Projet Avancé en Systèmes Embarqués conception d'objets connectés LoRa et interfaçage à TTN

Troisième année électronique - SE

Rédigé le 31 janvier 2019

par LAMRAOUI Mouad, HERBRETEAU Sylvain, TOURARI Oula & JAMAI Imane

Table des matières

1 Introduction **2**

2 Cahier de charge **3**

3 Etat de l'Art sur le protocole LoRaWAN **4**
3.1 Présentation générale 4
3.2 LoRa LoRaWAN 5
3.2.1 ABP - Activation By Personalization 6
3.2.2 OTAA - Over The Air Activation 7
3.2.3 Choix des bandes de fréquences 8

4 Présentation du matériel **9**
4.1 Carte cible Raspberry Pi 9
4.2 Distribution Linux pour RaspberryPI 10

5 LMIC et The Things Network **11**

6 Le protocole MQTT **14**
6.1 Présentation générale 14
6.2 Utilisation de l'application MQTTBox 14

7 L'interface graphique sous Qt **17**
7.1 Présentation de Qt et des librairies utilisées. 17
7.2 Présentation de l'application 19

8 Conclusion **21**

9 Références **22**

1 Introduction

Ce rapport reprend l'ensemble des travaux réalisés dans le cadre du projet avancé système embarqué du semestre 9 proposé par M. Patrice Kadionik, directeur de l'option Système Embarqué, Électronique 3ème année.

L'ensemble des travaux présentés ci-dessus ont été menés conjointement par les élèves ingénieurs suivants :
— Imane Jamai
— Oula Tourari
— Mouad Lamraoui
— Sylvain Herbreteau

Nous tenons à remercier en particulier M. Patrice Kadionik pour ce projet qui nous a permis de mettre en place de nombreuses techniques et compétences acquises lors de notre cursus à l'ENSEIRB-Matmeca, nous permettant ainsi de réaliser une application complète axée sur les objets connectés.

Le monde de l'internet des objets devient de plus en plus important et les prévisions montrent que le nombre d'objets connectés à travers le monde sera décuplé d'ici 2020, passant de plusieurs millions à plusieurs milliards. L'utilisation des réseaux LPWAN (Low Power Wide Area Network) devient donc incontournable. Ce type de réseau que l'on retrouve essentiellement dans ce cas de figure regroupe différents protocoles de communication, dont LoRaWAN.

L'objectif principal de ce projet est de créer un objet connecté en se servant de la plateforme Raspberry Pi 3. Il sera connecté par le biais du réseau LoRaWAN, il récolte différent données comme la température, humidité et pression et les envoie par le réseau LoRaWAN sur internet. Les différentes données vont pouvoir être récupérées sur une Interface graphique par le biais du protocole MQTT.

Pour résumer toutes les tâches accomplies lors de ce travail, nous commencerons par donner le cahier des charges et présenter généralement le protocole LoRaWAN. Ensuite, nous évoquerons tout ce qui a pu être effectué au cours du projet, de l'implantation du réseau jusqu'au réalisation de l'interface graphique. Enfin, nous conclurons en réalisant un bilan de ce qui a été réalisé et en donnant des évolutions possibles sur le sujet.

Le projet est hébergé sur un repository Github. On peut le récupérer avec le lien :

https://github.com/SylvainHerbreteau/IoTProject.git

2 Cahier de charge

Ce projet a pour objectif de présenter la conception par prototypage rapide d'un objet connecté au réseau LoRaWan. Comme nous avons pu le voir en cours, le logiciel libre et le matériel libre sont les composants de base essentiels dans cette conception. Nous n'allons pas réaliser la conception matérielle d'un objet connecté, nous partirons donc de matériels conçus pour le prototypage rapide et nous nous attacherons à la conception logicielle. Les objets connectés contiennent généralement un système d'exploitation.
On retrouve dans l'immense majorité des cas un socle Linux embarqué. L'objet est construit à la base d'une carte Raspberry Pi 3B+ complétée d'une carte fille Sence Hat, incluant différents capteurs et un afficheur, et qui contient une distribution Linux embarqué adaptée et optimisée comprenant les pilotes de périphériques nécessaires pour contrôler les capteurs de la cartes Sense HAT.
Le but est de pouvoir transmettre les données de la carte à The Thing Network via une passerelle en utilisant le protocole LoraWan, puis les récupérer depuis le cœur de réseau TTN et enfin les transférer vers notre base de données et notre interface de visualisation. Plusieurs approches sont possibles pour collecter nos données, notamment une intégration HTTP et MQTT. On privilégie dans notre projet la deuxième option qui est rapide et facile à mettre en œuvre.

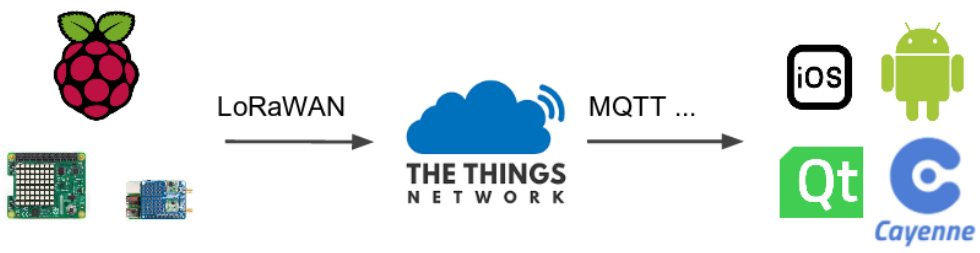

FIGURE 1 – Schéma de principe du projet

3 Etat de l'Art sur le protocole LoRaWAN

Après avoir correctement défini notre cahier des charges ce qui nous permettra d'avoir un fil conducteur pour la suite du projet, nous pouvons passer à la présentation de LoRaWAN.
Cette étape est primordiale et indispensable si l'on souhaite pouvoir déployer un réseau respectant ce protocole dans de bonnes conditions, pour savoir si l'on suit bel et bien les règles énoncées et comprendre concrètement ce qui se passe au sein de l'architecture. Il faut donc être au point sur LoRaWAN.

3.1 Présentation générale

Mis en place par la LoRa Alliance, LoRaWAN est un protocole de communication dédié à l'internet des objets puisque ses principales caractéristiques sont d'être peu coûteux à mettre en place et d'avoir une faible consommation énergétique. Son principal avantage est de permettre à des équipements de communiquer sur de longues distances, à savoir plusieurs kilomètres, et que ceux-ci aient une autonomie conséquente. Le débit des communications n'est pas l'intérêt principal du protocole comme plus généralement pour l'internet des objets.
D'un point de vue électronique, la certification s'appuie sur une technique de modulation par étalement de spectre appelée LoRa. Grâce à cet aspect, des communications ayant des débits différents ne peuvent pas interférer entre elles, ce qui permet à une station de base de gérer un millier d'équipements en théorie. L'une des forces de LoRaWAN provient de ce paramètre et surtout du fait que cela est géré automatiquement par le serveur réseau par un procédé de « débit adaptatif » (ADR). On optimise la durabilité de chaque équipement ainsi que la capacité du réseau en choisissant le débit idéal.

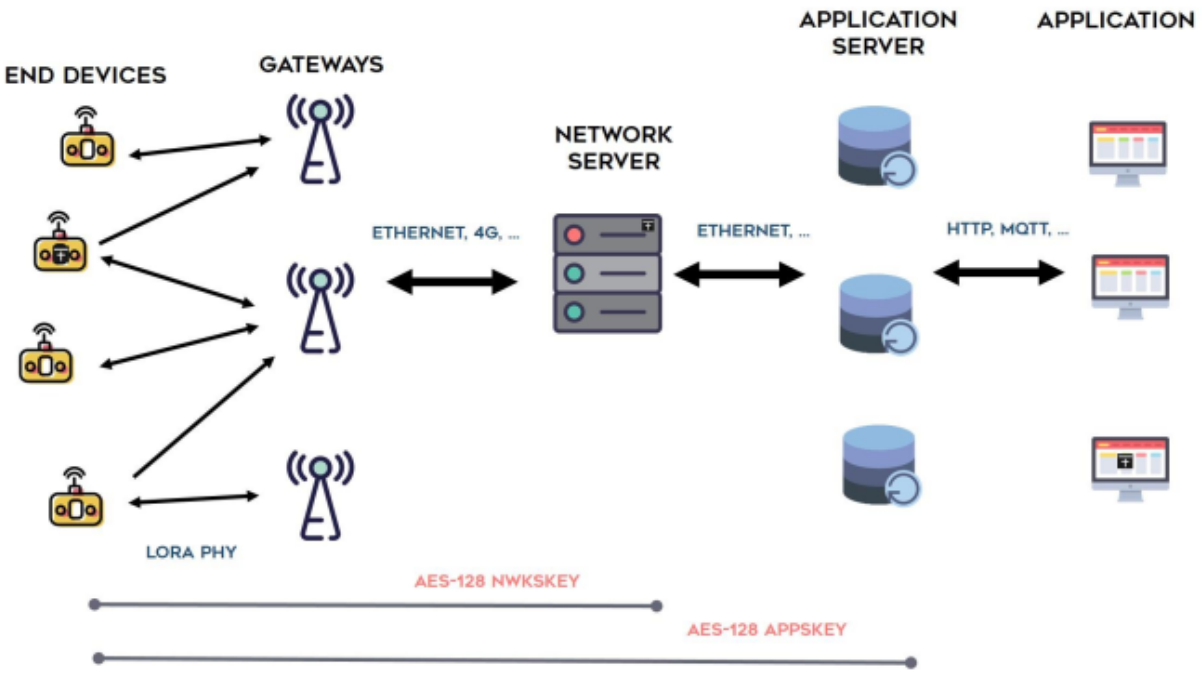

Figure 2 – Architecture réseau LoRaWAN

3.2 LoRa LoRaWAN

LoRaWAN est un protocole de communication longue distance et consommant peu d'énergie, on peut comparer cette technologie avec la technologie WiFi qui est lui aussi une technologie de communication sans fil. La WiFi consomme en moyenne 140mA pour une communication fonctionnant sur une distance de 100m, alors que la technologie LoRa ne consomme en moyenne que 40mA pour une communication fonctionnant sur une distance pouvant aller jusqu'à 15Km. Bien sur les tests qui ont été effectués pour le LoRa ont été réalisés dans des conditions optimales, c'est-à-dire qu'elles ont été réalisés sur des terrains généralement plats et sans bâtiments, le tout avec des conditions météorologiques adaptés pour des tests optimaux (sans vent, sans pluie...). En zone urbaine, comme par exemple l'école, le rayon de communication est d'environ 3Km. L'inconvénient de la technologie LoRa et qu'en dépit de la distance d'envoi la quantité de données envoyé et le débit est faible.

Les Ends-devices, ce sont généralement des cartes électroniques, tel que des cartes Arduino ou encore des Raspberry Pi, qui sont associées avec des modules LoRa et un ou plusieurs capteurs. La gateway LoRa, cette passerelles permet de recevoir les données envoyées par tout autre équipement LoRa, après la réception des données la gateway transfère ces dernières sur un système de raccordement.

Les passerelles LoRa sont connectées au serveur du réseau en utilisant des connections IP standard. Le Serveur, le serveur du réseau LoRa gère le flux de données arrivant des gateways. Le serveur permet d'éliminer les paquets dupliqués et adapte aussi le débit des données.

La technologie LoRa peut être une liaison bidirectionnelle, elle utilise une modulation par étalement de spectre, qui font qu'un signal généré avec une certaine bande-passante est étalé sur le spectre de fréquence, cela permet de :

•Sécuriser les communications.

•Une plus grande résistance aux interférences et aux bruits.

•Limiter la consommation.

L'image suivante compare la technologie LoRa et son protocole LoRaWAN avec la technologie WiFi : Sur cette image nous pouvons voir que pour la technologie LoRa il y a deux couches, la couche physique

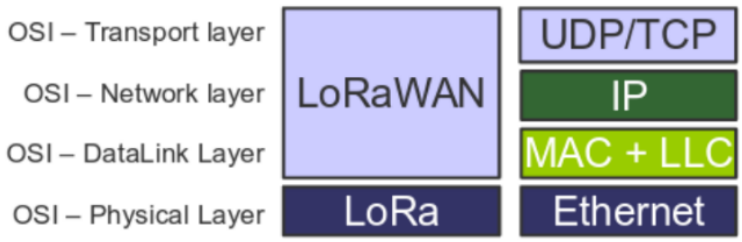

FIGURE 3 – Les couches des deux protocoles

(LoRa) et la couche MAC (LoRaWAN) qui regroupe la gestion des données, du réseau et du transport de données.

La couche physique configure et gère tous les aspects du signal transmis entre les ends-devices et la gateway LoRa. C'est-à-dire qu'elle gère les fréquences, le mode de modulation, les niveaux de puissance et les largeurs de bande. LoRa utilise une forme de modulation à étalement de spectre, elle permet de rendre le signal moins sensible aux fluctuations sélectives en fréquences. Le signal est transmis sur une bande de fréquences beaucoup plus large que nécessaire.

La modulation peut être modifiée en FSK ou LoRa. Le Spreading Factor (SF- Facteur d'étalement) représente la longueur du code envoyée. Pour éviter les interférences, les codes d'étalements sont orthogonaux, cela permet la transmission simultanée de plusieurs canaux, chacun étant étalé en temps et en fréquence.

Le tableau suivant représente la plage des valeurs accessible avec un module LoRa quelconque :

Spreading Factor	Chips/Symbol	LoRa demodulator SNR	LoRa MAC (Class compatibility
6	64	-5 dB	No
7	128	-7.5 dB	Yes
8	256	-10 dB	
9	512	-12.5 dB	
10	1024	-15 dB	
11	2048	-17.5 dB	
12	4096	-20 dB	

FIGURE 4 – Tableau des différentes valeurs du SF et du SNR

Nous pouvons voir ici que plus le SF est important, plus le SNR est faible, la capacité du récepteur LoRa à recevoir des signaux avec un SNR négative augmente sa sensibilité et son bilan de liaison. Le débit des données étant assez faible, plusieurs bandes passantes sont nécessaires aux multiples communications. On retrouve celles comprises entre 7.8kHz et 500kHz (elles ont des valeurs précises), les deux bandes passantes les plus utilisés sont 125kHz et 500kHz. La puissance utilisée pour l'envoie d'un message est adaptative, c'est-à-dire qu'elle dépend de la taille des données, du débit d'envoi, la portée d'émission... Un algorithme est donc utilisé pour déterminer le niveau de puissance requis pour l'envoie du message. L'adaptabilité de la puissance de transmission permet donc de maximiser la durée de vie de la batterie. Ces communications, entre les divers dispositifs, utilise des canaux de fréquences différents, et utilisent des débits de données distincts. Donc la technologie à étalement de spectre permet la communication avec des débits de données divers et garantie que les communications se déroulent sans interférences. Ce protocole utilisé par la technologie LoRa a été conçu pour fonctionner de façon optimale avec des dispositifs alimentés par batterie, mais aussi sans fil, qui nécessite une connexion à faible coût.

Les réseaux LoRaWAN utilise une topologie de réseau en étoiles, c'est-à-dire que les ends-devices échangent avec la gateway. Les ends-devices sont ressemblé dans trois classes de dispositifs (les classe A,B et C), la classe du dispositif change en fonction du types de communications nécessaires. La solution qui pour le moment est utilisé dans la plupart des cas est la classe A car les deux autres classes ne sont implémentés que sur quelques modules spécifiques.

La classe A est une implémentation d'end-devices qui utilise, du point de vue de l'utilisateur, des communications unidirectionnelles. L'utilisateur peut recevoir des données de ces dispositifs embarqués mais ne peut pas piloter les dispositifs à distance. Mais, les dispositifs qui communiquent avec la gateway reçoivent et traitent des requêtes de la gateway, ces requêtes ont pour but d'ajuster le débit, l'étalement du spectre et la puissance d'envoi du module. Elles permettent le maintien d'une bonne connexion entre les dispositifs et la gateway.

3.2.1 ABP - Activation By Personalization

Ce mode de connexion utilise six informations qui permettent d'identifier et de permettre la connexion d'un module. Ces informations sont :

• AppEUI, cet identifiant est unique à l'application, il permet de regrouper les modules LoRa. Cette adresse est sur 64 bits, il permet de classer les périphériques par application, ce paramètre est bien sur modifiable sur la plupart des modules vendu sur le marché.

• DevEUI, cet identifiant rend chaque module unique. Cet identifiant est programmé en usine, et donc n'est normalement pas modifiable.

• AppKey, cet identifiant secret qui n'est partagé qu'entre le module et la gateway. Cet identifiant est utilisé pour calculer les clefs de chiffrement utilisées pour les communications entre le module et la gateway.

Il y aussi les clés de chiffrements qui sont préprogrammé dans ce mode, l'utilisateur les fixes et elles ne sont donc pas calculables. Ces clefs sont les suivantes :

• DevAddr, cet identifiant est l'adresse du module, il sert à l'identification du module dans le réseau.

• NetSKey, cet identifiant est une clefs de chiffrement entre le module et la gateway qui est utilisé pour les transmissions et valider l'intégrité des messages envoyés.

• AppSKey, cet identifiant est aussi une clefs de chiffrement entre le module et la gateway qui est utilisé pour les transmissions et valider l'intégrité des messages envoyés.
Ce mode de connexion possède un avantage et un inconvénient, l'inconvénient de ce mode est que les clefs de de chiffrement permettant les communications avec le réseau sont pré-configurées dans l'objet ce qui rend la sécurité plus faible. L'avantage de ce mode est que le raccordement au réseau est très simple ce qui rend l'objet rapidement opérationnel.

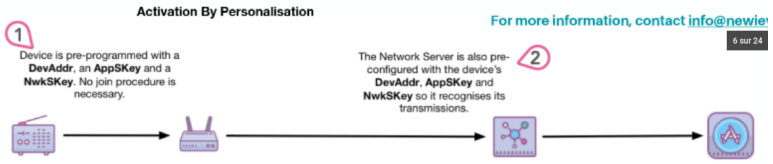

FIGURE 5 – Schéma explicatif du mode ABP

3.2.2 OTAA - Over The Air Activation

Ce mode de connexion utilise six informations qui permettent d'identifier et de permettre la connexion d'un module. Ces informations sont :
• AppEUI, cet identifiant est unique à l'application, il permet de regrouper les modules LoRa. Cette adresse est sur 64 bits, il permet de classer les périphériques par application, ce paramètre est bien sur modifiable sur la plupart des modules vendu sur le marché.
• DevEUI, cet identifiant rend chaque module unique. Il est programmé en usine, et donc n'est normalement pas modifiable.
• AppKey, cet identifiant secret n'est partagé qu'entre le module et la gateway. Cet identifiant est utilisé pour calculer les clefs de chiffrement utilisées pour les communications entre le module et la gateway.

Il y aussi les clés de chiffrements qui sont partagé lors d'un début de communication. Si l'objet est autorisé à rejoindre le réseau, le module et la gateway vont échanger des clefs de chiffrement qui seront propre à cette communication, la gateway va donc attribuer ces clefs de chiffrement au module, pour la suite de la communication avec le module. Ces clefs sont les suivantes :
• DevAddr, cet identifiant est l'adresse du module, il sert à l'identification du module dans le réseau.
• NetSKey, cet identifiant est une des clefs de chiffrement entre le module et la gateway qui est utilisé pour les transmissions et valider l'intégrité des messages envoyés.
• AppSKey, cet identifiant est aussi une des clefs de chiffrement entre le module et la gateway qui est utilisé pour les transmissions et valider l'intégrité des messages envoyés.
Ce mode de connexion possède un avantage et un inconvénient, l'inconvénient de ce mode est que l'objet doit implémenter ce mécanisme pour effectuer des communications avec le réseau, ce qui introduit une complexité supplémentaire. L'avantage de ce mode est que la création des clefs de chiffrement est effectué par la gateway, ce qui renforce la sécurité des communication effectué sur le réseau. Ce mode est donc le mode le plus utilisé dans le mode de l'IoT, en ce qui concerne les réseaux LoRaWAN car elle est plus sécurisé.

3.2.3 Choix des bandes de fréquences

Pour savoir quelle bande de fréquences le module doit utiliser pour communiquer à un instant donné, le module connecté au réseau utilise une procédure "d'écoute avant transmission" afin d'évaluer si un canal de communication est disponible, si la bande de fréquence testée n'est pas libre à l'instant donnée, alors le module passe à la bande de fréquence suivante et répète l'opération. Chaque module change de bande de fréquence à chaque communication de façon pseudo-aléatoire, ce qui permet de rendre les réseaux plus robuste vis-à-vis des interférences. Pour permettre une bonne communication sur l'ensemble du réseau, tous les équipements doivent gérer le facteur d'étalement, le débit et la puissance d'émission. Le moyen utilisé par les réseaux LoRaWAN est Adaptive Data Rate(ADR).
Tout d'abord voyons plus en détail comment le réseau gère les changements de canal et comment il choisit le canal le plus adapté pour garantir une communication fiable. Les modules qui veulent émettre des données effectuent une détection d'activité de canal, cette détection permet de connaître l'état d'une bande fréquence avant toute transmission. Après cette opération, les modules voulant émettre utilisent la procédure LBT qui garantit un RSSI. Cette mesure permet de calculer la puissance du signal elle est exprimée en dBm, moins cette valeur est élevé plus la qualité de la communication, et donc sa fiabilité, et grande. C'est pour cela que si la valeur calculée est supérieure à-90 dBm alors le module sélectionné une nouvelle bande de fréquence et recommence l'opération. Suite à l'accès à une bande de fréquence une trame PDU PHY est envoyée par la gateway. Elle précise les paramètres physiques qui vont servir à la prochaine communication.
Pour toutes les communications suivantes les modules devront sélectionner une nouvelle bande de fréquence en respectant toujours la même démarche. Voyons comment la couche LoRaWAN gère les paramètres de communication de la couche physique. Le mécanisme d'ADR utilise des commandes MAC pour régler ces paramètres. Cette modification ce fait de façon automatique dès lors qu'un module se connecte à une gateway pour une transmission, ces opérations sont réalisées pour chaque session.

Le tableau suivant vous montrera quelques commandes qui sont effectuées lors de ces échanges qui ont pour but de configurer le module :

Commande	Node to Gateway	Gateway to Node
LinkCheckReq	X	
LinkCheckAns		X
LinkADRReq		X
LinkADRAns	X	
DevStatusReq		X
DevStatusReq	X	

Table 1 – Commandes MAC entre Node et Gateway

Les commandes LinkADRReq et LinkADRAns sont les commandes MAC relatives à l'ADR, ces requêtes sont envoyées après la confirmation de la connexion au réseau. La gateway choisie donc les paramètres adéquats pour la communication à venir.

4 Présentation du matériel

4.1 Carte cible Raspberry Pi

La carte Raspberry Pi ou RPi est une carte bon marché largement utilisée pour le DIY (Do It Yourself) afin de développer de petits systèmes embarqués ou des objets connectés. Le modèle mis en œuvre dans ce projet est la carte Raspberry Pi 3B+. Notre carte RPi possède ainsi les éléments suivants :

- Un SoC (System on Chip) Broadcom BCM2837 avec un processeur quadricœur ARM Cortex-A53 à 1,2 GHz.
- 1 Go de RAM.
- 4 ports USB.
- Sortie vidéo HDMI.
- Sortie audio HDMI et Jack 3,5 mm.
- Support microSD.
- Ethernet 10/100 Mb/s, Wifi 802.11n et Bluetooth 4.1.
- 7 E/S GPIO, 1 UART, 1 bus I2C et 1 bus SPI. L'image suivante présente la carte cible RPi 3B+ :

FIGURE 6 – Raspberry PI

La carte RPi 3B+ est complétée d'un HAT, le Sense HAT pour ajouter différents capteurs et un affichage. La carte Sense HAT possède ainsi les éléments suivants :

- 1 gyroscope.
- 1 accéléromètre.
- 1 capteur d'accélération (2/4/8/16 g).
- 1 magnétomètre (4/8/12/16 Gauss)
- 1 baromètre (260 - 1260 hPa).
- 1 capteur de température (0-65°C).
- 1 capteur d'humidité relative (20-80• 1 affichage par leds avec une matrice de 8x8.
- 1 joystick 5 boutons.

L'image suivante présente la carte Sense HAT :

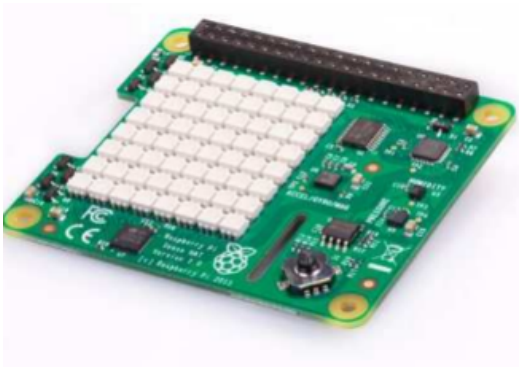

FIGURE 7 – SenseHAT

La carte RPi 3B+ est aussi complétée d'un Chistera PI shield pour connecter l'objet au réseau LoraWAN afin de pouvoir envoyer des paquets Lora :

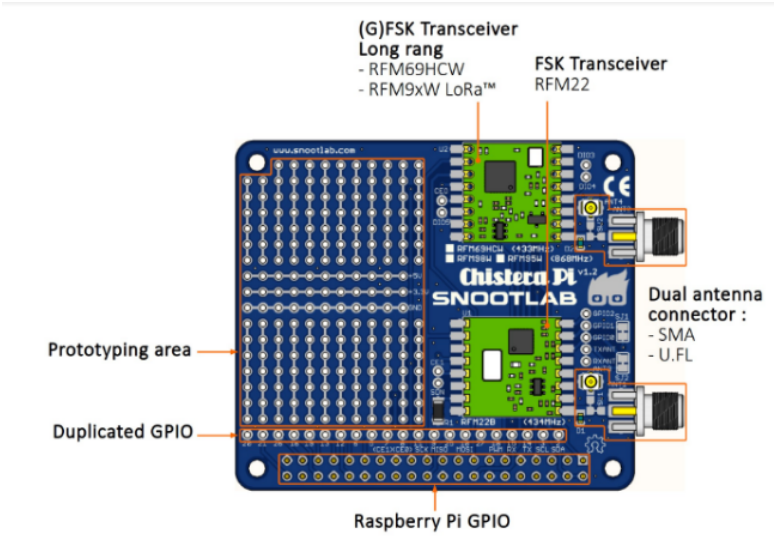

FIGURE 8 – ChisteraPI

Le kit comprend :

- Shield avec ses modems RFM22 et RFM95
- 2 antennes 434 et 868 MHz
- Connecteur 2x20 broches et SMA
- Connecteurs uFL-SMA

4.2 Distribution Linux pour RaspberryPI

Notre objet connecté contient une distribution Linux officielle de la carte cible Raspberry Pi comprenant tous les outils nécessaires pour un développement natif directement sur la carte RPi : La distribution Raspbian, un mot-valise formé par la fusion des mots "Raspberry Pi" et "Debian", il s'agit d'une modification de Debian spécifiquement adaptée pour les systèmes sur puce de type ARMv6.

5 LMIC et The Things Network

Il existe une librairie en C qui assure l'implémentation de la couche de liaison de données du réseau LoRa. Elle contient un ensemble de fonctions et de structure de données permettant d'envoyer les données plus facilement.

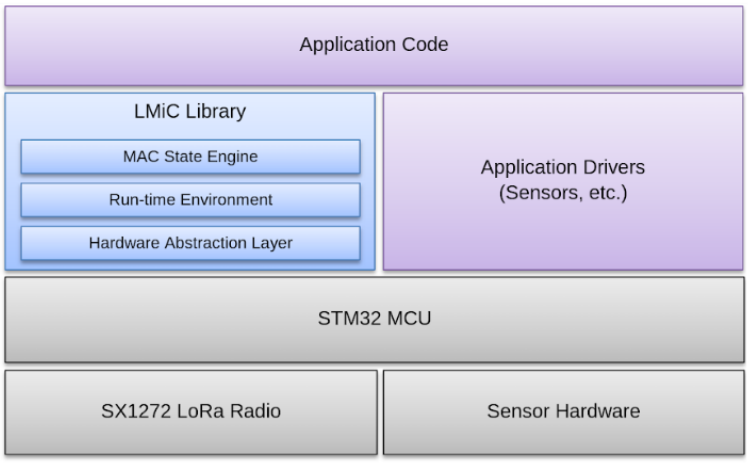

FIGURE 9 – LoRa et LMIC

Pour ce, LMIC permet de définir des jobs qui ne sont décrites de façon similaire à la description de fonctions en C mais qui peuvent être manipulées comme des objets par les fonctions run-time lors de l'exécution du programme. Le développement de l'application est réalisé à partir de jobs auxquels on fait appel dans la fonction main.

Mais avant il a fallu créer l'application sur TTN :

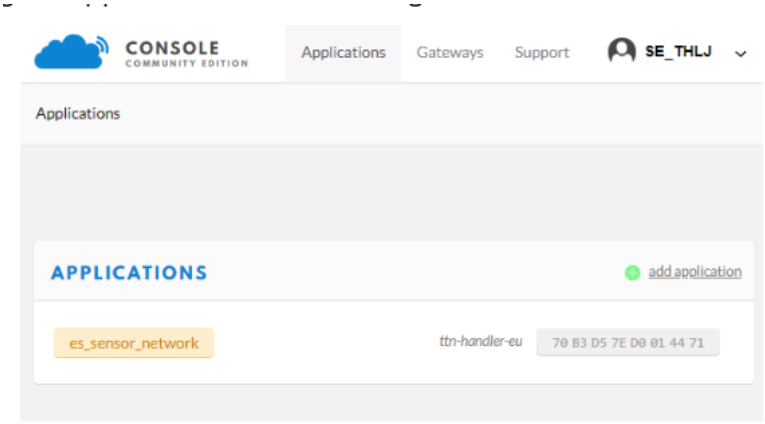

FIGURE 10 – Création de l'application sur TTN

On définit donc un job initial permettant d'envoyer les données acquises, une fonction call-back permettant de gérer les états d'interruptions ou d'échec d'envoi onEvent() que l'on distingue grâce aux macros suivants : EV-JOINING, EV-JOINED, EV-JOIN-FAILED, EV-REJOIN-FAILED, EV-TXCOMPLETE, EV-RXCOMPLETE

L'acquisition des données est réalisée par un script python. On y importe la librairie liée au module sense Hat ce qui donne la possibilité de déployer les fonctions get() de chacune des données météo. Celles-ci sont ensuite écrites sur un fichier texte que nous lisons dans le job initial de notre application.

Afin d'améliorer la lisibilité des données sur la plateforme TTN mais aussi pour le traitement et la récupération des données au niveau applicatif, le formatage s'avère nécessaire. Un des format les plus utilisés est CAYENNE LPP (Low Power Payload). Le formatage est ainsi établi par remplissage des champs d'une classe appelée CayenneLPP définie dans "CayenneLPP.h".

Il suffit de construire la classe avec la taille correspondant à la somme des tailles des champs que l'on veut remplir et qui sont prédéfinis (LPP-TEMPERATURE-SIZE, LPP-RELATIVE-HUMIDITY-SIZE, LPP-BAROMETRIC-PRESSURE-SIZE) Les champs sont ensuite modifiés grâces aux méthodes : addTemperature(), addRelativeHumidity(), addBarometricPressure(). Chacune demande deux arguments, le premier correspondant au numéro du canal et le deuxième à la donnée à saisir. Une fois formaté, il suffit d'extraire de buffer buf-LPP qui sera envoyé au serveur réseau, avec la méthode getBuffer().

Ce dernier est envoyé par la fonction LMIC-setTxData2 qui prépare le upstream afin qu'il soit envoyé. Ainsi, au lieu de recevoir une série de nombres ininterprétables, on spécifie dans la console TTN que les paquets attendus sont formatés par cayenne comme l'illustre la figure ci-dessous :

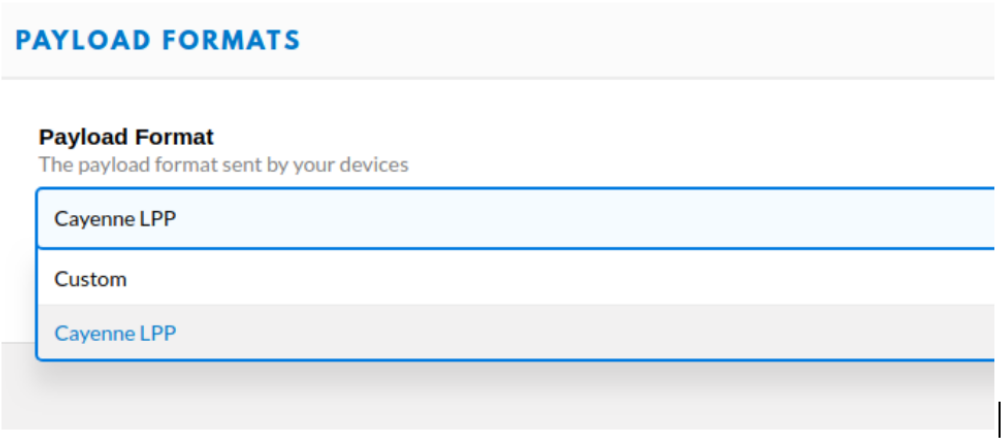

FIGURE 11 – Formats du payload

Les messages uplink reçus sont bien lisibles et il n'est pas nécessaire de connaître l'ordre avec lequel les données ont été envoyées :

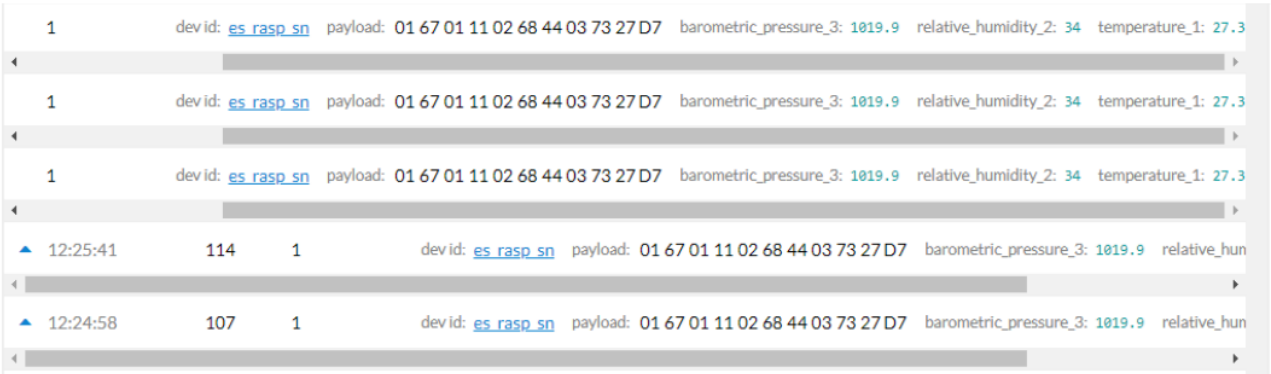

FIGURE 12 – Messages reçues sur TTN

Cayenne est un environnement de construction d'application IoT par simple glisser-déposer. Il permet aux développeurs, concepteurs et ingénieurs de rapidement prototyper et partager leurs projets d'appareils connectés. Cayenne a été conçu pour aider les utilisateurs à créer des prototypes d'application d'Internet des objets, puis à les mettre en production.Il est intégrable depuis la console TTN. Il y a plusieurs composants majeurs sur la plateforme :

• Cayenne Mobile Apps - Surveiller et contrôler à distance vos projets IoT à partir des applications Android ou iOS.

• Tableau de bord en ligne Cayenne - Utilise des widgets personnalisables pour visualiser des données, définir des règles, planifier des événements, etc.

Notre but était de construire un dashboard illustrant les mesures comme suit : L'utilisation de The

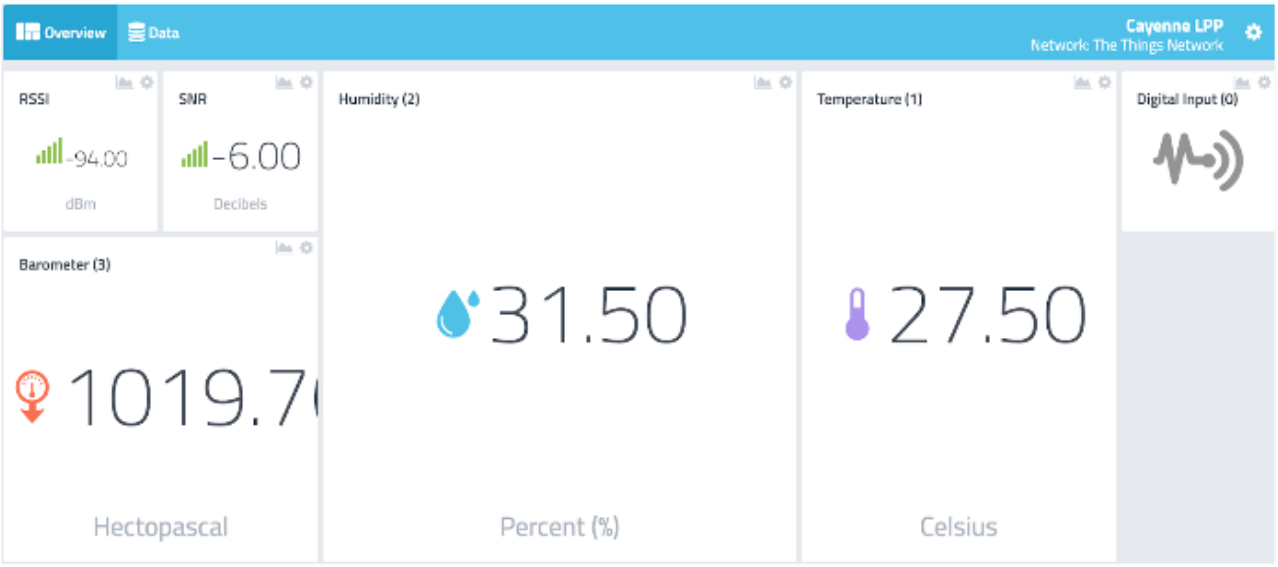

FIGURE 13 – Messages reçues Cayenne

Things Network combiné à Cayenne my devices fournit un environnement très pratique et puissant pour mettre à disposition des données issues de capteur ou pour piloter un système à distance en s'appuyant sur un réseau IoT LoRa longue distance et faible énergie. Le format Cayenne LPP peut représenter une certaine difficulté pour la mise en œuvre du programme dans le device. Cependant, la librairie CayenneLPP présentée plus tôt rend la tâche plus simple à réaliser.

6 Le protocole MQTT

6.1 Présentation générale

MQTT (Message Queuing Telemetry Transport) est l'un des protocoles les plus couramment utilisés dans les projets IoT, c'est un protocole de messagerie TCP/IP qui fonctionne sur le principe de souscription/ publication et qui a été développé à la base pour simplifier la communication entre les machines. Il permet aux clients d'échanger des messages à travers un serveur/broker MQTT. Les messages sont envoyés par des publieurs sur un canal appelé Topic. Ces messages peuvent être lus par les abonnés en se connectant au broker qui joue le rôle de médiateur entre deux clients. Chaque client peut souscrire à des topics particuliers. Lorsqu'un autre client publie un message sur un topic, le broker transfère le message à tous les autres clients qui y ont souscrit. Comme tout autre protocole d'internet, MQTT est basé sur des clients et un serveur, le serveur MQTT est appelé un "Boker".

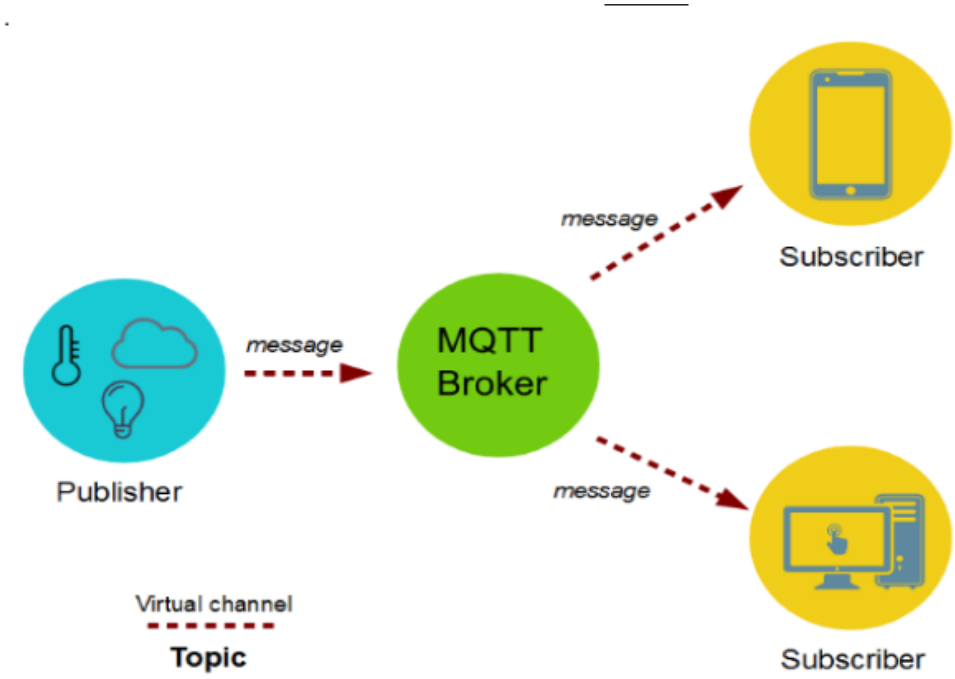

FIGURE 14 – Graph explicatif de MQTT

Ce graphique décrit l'architecture du protocole MQTT : le broker reçoit les messages du publieur et les envoie aux abonnés. Lors de la distribution des messages, le broker utilise le "Topic" pour filtrer les abonnés qui recevront le message. Un topic est un canal virtuel qui connecte un publieur à ses abonnés. Le broker MQTT gère ce topic.

En plus de simplifier la communication, MQTT a été conçu pour économiser au maximum la batterie des appareils mobiles sur lequel il est employé. On est 11 fois moins consommateur d'énergie pour envoyer des messages et 170 fois moins pour en recevoir que l'HTTP. MQTT est également 93 fois plus rapide que le HTTP.

De plus, sa petite taille, ses paquets de données minimisés et sa réponse en temps réel rendent le protocole idéal pour le monde de l'Internet des objets.

6.2 Utilisation de l'application MQTTBox

Pour une première approche, on utilise l'application MQTTBox pour créer notre client MQTT. MQTTBox est une application multiplateforme qui nous permet de souscrire à notre topic afin de récupérer les données du TTN. Une fois l'application est installée, on click sur la barre "create MQTT Client" pour ajouter un nouveau abonné :

Ensuite, on crée le client avec des paramètres de connexion spécifiques : nous définissons donc le nom du client, le protocole utilisé par le client pour se connecter au broker MQTT, l'hôte à connecter,

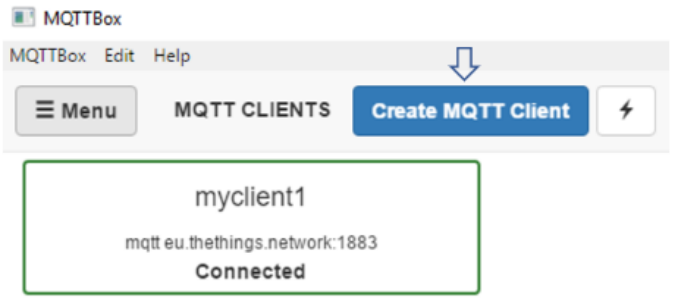

FIGURE 15 – Création client MQTTBox

qui est le TTN , avec le bon numéro de port, le nom d'utilisateur et le mot de passe de notre application qui sont déjà générés :

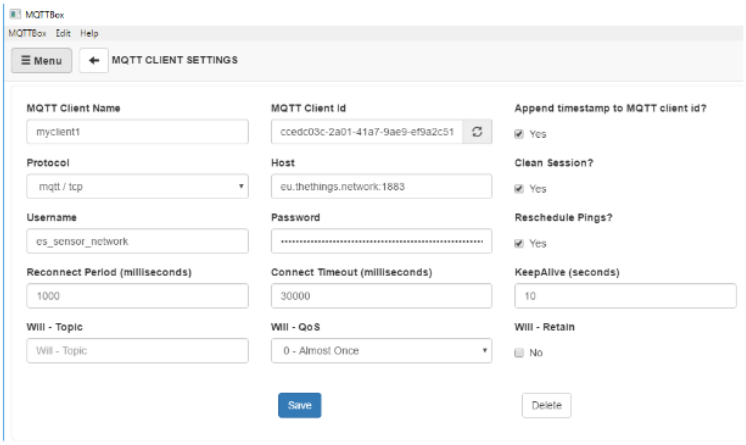

FIGURE 16 – Paramètres client MQTTBox

Une fois le client se connecte correctement au broker, la connexion nouvellement créée apparaît en vert pour indiquer que la connexion fonctionne :

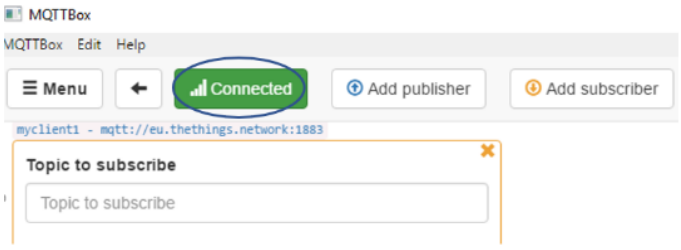

FIGURE 17 – Client created MQTTBox

Puis on précise le topic auquel le client sera abonné et après on click sur la barre “Subscribe” :

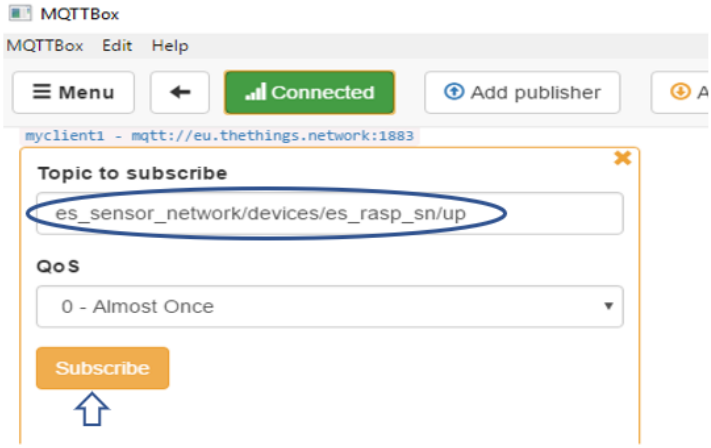

Figure 18 – Topic subscription MQTTBox

Lorsque le serveur du TTN, qui est automatiquement connecté au broker MQTT depuis la plateforme, publie un message sur ce topic, le broker le distribue à tous les clients connectés abonnés au sujet. Donc, une fois que notre client s’est connecté avec succès au broker nous recevons le message publié par le TTN sous format json.
On voit apparaître le payload reçu et qui comprend les données de la station météo.

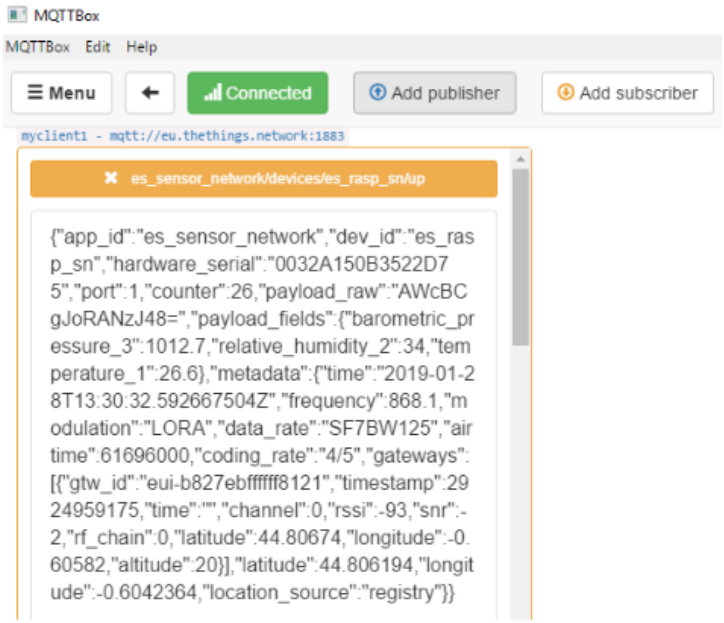

Figure 19 – Payload received MQTTBox

7 L'interface graphique sous Qt

L'utilisation des applications clients MQTT est très utile au départ pour tester le bon fonctionnement de l'objet connecté, cependant, elles ne permettent pas d'extraire et d'analyser les données reçues. L'utilisation d'une application web comme Cayenne permet de réaliser ce genre de traitements. En plus de cette solution, nous avons décidés de programmer une application en Qt. Le cahier des charges de l'application est le suivant :
•L'application doit pouvoir se connecter à un serveur MQTT et s'abonner à un topic particulier.
•Les messages reçues doivent être affichés sous une forme brut et sous la forme d'un "jolie" Json.
•Les données provenant des capteurs doivent être extraite du message et affichées sous la forme de graphes.

7.1 Présentation de Qt et des librairies utilisées.

Pour réaliser l'application, le framework Qt a été utilisé. Qt est un framework libre écrit en C++ permettant de programmer des interfaces graphiques. Les éléments graphiques sont appelés des widgets, chaque widget possède ses propres méthodes particulières permettant d'interagir avec lui même et avec les autres widgets de la fenêtre. Qt utilise le principe des signals/slots ; lorsqu'un événement se produit sur un objet (clic de la souris, fermeture de la fenêtre, connection au serveur ...), un signal est émis. Un slot est une méthode qui a la particularité de pouvoir être appelée lors de l'émission d'un signal. La connection entre un signal et un slot se fait grâce à la fonction connect. Le principe de signal/slot permet de facilement gérer les interactions entre les différents objets de notre application. Le schéma suivant résume l'utilisation des signaux/slots.

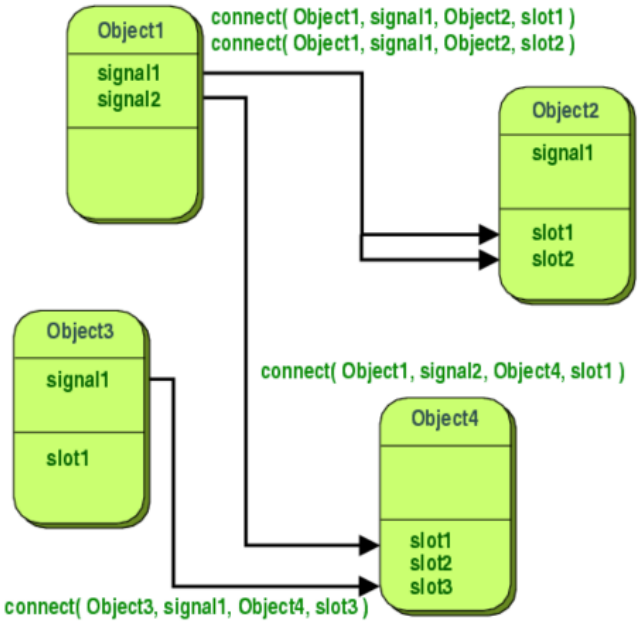

FIGURE 20 – Utilisation des signaux et slots

Pour faire les interfaces graphiques, l'outil QtDesigner a été utilisé, cette outil permet de réaliser à la main des interfaces graphiques sans une seul ligne de code, il suffit seulement de faire glisser les widgets dans l'interface et ils seront automatiquement ajoutés au projet. La création d'une fenêtre se fait via l'outil fournit par Qt, elle est composée d'un fichier .ui, d'un .h et d'un .cpp. La fenêtre Qt designer est la suivante, elle est composée de 5 parties :
•La liste des widgets à gauche
•L'aperçu de l'interface au milieu
•Un éditeur de signaux et slots au milieu en bas
•La liste des éléments de notre fenêtre en haut à gauche
•Les options de l'élément sélectionné en bas à gauche

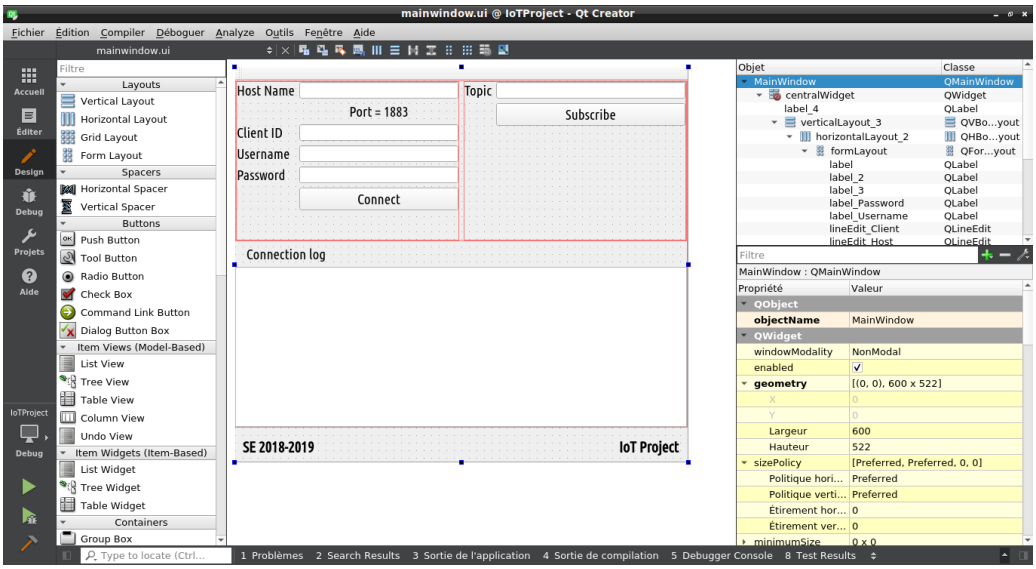

FIGURE 21 – Présentation de Qtdesigner

Les deux fenêtres de notre projet ont été réalisées grâce à l'outil Qt designer, tandis que les interactions entre les fenêtres a été fait dans les fichiers cpp.

Qt offre un ensemble de libraires permettant de programmer des interfaces graphiques, cependant, pour notre projet, d'autres librairies plus spécifique ont dû être installées. Certaines de ces librairies font parties des librairies de Qt mais doivent être installées séparément.

•QtMQTT (https ://github.com/qt/qtmqtt)

Cette librairie fournies par Qt offre toutes les méthodes permettant de se connecter à un serveur MQTT. Nous avons choisis cette libraries car elle est simple d'utilisation, elle permet d'utiliser les principes déjà vu de MQTT. Elle a l'avantage d'être intégrée à Qt, c'est à dire qu'elle utilise les principes de signaux et des slots.

•QtChart (https ://github.com/qt/qtcharts)

Pour afficher les données reçues sous la forme de courbe, nous avons utilisé la libraires QtChart, elle fournit tous les éléments nécessaires à la création de graphes telle que des classes permettant de gérer les axes, les légendes et les données en elle même.

•Json parser (https ://github.com/nlohmann/json)

Les messages reçues par le client sont sous la forme d'un Json qui a besoin d'être analysé afin d'extraire les données. Le parser que nous avons utiliser a l'avantage d'être en un seul fichier ce qui évite tous les soucis d'installation et rend notre application plus polyvalente.

7.2 Présentation de l'application

L'application est composée de 3 classes, 2 étant des fenêtres et une permettant de gérer les données reçues. Nous allons présenter plus en détails ses classes.

- Mainwindow (.ui, .h, .cpp)

Cette classe correspond à la première fenêtre de notre programme, l'interface est présentée en figure 22. L'utilisateur rentre les informations de connection dans les champs prévus à cet effet, il appuie ensuite sur le bouton connect. Cet appui envoie un signal qui active un slot, le slot extrait les informations, initialise le client et se connecte au serveur (méthode connectToHost()). A chaque interaction avec le serveur, des détails sont affichés dans le champ "connection log". Si la connection est établie entre le client et le serveur, une deuxième fenêtre s'ouvre. Le champ topic permet de renseigner le topic auquel le client doit s'abonner, là aussi des détails sont affichés dans les logs.

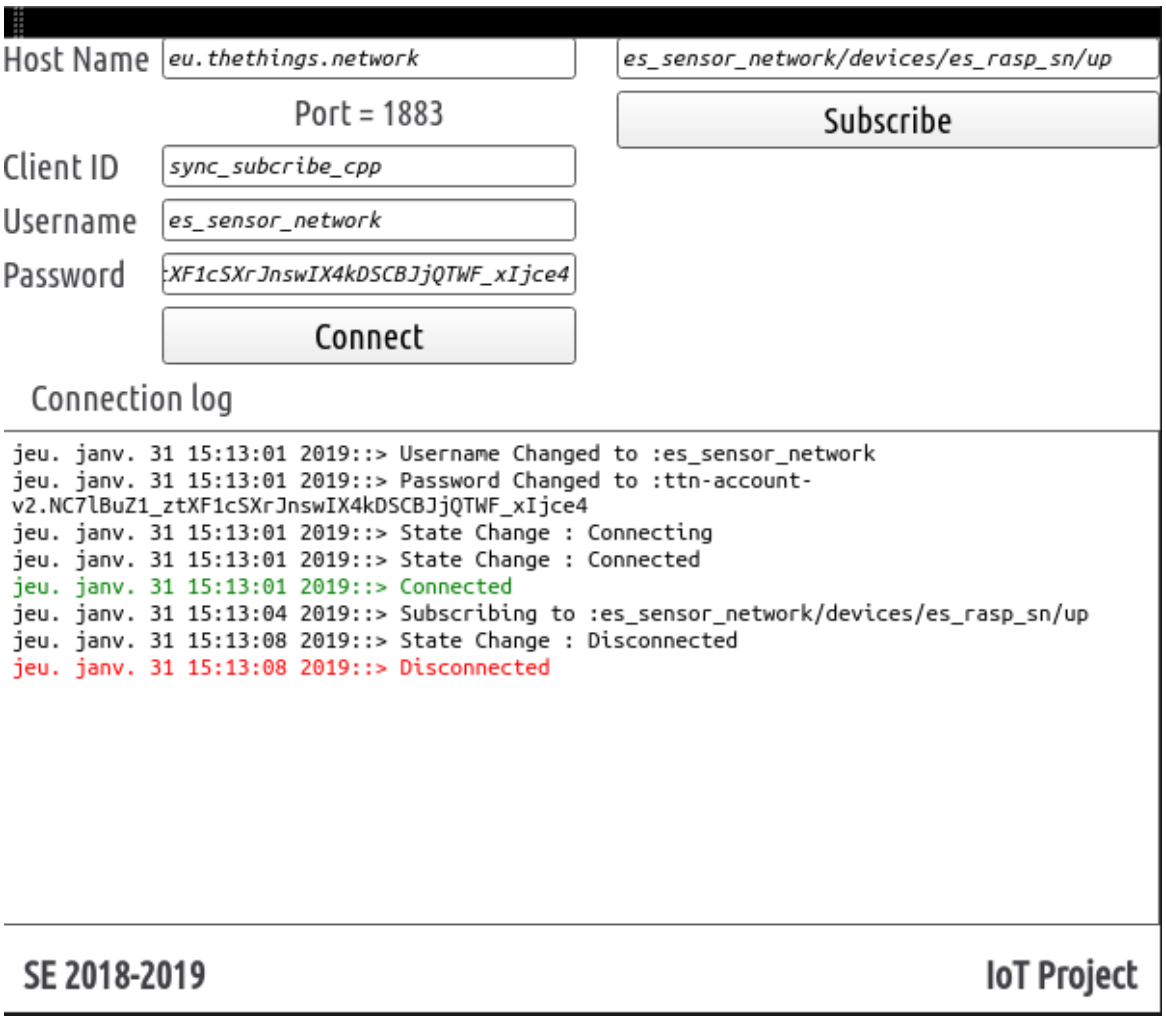

FIGURE 22 – Mainwindow

• Chartwindow (.ui, .h, .cpp)

Cette fenêtre s'ouvre après une connection réussi avec le serveur MQTT, elle permet d'afficher les informations extraites des messages reçus. Une fois la connection avec le serveur établie, il faut encore s'abonner à un topic. Si ses deux conditions sont réunies, les messages peuvent être reçus. Lorsqu'un message est reçu, un signal est émis par le client, il active plusieurs slots : un premier slots affiche le message en "brut" dans le champ "connection log" de la fenêtre mainwindow, un deuxième va extraire les informations du message en utilisant les méthodes fournies par le Json parser, les informations sont traitées ce qui met automatiquement à jour les graphes et la liste des messages reçues. Un champ affiche le message courant sous la forme d'un "jolie" Json. La fenêtre chartwindow est présentée en figure 23.

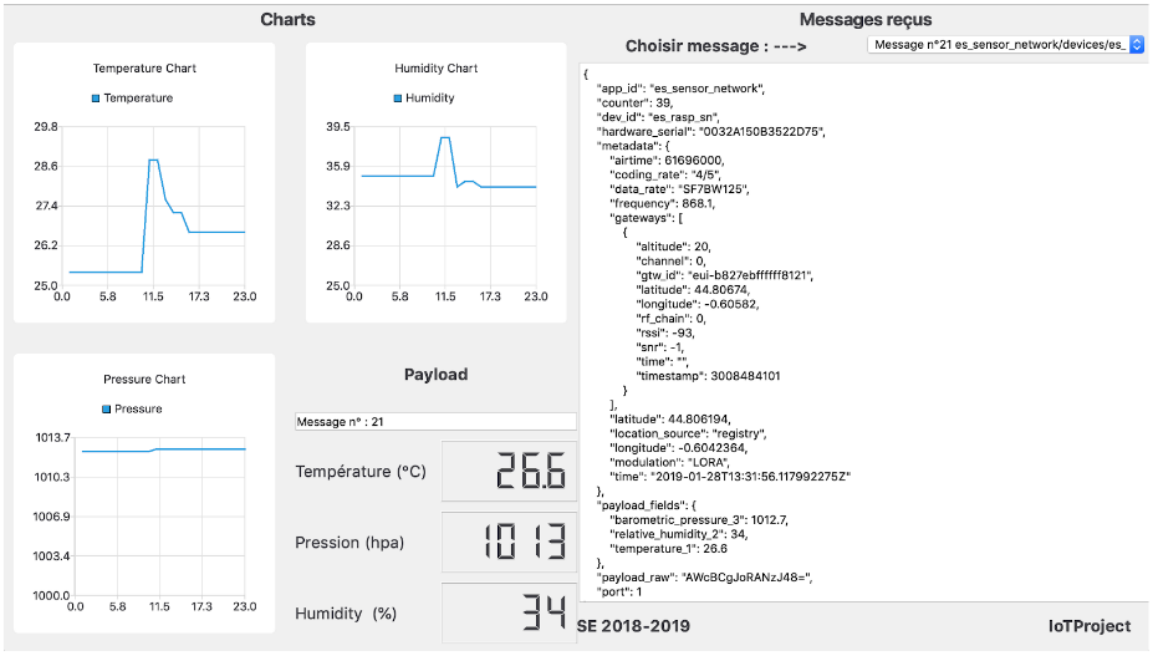

FIGURE 23 – Chartwindow

• Datachart (.h, .cpp)

Cette classe permet de gérer les données reçues par le client, elle contient trois QLineSeries qui stockent la température, la pression et l'humidité et trois QCharts qui permettent l'affichage des graphes. De plus, une liste contenant les messages reçues est utilisée afin de pouvoir afficher à tout moment n'importe quelle message reçu.

8 Conclusion

Arrivés à termes de ce projet, nous avions réussi à répondre aux spécifications qui nous ont été établies dans le cahier de charge. La station météo est maintenant fonctionnelle.

Ce projet fut très intéressant par la découverte et la mise en oeuvre d'un ensemble de protocoles et de librairies connues dans le monde de l'Internet des Objets.

Ce projet nous a permis de prototyper un objet connecté rapidement en mettant en œuvre des connaissances acquises lors de nos trois années d'école d'ingénieur. Les différentes fonctionnalités développées ont fortement mobilisé notre intérêt et contribué à la mise en place d'un travail d'équipe. Des compétences indéniablement appréciées dans ce domaine qu'est l'ingénierie.

9 Références

[1] Guide QT
http://doc.qt.io/qtcreator

[2] Disposition des broches de la Raspberry Pi 3
https://pinout.xyz/pinout/pin12gpio18

[3] Calibration des capteurs du Sense Hat - README.md
https://github.com/raspberrypi/documentation/blob/master/hardware/sense-hat/README.md

[4] Mosquitto MQTT
https://projetsdiy.fr/mosquitto-broker-mqtt-raspberry-pi/

[5] LMIC Library
https://github.com/Snootlab/lmicchisterapi

[6] Etat de l'art du protocole LoRaWAN et illustrations
https://www.lora-alliance.org/
https://www.frugalprototype.com/technologie-lora-reseau-lorawan/
http://www.ioi-labs.com/technologies/comprendre-reseau-lorawan7

[7]Décodeur de paquets LoRaWAN
https://lorawan-packet-decoder-0ta6puiniaut.runkit.sh/

Table des figures

1 Schéma de principe du projet . . . 3
2 Architecture réseau LoRaWAN . . . 4
3 Les couches des deux protocoles . . . 5
4 Tableau des différentes valeurs du SF et du SNR . . . 6
5 Schéma explicatif du mode ABP . . . 7
6 Raspberry PI . . . 9
7 SenseHAT . . . 10
8 ChisteraPI . . . 10
9 LoRa et LMIC . . . 11
10 Création de l'application sur TTN . . . 11
11 Formats du payload . . . 12
12 Messages reçues sur TTN . . . 12
13 Messages reçues Cayenne . . . 13
14 Graph explicatif de MQTT . . . 14
15 Création client MQTTBox . . . 15
16 Paramètres client MQTTBox . . . 15
17 Client created MQTTBox . . . 15
18 Topic subscription MQTTBox . . . 16
19 Payload received MQTTBox . . . 16
20 Utilisation des signaux et slots . . . 17
21 Présentation de Qtdesigner . . . 18
22 Mainwindow . . . 19
23 Chartwindow . . . 20