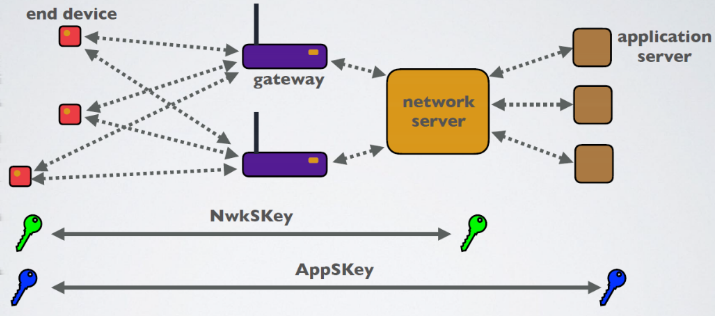
Connexion entre le node et le serveur à la méthode d’activation

L’activation a pour but d’établir une communication sécurisée entre les équipements.

Elle est réalisée à partir d’une paire de clés :

NwkSKey pour chiffrer/déchiffrer les données entre le node et le serveur réseau

Et AppSKey pour chiffrer/déchiffrer les données entre le node et le serveur d’applications.



Nous avons 2 méthodes d’activation.

Méthode ABP :

Les clés de session NwkSKey et AppSKey ainsi que l’adresse de l’équipement (DevAddr) sont directement inscrits dans l’équipement LoRaWAN. Ainsi, l’équipement n’a plus besoin d’envoyer de requête avant de communiquer sur le réseau.

L’utilisation de cette méthode implique que les équipements communiquent avec un réseau spécifique car les clés de session sont connues par avance. Et contrairement à la méthode OTAA , les clés de session sont statiques. Dans le cas d’un prototypage ou pour une utilisation sur un réseau connu, la méthode ABP suffit.

Méthode OTAA :

Elle offre un niveau de sécurité supérieur plus agile et pour un déploiement à plus grande échelle.

L’équipement transmet une demande d’accès (join request), pour le faire celui-ci doit être en possession de trois paramètres **DevEUI** (identifiant unique de type EUI 64 peut être fournit par l’équipement).

**AppEUI** (identifiant du fournisseur de l’application type EUI 64).

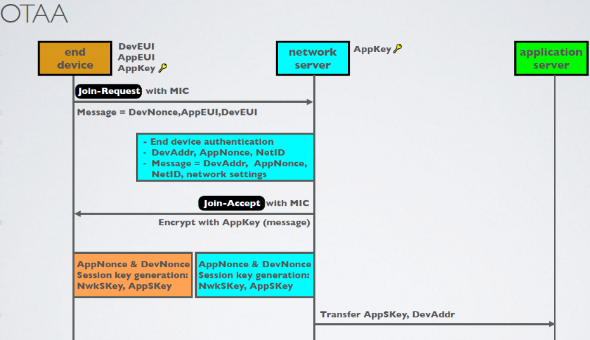
**AppKey** clé AES 128 déterminée par le fournisseur de l’application (pour le chiffrement).

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

L’équipement envoie, à travers le réseau, la requête join request contenant DevEUI , AppEUI , ainsi qu’un

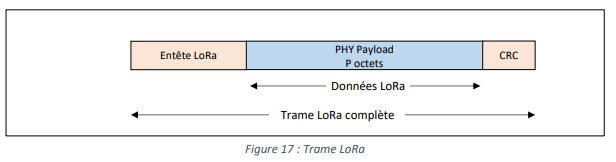
MIC (Message Integrity Code) calculé via la clé AppKey. Cette requête est transmise au serveur l’enregistrement qui vérifie le MIC via la clé AppKey (qui lui a été communiquée au préalable). Si l’équipement est autorisé par le serveur d’enregistrement, la requête join accept est transmise en réponse à l’équipement. Cette réponse contient des données à partir desquelles l’équipement va pouvoir calculer les clés de session (réseau : NwkSKey et applicative : AppSKey).

Parmi les données contenues dans cette réponse, se trouve également l’adresse Device Adress (DevAddr) sur 32 bits – qu’utilisera l’équipement pour communiquer sur le réseau.



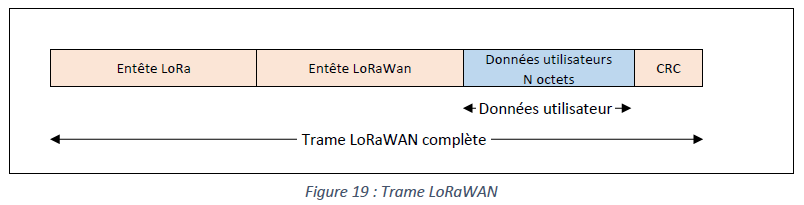
Différence entre les trames

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.

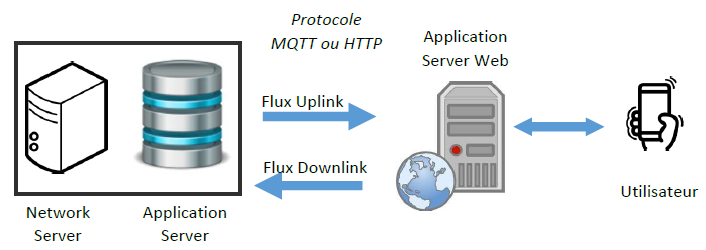
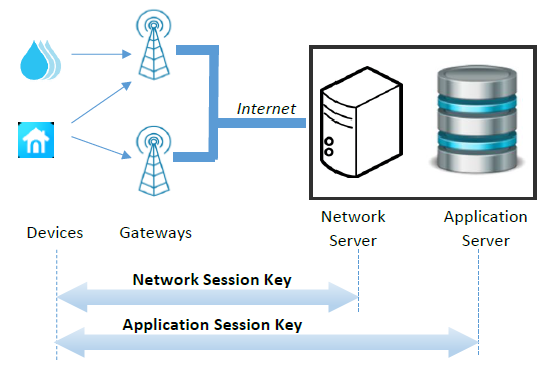
**Le Network Server**

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)

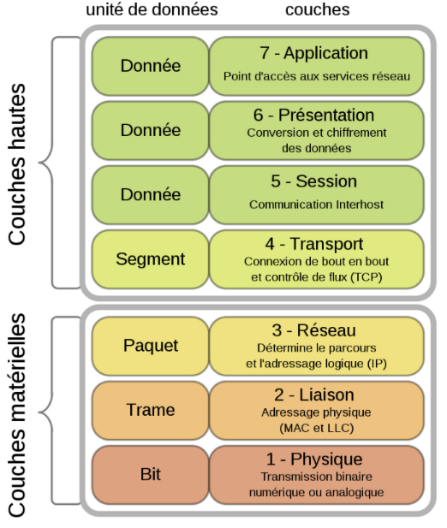
**Application Server**

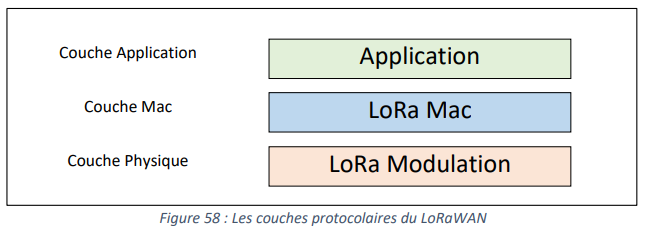
Il est souvent sur le même support physique que le Network Server. Il permet de dissocier les applications les unes des autres. Chaque application enregistre des Devices LoRa qui auront le droit de stocker leurs données (Frame Payload). Les messages transmis à l’application server sont chiffrés

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

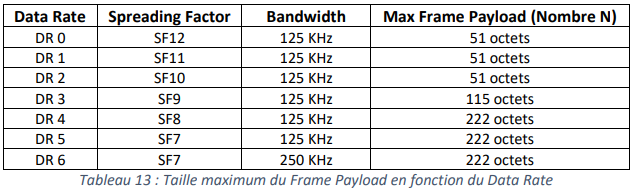


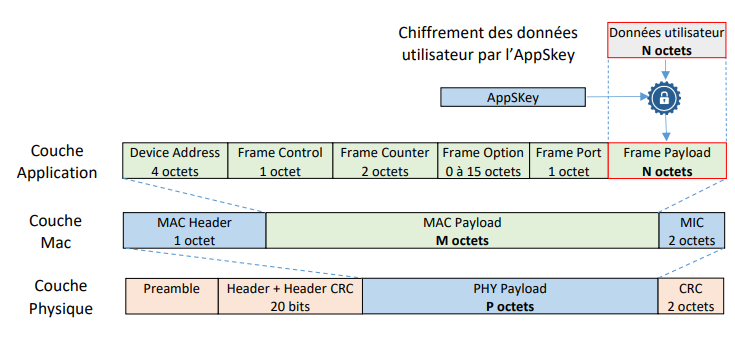
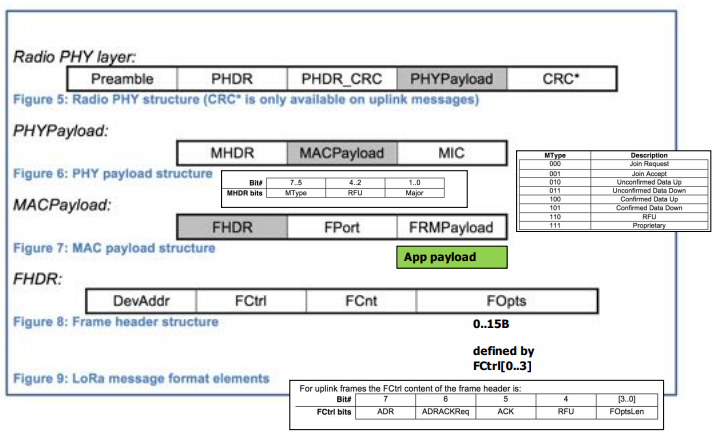
Les couches du protocole LoRaWAN

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

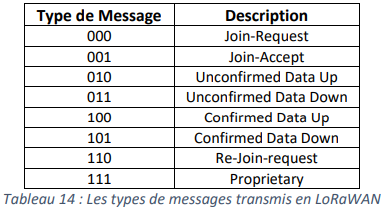






Le protocole LoRa de la couche MAC composé de :

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.

**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.

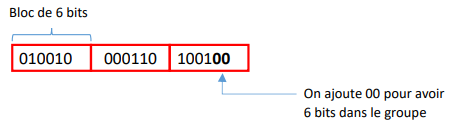
Le Payload correspond aux données transmises dans la trame LoRa. Elles sont brutes et non déchiffrées.

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.

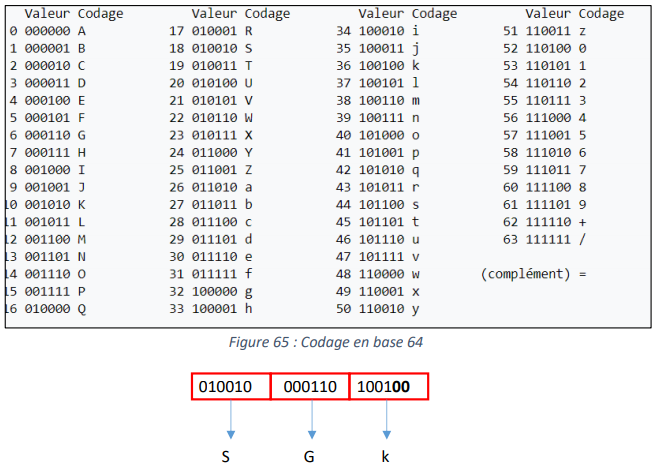
1. On écrit les données à transmettre en binaire

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

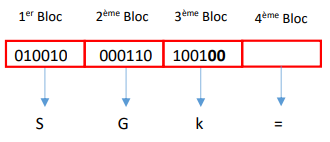
1. On regroupe en bloc de 6 bits et doit être un multiple de 4 (minimum 4 blocs). S’il manque on rajoute des zéros



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

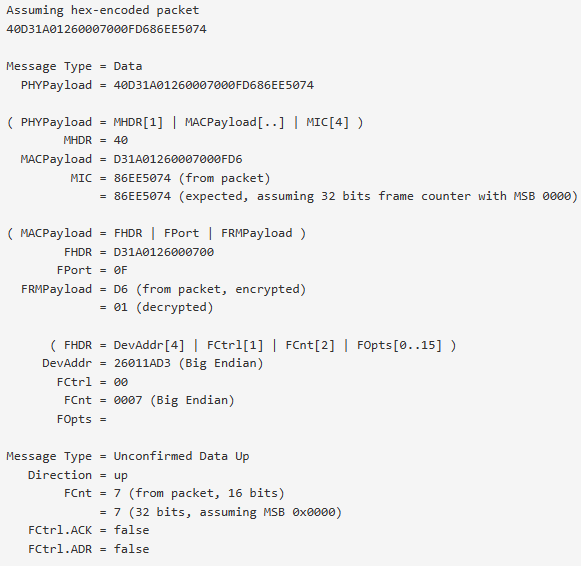
C’est un choix qui a été fait pour le protocole LoRa / LoRaWAN afin de rendre les données binaires lisibles. Le problème du code ASCII c’est qu’il est composé d’un certain nombre de caractères non imprimables (EOF, CR, LF …). Pour éviter ces soucis, la base 64 ne comporte que 64 caractères imprimables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

Test decoder de trame lorawan base 64 : [LoRaWAN 1.0.x packet decoder (runkit.sh)](https://lorawan-packet-decoder-0ta6puiniaut.runkit.sh/)

40D31A01260007000FD686EE5074

NwKSKey : E3D90AFBC36AD479552EFEA2CDA937B9

AppSKey : F0BC25E9E554B9646F208E1A8E3C7B24



# **Discussion générale de la création du protocole projetcns**

Le format des données capteurs envoyés par un nœud lora. [Spécification]

Le format de données mis en place cherche à réduire la taille des données transmises, tout en étant générique pour supporter une grande variété de capteurs.

Les données sont codées en binaire pour plus de compacité, en orientation petit-boutiste (little-endian bit-poids-faible-en-premier)

Le format proposé n’est pas d’une très grande régularité ou simplicité, en particulier il n’est pas composé d’un motif fixe reproduit à l’identique pour toutes les données, comme c’est le cas par exemple pour le format Cayenne qui a servi de source d’inspiration : [Cayenne Docs (mydevices.com)](https://developers.mydevices.com/cayenne/docs/lora/)

([https://http://mydevices.com/cayenne/docs/lora/](https://http:/mydevices.com/cayenne/docs/lora/)).

Le but est de réduire la taille des données transmises.

Le nombre d’octets d’un champ peut varier et certaines valeurs peuvent « déborder » de leur cadre (ont une portée globale). [Pour être plus clair sur ce dernier point j’évoquerais le cas de l’heure (timestamp)]

Toutes les mesures de capteurs sont horodatées.

Il se peut que plusieurs mesures aient une même date, parce qu’elles proviennent d’un même capteur ou encore parce que nous avons reçu simultanément des mesures de plusieurs capteurs par exemple.

La répétition à l’identique de données dans une même trame représente une perte de bande passante.

Aussi, dans le format proposé ici, les dates d’horodatage (timestamp) sont valides jusqu’à la fin de la trame de données envoyée, ou jusqu’à ce qu’un nouveau timestamp soit défini.

Toutes les valeurs de mesure transmises ont donc un timestamp qui est celui en cours de validité, qu’il ait été défini juste avant elles ou bien au début de la trame.

MSB big-endian gros-boutiste démarre avec les octets de poids fort (TCPIP,PCIexpr)

LSB little-endian petit-boutiste démarre inverse avec les octets de poids faible

