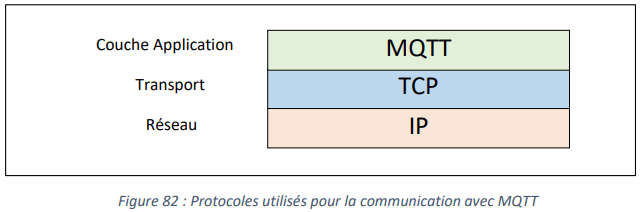
Evènementiel (par topic) (publish/subscribe)

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.

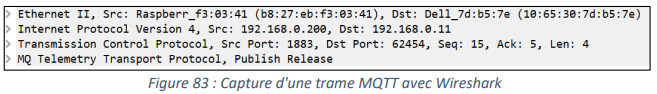
(Serveur central)



MQTT repose sur TCP l’encapsulation est donc la suivante



Vérification sur 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 connaitre et ne sont pas obligés de s’exécuter en même temps.

Connexion au Broker MQTT

On s’intéresse principalement aux options de connexion qui permettrons 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 ou le Client se connecte et le moment où :

Soit il se déconnecte explicitement (Close connexion)

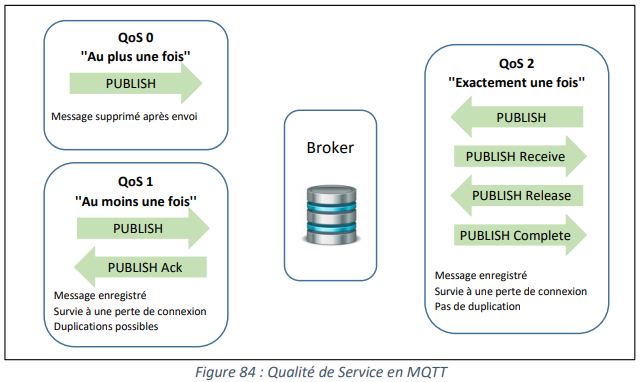
Soit il n’a rien émis, ni fait signe de vie pendant le temps KeepAlive.

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

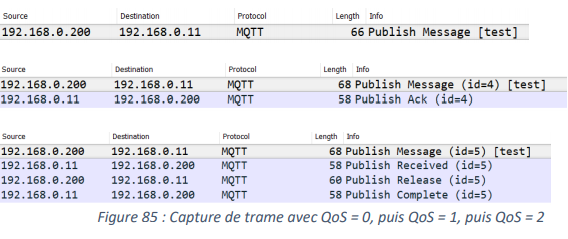
QoS 0 ''At most once'' (au plus une fois). : Le premier niveau de qualité est "sans acquittement". Le Publisher envoie un message une seule fois au Broker et le Broker ne transmet ce message qu'une seule fois aux Subscribers. Ce mécanisme ne garantit pas la bonne réception des messages MQTT.

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

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



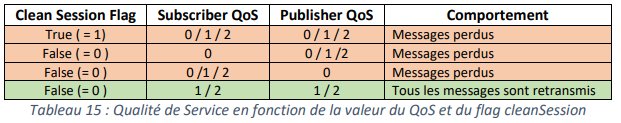
Les trames émises



Qualité de service après une reconnexion

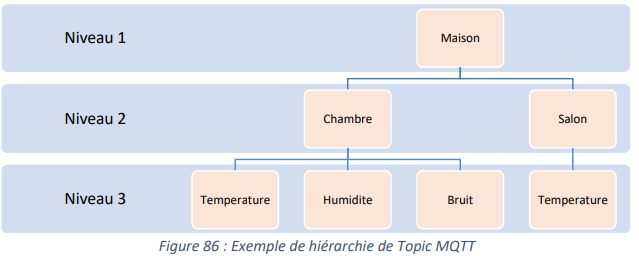
Si les subscribers sont injoignables. Il est possible de conserver les messages qui ont été publiés sur le Broker afin de les retransmettre lors de la prochaine connexion. Cette possibilité doit être activée à l’ouverture de connexion grâce au flag cleanSession = 0. La connexion sera persistante le Broker enregistre tous les messages qu’il n’a pas réussis à diffuser au Subscriber.

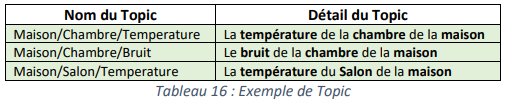
Résumé de l’effet du flag cleanSession et du QoS



**Les Topics du protocole MQTT**

Organisation des Topics



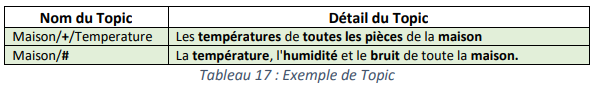


Un client peut s’abonner (désabonner) à plusieurs branches de l’arborescence à l’aide de jokers englobant plusieurs Topics. Deux caractères jockers existent :

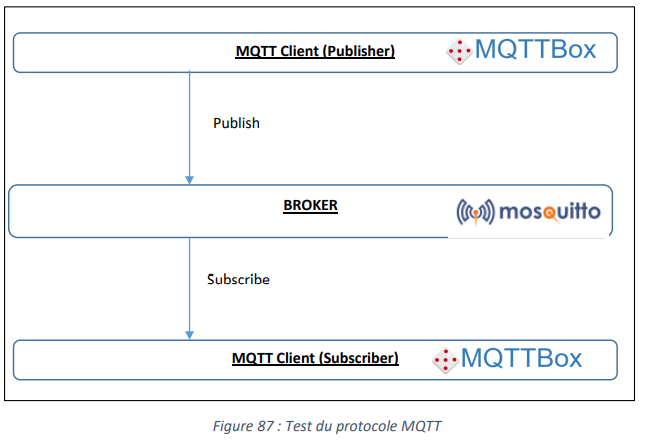
Le + remplace n’importe quelles chaines de caractères sur le niveau où il est placé

Le # remplace n’importe quelles chaines de caractères sur tous les niveaux suivants

il est obligatoirement placé à la fin.



**MISE EN PLACE D’UN BROKER MQTT**



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

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

1. apt-get install mosquito // 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

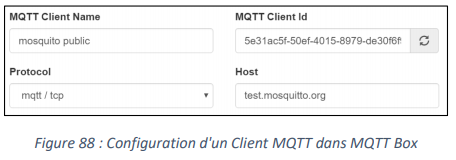
MQTTBox

MQTTBox>Create MQTT Client

Dans ce client les seuls champs indispensables sont :

Protocol : MQTT/TCP

Host : test.mosquitto.org

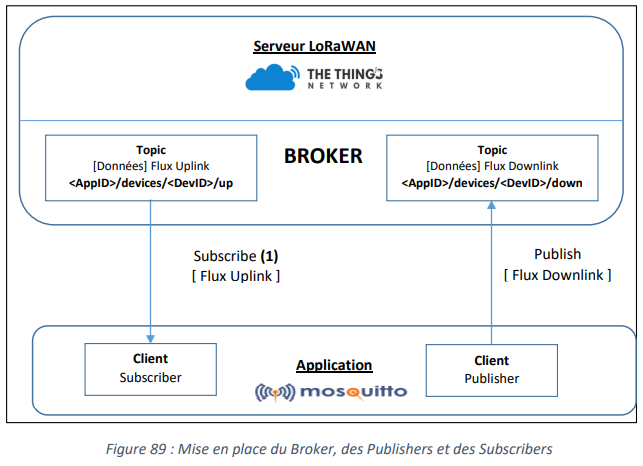


Tester l’envoi et la réception sur les Topics de votre choix.

RECUPERER LES DONNEES EN PROVENANCE DE TTN

En uplink TTN joue le rôle du Broker

Notre Application le rôle de Subscriber



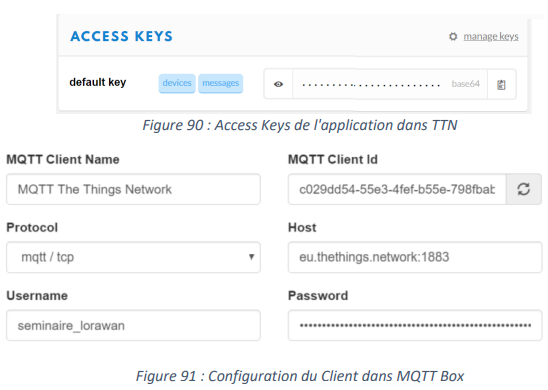
TTN intègre le Broker et le client nous utilisons MQTTBox mais avec les configurations suivantes

Protocol : mqtt/tcp

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

Username : connexion vers le Broket MQTT nom de l’application.

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

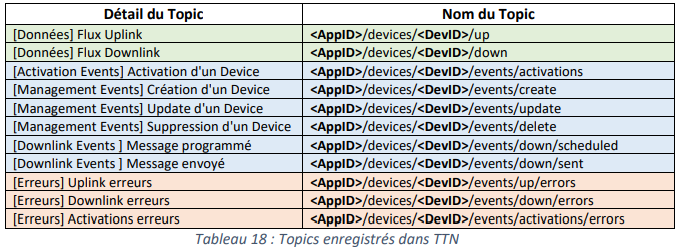


Le client peut se connecter au Broker il reste à définir le fait que le client sera Subscriber et il recevra les informations dépendent du Topic auquel nous souscrivons

Topic disponible sur le broket TTN

AppID correspond au nom de votre Application

DevID correspond au nom de votre Device LoRa



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



Envoyer des données à destination de TTN

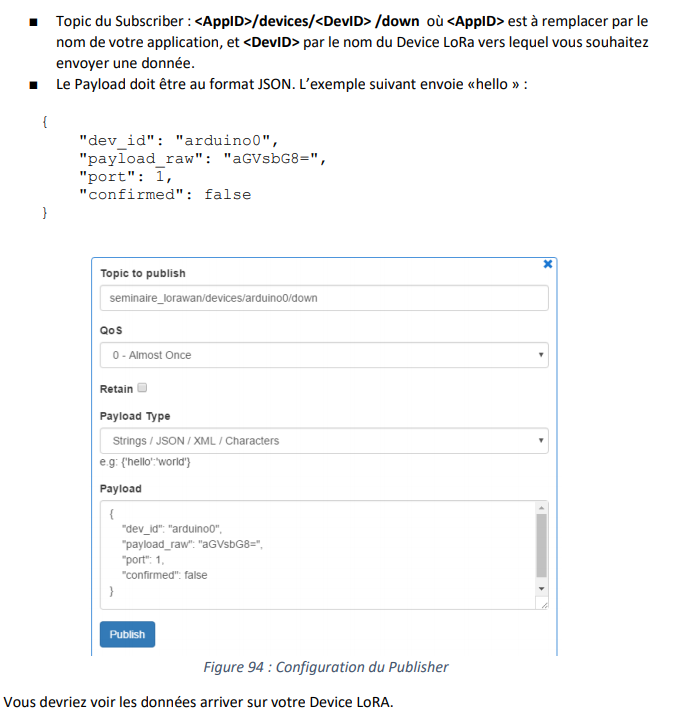
Downlink dans ce cas

L’application est le Publisher

TTN jour le rôle de Broker

Donc la trame (2)

Il reste à configurer le client Subscriber



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 Architecturent

## 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 LORIOT Gateway Software ou TTN Packet Forwarder.

Mise en place de l’environnement de travail

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

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

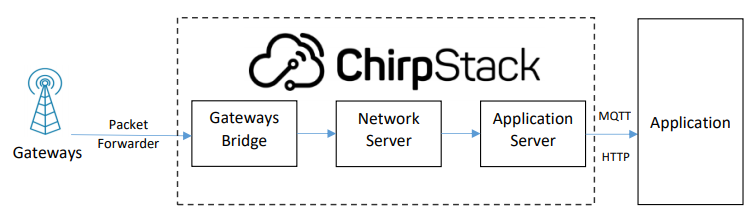
Dans les autres cas on peut utiliser Docker et Docker compose



Présentation de Chirpstack

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

Architecture de Chirpstack



Gataway 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

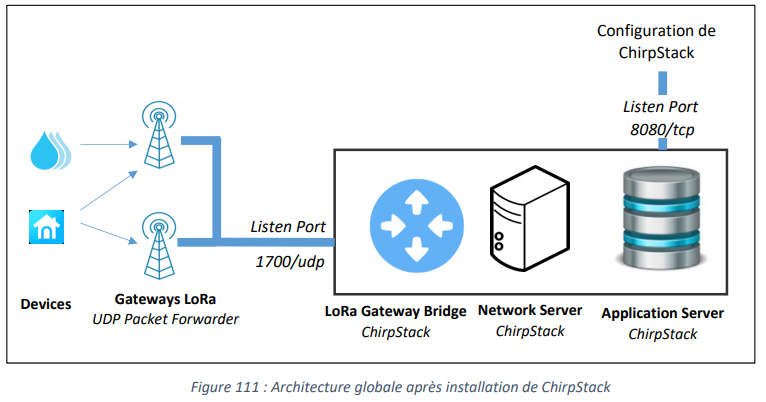
[GitHub - brocaar/chirpstack-docker: Setup ChirpStack using Docker Compose](https://github.com/brocaar/chirpstack-docker)

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

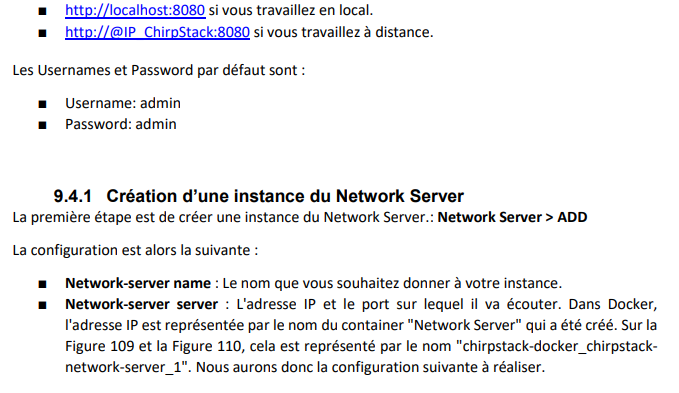


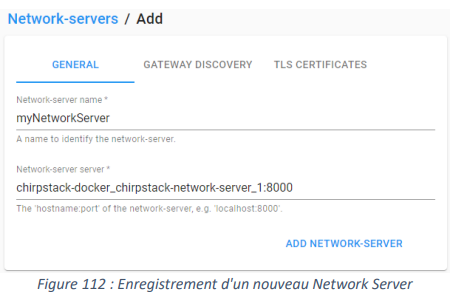


Application server écoute sur le port 8080/tcp c’est l’accès vers l’interface web pour la configuration de notre serveur lorawan



Configuration :





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 APB ou en OTAA. Cliquer sur "Device supports OTAA" puisque nous faisons l'essai en OTAA.

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

■ Device name : myDevice

■ Device description : Mettre la description de votre Device LoRa.

■ Device EUI : Rentrer votre Device EUI.

■ Device-profile : myDeviceProfile

■ Puisque nous travaillons en OTAA, nous n'avons pas besoin de "Disable frame-counter validation" (voir chapitre 4.5.2)

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

■ Application key : Rentrer votre AppKey.

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.

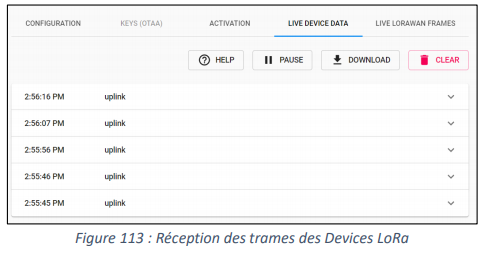
■ Port Up d'écoute de ChirpStack: 1700

■ Port Down : 1700

Visualisation des trames reçues

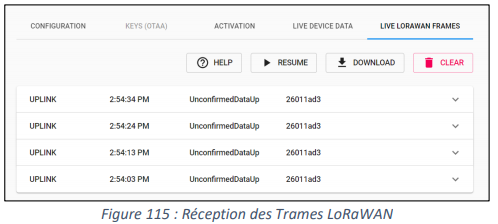
La configuration minimale de ChirpStack est maintenant terminée. Nous pouvons donc brancher un Device LoRa qui possède les attributs (DevEUI et AppSKey) que nous avons enregistrés dans ChirpStack et le faire émettre.

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)





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.



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 :

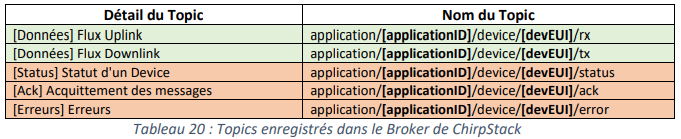
■ Payload marshaler : Le format utilisé pour transférer les données (JSON est un bon choix). ■ EndPoint : L'URL de votre serveur HTTP POST.

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.

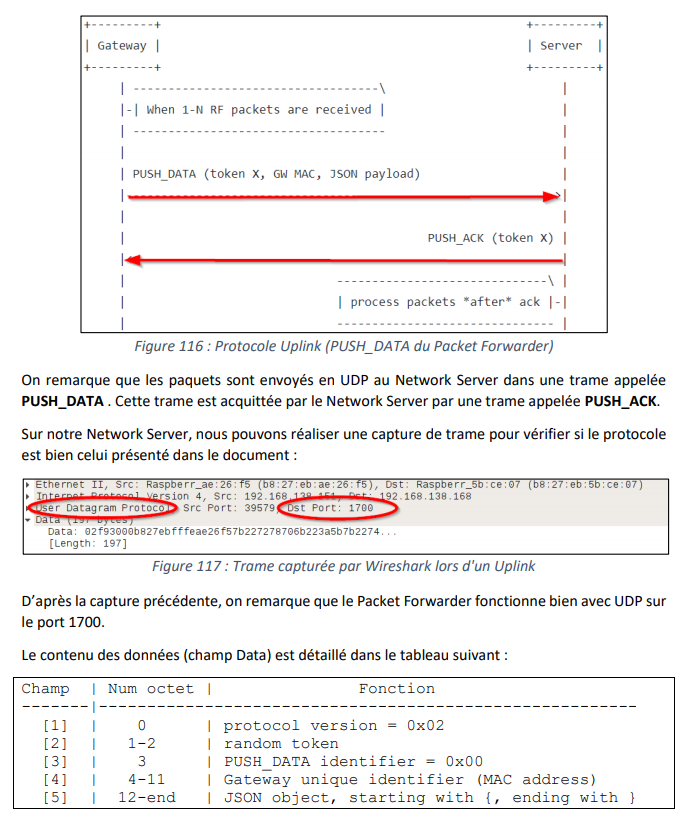
■ [devEUI] correspond au nom de votre Device LoRa.

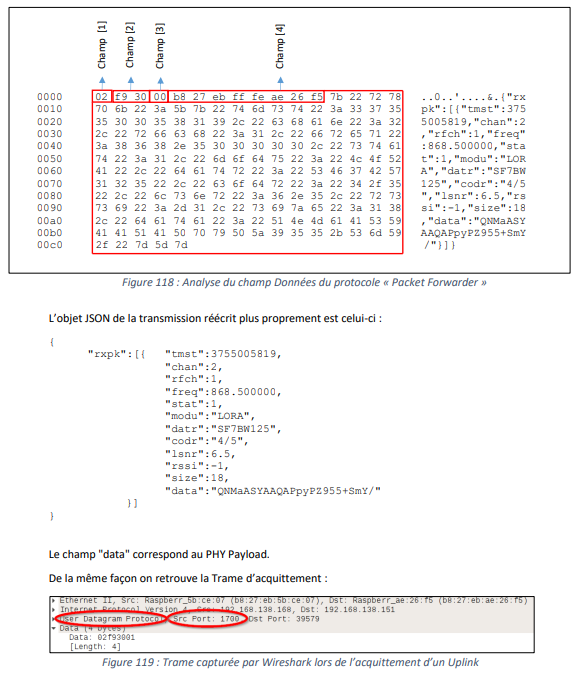


Analyse du "Packet Forwarder"

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 . Dans ce dossier, un fichier nommé PROTOCOL.TXT explique parfaitement ce protocole qui travaille au-dessus de UDP







## 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)

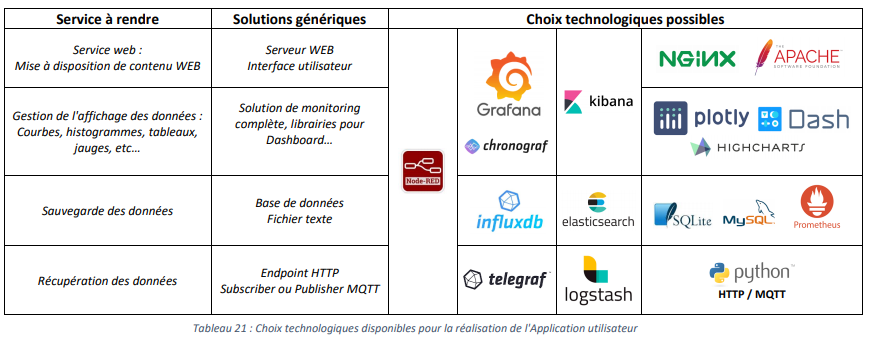
vous retrouverez une structure plus robuste organisée autour des services : Telegraf, InfluxDB, Chronograf et Grafana.

■ Gérer la récupération des données > Telegraf

■ Stocker les données. > InfluxDB

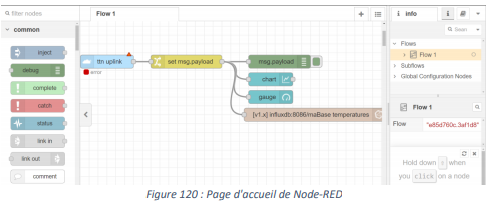
■ Système de monitoring et Serveur WEB > Chronograf ou Grafana

Mais aussi en cloud IOT AWS ou Microsoft Azure



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 ].



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.

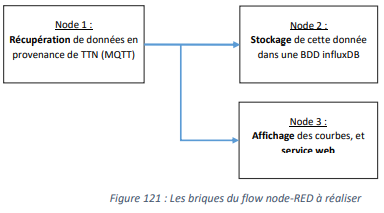
Pour notre cas d'étude, nous nous placerons dans un cas simple qui est l'envoi d'une température sur un octet par un capteur. Nous souhaitons :

■ Récupérer la valeur de température (node TTN).

■ La stocker dans une BDD (node InfluxDB).

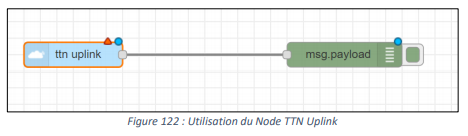
■ Afficher sa variation en fonction du temps (node Dashboard).

Réaliser l’application nodeRed



Récupération des données avec le node TTN

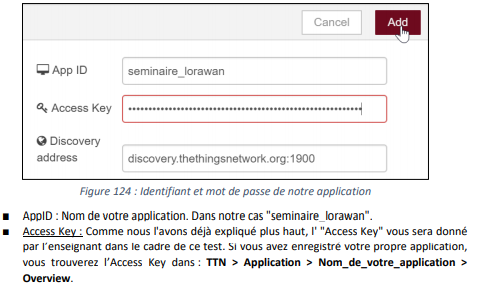
Nous allons gérer le flux Uplink. Apportezsur votre schéma un node ''ttn uplink" et un Node "Debug" puis reliez-les. Le Node "ttn uplink" nous permettra de configurer la communication avec TTN en MQTT. Le Node "debug" écrira sur le terminal les informations brutes qui seront reçues par le Node "ttn uplink".



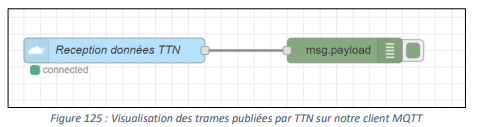
Double cliquer sur le Node "ttn uplink" pour le configurer comme sur la Figure 124 : Le champ "Name" sera le nom du node sur votre schéma. Le champ Device ID est le Device LoRa concerné dans votre application. De plus, il faut entrer les caractéristiques de votre application (Add new ttn app) pour que Node-RED puisse s'y connecter.



Configurer

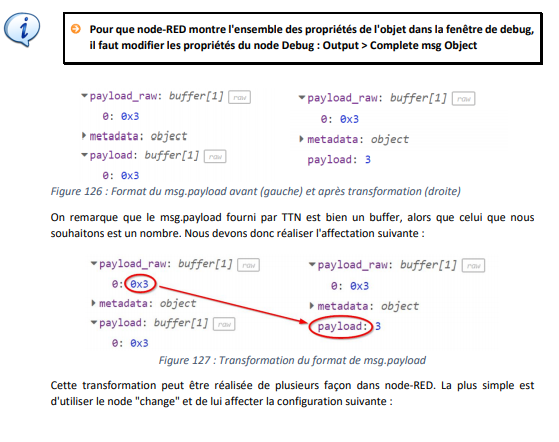


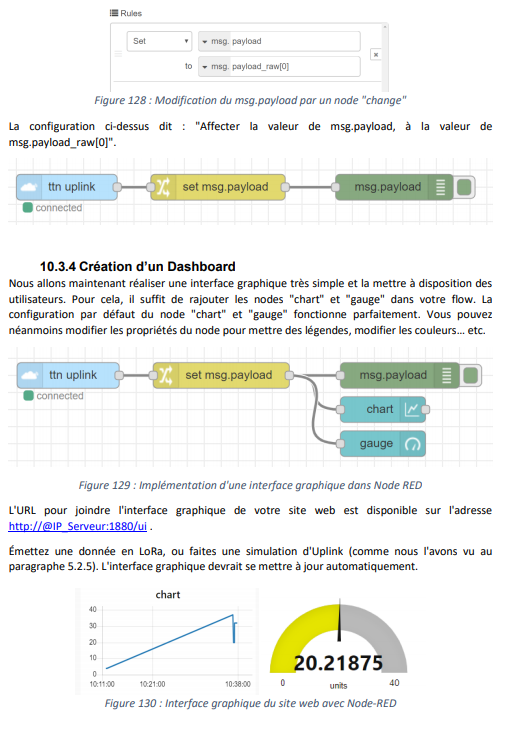
Le node "ttn uplink" doit maintenant avoir le nom que vous avez configuré. Cliquer sur Deploy pour lancer votre projet. Le bloc "ttn uplink" doit maintenant être connecté à TTN. Cela est spécifié "connected" sous le bloc



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





Stockage des informations dans une base de données

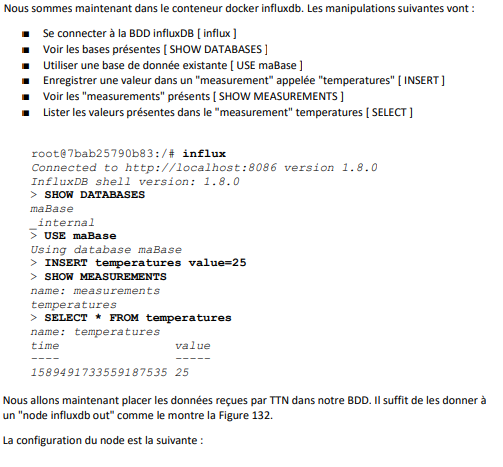
Nous utiliserons la base de données InfluxDB. Une librairie de node-RED fournie un node permettant de s'interfacer très facilement à InfluxDB. Nous utiliserons principalement le node "influxdb out" pour écrire des données dans la BDD. La documentation des nodes InfluxDB est fournie par le lien suivant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

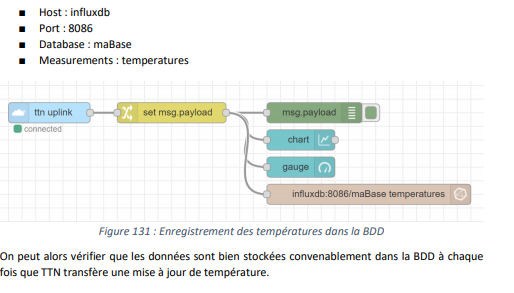
Elle montre bien que si le msg.payload est un nombre, alors sa valeur sera enregistrée. Les écritures dans la BDD se font grâce au protocole LINE dont la syntaxe est très bien expliquée ici

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

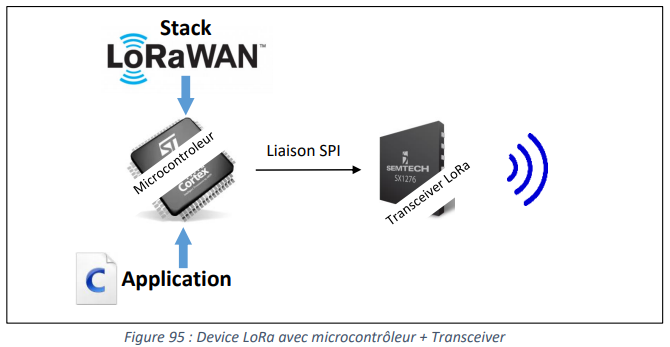
Dans notre cas, nous définirons un "measurement" appelé "temperatures" et les champs (fields) seront appelés "values". La BDD étant déjà installée d'après la mise en place de l'environnement de travail que nous avons vu au paragraphe 10.1, nous pouvons donc directement commencer à travailler. La BDD 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





## Architecture 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 SEMTCH.

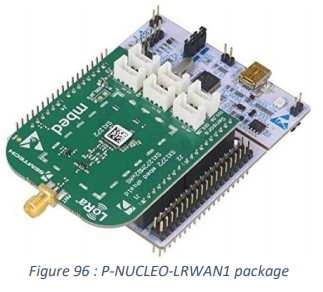
Ils gèrent la partie physique du protocole : Modulation, détection de préambule …

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

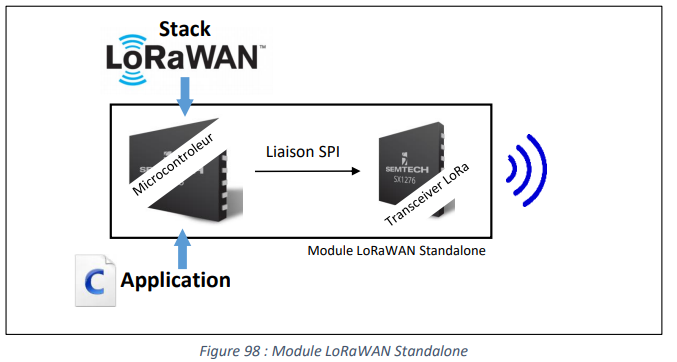
Ce choix-là est assez abouti et optimisé en termes de consommation, car il n'y a qu'un seul microcontrôleur qui gère tout. En revanche, c’est une solution beaucoup plus complexe en ce qui concerne la partie logicielle. En effet, il faudra se plonger dans la Stack. Même si la partie Application est bien 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



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



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

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