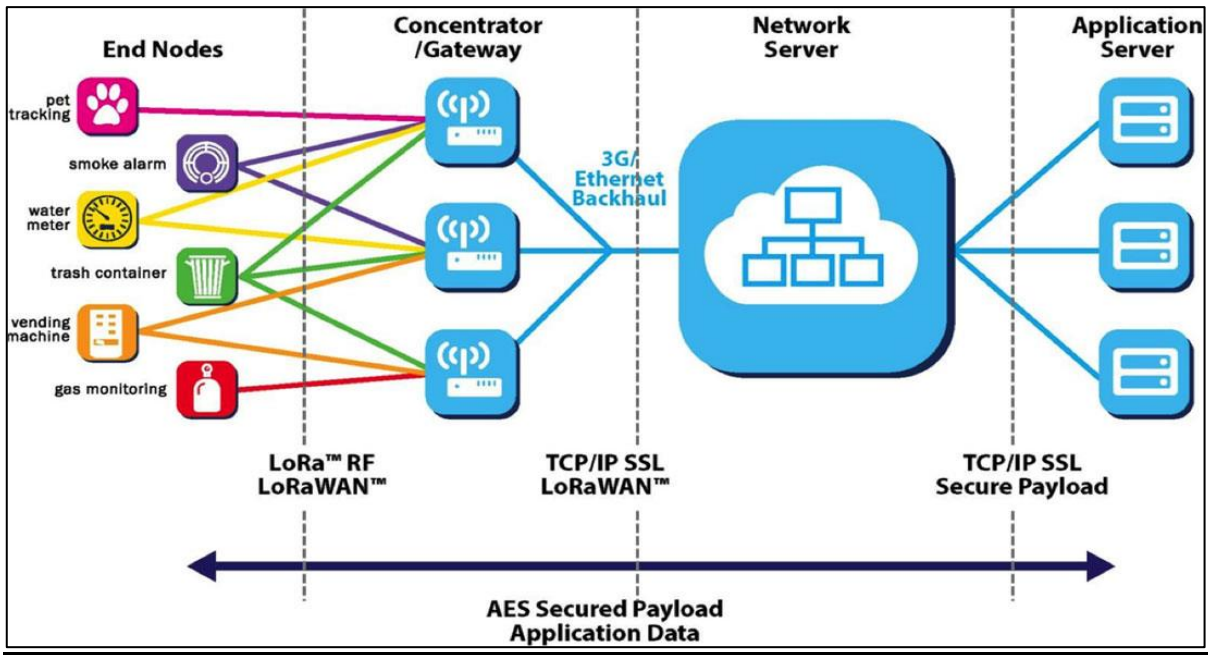
Complete LoRa transmission chain: from the data to the .csv file

Introduction

The following document will explain how a payload from an end-device (also called 'node') passes through different elements of the LoRa/LoRaWAN network and reaches the database of a server where it will be exported into a .csv file. This document will also explain each element of this transmission chain and how each of them behaves, what it does, what it adds to the transmission, …



A simplified view of a LoRa transmission

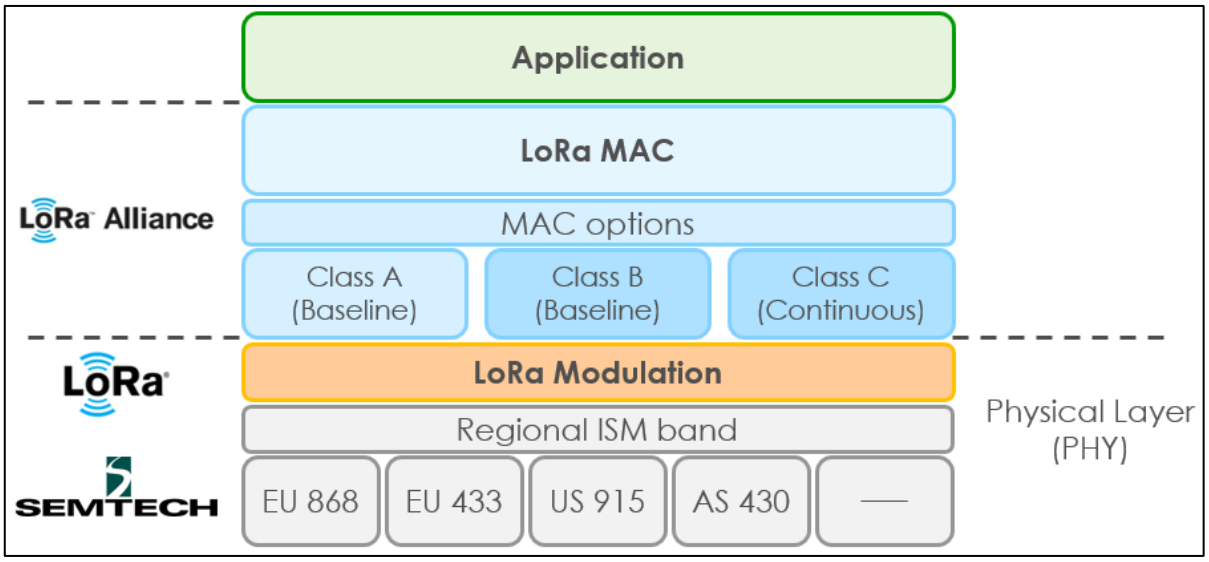
Difference between LoRa and LoRaWAN

Before getting into the subject, let's briefly explain the difference between the two terms.

LoRa is the PHY (physical) layer of the OSI model that uses Chirp Spread Spectrum modulation in order to send data. This modulation is wireless and allows for low-power emitters (usually battery life >1 year) and also high range (1km – 15km approx.).

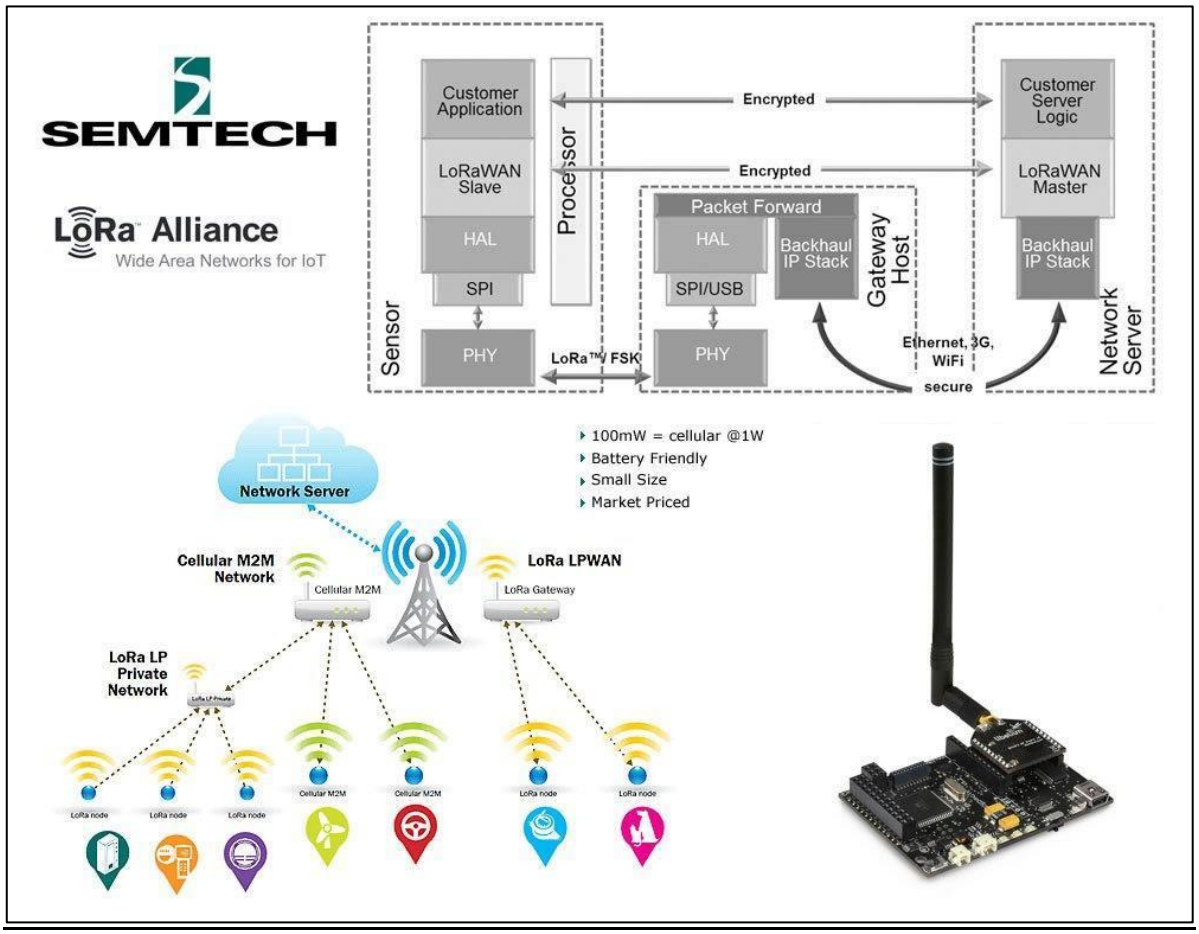
LoRaWAN is a network protocol that uses LoRa for communications. LoRaWAN also adds a MAC layer which includes network registration, addressing, encryption, decryption, message integrity, the possibility to use commands (called MAC commands) for various purposes (ex : join request, confirmed data up, join accept, …). In addition, LoRaWAN adds parameters to control the LoRa transmission (Spreading Factor, Coding Rate, Adaptive Data Rate, …). In the OSI model, LoRaWAN would cover the transport layer, IP layer and MAC layer.

It is possible to use only LoRa modulation without using LoRaWAN, but it wouldn't be practical because of the (very) limited functionality of the LoRa modulation alone.



OSI model of a LoRa/LoRaWAN transmission

Example of a LoRa transmission (from the node to the network server)



Different layers of a node and a gateway in a LoRa transmission

Activation of an end-device

The end nodes are the devices that will collect data (temperature, pressure, wind speed, activation of alarms, fill level of a tank, …) and will send it to application servers that will be accessed by the user. In some cases, the nodes are even simpler than what is described above and will not have any sensor on them (these nodes will just send some predetermined data at a certain period).

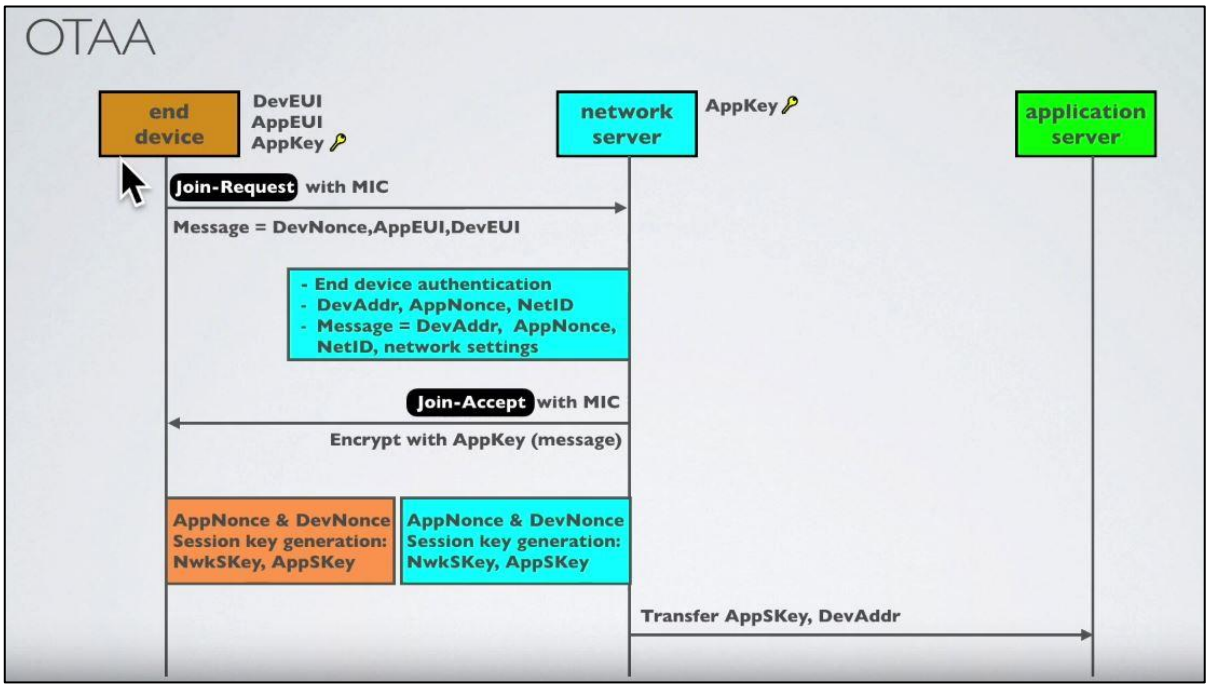
Before being able to communication with the IoT network (ex : TheThingsNetwork), the device must be activated. There are 2 methods: OTAA (Over The Air Activation) and ABP (Activation By Personalization). We will not discuss about the last one, as we will not use it for our project.

The OTAA method is a "join-request" and "join-accept" activation process, where the end-device sends the request to the network server, where it will respond by a "join-accept" if the device is properly authenticated. Before the activation, the node must store its DevEUI, AppEUI and AppKey. The network server stores the same AppKey as the node.

The DevEUI (64 bits – Extended Unique Identifier) uniquely identifies the end-device and is very similar to a MAC address. In most cases, end-devices come with a pre-loaded DevEUI.

The AppEUI uniquely identifies the application server we are going to send the data to. This parameter is like a port number.

The AppKey is an AES symmetric key (128 bits) and is used to generate a MIC (Message Integrity Code) to ensure the integrity of the exchanged messages.



OTAA activation process (Copyright Robert Lie – mobilefish.com)

The first step of the activation process is the "join-request" message which contains the DevNonce (randomly generated number to ensure that no other device can repeat this

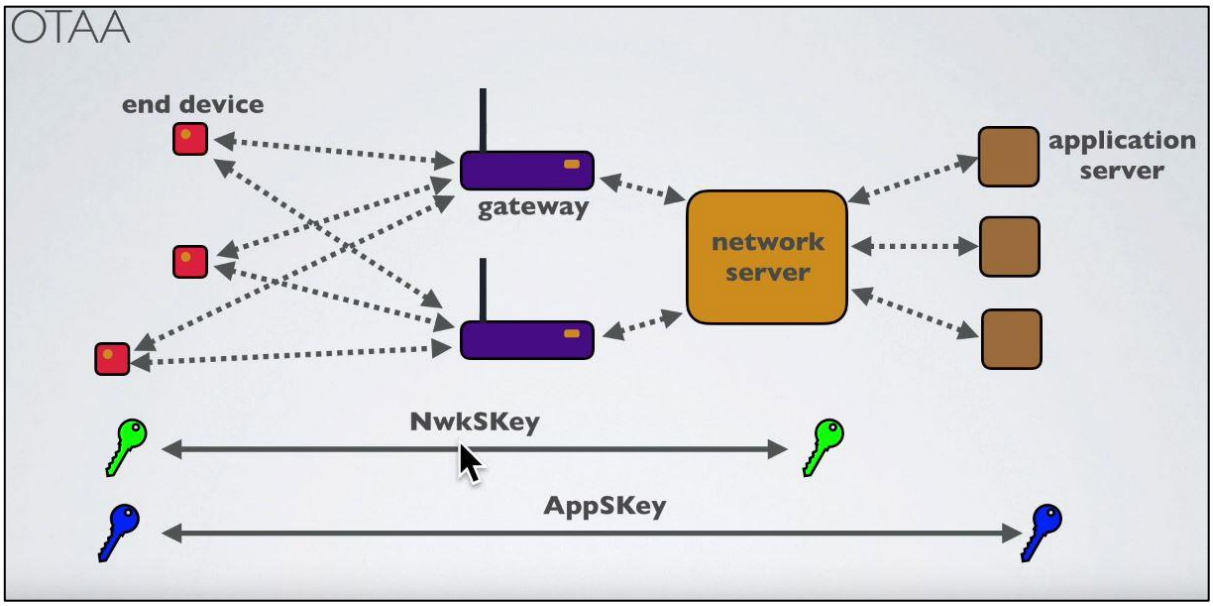
request), AppEUI and DevEUI. The AppKey generates the MIC for the request message but the message itself is not encrypted with the AppKey. When the network server receives the request, it will check if the DevNonce has already been used or not (a message "failure" will be sent if it is the case). It will do the same for the AppEUI and DevEUI. Then, the network server will generate its own MIC with its own AppKey. If it is the same as the MIC sent by the device, the node is successfully authenticated.

The network server will then generate, for the end-device, a DevAddr. This is a shorter 32 bits address that is mapped to the unique DevEUI and that has a similar meaning to a client IP address. The purpose of this shorter address is to reduce the transmission overhead during the communications.

The AppNonce is a randomly generated number (like DevNonce) and the NetID (24 bits value to identify LoRaWAN networks) is the network identifier of the end-device.

The "join-accept" message sent to the node contains all of this information and some network settings (data rates to use for reception, channel frequency list, reception delay, …) and is encrypted with the AppKey. The "join-accept" message also contains its MIC.

At the end of this exchange, the AppNonce and the DevNonce (which are the same for both end-device and network server) will generate a network session key (NwkSKey) and an application session key (AppSKey). The network server will transfer the AppSKey and the DevAddr to the application server.



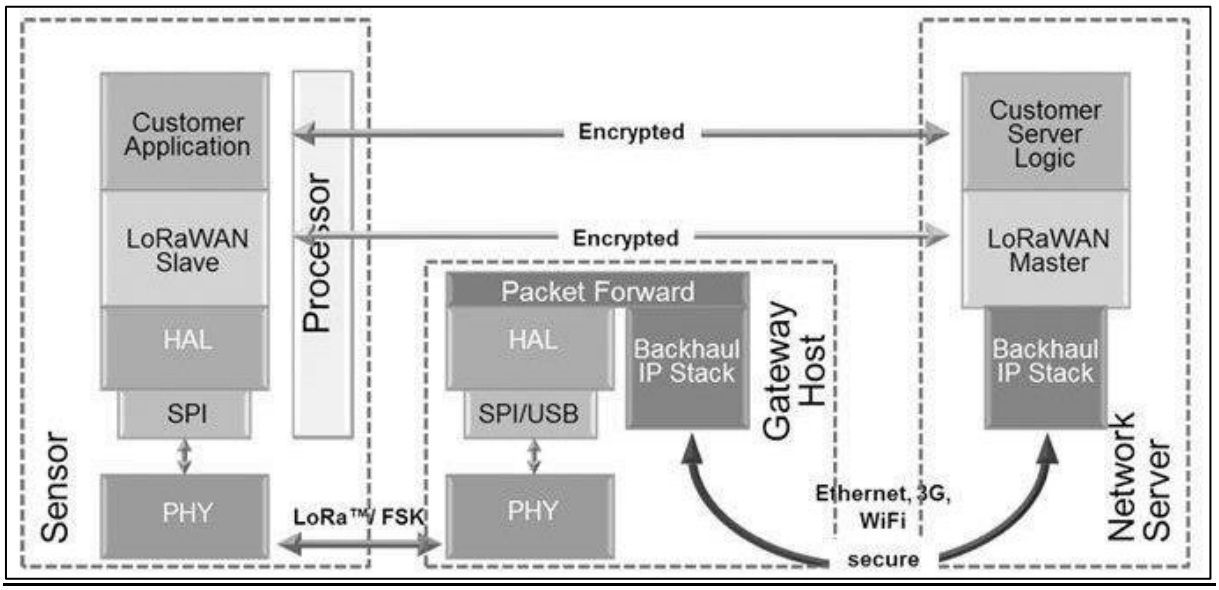
OTAA message exchange (Copyright Robert Lie – mobilefish.com)

The NwkSKey is used between the node and the network server to ensure the MIC of all data messages (ensure data integrity). The AppSKey is used to encrypt/decrypt the payload between the end-device and the application server.

Communicating with the gateways and the network server

After activation, the nodes are ready to send data the network. The first step before sending anything to the network is to encrypt the information with the AppSKey and ensure integrity of this message with the NwkSKey. After the payload has been encrypted and the MIC added to the message, the node will broadcast its data to all available gateways in the vicinity. This broadcasting is done through LoRa RF and it makes sure that the data has arrived to at least one gateway. Remember that in LoRa, we are speaking about a few dBm of power (which is relatively low compared to other technologies).

When the data has arrived at one (or more) gateway(s), it will be passed onto the network server. The gateway doesn't know what is inside the data (the payload), because it is encrypted, and the gateways don't have the keys to decrypt it.



Close-up of the link between the node, the gateway and the network server

The only purpose of the gateway is to identify the network where the node belongs to (ex : TheThingsNetwork) by looking at the NetID. The information is then sent from the gateway to the network server via WiFi, Ethernet, 4G, 3G, … One important thing to note is that the most recent versions of LoRaWAN allow roaming. This means that if my packet is sent to the wrong network (because the gateways that received my packet are registered to a different network than mine), the network server can redirect my packet to the right network.

As we have seen previously, several gateways can receive the same packet from my end-device. This is not a problem, as the network server will filter the received packets to only keep one.

When the network server finally receives the data, it will use the NwkSKey (which is the same as the node's Session Key) to check the integrity of the received data. The packet will be forwarded to the correct application server by looking at its DevAddr (which is like the IP address of the node) and it will be decrypted by the AppSKey. This will allow us to get the payload from the end-device (temperature, pressure, timestamp, alarm, fill-level, …)

MQTT broker

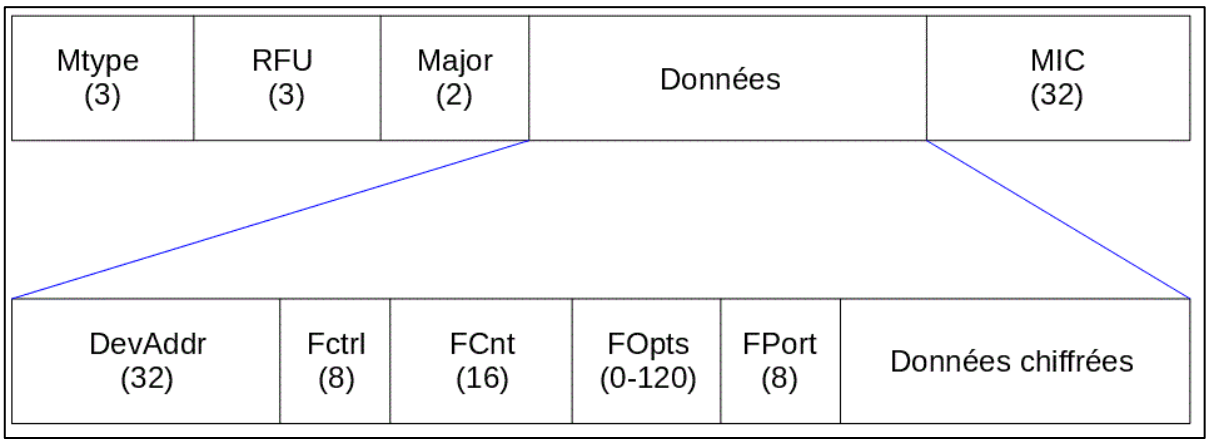
MQTT stands for "Message Queuing Telemetry Transport" and it is an application protocol. It is also known as a publish/subscribe protocol and it is designed to be lightweight and to minimise network bandwidth and resource requirements for the devices. It is well-suited for high-latency or unreliable networks (the last one is especially true for the LoRa link) and it attempts to ensure some reliability of packets delivery. MQTT is different from HTTP, where the last one is a request/response protocol used by the browser to navigate on the web pages and request content (text, image, video, …).

The MQTT broker is the centralised system that hands-over the messages to the subscribed clients (our application servers). This system is also able to store data temporarily in a buffer and push it to the subscribers. MQTT will only give the last received messages it has. It will not store them.

In an IoT architecture, the MQTT can be placed between the network server and the application server and simply relay the uplinks and downlinks between the nodes and the application servers. However, for some networks (like TTN), there is an additional application server that belongs to TTN and that decrypts the payload and then sends it to the MQTT broker.

To receive messages on a certain topic, you need to subscribe to it (ex : IoT/home/node1 -> my application server will receive messages from node1 through the broker).

LoRaWAN data frame



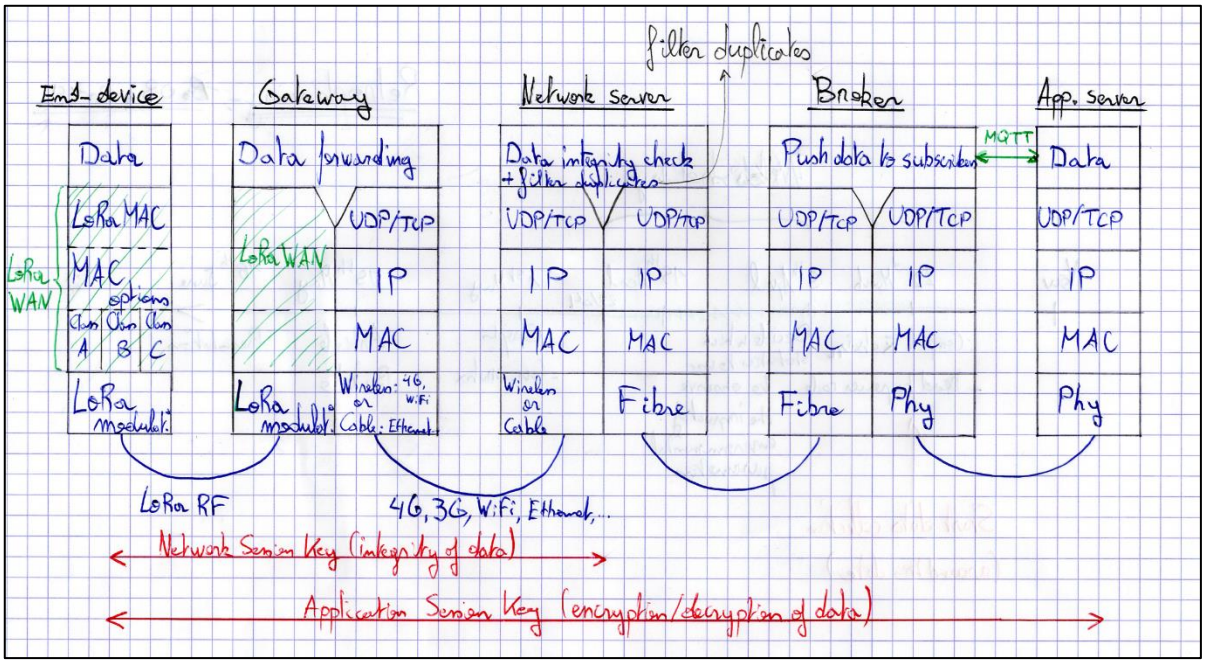
LoRaWAN data frame

The figure above shows the fields (the lengths are in bits) of the LoRaWAN data frame. The new ones (which we have not seen yet) are:

- Mtype: indicates the kind of message (uplink or downlink)
- RFU: this field is reserved for future implementation
- Major: the version of the LoRaWAN protocol that is used

- Fctrl: allows to adapt the data rate and is used for ACK messages
- FCnt: a counter that increments for every sent packet
- FOpts: this field is used to pass the MAC commands
- FPort: application or service port where the packet belongs to

Summary of the transmission chain (LoRa/LoRaWAN side)



Transmission chain between the node and the app. server (OSI model)

When the node sends a packet to the application server, it is called an uplink. When the application server sends a message to the node, it is called a downlink.

Exporting data from the database to a .csv file

It is possible to dump data from an SQL database to .csv file. One of the tools that will be presented here is sqlite3. First, the user must connect to its SQL database and type the "headers on" command to get the headers for the .csv file. Then, the user instructs the tool to set the output mode to csv and then he sends the output to a .csv file. After these steps, the user can issue the query to select the data that he wants from the table (ex: SELECT dev_eui, SELECT rssi, …)

Bibliography

Difference between LoRa and LoRaWAN

https://www.quora.com/What-is-the-difference-between-Lora-and-LoraWan

https://enablingsupport.zendesk.com/hc/en-us/articles/205089362-What-is-the-difference-between-LoRa-and-LoRaWAN-

Pictures of a LoRa/LoRaWAN transmission

https://www.postscapes.com/wp-content/uploads/2018/03/lora.jpg

https://cdn.mikroe.com/blog/2016/02/lorawan.jpg

Activation process and communication between gateways and network server

https://www.youtube.com/watch?v=KrNDOBzhxeM

https://lora-alliance.org/sites/default/files/2018-11/20181114_NetID_FAQ.pdf

https://www.thethingsnetwork.org/docs/lorawan/security.html

MQTT broker

http://mqtt.org/faq

https://www.atys-concept.com/blog-de-la-performance/articles-cybersecurite-et-reseaux/m2m-iot-lora-mqtt-comment-profiter-simplement-des-objets-connectes/

http://www.steves-internet-guide.com/mqtt-publish-subscribe/

LoRaWAN data frame

https://fr.wikipedia.org/wiki/LoRaWAN#Le_protocole_LoRaWAN

LoRa/LoRaWAN and the OSI model

http://www.rfwireless-world.com/Tutorials/LoRa-protocol-stack.html

Export database to a .csv file

http://www.sqlitetutorial.net/sqlite-tutorial/sqlite-export-csv/