MASTER OF SCIENCE IN ENGINEERING

Master of Science HES-SO in Engineering
Av. de Provence 6
CH-1007 Lausanne

Hes·so
Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences and Arts Western Switzerland

Master of Science HES-SO in Engineering

Orientation : Technologies de l'information et de la communication (TIC)

Projets LoRa pour le SmartCanton de Genève

Fait par

David Da Silva Andrade

Sous la direction de
Prof. Fabien, Vannel
Dans l'institut inIT de la Haute école du paysage, d'ingénierie et d'architecture de Genève

Expert externe : Maxime Feroul

Genève, HES-SO//Master, 2018

Résumé

Projets LoRa pour le SmartCanton de Genève

par David Da Silva Andrade

Ce document contient la description, l'analyse ainsi que le développement nécessaire à la réalisation de ce travail de master. Ce travail a été réalisé en collaboration avec la Direction Générale des Systèmes d'Information (DGSI) de la ville et du canton de Genève.

Le projet avait pour but de créer une carte électronique pouvant être utilisée pour divers tests de développement d'un réseau LoRaWAN. L'infrastructure de ce réseau a été mise en place sous forme d'un *proof of concept* (poc), via un projet nommé SmartCanton avancé par la ville et le canton de Genève. Le but de ce réseau est d'offrir une plateforme utilisable par les citoyens et les entreprises autour de Genève tout en garantissant une sécurité accrue des données.

Dans le cadre de ce projet, une carte électronique a donc été développée afin de répondre à plusieurs critères définis par des *use cases* souhaités par la DGSI. Ces *use cases* ont influencé le choix des technologies utilisées, des composants d'alimentation, la portabilité, les périphériques, etc. Cette carte est équipée de deux types d'interfaces de communication sans fil. Un module LoRa avec une implémentation du protocole LoRaWAN, ainsi qu'un circuit intégré supportant les liaisons Bluetooth Low Energy et IEEE 802.15.4. Cette carte a été dénommée SmartCanton DevBox.

Pour pouvoir changer les clés utilisées par la spécification LoRaWAN, une application Android a été développée. Celle-ci se connecte en Bluetooth Low Energy sur la carte électronique pour récupérer les données de certains capteurs, mais surtout afin de pouvoir modifier les informations du réseau LoRaWAN sur lequel la carte se connecte. En parallèle, un serveur exposant une API REST offre la possibilité de gérer des utilisateurs, ainsi que les divers périphériques LoRaWAN assignés à ceux-ci. L'application Android communique avec ce serveur pour authentifier ces utilisateurs et récupérer les diverses informations sur les SmartCanton DevBox connectées via Bluetooth.

L'affichage final des données capturées par la DevBox est réalisé sur un tableau de bord de la plateforme en ligne Cayenne de l'entreprise MyDevices. Il est ainsi possible de contrôler les données que l'on souhaite recevoir, la fréquence d'émission, de même que changer l'état de certains périphériques via des paquets LoRaWAN.

Remerciements

J'adresse mes remerciements aux personnes qui m'ont aidé dans la réalisation de ce travail de master, ainsi que dans la rédaction de ce mémoire.

En premier lieu, je remercie M. Fabien Vannel, professeur à la Haute école du paysage, d'ingénierie et d'architecture de Genève. En tant que Directeur de mémoire, il a proposé ce sujet fort intéressant et il a me guider dans mon travail afin de le concrétiser au mieux. Il a également pris sur son temps pour m'aider à trouver des solutions aux différentes problématiques rencontrées durant l'avancement du présent travail.

Je remercie également M. Christopher Larraz, conseiller au numérique et à l'innovation pour la République et Canton de Genève, pour m'avoir cette opportunité de travailler sur le projet SmartCanton. C'est grâce à lui que ce sujet a été proposé comme travail de Master en collaboration avec la HES-SO.

Je souhaite tout particulièrement remercier M. Maxime Feroul, Directeur Technique de la société Kyos, qui a travaillé en collaboration avec la DGSI sur la création du SmartCanton. Il m'a non seulement aider à avoir la vue d'ensemble du projet SmartCanton, mais également lors de la compréhension des besoins du projet. Grâce à lui, j'ai pu affiner le cahier des charges de ce travail et ainsi accélérer le développement de celui-ci. Malgré son emploi du temps chargé, il a toujours su trouver du temps pour répondre à mes e-mails et me rencontrer, de sorte à m'aider lors de l'avancement du projet et effectuer des tests d'implémentation.

Je remercie aussi ma famille pour leur soutien et compréhension durant ces derniers mois. Tout particulièrement Mme Marques et Mme De Lima Ferreira pour leur précieuse aide à la relecture et à la correction de ce mémoire.

Un grand merci à tous les collègues et amis au laboratoire de systèmes numériques d'hepia. Ils ont toujours su rire à mes mauvaises blagues et n'ont jamais hésité à me donner un coup de main pour résoudre les divers obstacles rencontrés dans la réalisation de ce travail.

Table des matières

1 Introduction **1**
1.1 Contexte . . . 1
1.2 Objectifs . . . 2

2 État de l'art **3**
2.1 *Smart Cities* . . . 3
2.2 LPWAN . . . 5
2.3 LoRa . . . 6
2.3.1 Sensibilité . . . 7
2.3.2 *Coding rate* . . . 7
2.3.3 *Spreading factor* . . . 8
2.3.4 Structure des paquets . . . 8
2.3.5 Distance d'émission . . . 8
2.3.6 Cycle d'opération . . . 9
2.4 LoRaWAN . . . 10
2.5 Communication sans fil à 2.4 GHz . . . 10
2.6 Bluetooth Low Energy . . . 12
2.7 IEEE802.15.4 . . . 13
2.7.1 ZigBee . . . 13
2.7.2 Thread . . . 14
2.8 Équipements matériels existants . . . 16
2.9 Développement de logiciels sur microcontrôleur . . . 17
2.9.1 Stacks Bluetooth . . . 17
2.9.2 Stacks LoRaWAN . . . 18
2.9.3 RTOS . . . 18
2.9.3.1 FreeRTOS . . . 19
2.9.3.2 Zephyr . . . 20
2.9.3.3 Mynewt . . . 21
2.9.3.4 Couches d'abstraction . . . 22
2.9.4 Drivers de périphériques . . . 23
2.10 SmartCanton . . . 23
2.10.1 Architecture . . . 24
2.10.2 Sécurité . . . 25
2.10.3 Mise à disposition des données . . . 25
2.10.4 Fédération d'opérateurs . . . 25

3 Protocoles **27**
3.1 Bluetooth Low Energy . . . 27
3.1.1 Interopérabilité entre les dispositifs . . . 28
3.1.2 Rôles . . . 28
3.1.3 *Advertisements* . . . 29
3.1.4 Machine des états de la connexion . . . 30
3.1.5 Architecture d'une *stack* Bluetooth . . . 31

3.1.5.1 *Controller* 32
3.1.5.1.1 *Physical Layer* 32
3.1.5.1.2 *Link Layer* 32
3.1.5.2 *Host Control Interface* 33
3.1.5.3 *Host* 33
3.1.5.3.1 *L2CAP* 33
3.1.5.3.2 *GAP* 33
3.1.5.3.3 *Security Manager* 34
3.1.5.3.4 *ATT* 34
3.1.5.3.5 *GATT* 35
3.1.5.3.6 *Profils et services* 35
3.1.5.4 *Application* 37
3.1.6 Beacons 37
3.1.7 Consommation énergétique 40
3.2 LoRaWAN 40
3.2.1 Classes des périphériques 41
3.2.2 Architecture d'un réseau LoRaWAN 42
3.2.2.1 *End nodes* 42
3.2.2.2 *Gateways* 42
3.2.2.3 *Network Server* 43
3.2.2.4 Join Server 44
3.2.2.5 Applications Server 44
3.2.3 *Spreading Factor* et *Data Rate* 45
3.2.4 Paquets LoRaWAN 46
3.2.5 Clés de chiffrement 47
3.2.6 Canaux d'émission 47
3.2.7 Temps d'émission et nombre de paquets 48
3.2.8 Spécification LoRaWAN 1.1 48

4 Sécurité **51**
4.1 Bluetooth Low Energy 51
4.1.1 Modes et niveaux de sécurité 52
4.1.2 Appairage et attachement 52
4.1.2.1 Phase 1 : *Pairing Feature Exchange* 53
4.1.2.2 Phase 2 : *LE Legacy Pairing* ou *LE Secure Connections* 53
4.1.2.3 Phase 3 : *Transport Specific Key Distribution* 56
4.1.3 Recommandations de sécurité 56
4.2 LoRaWAN 58
4.2.1 Rejoindre un réseau à l'aide d'OTAA 58
4.2.2 Protection des données 59
4.2.2.1 Chiffrement des données 59
4.2.2.2 Signature des messages 60
4.2.2.3 Infrastructure des opérateurs LoRaWAN 60
4.2.3 LoRaWAN spécification 1.1 61
4.2.4 Vulnérabilités et recommandations de sécurité 62
4.2.4.1 Infrastructure du réseau 63
4.2.4.2 Gestion des clés pour les périphériques 64
4.2.4.3 Gestion des clés pour les serveurs 64
4.2.4.4 Compteur de paquets 64
4.2.4.5 Renouvellement des clés OTAA 65
4.2.4.6 Usurpation du microcontrôleur 65

4.2.4.7 Accès aux *gateways* . . . 65
4.2.5 SmartCanton . . . 66
4.2.5.1 *Hardware Security Module* . . . 66
4.2.5.2 *Join Server* . . . 66
4.2.5.3 Cheminement des données et répartition des clés . . . 68
4.3 Serveur Web . . . 69
4.3.1 HTTPS . . . 69
4.3.2 JWT . . . 69
4.4 Protections matérielles . . . 70
4.4.1 Détection d'ouverture de boitier . . . 71
4.4.2 Protection de la mémoire flash . . . 71
4.4.3 Coffres-forts matériels . . . 72

5 Conception matérielle **73**
5.1 Introduction . . . 73
5.2 Architecture . . . 73
5.3 Choix des composants et leurs caractéristiques . . . 75
5.3.1 Microcontrôleur . . . 75
5.3.1.1 Critères de comparaison . . . 75
5.3.1.2 Comparaisons et choix . . . 76
5.3.1.3 NXP Kinetis KW41Z . . . 79
5.3.2 Gestion de la batterie . . . 80
5.3.3 Coprocesseur LoRa . . . 81
5.3.3.1 Murata CMWX1ZZABZ-078 . . . 82
5.3.3.2 IMST iM880B . . . 85
5.3.3.3 USI WM-SG-SM-42 . . . 86
5.3.3.4 Module choisi . . . 88
5.3.4 Périphériques . . . 89
5.3.4.1 LED . . . 89
5.3.4.2 Buttons . . . 89
5.3.4.3 Plateforme inertielle 9 axes . . . 89
5.3.4.4 Capteur environnemental et qualité de l'air ambiant . . . 90
5.3.4.5 GPS . . . 90
5.3.4.6 Connecteur PMOD . . . 93
5.4 Schéma bloc général final . . . 94
5.5 Réalisation et production du circuit électronique . . . 94
5.6 Erreurs de conception et corrections possibles . . . 95
5.7 Boitier en plastique . . . 95

6 Développement logiciel **97**
6.1 Architecture générale . . . 97
6.2 Scanneur de périphériques Bluetooth Low Energy . . . 98
6.3 DevBox : module LoRaWAN . . . 99
6.3.1 Architecture logicielle . . . 99
6.3.2 Exemples et modifications . . . 99
6.4 DevBox : NXP Kinetis KW41Z . . . 100
6.4.1 *OS Abstraction Layer* et *Connectivity Framework* . . . 100
6.4.2 Interfaces de programmation . . . 102
6.4.2.1 Bibliothèques et *drivers* repris . . . 103
6.4.2.1.1 AT Commander . . . 103
6.4.2.1.2 minmea . . . 103

6.4.2.1.3 BNO055 *driver* . . . 103
6.4.2.1.4 BME680 *driver* . . . 104
6.4.2.1.5 Bosch Sensortec Environmental Cluster . . . 104
6.4.2.2 Bibliothèques et *drivers* développés . . . 105
6.4.2.2.1 `pin_mux` . . . 105
6.4.2.2.2 `board` . . . 105
6.4.2.2.3 `bme680 et bno055 support` . . . 105
6.4.2.2.4 `lorawan_controller` . . . 106
6.4.2.2.5 `neo-m8` . . . 107
6.4.2.2.6 `string_utils` . . . 108
6.4.3 Architecture des tâches . . . 108
6.4.4 Algorithmes des tâches . . . 109
6.4.4.1 DevBox Main . . . 109
6.4.4.2 GPS Neo M8 . . . 110
6.4.4.3 BNO055 . . . 111
6.4.4.4 BME680 . . . 112
6.4.4.5 Scanneur Bluetooth Low Energy . . . 112
6.4.4.6 LoRaWAN *Controller* . . . 113
6.4.5 KW41Z *stack* Bluetooth . . . 114
6.4.6 Services Bluetooth . . . 115
6.4.6.1 Création d'un profil personnalisé . . . 116
6.4.6.1.1 Fonctions minimales d'un profil . . . 117
6.4.6.1.2 Écritures et lecture depuis la base de donnée GATT . 118
6.4.6.1.3 Notifications . . . 119
6.4.6.2 Services BLE disponibles sur la DevBox . . . 120
6.4.6.2.1 *Generic Access*, *Generic Attribute* et *Battery* . . . 120
6.4.6.2.2 SmartCanton DevBox LoRaWAN . . . 121
6.4.6.2.3 SmartCanton DevBox GPS . . . 121
6.4.6.2.4 SmartCanton DevBox BME680 . . . 122
6.4.6.2.5 SmartCanton DevBox BNO055 . . . 122
6.4.6.2.6 SmartCanton DevBox BLE Scanner . . . 122
6.4.7 Sécurité . . . 123
6.4.7.1 Adresses MAC Bluetooth . . . 123
6.4.7.2 Sécurité des services Bluetooth . . . 123
6.4.7.3 *Passkey* Bluetooth Low Energy . . . 124
6.4.7.4 Stockage des clés LoRaWAN . . . 124
6.5 Application Android SmartCanton Manager . . . 125
6.5.1 Frameworks utilisés . . . 125
6.5.1.1 SweetBlue . . . 125
6.5.1.2 Retrofit . . . 126
6.5.2 Manuel d'utilisation de l'application . . . 126
6.5.3 Activités . . . 126
6.5.4 Sécurité . . . 127
6.6 Serveur de management de SmartCanton DevBox . . . 128
6.6.1 Frameworks utilisés . . . 128
6.6.1.1 Flask . . . 128
6.6.1.2 Flask-JWT-Extended . . . 129
6.6.2 Base de données . . . 129
6.6.3 Routes REST . . . 130
6.6.4 Certificats SSL . . . 131

7 Démonstrateur **133**
7.1 Architecture finale . . . 133
7.2 Cayenne de MyDevices . . . 134
7.2.1 Format des données . . . 134
7.2.2 Fonctionnalités implémentées . . . 134
7.2.3 Dashboard . . . 135
7.3 Échangeur d'AppKey . . . 135
7.4 Opérateurs testés . . . 135
7.5 Consommation électrique . . . 136

8 Gestion du projet **139**
8.1 Planning . . . 139
8.2 Séparation des dépendances . . . 139
8.3 Gestionnaire de version . . . 140
8.4 Documentation du code . . . 140

9 Discussions **141**
9.1 Problèmes rencontrés . . . 141
9.1.1 Réalisation matérielle de la DevBox . . . 141
9.1.2 Développement logiciel de la DevBox . . . 141
9.1.3 Serveur . . . 143
9.1.4 Application Android . . . 143
9.1.5 Démonstrateur . . . 143
9.2 Améliorations possibles . . . 144
9.2.1 Sécurité . . . 144
9.2.1.1 Stockage des clés . . . 144
9.2.1.2 Confiance des données du périphérique . . . 145
9.2.1.3 Bluetooth . . . 145
9.2.2 Réalisation matérielle de la DevBox . . . 145
9.2.3 Programmation de la DevBox . . . 146
9.2.3.1 RTOS . . . 146
9.2.3.2 Mise en place de la basse consommation . . . 146
9.2.3.3 Mises à jour logicielles . . . 146
9.2.4 Serveur . . . 147
9.2.5 Application Android . . . 147
9.2.6 Démonstrateur . . . 147
9.3 Conclusions . . . 148

Bibliographie **151**

A SmartCanton DevBox : Bluetooth Services **A-1**
A.1 Generic Access . . . A-1
A.2 Generic Attribute . . . A-1
A.3 SmartCanton DevBox LoRaWAN . . . A-1
A.4 SmartCanton DevBox GPS . . . A-3
A.5 SmartCanton DevBox BME680 . . . A-3
A.6 SmartCanton DevBox BNO055 . . . A-5
A.7 SmartCanton DevBox BLE Scanner . . . A-6
A.8 Batterie . . . A-6

B AppKey Server : REST API Documentation **B-1**
B.1 Authentification . . . B-2
B.2 User routes . . . B-3
B.2.1 Get all users . . . B-3
B.2.2 Get a specific user information from a public id . . . B-3
B.2.3 Update information about a specific user . . . B-4
B.2.4 Create a new user . . . B-4
B.2.5 Delete an existing user . . . B-5
B.3 SmartCanton DevBox routes . . . B-5
B.3.1 Get all devices . . . B-5
B.3.2 Get information about a specific device . . . B-6
B.3.3 Create a new device . . . B-6
B.3.4 Update an existing device . . . B-7
B.3.5 Delete an existing device . . . B-7

C Android SmartCanton Manager : User's Guide **C-1**
C.1 Login Activity . . . C-1
C.2 Bluetooth Activity : Scanner BLE Fragment . . . C-1
C.3 Bluetooth Activity : Device Management Fragment . . . C-2
C.4 Profile Activity . . . C-5
C.5 Beacon Generator Activity . . . C-6

D Planification **D-1**
D.1 Temps planifié pour chaque tâche . . . D-1
D.2 Temps écoulé pour chaque tâche . . . D-2

E SmartCanton DevBox : Electronic Designs **E-1**

Table des figures

2.1 Carte du trafic des principaux axes routiers de Genève en temps réel sur le portail SITG . . . 4
2.2 Comparaison de LPWAN avec d'autres technologies . . . 5
2.3 Modulation *chirp spread spectrum* de LoRa . . . 6
2.4 Comparaison de la sensibilité entre la modulaton FSK et LoRa . . . 7
2.5 Composition d'un paquet LoRa . . . 8
2.6 Tests de portée à l'aide de deux dispositifs LoRa avec visualisation de la topologie du terrain . . . 9
2.7 Résultats des distances maximales LoRa réalisées par Libelium . . . 9
2.8 Description des modes utilisés par Libelium . . . 9
2.9 Comparaison Bluetooth et Bluetooth Low Energy . . . 12
2.10 Exemples d'architectures avec la technologie ZigBee . . . 13
2.11 Architecture d'une *stack* logicielle Thread . . . 15
2.12 Architecture d'un réseau Thread . . . 15
2.13 Comparaison entre RTOS et bare metal . . . 19
2.14 Résultats d'une étude de marché sur les OS réalisée par EE Times et embedded.com en avril 2017 . . . 20
2.15 Logo *Zephyr Project* . . . 20
2.16 Logo *Apache Mynewt* . . . 22
2.17 Architecture générale du projet SmartCanton . . . 24
2.18 Infrastructure LoRa du SmartCanton . . . 24
2.19 Fédération des réseaux . . . 26

3.1 Bande ISM utilisée par le Bluetooth Low Energy . . . 27
3.2 Interopérabilité entre les dispositifs Bluetooth . . . 28
3.3 Terminologies des connexions et des échanges de données en Bluetooth Low Energy . . . 29
3.4 Contenu d'un paquet de type *advertisement* . . . 29
3.5 Les 5 états possibles pour un dispositif Bluetooth Low Energy . . . 30
3.6 Étapes lors de la connexion Bluetooth entre une centrale et un périphérique . . . 31
3.7 Architecture complète de la stack BLE . . . 32
3.8 Opérations possibles entre un serveur et un client Bluetooth Low Energy 34
3.9 Profil *Heart Rate Sensor* standardisé . . . 36
3.10 Spécification des paquets iBeacons . . . 38
3.11 Spécification des paquets AltBeacon . . . 38
3.12 Spécification des paquets Eddystone . . . 39
3.13 Consommation des *advertisements* en Bluetooth Low Energy . . . 40
3.14 Architecture du protocole LoRaWAN . . . 41
3.15 Réception en tant que périphérique de type classe A . . . 41
3.16 Réception en tant que périphérique de type classe B . . . 41
3.17 Réception en tant que périphérique de type classe C . . . 42
3.18 Architecture d'un réseau LoRaWAN . . . 42

3.19 Concentrateur LoRaWAN Multitech Conduit 43
3.20 Comparatif du temps d'émission d'une trame LoRaWAN en fonction de ses paramètres 45
3.21 Format des messages du protocole LoRaWAN 46
3.22 Rôles de la *Network Session Key* et *Application Session Key* 47
3.23 Roaming LoRaWAN 48

4.1 Diagramme du processus d'appairage 52
4.2 Capacité des entrées disponibles sur un dispositif Bluetooth 53
4.3 Capacité des sorties disponibles sur un dispositif Bluetooth 53
4.4 Comparaison des algorithmes de générations de clés en fonction des versions BLE 54
4.5 Choix de la méthode de sécurité en fonction des entrées/sorties 55
4.6 Format des messages du protocole LoRaWAN 58
4.7 Infrastructure d'un réseau LoRaWAN comme imaginée dans la spécification 60
4.8 Infrastructure d'un réseau LoRaWAN réellement implémenté avec les opérateurs du marché 61
4.9 Clés utilisées dans un réseau LoRaWAN avec la nouvelle spécification 1.1 62
4.10 Recommandation pour l'implémentation d'une infrastructure LoRaWAN sécurisée 63
4.11 Clé asymétrique utilisée pour chiffrer les données 63
4.12 Vulnérabilité des modules LoRaWAN avec le remplacement du microcontrôleur 65
4.13 Thales Nshield 66
4.14 Périphérique enregistré sur un réseau OrbiWise sans AppKey 67
4.15 Déchiffrement des données par le HSM 67
4.16 Déchiffrement des données par la plateforme IoT (FIWARE) 68
4.17 Cheminement des données depuis le périphérique jusqu'à l'application en utilisant le HSM pour le déchiffrement 68
4.18 Génération et utilisation d'un *token* JWT 69
4.19 Déchiffrement d'un JWT 70
4.20 Authentifications possibles avec un STSAFE 72

5.1 Architecture option numéro 1 74
5.2 Architecture option numéro 2 74
5.3 Diagramme bloc interne du KW41Z 79
5.4 Cartes de développement pour KW41Z 80
5.5 Comparatif SX1272 et SX1276 81
5.6 Comparaison de la consommation de courant entre le sx1272 et sx1278 82
5.7 Module LoRa CMWX1ZZABZ-078 82
5.8 Contenu du module LoRa CMWX1ZZABZ-078 83
5.9 Consommation en courant du module CMWX1ZZABZ-078 84
5.10 Consommation en courant du microcontrôleur STM32L082 85
5.11 Module LoRa iM880B du fabricant IMST 85
5.12 Contenu du module LoRa iM880B 86
5.13 Consommation en courant du module iM880B 86
5.14 *Shield* I-NUCLEO-LRWAN1 de ST et module USI WM-SG-SM-42 . . 87
5.15 Contenu du module WM-SG-SM-42 87
5.16 Consommation en courant du module WM-SG-SM-42 88

5.17 Carte de développement ST B-L072Z-LRWAN1 88
5.18 Plateforme 9 axes BNO055 89
5.19 Capteur environnemental et qualité de l'air BME680 90
5.20 VOC mesurés par le BME680 90
5.21 GPS U-blox NEO-M8 91
5.22 Comparaison de la gamme de GPS U-blox NEO-M8 91
5.23 Schéma de connexion d'un NEO-M8 avec une antenne active 92
5.24 Consommation du GPS NEO-M8 92
5.25 Pinout du standard PMOD en fonction des types choisis 93
5.26 Schéma bloc général de la DevBox 94
5.27 Boitier imprimé en 3D pour y loger le circuit électronique 96

6.1 Schéma général du projet avec les différentes technologies utilisées . . 97
6.2 Architecture logicielle du CMWX1ZZABZ 99
6.3 Connexion entre un AT *Slave* et un AT *Master* utilisant la suite de bibliothèque STM32Cube LoRa 100
6.4 *Connectivity Framework* de Freescale (NXP) 101
6.5 Bibliothèques et *drivers* utilisés sur la DevBox 102
6.6 Niveaux de l'indice IAQ 104
6.7 Niveaux de l'indice IAQ Accuracy 104
6.8 Utilitaire de configuration du multiplexeur de pins des microcontrôleurs NXP 105
6.9 Diagramme de la fonction d'initialisation du module LoRaWAN 107
6.10 Vue générale des communications entre les tâches 108
6.11 Diagramme de fonctionnement de la tâche principale lors de la gestion des événements 110
6.12 Diagramme de fonctionnement de la tâche GPS 111
6.13 Diagramme de fonctionnement de la tâche BNO055 111
6.14 Diagramme de fonctionnement de la tâche BME680 112
6.15 Diagramme de fonctionnement de la tâche scanneur Bluetooth lors de la sauvegarde des nouveaux périphériques 113
6.16 Diagramme de fonctionnement de la tâche de scanneur Bluetooth lors de son initialisation et de la capture des données 113
6.17 Diagramme de fonctionnement de la tâche `LoRaWAN Controller` lors de la gestion des événements 114
6.18 MCUXpresso SDK Builder interface 115
6.19 Bluetooth Developer Studio avec les services BLE du projet SmartCanton DevBox 116
6.20 Fichiers de déclaration des services Bluetooth 117
6.21 Lecture des caractéristiques du service de GPS à l'aide de l'application *NRF Connect* 120
6.22 Rôle de l'application mobile 125
6.23 Diagramme d'interconnexions entre les différentes activités Android . 126
6.24 Diagramme de communication entre les utilisateurs et le serveur de gestion des AppKey 128
6.25 Contenu de la base de données 130

7.1 Diagramme du projet avec l'application de visualisation des données . 133
7.2 *Dashboard* du capteur DevBox à l'aide de Cayenne sur un navigateur Web 136

7.3 *Dashboard* du capteur DevBox à l'aide de Cayenne sur l'application Android 136

9.1 Mesure de la température faussée sur le BME680 141
9.2 Amélioration du stockage des clés sur le KW41Z 145

B.1 Database model B-1

C.1 SmartCanton Manager login activity C-1
C.2 SmartCanton Manager scanner fragment C-2
C.3 SmartCanton Manager device found on the server C-2
C.4 SmartCanton Manager connection progress C-3
C.5 SmartCanton Manager device connected and characteristics read . . . C-3
C.6 SmartCanton Manager update LoRaWAN config from server configuration C-4
C.7 SmartCanton Manager enable sensors notifications C-4
C.8 SmartCanton Manager edit bno055 or ble scanner interval C-5
C.9 SmartCanton Manager profile activity C-5
C.10 SmartCanton Manager beacon generator activity C-6

Liste des tableaux

3.1 Paramètres d'un *Client Characteristic Configuration Descriptor* 36
3.2 Différents *data rate* LoRaWAN avec leurs débits pour l'Europe 45
3.3 Différents champs contenus dans un MHDR 46
3.4 Différents types de messages MAC possibles 46

5.1 Comparaison des microcontrôleurs envisagés pour le projet 78

6.1 Format du AltBeacon utilisé par le scanner Bluetooth 98
6.2 Paramètres d'un *Characteristic User Description Descriptor* 116

A.1 Description du Generic Access service A-1
A.2 Description du Generic Attribute service A-1
A.3 Description du SmartCanton DevBox LoRaWAN service A-2
A.4 Description du SmartCanton DevBox GPS service A-3
A.5 Description du SmartCanton DevBox BME680 service A-4
A.6 Description du SmartCanton DevBox BNO055 service A-5
A.7 Description du SmartCanton DevBox BLE Scanner service A-6
A.8 Description du Battery Service A-6

Convention topographique

Afin d'aider le lecteur à parcourir ce document, les conventions typographiques ci-dessous ont été adoptées. Tous les mots empruntés à la langue anglaise ont été écrits en *italique*. Pour les mots de la langue française, la rectification de l'orthographe de 1990 (mise en oeuvre en 2016) a été utilisée, ayant pour conséquence la disparition des accents circonflexes sur une majorité de mots.

Toute référence à un nom de fichier (ou dossier), un chemin d'accès, une utilisation de paramètre, variable ou commande utilisable par l'utilisateur, est écrite avec le type d'écriture `Teletypefont Family`.

Tout extrait de fichier, de code ou de configuration est écrit selon le format suivant :

HelloWorld.c

```c
/* Hello World example */
#include<stdio.h>
int main()
{
    printf("Hello World");
    return 0;
}
```

Abréviations

hepia	**H**aute **É**cole du **P**aysage, d'**I**ngénierie et d'**A**rchitecture de Genève
HES-SO	**H**aute **É**cole **S**pécialisée de **S**uisse **O**ccidentale
POC	**P**roof **O**f **C**oncept
DGSI	**D**irection **G**énérale des **S**ystèmes d'**I**nformation
CERN	**C**onseil **E**uropéen pour la **R**echerche **N**ucléaire
SITG	**S**ystème d'**I**nformation du **T**erritoire à **G**enève
SIG	**S**ervices **I**ndustriels de **G**enève
ISRG	**I**nternet **S**ecurity **R**esearch **G**roup
IEEE	**I**nstitute of **E**lectrical and **E**lectronics **E**ngineers
NIST	**N**ational **I**nstitute of **S**tandards and **T**echnology
IoT	**I**nternet **o**f **T**hings
HSM	**H**ardware **S**ecurity **M**odule
PIN	**P**ersonal **I**dentification **N**umber

Électronique

UART	**U**niversal **A**synchronous
SPI	**S**erial **P**eripheral **I**nterface
I2C	**I**nter-**I**ntegrated **C**ircuit
BGA	**B**all **G**rid **A**rray
VOC	**V**olatile **O**rganic **C**ompounds
LNA	**L**ow-**N**oise **A**mplifier
SAW	**S**urface **A**coustic **W**ave
IAQ	**I**ndoor **A**ir **Q**uality
TRNG	**T**rue **R**andom **N**umber **G**enerator
JTAG	**J**oint **T**est **A**ction **G**roup

Programmation

API	**A**pplication **P**rogramming **I**nterface
RTOS	**R**eal-**T**ime **O**perating **S**ystem
HTTPS	**H**yperText **T**ransfer **P**rotocol **S**ecure
JSON	**J**avaScript **O**bject **N**otation
JWT	**JSON W**eb **T**okens
FSCI	**F**reescale **S**erial **C**ommunication **I**nterface
BSEC	**B**osch **S**ensortec **E**nvironmental **C**luster
BDS	**B**luetooth **D**eveloper **S**tudio

Réseaux sans fil et protocoles

RF	**R**adio **F**réquence
LPWAN	**L**ow-**P**ower **W**ide-**A**rea Network

LR WPAN	**L**ow **R**ate **W**ireless **P**ersonal **A**rea **N**etwork
UNB	**U**ltra **N**arrow **B**and
ISM Band	**I**ndustrial, **S**cientific and **M**edical Band
GFSK	**G**aussian **F**requency **S**hift **K**eying
RSSI	**R**eceive **S**ignal **S**trength **I**ndicator
SNR	**S**ignal to **N**oise **R**atio
OSI	**O**pen **S**ystems **I**nterconnection
MAC	**M**edium **A**ccess **C**ontrol
6LowPAN	IPv**6** over **L**ow-**P**ower **W**ireless **P**ersonal **A**rea **N**etworks
UDP	**U**ser **D**atagram **P**rotocol
TCP	**T**ransmission **C**ontrol **P**rotocol
NFC	**N**ear-**F**ield **C**ommunication
SSH	**S**ecure **Sh**ell

Bluetooth Low Energy

BLE	**B**luetooth **L**ow **E**nergy
B SIG	**B**luetooth **S**pecial **I**nterest **G**roup
CCCD	**C**lient **C**haracteristic **C**onfiguration **D**escriptor
CUDD	**C**haracteristic **U**ser **D**escription **D**escriptor
MITM	**M**an-**I**n-**T**he-**M**iddle
OOB	**O**ut-**O**f-**B**and
DTLS	**D**atagram **T**ransport **L**ayer **S**ecurity

LoRa et LoRaWAN

LoRa	**Lo**ng **Ra**nge
LoRaWAN	**Lo**ng **Ra**nge **W**ide-**A**rea **N**etwork
CSS	**C**hirp **S**pread **S**pectrum
BW	**B**and**w**idth
SF	**S**preading **F**actor
CSS	**C**hirp **S**pread **S**pectrum
ToA	**T**ime of **A**rrival
DR	**D**ata **R**ate
ADR	**A**daptative **D**ata **R**ate
TTN	**T**he **T**hings **N**etwork
DASS	**D**ata **A**ccess **S**ub-**S**ystem
RNSS	**R**adio **N**etwork **S**ub-**S**ystem
ABP	**A**ctivation **B**y **P**ersonalization
OTAA	**O**ver-**T**he-**A**ir **A**ctivation
MIC	**M**essage **I**ntegrity **C**ode
FCntDown	**F**rame **C**ounter **D**own
FCntUp	**F**rame **C**ounter **U**p
AppKey	**App**lication **Key**
AppSKey	**App**lication **S**ession **Key**
NwkSKey	**N**etwork **S**ession **Key**
App EUI	**App**lication **EUI**
Dev Addr	**Dev**ice **Addr**ess

Chapitre 1

Introduction

De nos jours, de plus en plus de villes offrent à leurs habitants un accès à une multitude de données. Ces données sont acquises en utilisant divers capteurs disséminés à travers la ville.

L'expression *Smart City* est de plus en plus présente dans notre quotidien. Principalement dans l'optique d'améliorer la qualité des services urbains et de réduire leurs couts que les institutions publiques se sont intéressées au concept.

Dans le canton de Genève plusieurs données sont librement accessibles à la population. Ces données sont mises à disposition par le Système d'Information du Territoire à Genève (SITG) [1]. Parmi ces données, il est possible de récupérer le flux représentant la concentration du trafic routier sur les grands axes de la ville de Genève. Certaines données, plus sensibles, sont uniquement mises à disposition des clients professionnels, notamment, le réseau des conduites de gaz.

1.1 Contexte

Le canton de Genève a décidé de proposer un projet nommé SmartCanton, afin de laisser la possibilité à des entreprises ou des particuliers la possibilité d'alimenter une base de données commune ou privée.

Afin de couvrir la plus grande zone possible, une technologie de type LPWAN (*Low-Power Wide-Area Network*) a été utilisée. La technologie retenue pour ces capteurs a été le LoRa [1] car il s'agit d'une technologie prometteuse pour les *Smart Cities*. LoRa est une modulation de fréquence (couche 1 du modèle OSI) propriétaire de l'entreprise Semtech. LoRa utilise différentes fréquences dans le monde des fréquences libres. En Europe il s'agit de la fréquence 868 MHz, alors qu'en Amérique du Nord, la fréquence de 915 MHz est utilisée. Au dessus de cette couche physique, il est possible de trouver différentes spécifications. Celle utilisée dans ce projet est le LoRaWAN, soit la spécification proposée par la LoRa Alliance [2].

Le projet SmartCanton doit également pourvoir à la sécurité, que ce soit au niveau du transfert des données ou de l'authentification sur le réseau. Une fois un dispositif implanté sur le réseau, les données échangées sont toujours chiffrées. Ceci est géré par la spécification LoRaWAN.

Dans le cadre du projet SmartCanton, il est souhaité que la génération les clés d'authentification du réseau LoRa aient une durée de validité limitée dans le temps.

1. http://ge.ch/sitg/

Se pose dès lors la question du réapprovisionnement de celles-ci aux périphériques lorsqu'elles sont périmées. À l'heure actuelle, ces périphériques reçoivent les clés lorsqu'ils sont programmés (en mode ABP) ou quand ils s'authentifient auprès d'un réseau (OTAA).

1.2 Objectifs

Le projet a été structuré en trois segments. Le premier étant la réalisation d'une carte électronique suivant plusieurs restrictions telles que les types de capteurs l'équipant, la consommation et la modularité. Deux technologies sont imposées dans le choix des composants intégrant cette carte : Bluetooth et LoRa.

Le deuxième segment se consacre à la programmation de la carte électronique avec la mise en avant d'une modularité en utilisant des tâches séparées pour chaque périphérique. Le but de cette approche est de faciliter la compréhension de la structure pour l'utilisateur souhaitant modifier les données envoyées. Deux interfaces de communication sans fil doivent être implémentées pour le LoRa et le Bluetooth.

Pour conclure ce travail, un démonstrateur doit afficher les différentes données récupérées, mais également contrôler quels flux de données sont récupérés par l'utilisateur. Pour ce faire, la communication LoRaWAN doit être bidirectionnelle.

La sécurité doit être prise en compte tout au long de l'implémentation logicielle de ce travail, afin de respecter la philosophie adoptée dans le projet SmartCanton. Pour cela, un système d'échange d'informations sécurisé doit être mis en place. Ceci afin de pouvoir programmer les différents paramètres nécessaires à la connexion et la communication sur un réseau LoRaWAN.

Chapitre 2

État de l'art

2.1 *Smart Cities*

Les Smart Cities, également connues sous le terme villes intelligentes, sont des villes qui utilisent des technologies de l'information et de la communication afin d'améliorer les services urbains en faveur des citoyens [3]. Pour cela, ces villes utilisent un large panel de capteurs qui ont pour but d'acquérir des données afin d'améliorer la gestion des ressources et des actifs. De plus en plus de ces villes offres des flux de données en temps réel pour informer les déplacements de ses citoyens. La gestion des feux automatiques afin d'optimiser les déplacements des automobilistes et ainsi réduire les ralentissements du réseau routier en son un exemple.

Aux États-Unis, en 2017, 66 % des villes ont investis dans les technologies liées aux Smart Cities [4] selon une étude de la *National League of Cities* (NLC). Et parmi les 34 % restants, 25 % sont actuellement en train d'explorer les différentes possibilités d'investissement. La NLC indique que les principaux domaines connectés sont :

- les compteurs intelligents pour les services publics ;
- les signaux de trafic intelligents ;
- les applications de gouvernance électroniques ;
- les kiosques Wi-Fi ;
- les capteurs d'identification par radiofréquence (RFID) dans la chaussée.

Selon le rapport *Smart Cities Market* par Focus Area, le marché des Smart Cities se montera à 1 201,69 milliards USD en 2022 avec un taux de croissance annuel de 23.1 % entre 2017 et 2022 [5]. Ceci montre bien qu'il s'agit là d'un marché en plein développement qui attire beaucoup d'investisseurs. Toutes ces technologies doivent se baser sur des plateformes communes, afin d'éviter la fragmentation des données. Le véritable *challenge* réside justement sur cette plateforme qui doit être pensée afin d'être générique sans ajouter une trop grande complexité pour les différents partenaires. Il y a également la problématique des flux de données, car certains capteurs génèrent de grands flux de données (ex. capteurs en temps réel) alors que certains n'effectuent qu'une à deux mesures journalières. Ce sont donc deux types de capteurs ayant des besoins bien différents, que ce soit en termes de consommations ou de débit de données. L'une des raisons pour lesquels ces plateformes doivent accepter plusieurs protocoles en fonctions des différents besoins.

De plus en plus de villes, à l'image de celles des États-Unis, sont intéressées par ces nouvelles opportunités et Genève n'est pas en reste. En effet, Genève met déjà à dispositions de ses citoyens, ainsi que des entreprises, plusieurs données publiques.

Une partie de ces données sont disponibles sur le portail du SITG [1] sous forme de cartes interactives. La Figure 2.1 contient un visuel sur l'état des différents axes routiers à l'aide du portail du SITG. D'ici 2030, le canton de Genève s'est donné pour objectif stratégique de devenir un pilier dans le domaine des Smart Cities [6] en développant de nouveaux projets et services autour de cette thématique.

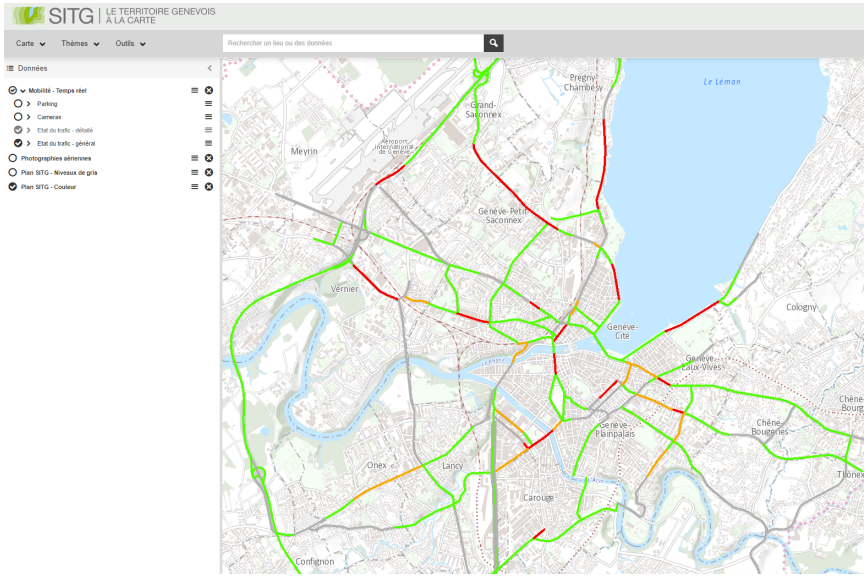


FIGURE 2.1 – Carte du trafic des principaux axes routiers de Genève en temps réel sur le portail SITG

En Suisse, il existe une initiative de l'Office fédéral de l'énergie et de SuisseEnergie nommée Smart City Suisse [7]. Il s'agit d'un programme ayant pour but de créer une plateforme d'information et d'échange pour les différentes villes de Suisse. Cette initiative est destinée aux communes qui souhaite planifier leur avenir afin d'accroitre leur statut de *Smart City* à l'aide de différents partenaires. Le programme se considère comme plateforme d'information et d'échange. À l'heure actuelle, cette plateforme a déjà aidé plus de 200 projets [2] à se développer en Suisse et en Europe dans le but d'améliorer la qualité de vie dans diverses villes. Les projets sont divisés en 10 thématiques, allant de la gestion de l'énergie à la santé, en passant par l'éducation et la sécurité.

Genève a annoncé, dans le cadre de son projet de SmartCanton [8], l'installation de capteurs sur 650 places de parkings. L'objectif de cette installation est de connaitre en temps réel l'occupation de ces places de stationnement dans le but d'informer les automobilistes des cases libres, mais également de faciliter le paiement du temps de stationnement. Le conducteur est guider à l'aide de panneaux de guidage dynamique ou d'une application smartphone [8]. En 2015, un premier projet pilote a été mis en place dans la commune de Carouge en utilisant des capteurs développés par l'entreprise genevoise IEM [3].

1. Système d'information du territoire à Genève (SITG) http://ge.ch/sitg/
2. http://ds1.dreifels.ch/smartcity/wprlist.aspx?LA=de
3. http://www.iemgroup.com/fr/solutions-produits/capteurs/presto-sense/

2.2 LPWAN

Les réseaux *Low-Power Wide-Area Network* (LPWAN) sont des réseaux qui sont de nos jours en pleine expansion, en partie grâce à la croissance des villes intelligentes [9]. Le placement des LPWAN à comparer aux différentes technologies sans fils est visible sur la Figure 2.2 [10]. On peut constater que les LPWAN s'associent bien avec les technologies existantes, en comblant des vides laissés par celles-ci. Les débits disponibles sur les LPWAN oscillent entre 0.3 kb/s et 50 kb/s [11]. Cela limite donc drastiquement les domaines d'application des capteurs qui peuvent utiliser ces technologies. Le débit de ces capteurs est restreint, mais les LPWAN sont surtout attractifs pour leur portée, et donne la possibilité de créer des périphériques de basse consommation, avec un faible cout unitaire. L'autonomie des dispositifs LPWAN peut atteindre plusieurs années en fonction de la fréquence des paquets envoyés et la taille de la batterie utilisée.

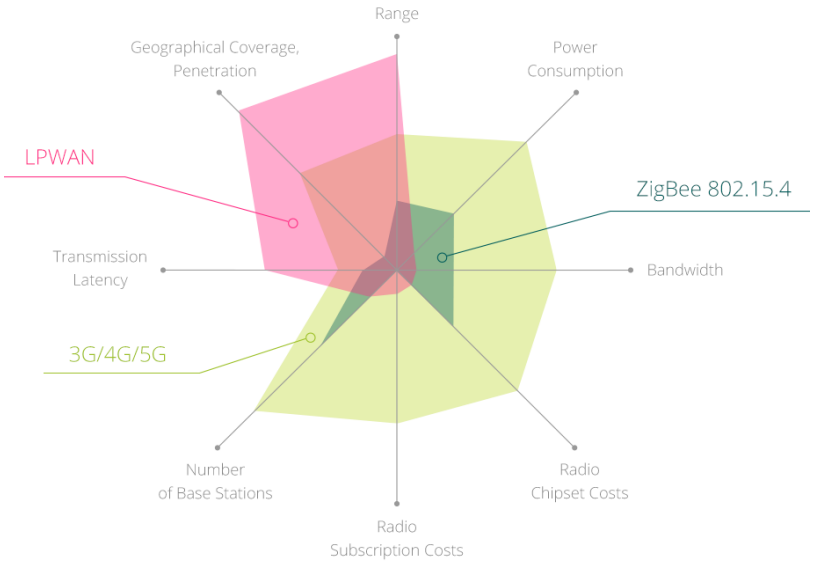


FIGURE 2.2 – Comparaison de LPWAN avec d'autres technologies

Il existe principalement deux types de technologies sur lesquels se base les différents protocoles :

1. LoRa : LoRa est une modulation radio utilisée par plusieurs protocoles tels que LoRaWAN [1], Haystack Technologies [2] ou encore Symphony Link [3] [9]. Cette modulation est la propriété du fabricant de semiconducteurs Semtech depuis 2012. Elle est principalement présente dans la gamme de fréquence *sub-gigahertz*, avec des fréquences comme 433 MHz et 868 MHz.
2. Ultra Narrow Band (UNB) : il s'agit là de technologies qui transmettent dans un spectre fréquenciel très faible (environ 1 kHz), afin d'émettre sur de très longues distances (5 km en zone urbaine et à plus de 25 km en champ libre). Ces distances peuvent être atteintes grâce à l'excellent bilan de liaison disponible sur ces connexions [12]. SigFox [4] et Nwave [5] sont deux grands acteurs du marché qui utilisent sur cette technologie.

1. https://www.lora-alliance.org/technology
2. http://haystacktechnologies.com/
3. https://www.link-labs.com/symphony
4. https://www.sigfox.com/en
5. https://www.nwave.io/

2.3 LoRa

LoRa n'est pas un protocole, mais une modulation de fréquence basée sur le principe du *chirp spread spectrum* (CSS) [9]. Du point de vue du modèle OSI, LoRa définit la couche 1 du module OSI [1] (couche physique), ainsi que quelques critères sur la couche 2 (liaisons) avec la spécification du format de la correction de données. La modulation la propriété de l'entité qui l'a développée, à savoir l'entreprise Semtech [2], et donc la seule à pouvoir la commercialiser. De ce fait, aucune documentation officielle décrivant le fonctionnement complet, n'est fournie. Semtech a déposé le 9 septembre 2015 un brevet avec le numéro de référence *US 20160094269 A1* [3] décrivant les principes de la modulation avec l'utilisation des *chirps* mais sans discuter en détails les *timings* et le codage utilisés. En 2016, au Jailbreak Security Summit [4], l'expert en sécurité Matt Knight, de l'entreprise Batille Networks, a effectué une présentation d'une heure sur le *reverse engineering* de LoRa, nommé *Reversing LoRa : Exploring Next-Generation Wireless* [5]. Il a exploré en détails comment la modulation LoRa se comportait en fonction des paquets envoyés, jusqu'à réussir à réaliser lui-même l'implémentation complète de la modulation à l'aide d'un *Universal Software Radio Peripheral* B210 [6].

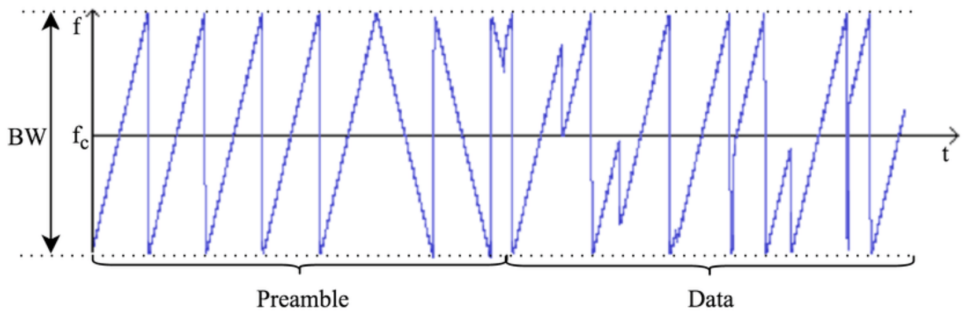


FIGURE 2.3 – Modulation *chirp spread spectrum* de LoRa

Le principe CSS de la modulation LoRa est visible sur la Figure 2.3 illustrant une trame LoRa utilisant cette même modulation. Le codage des informations consiste donc en la dérivée de la fréquence en fonction du temps, afin de coder les diverses informations désirées. Les états prennent en compte des valeurs précédentes et incluent des alignements sur les données afin de réduire l'impact du bruit sur le signal. Les tables expliquant le codage de la modulation sont disponibles dans le document de la présentation de Matt Knight au Jailbreak Security Summit.

L'un des défauts du LoRa est celui du choix du contrôleur effectuant l'implémentation matérielle de la modulation. À l'heure actuelle, un seul fabricant est autorisé à produire les puces avec la modulation LoRa, il s'agit de la société Semtech. Cela réduit considérablement le nombre de possibilités et détruit également la concurrence dans le domaine, puisque le prix est dicté par le créateur du standard. Semtech a annoncé une collaboration avec STMicroelectronics pour développer un processeur dans lequel on retrouve un STM32, avec en interne, un cœur attribué uniquement à la modulation LoRa et l'implémentation du protocole LoRaWAN. Mais en attendant

1. https://en.wikipedia.org/wiki/OSI_model
2. http://www.semtech.com/
3. http://www.google.ch/patents/US20160094269
4. http://www.jailbreaksecuritysummit.com/
5. https://github.com/matt-knight/research/tree/master/2016_05_20_jailbreak
6. https://www.ettus.com/product/details/UB210-KIT

ce microcontrôleur, Murata, un célèbre fabricant de semiconducteurs, s'est allié avec STMicroelectronics afin d'offrir un module nommé CMWX1ZZABZ-078. Ce module est composé, en interne, d'un STM32L0 et d'un *tranceiver* SX1276 de Semtech. La communication entre ces deux éléments est possible, via l'utilisation d'un bus SPI.

2.3.1 Sensibilité

L'augmentation de la bande passante du signal est le principe fondamental des différentes technologies de communications de type *spread-spectrum*. Sans cela, le transport sur de longues distances ne serait pas possible. La sensibilité de réception du LoRa par rapport à la modulation concurrence FSK est visible sur la Figure 2.4 [13]. On constate que le LoRa dispose d'une sensibilité plus importante pour un débit de données égal. Semtech propose une note d'application nommée `AN1200.22`[1] explorant plus en détail la modulation LoRa et ses puissances. Ce même document explique également les différents *noise floor* illustrés en Figure 2.4 à l'aide une approche mathématique.

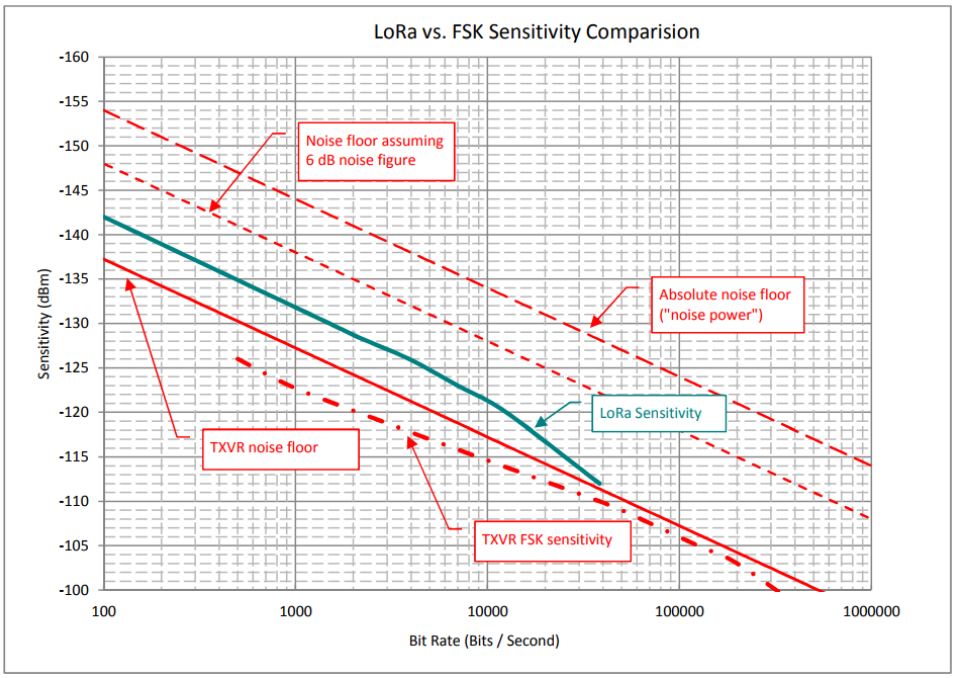


FIGURE 2.4 – Comparaison de la sensibilité entre la modulaton FSK et LoRa

2.3.2 *Coding rate*

En télécommunications, le *coding rate* est utilisé pour ajouter une correction d'erreur lors de la récupération d'un paquet. Si le *code rate* est de `k/n`, pour chaque `k` bits d'information utile, l'encodeur génère au total `n` bits de données [14]. Cela a pour résultat d'ajouter des données dans un paquet. Dans le cas de LoRa, cela augmente temps d'émission, réduisant ainsi le nombre de paquets qui peuvent être envoyés par unité de temps.

1. https ://www.semtech.com/uploads/documents/an1200.22.pdf

2.3.3 *Spreading factor*

Le *Spreading Factor* (SF) est un facteur qui est calculé en tenant compte de la durée d'un décalage en fréquence. Par exemple, sur la Figure 2.3 il s'agit du temps que met le *shift* pour parcourir BW. LoRa opère avec des *Spreading Factors* de 7 à 12. Chaque incrémentation de 1 du SF, double le temps d'émission d'un *chirp* [15]. Ce qui a pour conséquence — avec la même bande passante — d'augmenter le temps d'émission et donc réduire le débit de données.

2.3.4 Structure des paquets

La spécification LoRa décrit essentiellement l'implémentation d'une modulation. Cependant, une partie de la couche 2 du modèle OSI [1] est également affectée par cette spécification. Ce paquet est constitué des éléments présentés sur la Figure 2.5.

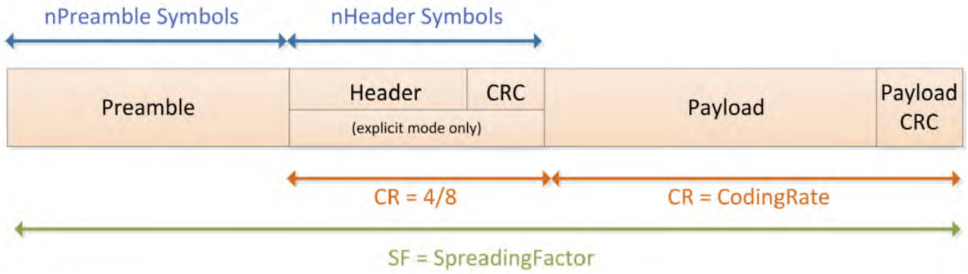


FIGURE 2.5 – Composition d'un paquet LoRa

Un paquet LoRa commence toujours avec un préambule constant qui couvre toute la bande de fréquence contenant à la fin un byte de synchronisation [16]. Ce byte permet ainsi de différencier les différents réseaux LoRa qui utilisent les mêmes bandes de fréquences. Un périphérique configuré pour écouter un byte de synchronisation précis va s'arrêter si le byte décodé ne correspond pas au byte attendu. La taille du préambule est configurable entre 10.25 et 65539.25 symboles [16]. Une image du préambule dans le domaine fréquentiel est visible sur la Figure 2.3.

Suivant le préambule, il y a un entête qui est optionnel. Si celui-ci est présent, il est toujours envoyée avec un *coding rate* de 4/8. Cet entête a pour but d'indiquer la taille du *payload* en bytes, le *coding rate* utilisé pour le reste de la transmission, ainsi que spécifier si la trame finit par un CRC 16-bit ou non. L'entête inclut son propre CRC pour éviter les erreurs et ainsi ignorer le paquet au cas où il aurait été corrompu. La taille du *payload* est définie uniquement à l'aide d'un *byte*, ce qui limite la taille totale du *payload* à 255 *bytes* [16]. L'entête n'est pas requis lorsque le *payload* a une taille fixe et que le récepteur connait la configuration du paquet (taille *payload*, *coding rate* et si un CRC est présent ou non).

2.3.5 Distance d'émission

L'entreprise Libelium [2], spécialisé dans la production et la vente de plateformes de développement IoT, a effectué plusieurs tests sur la portée du LoRa [17]. Pour ce faire, deux périphériques ont été placés à 21 km de distance dans une zone non urbaine. Le placement de ces périphériques est visible sur la Figure 2.6, où il est possible de voir à gauche une localisation GPS et à droite le profil du terrain. On constate un

1. https://en.wikipedia.org/wiki/OSI_model
2. http://www.libelium.com/

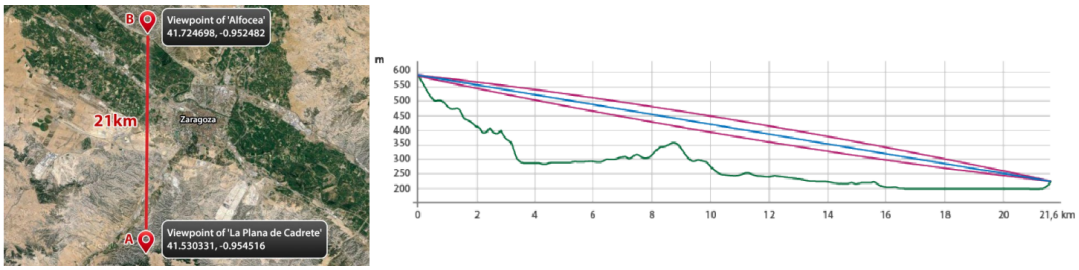


FIGURE 2.6 – Tests de portée à l'aide de deux dispositifs LoRa avec visualisation de la topologie du terrain

dénivelé entre les deux périphériques d'environ 400 m.

Plusieurs mesures ont été effectuées entre les deux périphériques en appliquant différents paramètres. Ces résultats sont visibles en Figure 2.7. Les modes définissent une combinaison de *bandwidth* (BW), de *coding rate* et de *spread factor* (SF) décrits en détail à l'aide de la Figure 2.8. La modification de ces paramètres change la sensibilité d'émission et de réception. En analysant les différents résultats, on constate que le taux de réception des paquets est très élevé, avec des résultats proches de 100 % dans trois modes [17]. Le mode 9 se révèle quant à lui inadapté pour ce type de transfert, il est donc à proscrire pour de longues distances.

LoRa Mode	Range	Power	Channel	Success (%)	Mean SNR (dB)	Mean RSSI (dBm)	Mean RSSI packet(dBm)	Sensitivity (dB)	Margin (dB)
Mode 1	21.6 km (13.4 miles)	High	CH_12_868	100	-9.79	-113.72	-126.79	-134	7.21
		Max		100	-4.33	-113.76	-121.76	-134	12.24
		High	CH_16_868	100	-10.06	-114.28	-127.06	-134	6.94
		Max		100	-3.20	-113.97	-120.21	-134	13.79
Mode 3	21.6 km (13.4 miles)	High	CH_12_868	95	-10.29	-114.16	-127.29	-129	1.71
		Max		95	-3.73	-114.08	-120.73	-129	8.27
Mode 6	21.6 km (13.4 miles)	High	CH_12_868	99	-14.77	-107.22	-125.77	-125.5	-0.27
		Max		100	-8.42	-106.60	-119.43	-125.5	6.07
Mode 9	21.6 km (13.4 miles)	High	CH_12_868	0	-	-	-	-117	-
		Max		49	-9.95	-107.68	-120.95	-117	-3.95

FIGURE 2.7 – Résultats des distances maximales LoRa réalisées par Libelium

Mode	BW	CR	SF	Sensitivity (dB)
1	125	4/5	12	-134
3	125	4/5	10	-129
6	500	4/5	11	-125,5
9	500	4/5	8	-117

FIGURE 2.8 – Description des modes utilisés par Libelium

2.3.6 Cycle d'opération

Le cycle d'opération (*duty cycle* en anglais) est grandement limité sur les bandes de fréquences utilisées par le LoRa en Europe. Ces fréquences diffèrent en fonction

des pays, mais en Europe les bandes de fréquences autorisées sont 863-870 MHz et 433 MHz. Ces cycles d'opérations sont réglementés par la section 7.10.3 du standard ETSI EN300.220 [18] pour l'Europe. Voici le pourcentage de temps que les différentes bandes de fréquences peuvent être utilisés en pourcentage :

1. g (863.0 – 868.0 MHz) : 1 %
2. g1 (868.0 – 868.6 MHz) : 1 %
3. g2 (868.7 – 869.2 MHz) : 0.1 %
4. g3 (869.4 – 869.65 MHz) : 10 %
5. g4 (869.7 – 870.0 MHz) : 1 %

On constate qu'il s'agit d'une restriction considérable sur l'envoi des données. Le pourcentage est basé sur le temps total d'une heure. C'est-à-dire que pour le cas des 1 %, on ne peut émettre que pendant 36 secondes par heure. Le temps d'émission d'un paquet peut être de plusieurs secondes, selon le type de *coding rate* et *spreading factor* choisi.

La puissance est également spécifiée par ETSI pour les différents canaux possibles dans leur document.

2.4 LoRaWAN

LoRaWAN définit le protocole de communication alors que LoRa est la couche physique sur laquelle se repose le protocole LoRaWAN. LoRaWAN opère principalement en couche 2 du modèle OSI, en tant que *medium access control* (MAC). Les données envoyées au travers d'un réseau LoRa sont chiffrées en AES-128 [1]. Le protocole LoRaWAN est exploré plus en Section 3.2 de ce document. La sécurité LoRaWAN est, quant à elle, explorée en Section 4.2.

LoRaWAN autorise l'envoi de données bidirectionnelles, mais les messages *downlink* (de l'application jusqu'aux périphériques) ne sont transmissibles que sous certaines conditions.

L'implémentation du protocole LoRaWAN est fait uniquement en logiciel. La plupart des implémentations proposées par les fabricants de microcontrôleurs sont basées sur le code de Semtech nommé LoRaMAC-node [2]. Ceci permet ainsi de faciliter ainsi le développement sur les microcontrôleurs qui sont couplés aux contrôleurs LoRa vendus.

2.5 Communication sans fil à 2.4 GHz

La plage de fréquence autour de 2.4 GHz est l'une des plus utilisée. Ceci est dû au fait que celle-ci peut être exploitée sans nécessiter l'obtention d'une licence provenant d'une autorité de certification et cela dans tous les pays [19]. Plusieurs autres fréquences sont également ouvertes dans le monde, mais le succès de cette plage provient de trois facteurs :

— la taille de l'antenne. En effet, un principe simple en radio communication est que plus la fréquence est élevée et plus l'antenne peut être compacte. On peut facilement le constater avec l'Équation 2.1 représentant la longueur d'un dipôle [20]. La constante A dépend principalement de la longueur de l'antenne

1. https://en.wikipedia.org/wiki/Advanced_Encryption_Standard
2. https://github.com/Lora-net/LoRaMac-node

en fonction de l'épaisseur du matériau utilisé. Cette constante se situe typiquement entre 0.96 et 0.98 [20]. Il est possible de trouver divers graphiques permettant d'obtenir la valeur de cette constante [1], facilitant ainsi son dimensionnement.

$$\text{Longueur dipôle } [m] = \frac{150 \times A}{freq[MHz]} \tag{2.1}$$

$$\text{Longueur dipôle } [m] = \frac{150 \times 0.98}{2400} = 0.06125 = 6.125[cm] \tag{2.2}$$

$$\text{Longueur dipôle } [m] = \frac{150 \times 0.98}{433} \approx 0.33949 = 33.949[cm] \tag{2.3}$$

En utilisant l'Équation 2.1 avec 2.4 GHz (Équation 2.2) et 433 MHz (Équation 2.3) on observe bien l'influence de la fréquence sur ce dipôle et la place qui devra être allouée à une antenne sur une carte électronique. Grâce à une miniaturisation qui devient raisonnable, on peut facilement offrir des périphériques compacts.

— la technologie électronique disponible pour gérer ces hautes fréquences. Grâce à l'avancée dans les domaines de l'électronique et de la microélectronique, il est maintenant possible d'avoir des circuits intégrés pouvant offrir la modulation à cette fréquence et cela en ayant une taille raisonnable. Par exemple, le microcontrôleur utilisé dans ce projet, le KW41Z de NXP, offre un *package* de 3.9x3.8 mm ne nécessitant aucun élément électronique externe pour l'amplification de l'antenne. Ceci permet de créer une carte électronique extrêmement compacte. Le prix de ce type de technologie est de plus en plus négligeable lors du développement(cf. sous-section 5.3.1 pour un comparatif plus approfondi des microcontrôleurs compatibles Bluetooth avec des indications de prix).

— la distance couverte par cette plage de fréquence. Cette fréquence permet d'avoir une distance tout à fait raisonnable entre deux périphériques qui souhaitent interagir. Le Bluetooth 5.0 peut ainsi atteindre, en théorie, jusqu'à 500 m dans un environnement ouvert, avec son mode *long range*[21]. Bien entendu, selon le type de réseau que l'on souhaite créer cette distance est peut-être trop faible. Cependant, il est toujours possible d'utiliser des protocoles qui implémentent des principes de *mesh networking* afin de propager les messages à travers un réseau de nœuds.

La fréquence de 2.4 GHz est utilisée par de nombreux protocoles. On y trouve le Bluetooth, le Wi-Fi, ou une multitude d'autres standards, comme le IEEE 802.15.2. Si on prend l'exemple d'un smartphone ou d'une tablette, les seuls protocoles supportés universellement sur toutes ces plateformes sont le Bluetooth, ainsi que le Wi-Fi. Le problème de ce dernier est l'impact sur la consommation des appareils, mais avec comme avantage d'avoir un grand débit de données possibles. Le Bluetooth est lui plutôt utilisé lors de la communication avec des périphériques proches de l'utilisateur et nécessitant un débit de données moyen, mais offrant en contrepartie une plus grande autonomie. Le Wi-Fi est quant à lui utilisé dans le cadre d'un besoin de communication avec des données externes, provenant par exemple d'Internet.

1. http://www.radio-electronics.com/info/antennas/dipole/length-calculation-formula.php

2.6 Bluetooth Low Energy

Le standard Bluetooth existe depuis 1994 et a été inventé par Ericsson [22]. Il a été premièrement conçu pour créer une alternative à la connexion filaire RS-232. Le standard est maintenu par le *Bluetooth Special Interest Group*[1].

Le Bluetooth Low Energy initiliament nommé Wibree, a été pensé par Nokia avec une collaboration entre STMicroelectornics et Logitech [22]. La sortie publique a été annoncée en 2006. En 2007, le *Bluetooth Special Interest Group* a pris la décision d'implémenter au sein du protocole Bluetooth les concepts avancés par le Wibree. En 2010, la spécification Bluetooth Low Energy en version 4.0 a été annoncée. Il a fallu attendre fin 2011 pour qu'elle fasse son apparition dans les premiers équipements grand public, comme l'iPhone 4S. Les principales différences entre le Bluetooth et le Bluetooth Low Energy sont illustrées sur la Figure 2.9. Il est possible de voir que la grande différence provient du débit. Avec le Bluetooth Low Energy, on limite le nombre de communications afin d'économiser un maximum d'énergie. Cela ouvre la porte à des dispositifs très basse consommation, qui peuvent communiquer durant plusieurs mois, mais en contrepartie limite le nombre de données transitant sur ce type de communication.

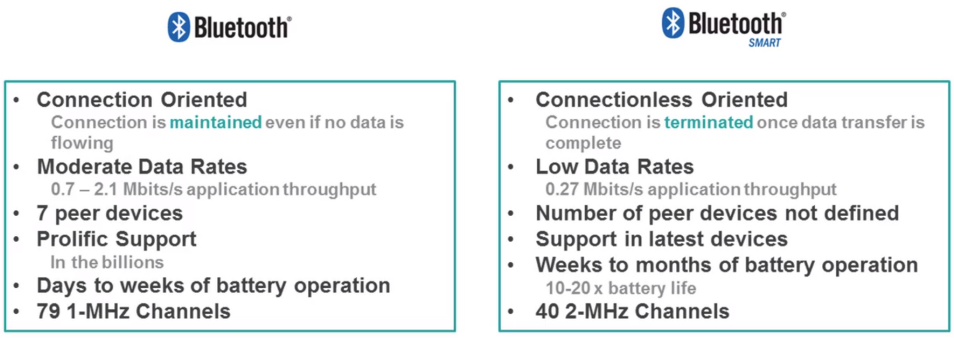


FIGURE 2.9 – Comparaison Bluetooth et Bluetooth Low Energy

Le Bluetooth 5.0 a été officiellement annoncé le 16 juin 2016. Néanmoins, les premiers microcontrôleurs implémentant toutes les fonctionnalités de ce protocole n'ont été annoncés que courant 2017. Une partie des prérequis pour passer au Bluetooth 5.0 ne nécessitent qu'une mise à jour logicielle. Par exemple, l'augmentation de la taille des *advertissements*, qui passent de maximum 31 bytes à 255 bytes. Le Bluetooth 5.0 crée également deux nouveaux modes, le mode *long range* (nommé LE Coded) et le mode *high speed* (nommé LE2M)[21]. Le premier offre une distance accrue au détriment du débit, alors que le deuxième mode est son opposé.

Le marché du Bluetooth est en pleine expansion depuis plusieurs années. Cette tendance est due à la démocratisation des smartphones. Le Bluetooth est en effet l'une des rares interfaces de communication présente sur ces dispositifs avec les communications cellulaires et WiFi.

1. https://www.bluetooth.com/

Le protocole Bluetooth est exploré plus en Section 3.1. La sécurité du Bluetooth est quant à elle explorée en Section 4.1 de ce document.

2.7 IEEE802.15.4

L'IEEE 802.15.4 est un protocole utilisé dans les communications sans fil et définit par l'association professionnelle *Institute of Electrical and Electronics Engineers* (IEEE)[23]. Les protocoles utilisant ce standard comme base font partie d'une catégorie nommée *Low Rate Wireless Personal Area Network* (LR WPAN) due à la basse consommation, au faible débit, ainsi que la faible portée de communication. Ce protocole définit trois fréquences d'utilisation possibles :

1. 2.4 à 2,4835 GHz avec l'utilisation de 16 canaux ;
2. 902 à 928 GHz avec l'utilisation de 10 canaux ;
3. 868 à 868,6 GHz avec l'utilisation de 1 canal.

Ces types de protocoles sont principalement présents dans la domotique par le fait de leur basse consommation, mais surtout de l'avantage des réseaux maillés (*mesh networking*).

Il est important de faire un petit comparatif des différents protocoles disponibles afin de les comparer au Bluetooth. En effet, plusieurs microcontrôleurs implémentant du Bluetooth peuvent, en parallèle, être membre d'un réseau IEEE802.15.4 (cf. KW41Z de NXP).

2.7.1 ZigBee

Le ZigBee[1] est un protocole très puissant en raison de sa conception, car ce protocole a été pensé pour interconnecter plusieurs périphériques avec la possibilité d'intégrer directement le principe de réseau maillé (*mesh*). Plusieurs architectures sont possibles avec ce réseau, illustrées par la Figure 2.10. Il a été conçu en 1998 et standardisé en 2003[24], ce qui fait de lui un protocole reconnu et présent dans beaucoup de réseaux liés à l'IoT.

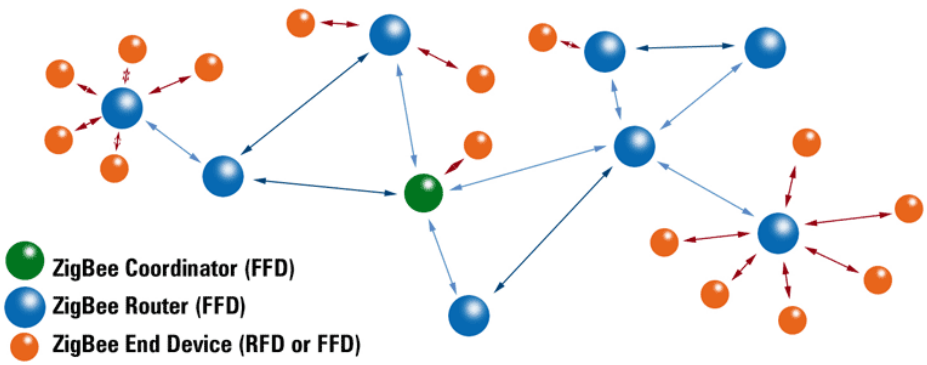


FIGURE 2.10 – Exemples d'architectures avec la technologie ZigBee

Un exemple de réseau ZigBee est visible sur la Figure 2.10. Ces réseaux sont composés de 3 éléments :

1. http://www.zigbee.org/

1. `Périphériques` (*End Device*) : sont les éléments qui fournissent les données au réseau, c'est sur ces derniers que les capteurs acquérant les données sont présents. Ce sont ces dispositifs qui sont les moins gourmands en énergie sur le réseau, car ils peuvent dormir la plus grande partie du temps, puis se réveillent pour acquérir les données et/ou les envoyer sur le réseau. Le périphérique doit informer le réseau quand il a été déplacé, mais également interroger son parent pour savoir s'il a reçu un nouveau message.
2. `Routeurs` : sont les éléments responsables pour router le trafic entre les différents nœuds. Dans certaines implémentations du protocole (version 3.0) [25], ces routeurs ne peuvent pas dormir. Ce sont donc des éléments qui ne sont pas adéquats pour être alimentés uniquement par batterie.
3. `Coordinateur` : est un routeur avec des caractéristiques spéciales[25]. En plus de toutes les tâches du routeur, il a pour rôle de former la topologie du réseau. Pour cela, il doit sélectionner les canaux de communication, les identifiants des éléments du réseau, ainsi que l'adresse du réseau elle-même. Il a également un rôle principal dans la sécurité, car il valide les nouveaux périphériques et contrôle que les données fournies sont de confiance. Les clés de tous les dispositifs du réseau sont également générées par ce dernier.

Le ZigBee est disponible sur les 3 bandes de fréquences autorisées par le standard 802.15.4. Mais certaines de ces fréquences sont limitées géographiquement. La fréquence la plus utilisée est le 2.4 GHz, du fait de son autorisation internationale.

Le routage des paquets ZigBee se produit à deux niveaux[24]. Le premier est au niveau de la couche réseau et peut être de type direct ou indirect. Le routage indirect se produit lorsque le dispositif ne connait pas l'adresse du destinataire. Un nœud de type routeur ou coordinateur fait alors la relation avec le destinataire final en utilisant sa table de routage et sa table de découvertes des routes. Le routage est direct, lorsque le dispositif qui transmet les données connait l'adresse réseau du destinataire.

L'autre type de routage se produit au niveau de l'applicatif en utilisant la table de liaisons stockée dans un coordinateur ou routeur. Les liaisons sont des liens logiques entre des dispositifs d'application complémentaire et des capteurs/périphériques. Le routage au niveau de l'applicatif se fait grâce à la table de liaison, contenu dans le coordinateur ou un routeur.

2.7.2 Thread

Thread est un protocole basé sur le principe *IPv6 over Low-Power Wireless Personal Area Networks* (6LowPAN)[26]. Les objets d'un réseau Thread sont directement accessibles via des adresses IPV6. Le protocole a été créé par Nest Labs (racheté en 2013 par Google/Alphabet)[27] et est compatible avec tous les produits de la marque Nest. Une des forces de Thread réside dans les partenaires du projet, car on y trouve ainsi tous les grands fabricants de microcontrôleurs comme NXP, Silicon Labs et Texas Instruments, mais également ARM à l'aide des bibliothèques pour mbed OS [1].

Les différentes couches ajoutées par Thread sont visibles sur la Figure 2.11. On constate que Thread utilise UDP et non TCP pour l'échange de messages. Réduisant ainsi l'*overhead* rajoutée par les paquets UDP et ainsi réduire la complexité pour

1. https://www.mbed.com/en/platform/mbed-os/

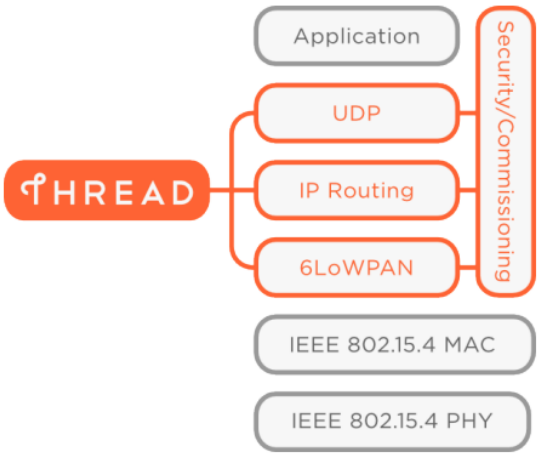


FIGURE 2.11 – Architecture d'une *stack* logicielle Thread

former les trames de données sur les différents périphériques du réseau.

Thread a été construit avec la sécurité comme élément primordial. En effet, tous les messages sont chiffrés en AES-128. La génération des clés et leur échange sont basés sur ECC et J-PAKE[28]. Pour ajouter un nouveau capteur au réseau, un périphérique de mise en service connu et approuvé au réseau est nécessaire (ex. : un smartphone). Pour rajouter un capteur, le périphérique de mise en service doit donner l'identifiant du capteur et fournit un mot de passe au réseau. L'échange de ces informations est effectué à l'aide d'une connexion sécurisée basée sur le protocole *Datagram Transport Layer Security* (DTLS), après l'authentification de l'utilisateur.

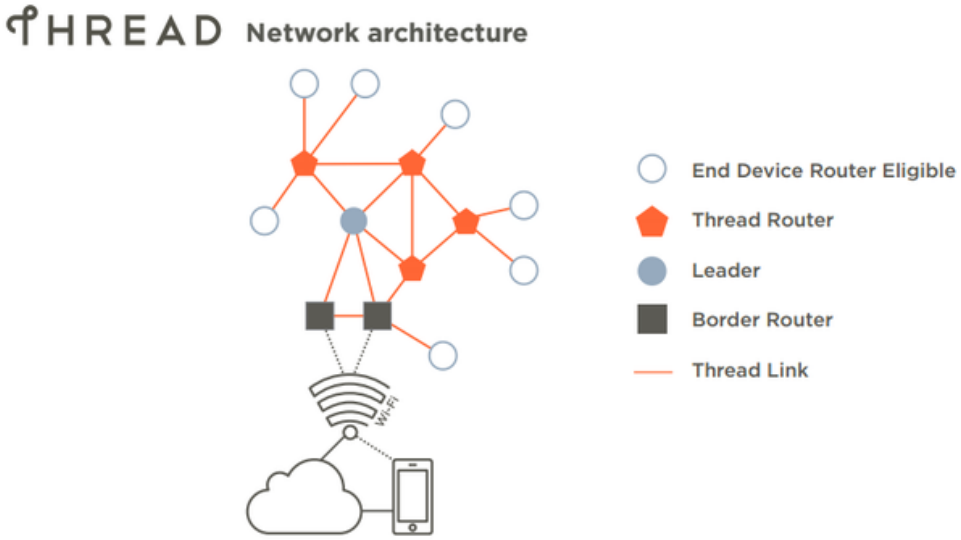


FIGURE 2.12 – Architecture d'un réseau Thread

Contrairement au protocole ZigBee, Thread est beaucoup plus dynamique dans son réseau. Cela s'explique par le fait que seulement deux types de nœuds sont réellement possibles dans un réseau Thread [28]. Un troisième type est ensuite élu par

ce même réseau. Voici la définition des types qui sont également visibles sur la Figure 2.12 :

1. `End Device` : est le capteur qui va fournir des données vers le réseau. Ceux-ci sont souvent alimenté par batterie et ne doivent s'activer que sporadiquement pour être efficients.
2. `Router Eligible` : s'agit d'un `End Device`, mais pas obligatoirement. Si par exemple, le réseau a besoin d'un nouveau routeur, par grand trafic de données ou lorsque les routes créées ne sont pas assez nombreuses, il devient un routeur sur le réseau. Son but est alors d'écouter l'arrivée de nouveaux paquets et de les rediriger vers le *End Device* de destination ou le routeur le plus proche. Étant donné qu'un routeur doit écouter toutes les communications, il ne pas être considéré comme un périphérique *low power*.
3. `Leader` : est un `Router Eligible` qui se fait élire à ce poste par le réseau. Ce leadeur peut changer au fil du temps, mais un seul peut être actif par réseau. Il doit s'occuper de tâches supplémentaires sur le réseau et de prendre la plupart des décisions.

Les *border router* sont des routeurs qui proposent un pont vers un autre réseau. Dans l'exemple exposé en Figure 2.12, un pont Internet ou un réseau local est présent pour que les capteurs soient directement accessibles depuis un smartphone ou un service quelconque. Ces *border routers* peuvent aussi convertir vers d'autres types de technologies, par exemple, vers un réseau Bluetooth.

Les principes de Thread sont similaires à ZigBee, mais l'apport de la technologie IPv6 est un avantage dans l'IoT, car cela facilite l'accès aux capteurs depuis un réseau externe.

2.8 Équipements matériels existants

Il existe sur le marché divers fabricants proposant des kits de développement orientés IoT et très facilement utilisables dans un concept *SmartCity*. L'un des plus connus est STMicroelectronics [1], car il propose une très grande panoplie de capteurs pouvant êtres utilisés dans un grand nombre de situations [2]. Les kits de développement de STM sont souvent utilisés par des passionnés, au vue de leur de disponibilité et de leur faible cout sur le marché. Des kits avec du WiFi, du NFC et du Bluetooth peuvent être achetés pour moins de 50 CHF. L'écosystème STM dispose d'une très grande communauté, ainsi que de bibliothèques très bien construites qui facilitent l'implémentation d'un projet personnel. Une grande partie de leurs kits de développement sont supportés par la plateforme mbed [3], maintenue par la société de conception d'architecture de processeurs ARM. La plateforme mbed offre ainsi une suite de bibliothèques facilitant l'implémentation et la programmation. Un problème provenant de ces kits est la disponibilité. Ces kits ne sont pas censés être utilisés pour des produits définitifs. Ils sont uniquement des plateformes de tests pour le produit vendu, ce qui n'oblige aucunement le vendeur à garantir une disponibilité ni même une compatibilité entre les futurs designs. La production de ces kits peut ainsi être interrompue du jour au lendemain. Générant une situation inacceptable si cette carte est utilisée dans un produit commercial. Ils ont comme objectif de réaliser très facilement des prototypes permettant ainsi de créer des *proof of concept* rapidement.

1. http://www.st.com/content/st_com/en.html
2. http://www.st.com/en/evaluation-tools/stm32-nucleo-expansion-boards.html
3. https://www.mbed.com/en/

Toutefois, lorsque l'utilisateur désire avoir un produit final, il est forcé d'implémenter le processeur directement dans une carte électronique sur mesure. Plusieurs acteurs proposent des kits sur le marché. Les plus grands fabricants de microcontrôleurs sont STM, NXP ou Texas Instruments, proposant chacun des plateformes et des exemples afin de tester leurs produits.

Les dispositifs matériels sur le marché implémentant une interface de communication utilisent parfois deux processeurs. Un premier processeur, souvent le moins puissant, est utilisé pour la gestion de l'interface de communication. Il s'occupe à lui seul de la communication avec l'extérieur. Ceci peut s'avérer une tâche couteuse selon le type de protocole implémenté. Par exemple, lorsque de la cryptographie est utilisée, le chiffrement et déchiffrement des paquets peut être couteux en temps de calcul. Ce processeur communique ensuite avec un processeur plus puissant via une interface de communication variant selon les implémentations. Un deuxième processeur est à disposition de l'utilisateur pour toutes les tâches qu'il souhaite effectuer. Mais il ne faut pas oublier que la communication avec le premier processeur aura un impact sur les performances. À cause de ce type de structure, mais aussi de la complexité à maintenir un système stable, un *Real-Time Operating System* (RTOS) est souvent utilisé.

Libelium [1] est également un acteur dans le marché des *Smart Cities*. La différence avec STMicroelectronics réside dans le fait que Libelium n'est pas un fabricant de microcontrôleurs en tant que tel. Cette entreprise propose plusieurs cartes dont la plupart sont basées sur un processeur d'Atmel, un ATMEGA1281. Cela permet une utilisation aisée de leurs équipements, mais limite malheureusement la puissance de calcul. Ces processeurs ne sont pas très puissants et l'utilisation de l'environnement Arduino pour la programmation réduit considérablement la flexibilité et la puissance de calcul pour l'utilisateur. Un autre défaut des plateformes Arduino est l'absence d'une interface de débogage (ex. JTAG [2]).

Dans l'industrie électronique, un périphérique produit à grande échelle sera toujours réalisé avec la plus grande maitrise possible sur l'implémentation, qu'elle soit matérielle ou logicielle. Un travail de Master doit refléter un besoin de l'industrie et valoriser de nouvelles compétences pour l'étudiant. Toutes ces raisons ont incité la réalisation d'une carte de développement personnalisée pour ce projet.

2.9 Développement de logiciels sur microcontrôleur

Cette section a pour but d'explorer l'état actuel des systèmes d'exploitation et des diverses bibliothèques utilisables sur des systèmes embarqués. Dans le cadre de ce projet, deux microcontrôleurs ont été utilisés, l'aspect logiciel a donc été exploré pour connaitre les différentes approches possibles.

2.9.1 Stacks Bluetooth

Le problème lorsque l'on utilise un microcontrôleur avec une interface Bluetooth ou de manière générale un quelconque protocole radio réside dans le fait que le code source s'occupant de la partie radio n'est que très rarement *open source*. Il est souvent

1. http://www.libelium.com/
2. https://en.wikipedia.org/wiki/JTAG

distribué sous forme binaire avec une documentation principalement applicative, sans aucune documentation du fonctionnement interne du code (via des machines d'états, etc.). Ce code est souvent référé en tant que *stack RF*.

On trouve tout de même certaines *stacks RF* tierces, comme la *stack* BTstack [1] développée par l'entreprise Bluekitchen [2] et la communauté *open source*. Toutefois le nombre de microcontrôleurs supporté se limite souvent à des microcontrôleurs assez anciens ou simplement ceux très appréciés par le milieu des amateurs de circuits électroniques ou de systèmes embarqués. On trouve ainsi le support des ESP32 du fabricant Espressif Systems, un des microcontrôleurs le plus prisé des hobbyistes par son très faible cout (moins de 4 dollars pour une seule unité sur des sites de ventes comme AliExpress). On a également ceux basés sur les MSP430 de Texas Instruments. Un microcontrôleur assez ancien (sorti en 2010), mais qui est encore l'un des plus connus de Texas Instruments par sa très basse consommation et sa simplicité.

2.9.2 Stacks LoRaWAN

La *stack* LoRaWAN est principalement développé par Semtech à l'aide du projet LoRaMac-node [3]. Cette *stack* requière uniquement au développeur de fournir des fonctions d'écriture et de lecture sur le bus série connecté au périphérique LoRa. La gestion des interruptions doit aussi être mise en place. Le protocole LoRaWAN est géré en intégralité par une machine. Son implémentation est entièrement séquentielle, la présence d'un RTOS n'est donc pas nécessaire.

2.9.3 RTOS

Pour l'implémentation logicielle, plusieurs alternatives sont possibles. Dans le cas de l'utilisation de processeurs implémentant des communications sans fil, un **Real Time Operating System** (*RTOS*) est très souvent nécessaire et conseillé. Cela est dû au fait que la partie radio est gérée par une tâche propre, souvent doté d'un niveau de priorité supérieur aux autres tâches, afin d'avoir la meilleure réactivité sur les différentes opérations.

La différence entre un système en *bare-metal* et un RTOS est visible sur la Figure 2.13 [29]. Un code complexe en *bare-metal* utilise une machine d'état dans une boucle infinie. Cette machine scrute des fanions modifiés par des événements (ex. une ISR [4]). Dans un RTOS, plusieurs tâches attendent en parallèles sur des événements générés par le déblocage de verrous (sémaphores ou mutexes). Le noyau principal d'un RTOS est l'ordonnanceur (*scheduler*), qui gère l'ordre d'exécution des tâches en fonction des priorités assignées à celles-ci.

Il est toujours possible de programmer en *bare-metal*, mais ce type de programmation devient de plus en plus difficile, suite à la complexification grandissante des systèmes connectés. Toutefois, un RTOS crée une surcouche trop importante pour garantir certaines contraintes. Cela arrive souvent avec des contraintes temporelles, quand une application nécessite de faire des opérations avec une grande réactivité [30]. Mais dans le cas de communication sans fil, une approche *bare-metal* est souvent trop complexe pour être possible. Une autre solution est l'utilisation de deux processeurs, un premier qui gère la communication sans fil, le deuxième s'occupe du reste

1. https://github.com/bluekitchen/btstack
2. http://bluekitchen-gmbh.com/
3. https://github.com/Lora-net/LoRaMac-node
4. https://en.wikipedia.org/wiki/Interrupt_handler

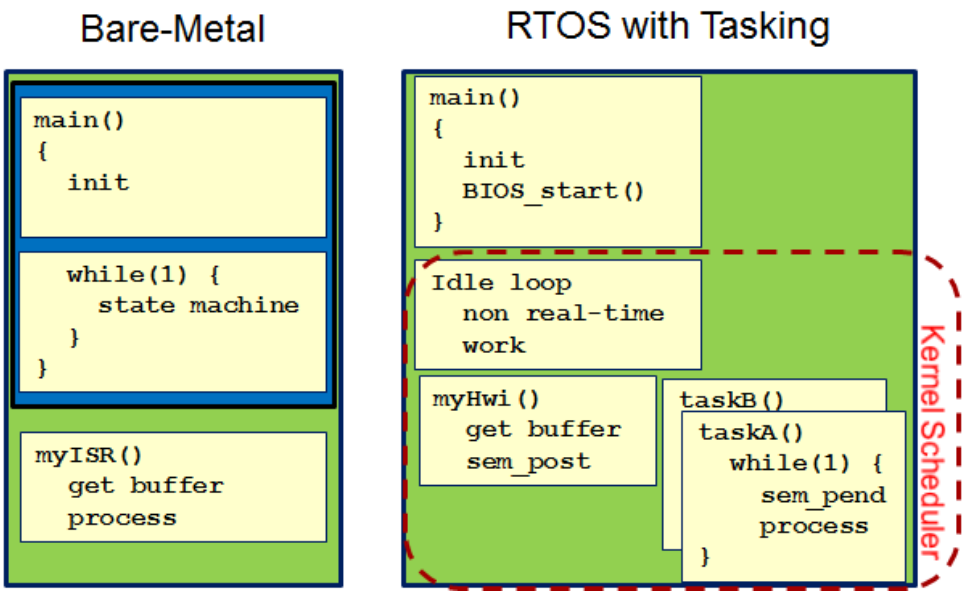


FIGURE 2.13 – Comparaison entre RTOS et bare metal

des opérations. Le processeur RF est donc considéré comme un coprocesseur dans cette architecture. La communication entre ces deux processeurs dépend ensuite des choix du fabricant et non du développeur.

2.9.3.1 FreeRTOS

Sur le marché on retrouve divers systèmes d'exploitation. L'un des plus connus est FreeRTOS, parce qu'il est intégralement *open source* [1]. Son succès a été porté sur plus d'une cinquantaine d'architectures [31].

Les dispositifs IoT sont de plus en plus complexes et connectés. L'une des raisons qui ont poussé l'un des leadeurs des produits *cloud* dans ce domaine, Amazon IoT, a développer avec l'organisation FreeRTOS un RTOS modifié [2] afin de s'intégrer au mieux avec leurs plateformes. Par défaut, les constructeurs de microcontrôleurs fournissent une variante d'un RTOS. Très souvent, cette variante est FreeRTOS, car elle peut être distribuée sans avoir de couts liés aux licences car étant basée sur une licence MIT.

En avril 2017, le magazine EE Times en collaboration avec le site Internet Embedded.com, tous les deux spécialisés dans les informations sur le monde des systèmes embarqués, ont à nouveau publiés une étude sur la part de marché des systèmes d'exploitation utilisés sur les divers systèmes embarqués[32]. Cette étude a été réalisée en questionnant les diverses personnes inscrites sur ces deux sites. Au total, 1'234 utilisateurs ont été retenus pour l'année 2017, avec un niveau de confidence de 95 % ± 2.8 %. Plusieurs questions ont été posées dans ce sondage, dont celle de nommer tous les OS sur lesquels ils risquent de travailler dans les douze prochains mois. La Figure 2.14 illustre les résultats de cette question. On constate que le premier OS est FreeRTOS, cité par 28 % des personnes. Néanmoins, cette étude ne se concentre pas sur les systèmes embarqués uniquement réservés à une approche sans fil, basse

1. https://sourceforge.net/projects/freertos/
2. https://aws.amazon.com/freertos/

FIGURE 2.14 – Résultats d'une étude de marché sur les OS réalisée par EE Times et embedded.com en avril 2017

consommation et avec une faible empreinte, comme dans le cadre de ce travail de Master. Certains de ces OS sont également bien trop complexes pour être utilisables sur des petits microcontrôleurs, c'est pour cela qu'on peut y voir des systèmes d'exploitation très complets comme Ubuntu. Les autres OS utilisables par des microcontrôleurs listés par cette étude et toujours visibles sur la Figure 2.14 sont ceux du développeur Micrium [1]. Il s'agit de deux RTOS nommés µC/OS-III et µC/OS-II. Ce sont par contre deux solutions propriétaires qui nécessitent le paiement de licences pour être utilisées. Cette dernière raison est probablement celle qui fait que FreeRTOS est aussi populaire dans le monde professionnel.

2.9.3.2 Zephyr

FIGURE 2.15 – Logo *Zephyr Project*

Le projet Zephyr, lancé le 17 février 2016 [33], est un système d'exploitation temps réel supportant un large éventail d'architectures. Ce projet est porté par la fondation Linux et a pour principal objectif de forcer la sécurité dans les systèmes embarqués avec des implémentations simples, mais efficaces [33]. Grâce au support de la fondation Linux, ce projet est extrêmement actif. Une nouvelle version est en moyenne disponible tous les 1 à 2 mois. La documentation est très complète avec plusieurs exemples afin de faciliter l'intégration aux développeurs.

Zephyr a plusieurs caractéristiques qui le différencie de certains concurrents, ce qui le rend attractif dans certains domaines :

1. *Single address-space* : l'application et le kernel utilisent la même plage d'adresse. Ces deux entités, une fois combinées créent une image monolithique qui sera ensuite exécutée sur le système embarqué.

1. https://www.micrium.com/rtos/

2. Hautement configurable : le *kernel* et l'application peuvent être configurés à l'aide d'une interface afin d'activer uniquement les éléments qui sont nécessaires. Cette interface se base sur le même principe que le noyau Linux à l'aide d'un *Makefile menuconfig*[1].
3. *Compile-time resource definition* : toutes les ressources sont définies à la compilation, ce qui réduit la taille du code source et augmente les performances des systèmes.
4. Vérification minimale des erreurs : afin d'augmenter les performances lors de l'exécution ainsi que la taille du code, il est possible de désactiver la vérification d'erreurs lors de la compilation de l'image. Une option est toutefois disponible pour ne pas les désactiver et ainsi faciliter le débogage des applications lors du développement.
5. Une suite de services pouvant être complétée au gré de l'utilisateur est disponible. Voici une liste exemplative de services sont déjà implémentés :
   — Un service de *multi-threading* avec la possibilité d'être préemptif avec un ordonnanceur en *round robin*.
   — Un service de gestion d'interruption qui gère toute la gestion de celles-ci, ainsi que leur enregistrement.
   — Un service de gestion de l'allocation et de désallocation de la mémoire dynamique (la taille de la mémoire allouée à ce service doit être prédéfinie en avance).
   — Service de synchronisation *inter-threads* avec l'implémentation de sémaphores, sémaphores compteurs et de mutexes.
   — Service d'envoi et de réception de données entre les *threads* à l'aide de queues de message et *bytes streams*.
   — Service de gestion de l'alimentation avec la possibilité de se mettre en mode *idle*, ainsi que d'autres modes basse consommation compatibles avec le matériel utilisé.

Zephyr supporte de multiples architectures telles que ARM Cortex-M, Intel x86, ARC, NIOS II, Tensilica Xtensa et RISC-V. Certaines cartes de développement sont supportées, telles que les ST Nucleo ou les nRF51 de Nordic Semiconductor. Toutefois, le grand défaut de ce framework réside dans le fait que les interfaces radio des différents microcontrôleurs ne sont pas supportées. Par exemple, seuls les microcontrôleurs Bluetooth nRF51 et nRF52 sont supportés. Trois cartes de développements de NXP sont prises en charge (NXP FRDM-K64F, NXP FRDM-KL25Z et NXP FRDM-KW41Z), toutes équipées de Bluetooth Low Energy, mais pas supportées par Zephyr.

2.9.3.3 Mynewt

Le RTOS Apache Mynewt est entièrement *open source* et a la même philosophie que Zephyr, c'est-à-dire, la sécurité avant tout . Il est développé avec le support de la très connue *Apache Software Foundation*. Son optique est le monde IoT avec des périphériques connectés, que ce soit via une technologie sans fils ou filaire. Voici les principales caractéristiques de cet OS :

— Support BLE 5.0 avec une stack Bluetooth personnalisée ;
— Support LoRa PHY et LoRaWAN avec le *transceiver* SX1276 ;
— Support du Bluetooth Mesh[2] ;

1. https://en.wikipedia.org/wiki/Menuconfig
2. https://blog.bluetooth.com/introducing-bluetooth-mesh-networking

FIGURE 2.16 – Logo *Apache Mynewt*

— Support pour TCP/IP et UDP ;
— Support pour des *Constrained-Node Networks*[1] (ex : CoAP, 6LoWPAN).

Sa première *release* officielle a été le 24 février 2016[2]. Un RTOS qui est très jeune, tout comme Zephyr, mais qui reste toutefois très intéressant et dont le développement autour de celui-ci semble prometteur. Une *release* majeure est déployée tous les trois mois, ce qui montre une forte implication de la part de la communauté dans le développement de ce projet.

La principale problématique de l'utilisation de ce RTOS est la même que Zephyr, à savoir, le nombre de périphériques actuellement supportés. Pour les périphériques Bluetooth, ce sont principalement les microcontrôleurs de Nordic Semiconductor qui sont supportés en intégralité. Il s'agit de la gamme de modèles NRF51 et NRF52. Dans la liste des microcontrôleurs on retrouve également plusieurs *chip* du fabricant NXP, comme le K64F ou le KW41Z. Ce premier a une interface Bluetooth et Ethernet. Cependant, à l'heure actuelle, uniquement l'interface Ethernet semble être supportée par Mynewt, ce qui n'as pas forcément de sens lorsque l'on souhaite utiliser un microcontrôleur offrant une interface Bluetooth.

Pour le LoRa, on peut utiliser n'importe quel type de microcontrôleur compatible avec Mynewt et ensuite connecter un SX1276 via une interface SPI. Mynewt propose ensuite tout un écosystème pour communiquer avec ce périphérique en lui assignant une tâche propre. Par défaut, les exemples de codes utilisant LoRaWAN reposent sur deux modules, le EE-02 et EE-04[3]. Ces modules ont été développés par l'entreprise Telenor Digital et les schémas de ces modules sont disponibles sous licence Open Hardware du CERN[4]. Ce module est équipé d'un nRF52832 du fabricant ainsi que du module SX1276 de Semtech. Malheureusement, aucun fournisseur ne semble proposer ce module directement à la vente à l'heure actuelle.

2.9.3.4 Couches d'abstraction

Certains constructeurs de microcontrôleurs supportent plusieurs RTOS et essayent de faciliter le changement de RTOS à ses utilisateurs. Freescale a donc créé un *framework* nommé Connectivity Framework. Celui-ci, uniquement compatible avec ses microcontrôleurs, a pour but de simplifier le changement de RTOS, d'uniformiser les diverses API et donc d'accélérer le processus de développement.

Les microcontrôleurs développés par Freescale (racheté par NXP) sont compatibles avec cette couche d'abstraction, car tous les RTOS sont basés sur des concepts

1. https://tools.ietf.org/html/rfc7228
2. https://github.com/apache/mynewt-core/releases?after=mynewt_0_9_0_rc1_tag
3. https://www.telenor.com/innovation/telenor-digital/
4. https://github.com/telenordigital/ee0x-hardware

similaires. En effets, on retrouve toujours les mêmes blocs, comme des sémaphores et mutexes pour gérer la concurrence entre les tâches. Il y a également des queues et des événements pour communiquer et synchroniser ces mêmes tâches.

Par défaut, Freescale/NXP propose FreeRTOS, mais il est possible de le remplacer par µC/OS-III et µC/OS-II du développeur Micrium. On retrouve également le support du MQX RTOS qui a été développé par Freescale.

2.9.4 Drivers de périphériques

De plus en plus de composants sont fournis avec des bibliothèques qui peuvent être facilement adaptées en fonction de la plateforme utilisée. C'est surtout le cas des périphériques externes. Ces bibliothèques demandent simplement à l'utilisateur d'implémenter des fonctions qui seront passées en paramètre à la bibliothèque. Deux fonctions sont toujours nécessaires, à savoir l'écriture et la lecture via les périphériques de communication.

Le fabricant de périphériques Bosch Sensortech dispose d'un dépôt Git public offrant des pilotes pour la plupart de ses capteurs :

`https://github.com/BoschSensortec?tab=repositories`

Une stratégie commerciale qui semble être bénéfique tant pour l'entreprise que pour l'utilisateur. En premier lieu, le code est souvent *open source*. Si une entreprise souhaite implémenter sa propre bibliothèque, il lui est possible d'analyser le code source des bibliothèques fournies pour s'en inspirer et aider à démystifier le contenu d'une *datasheet* ou à guider l'utilisateur. Il permet également une économie de temps pour les utilisateurs, ce qui constitue une raison supplémentaire pour utiliser le composant matériel vendu. Ces bibliothèques sont la plupart du temps maintenues par les fabricants des composants, mais on trouve également des implémentations communautaires.

Il y a tout de même des défauts à ces bibliothèques, notamment les ressources mémoires utilisées. Certaines de ces bibliothèques utilisent des fonctions de traitement de chaines de caractères qui sont lourdes, notamment, la fonction `sscanf`. Selon les capacités mémoires du microcontrôleur choisi, l'utilisation de fonctions aussi lourdes n'est pas envisageable. Des fonctions plus optimisées sont parfois disponibles sur Internet afin de réduire l'empreinte mémoire générée par celles-ci. Ces bibliothèques sont très complètes, parfois même trop selon l'utilisation qu'on souhaite en faire. Si l'utilisateur a un composant où il n'accède qu'à une donnée précise dans un registre fixe, la bibliothèque s'avère inutile car son surcoût devient trop important. L'utilisation doit ainsi être évaluée en fonction de l'utilité que celle-ci apportera au projet.

2.10 SmartCanton

Le canton de Genève s'est donné pour objectif stratégique de devenir un pilier dans le domaine des *Smart Cities* d'ici 2030 [6]. Celui-ci deviendrait ainsi un SmartCanton dans lequel les utilisateurs pourraient accéder aux données capturées et les utiliser dans le cadre de projets personnels et/ou commerciaux. Le but, une fois les données capturées, est de les faire parler. Pour ce faire, on peut imaginer plusieurs algorithmes de Machine Learning permettant de créer des modèles pour mieux conseiller les habitants, analyser les risques et réduire les couts de certaines actions qui nécessitent

actuellement la présence d'une personne physique à proximité.

Un *Proof Of Concept* (POC) a été lancé dans ce sens, afin de présenter différentes approches de ce SmartCanton, mais également les limitations et les différentes technologies possibles. La technologie retenue pour pouvoir remonter les données a été le LoRa. Plus spécifiquement, le protocole LoRaWAN.

2.10.1 Architecture

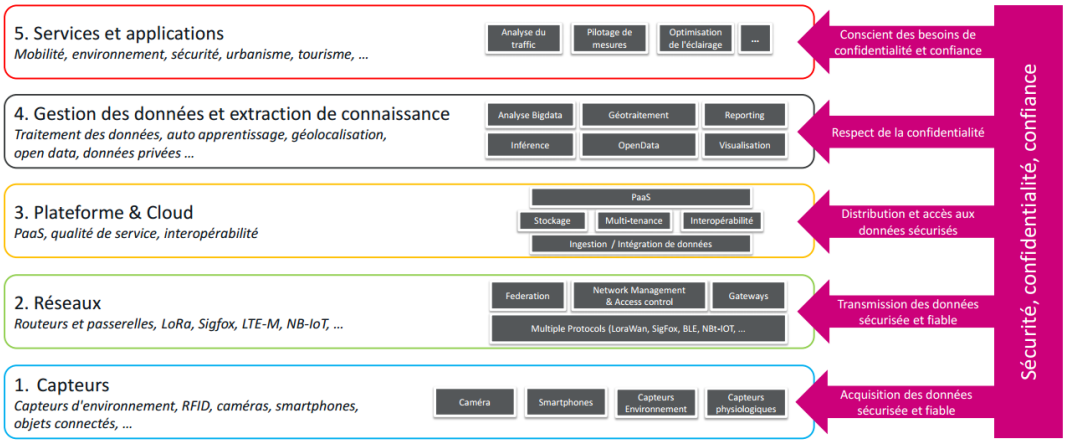


FIGURE 2.17 – Architecture générale du projet SmartCanton

Le POC peut être subdivisé en cinq couches interconnectées lesquelles sont détaillées sur la Figure 2.17. Le projet couvert par cette thèse de Master utilise principalement les deux premières de cette architecture. Le capteur est l'élément principal de ce travail, de même que son développement matériel et sa programmation. Un réseau doit ensuite être utilisé pour mettre à disposition les données capturées. Cette couche numéro 2 supporte ainsi plusieurs types de protocoles basés sur différentes technologies, tel que le LoRa, SigFox, etc. Dans le cadre de ce travail, l'accent porte sur la technologie LoRaWAN, couplée à la modulation LoRa. Parallèlement aux cinq couches de la Figure 2.17, on constate la présence de trois termes : Sécurité, Confidentialité et Confiance. Ceux-ci sont primordiaux dans les systèmes IoT modernes.

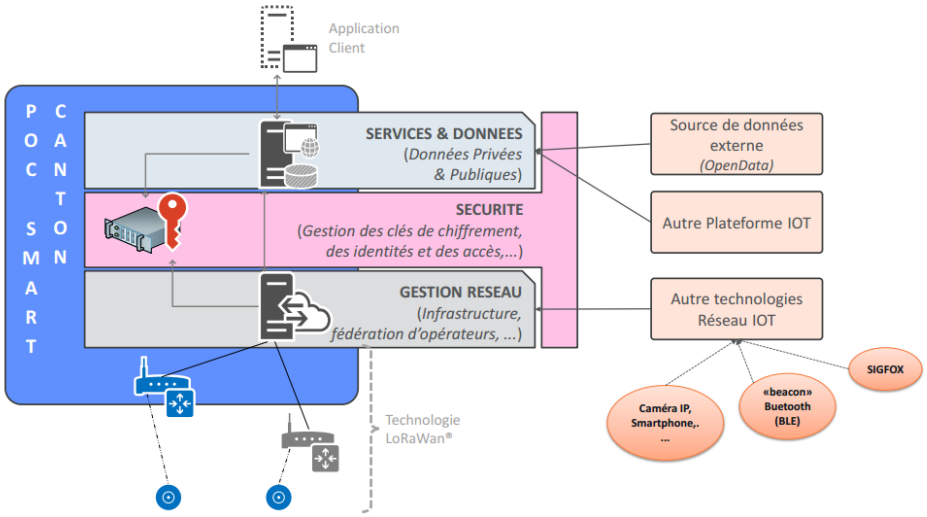


FIGURE 2.18 – Infrastructure LoRa du SmartCanton

La Figure 2.18 expose la situation d'un point de vue plus appliqué à un réseau LoRaWAN. Les ronds bleu sur l'images correspondent aux périphériques LoRaWAN qui souhaitent transférer leurs données. Sur la même image, deux *gateways* LoRaWAN différentes sont illustrées. Les raisons liées à cette différentiation sont exposés en Section 2.10.4.

2.10.2 Sécurité

Le POC a exploré en détail les problématiques liées à la sécurité, tout particulièrement celles liées à la technologie LoRaWAN. Un *Hardware Security Module* (HSM) (visible sur la Figure 2.18) a été intégré au coeur de la sécurité du SmartCanton. La sécurité utilisée au travers d'un réseau LoRaWAN est décrite à la Section 4.2. Celle spécifique au SmartCanton est quant à elle explorée à la Section 4.2.5.

2.10.3 Mise à disposition des données

L'accès au données générées par les capteurs s'effectue à l'aide d'une infrastructure FIWARE [1]. Cette plateforme a été développée dans le cadre du vaste projet Horizon 2020 [2], financé en partie par l'Union Européenne. L'environnement de FI-Ware est composé de plusieurs modules communicant entre eux à l'aide d'un protocole standardisé nommé NGSI. L'intégralité des composants de FI-Ware sont open-sources, prodiguant ainsi une liberté pour toute entité souhaitant implémenter une infrastructure complète. Le but de ce projet est de proposer une plateforme intégralement open-source pour les besoin des *smart technologies*.

2.10.4 Fédération d'opérateurs

Le POC SmartCanton utilise, grâce à un partenariat, une partie des infrastructures LoRa du réseau des Services Industriels de Genève [3] (SIG) et de l'entreprise Orbiwise [4]. Le but de cette collaboration est de couvrir au maximum le canton de Genève à l'aide des réseaux déjà existants. Le *roaming* des réseaux LoRaWAN ne sera couvert qu'avec l'arrivée de la nouvelle spécification LoRaWAN 1.1 (cf. Section 3.2.8 pour des informations supplémentaires sur le *roaming*). Celle-ci ne dispose, à l'heure actuelle, d'aucun périphérique ou réseau d'opérateurs compatible. Le concept de fédération des réseaux a donc du être mis en place dans le POC afin de récupérer des requêtes provenant de différents réseaux. La Figure 2.19 illustre les chemins d'accès disponibles depuis plusieurs opérateurs (Swisscom, SIG, Orbiwise et DGSI) qui, lorsqu'ils reçoivent des données d'un périphérique inconnu à leur propre réseau, peuvent rediriger les données vers la plateforme centralisée de la DGSI. Les divers composants illustrés sur ce diagramme (ex. *Network Server*, *Join Server*, etc.) sont explorés à la Section 3.2.

1. https://www.fiware.org/
2. https://ec.europa.eu/programmes/horizon2020/
3. https://www.sig-ge.ch/
4. https://www.orbiwise.com/home

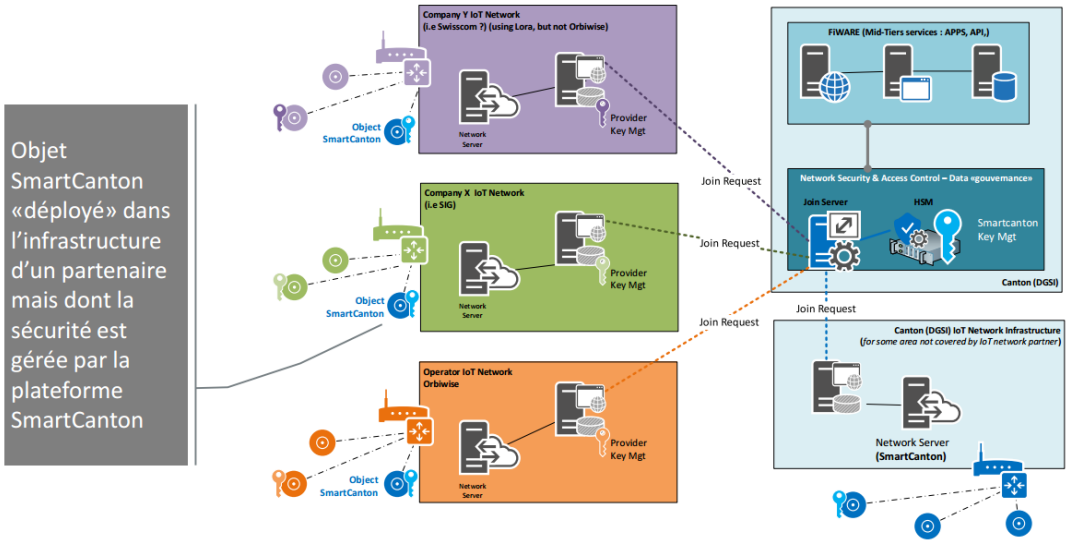


FIGURE 2.19 – Fédération des réseaux

Chapitre 3

Protocoles

Le but du présent chapitre est d'explorer plus les principaux protocoles utilisés dans le cadre de ce projet, c'est-à-dire le LoRaWAN ainsi que le Bluetooth Low Energy. Une brève introduction de ceux-ci est présentée au Chapitre 2.

3.1 Bluetooth Low Energy

Une brève introduction à la technologie Bluetooth est présentée dans la Section 2.6. La présente section a pour but d'expliquer plus en détail cette technologie. Celle-ci ayant été activement utilisée dans ce projet, il est important de comprendre ses bases.

Le Bluetooth opère dans la plage de fréquence ISM (*Industrial, Scientific et Medical*) de 2.4 à 2.4835 GHz. L'on peut visualiser les répartitions de cette bande à l'aide de la Figure 3.1 en examinant les 40 canaux du BLE. Parmi ces canaux, 37 sont réservés à la communication de données. Les trois autres sont réservés à l'émission d'*advertisements* (voir Section 3.1.3 pour plus d'informations) pour établir des connexions ou pour diffuser des informations. L'on remarque également que le WiFi (nommé 802.11 sur la Figure 3.1) se trouve sur la même fréquence. Les différents canaux WiFi (au nombre de 13 en Europe) se superposent sur cette bande, ce qui implique qu'il puissent perturber les communications sur cette bande de fréquence. Pour pallier à ces interférences, le Bluetooth effectue du *frequency-hopping spread spectrum*[1], ce qui signifie qu'il change régulièrement de canal dans un pattern pseudo aléatoire connu des deux dispositifs.

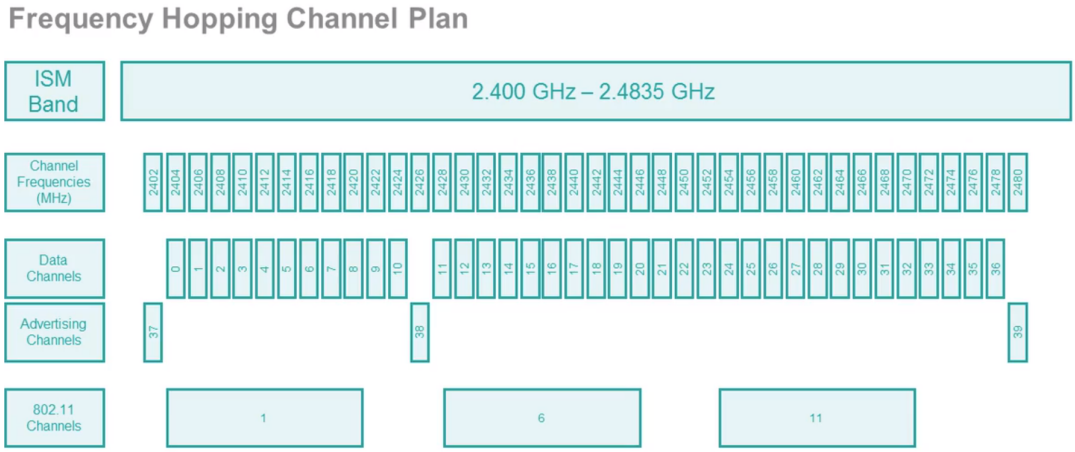


FIGURE 3.1 – Bande ISM utilisée par le Bluetooth Low Energy

1. https://en.wikipedia.org/wiki/Frequency-hopping_spread_spectrum

Le Bluetooth Low Energy utilise la modulation Gaussian Frequency Shift Keying [1] (GFSK). La différence entre le Bluetooth Low Energy et le Bluetooth Classic réside dans l'index de la modulation qui s'élève à 0.35 pour le *Classic* et 0.5 pour le Low Energy [34].

3.1.1 Interopérabilité entre les dispositifs

La Section 2.6 explore la différence entre le Bluetooth et le Bluetooth Low Energy, mais il faut savoir que certains problèmes d'incompatibilité persistent entre les deux technologies. En réalité, il existe deux mondes. Le premier est celui du Bluetooth *standard* ou *classic*, les dispositifs communiquant dans ce monde sont labellisés *Bluetooth*. Le deuxième monde est celui des dispositifs communiquant en Low Energy, ces dispositifs ont le label *Bluetooth Smart*. Ces derniers ne peuvent pas communiquer avec les premiers et vice-versa. La communication entre les deux mondes est réservée aux dispositifs de type *Bluetooth Smart Ready*. De moins en moins de dispositifs ne sont labellisés que Bluetooth, mais il existe encore de vieux appareils qui ne sont pas *Smart Ready*. De nos jours, tous les smartphones sont *Smart Ready*. La communication entre ces deux mondes est visible sur la Figure 3.2.

FIGURE 3.2 – Interopérabilité entre les dispositifs Bluetooth

3.1.2 Rôles

En Bluetooth Low Energy, quatre terminologies décrivent les objets souhaitant établir une connexion Bluetooth et échanger des informations une fois connectés. La Figure 3.3 illustre ces quatre terminologies. En premier lieu, il y a les objets en eux même, soit sur l'illustration un capteur de fréquence cardiaque. Ensuite, un smartphone ou PC qui souhaite récupérer les données ce cet objet. Un dispositif qui souhaite engager la connexion porte le nom de **centrale**. Le dispositif acceptant la connexion est quant à lui nommé **périphérique**. Une fois la connexion établie, il est possible d'échanger des données. Là, les noms ne sont plus les mêmes, car c'est maintenant le rôle des services Bluetooth de fournir les données (cf. Section 3.1.5.3.6). La centrale est nommée **client**, tandis que le périphérique est le **serveur**. Les dispositifs Bluetooth sont tous identifiés à l'aide d'une adresse MAC [2] unique sur 48 bits envoyée dans tous les paquets, y compris lorsqu'une connexion doit être établie.

Parfois les termes «central» et «périphérique» sont remplacés par les notions de **maitre** et **esclave**, car comme dans beaucoup de protocoles, le maitre initialise la connexion (ou communication) avec l'esclave.

1. https://en.wikipedia.org/wiki/Frequency-shift_keying#Gaussian_frequency-shift_keying
2. https://en.wikipedia.org/wiki/MAC_address

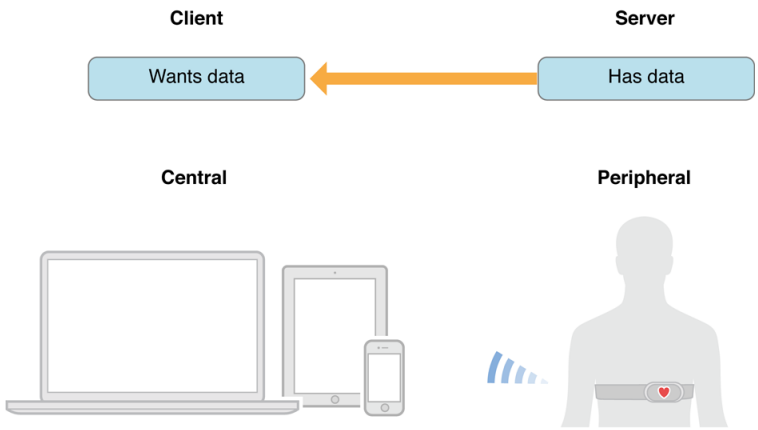


FIGURE 3.3 – Terminologies des connexions et des échanges de données en Bluetooth Low Energy

3.1.3 *Advertisements*

Un périphérique Bluetooth Low Energy doit s'annoncer lorsqu'il est prêt pour être connecté ou simplement quand il souhaite informer ses voisins de sa présence. Lors de la présentation des canaux Bluetooth (cf. Figure 3.1), on peut constater la présence de trois canaux réservés aux *advertisements*. Ceux-ci ont deux rôles. Le premier est d'annoncer à une centrale que le périphérique est disponible ou non pour une connexion BLE. Le second consiste à diffuser à toutes les centrales et périphériques aux alentours un paquet de données personnalisable par l'utilisateur. Cette dernière catégorie contient le concept de *beacons*, lequel est abordé dans la Section 3.1.6 de ce document. Le choix du canal d'*advertisements* est aléatoire (parmi les trois présentés) pour chaque nouveau paquet envoyé.

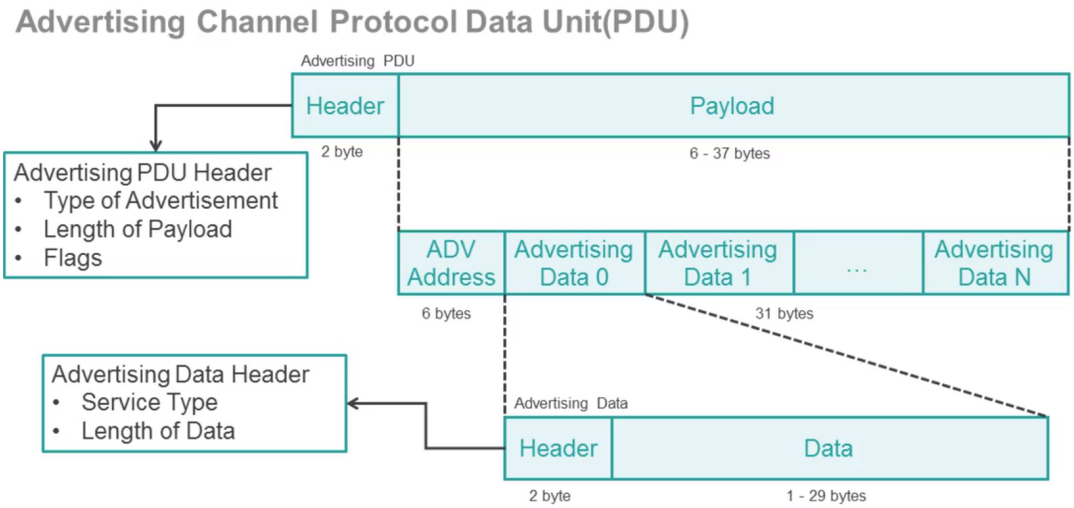


FIGURE 3.4 – Contenu d'un paquet de type *advertisement*

Le contenu d'un *advertisement* est illustré par la Figure 3.4. La taille complète d'un *advertisement* ne peut pas dépasser 39 bytes en Bluetooth 4.2. Les deux premiers bytes sont là pour renseigner les dispositifs qui scannent sur les différents paramètres et configurations offerts par le périphérique. Parmi ces informations, il est possible de savoir si le dispositif est connectable, si celui-ci utilise une adresse fixe ou variable, etc.

Il indique également s'il est possible de demander au périphérique des informations supplémentaires en effectuant un *Scan Request*, afin de récupérer un nouvel *advertisement* contenant un *Scan Response*. Pour information, le site Internet suivant offre une description très détaillée de chacun des champs et de la valeur qui peut lui être assignée :

`http://j2abro.blogspot.ch/2014/06/understanding-bluetooth-advertising.html`

A la suite de l'entête, se trouve l'adresse MAC du périphérique (fixe ou aléatoire). S'ensuivent différents paquets de données qui sont tous accompagnés d'une nouvelle entête. C'est avec ces derniers qu'il est possible d'implémenter des *beacons* (cf. Section 3.1.6).

L'intervalle entre les *advertisements* est configurable par le programmeur. Cet intervalle est normalisé et peut varier de 20 ms à 10.24 s avec des pas de 0.625 ms [35]. Plus cet intervalle est court, plus la centrale aura de chances de détecter le périphérique. Cependant, cette vitesse se fait au détriment de la consommation énergétique qui sera donc plus importante. L'intervalle inclut un facteur aléatoire dans le temps afin d'éviter les chances de collisions, par exemple, lorsque deux dispositifs avec le même intervalle commencent à émettre des *advertisements* en même temps.

3.1.4 Machine des états de la connexion

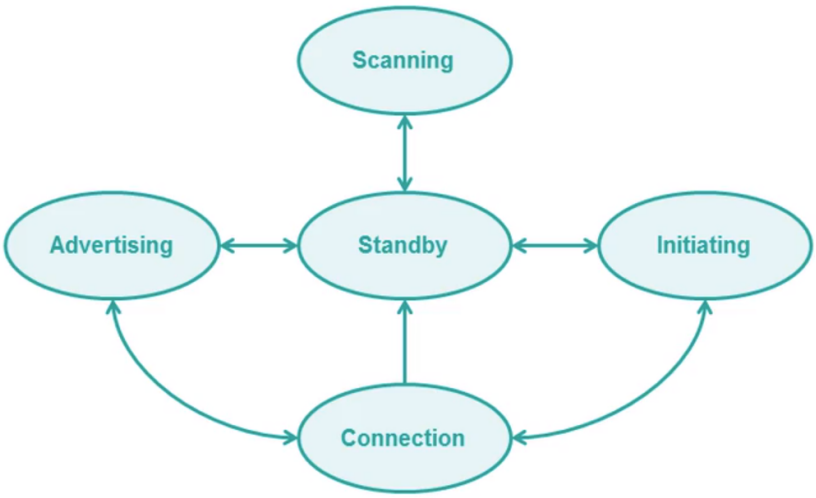


FIGURE 3.5 – Les 5 états possibles pour un dispositif Bluetooth Low Energy

Il existe 5 états possibles pour un appareil avec du Bluetooth Low Energy. Ces 5 états sont illustrés sur la Figure 3.5, de même que les liaisons possibles entre ceux-ci. L'explication de ces états est la suivante :

1. ***Standby*** : état par défaut. C'est dans ce mode que les dispositifs attendent qu'une nouvelle action leur soit demandée. C'est l'état où ils consomment également le moins d'énergie.
2. ***Advertising*** : état où le périphérique envoie des *advertisements*, afin de notifier s'il est connectable ou simplement lorsqu'un dispositif souhaite envoyer des données (par exemple des *beacons*).

3. *Scanning* : état où la centrale scanne ses environs afin de rechercher le dispositif auquel elle souhaite se connecter. Ce mode vise également la situation où la centrale est simplement intéressée par les paquets d'*advertisements* qui transitent, sans forcément vouloir établir une connexion.
4. *Initiating* : état où la centrale décide de se connecter avec un périphérique. Elle tente de l'atteindre afin d'effectuer une procédure de connexion.
5. *Connection* : état où les deux périphériques ont acceptés la connexion et peuvent maintenant commencer à s'échanger des données.

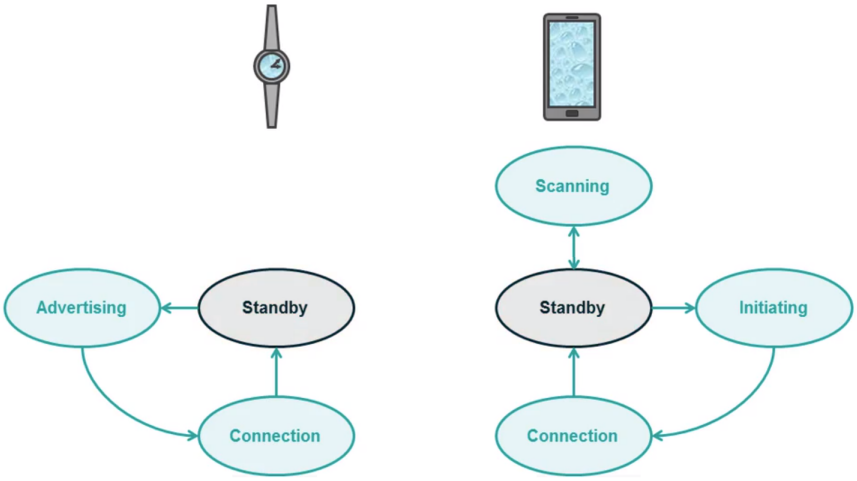


FIGURE 3.6 – Étapes lors de la connexion Bluetooth entre une centrale et un périphérique

Ces 5 états ne sont pas utilisés en parallèle. Un périphérique et une centrale n'en utilisent qu'un nombre limité. La Figure 3.6 illustre la connexion d'un smartphone en mode central à une montre connectée en tant que périphérique. Voici la séquence de connexion de ces deux dispositifs :

1. Prenons l'exemple où les deux dispositifs commencent en mode *Standby*.
2. Le périphérique se met en mode *Advertising* lorsqu'il est prêt à être connecté. En parallèle, le smartphone (centrale) se met en mode *Scanning* pour détecter la présence du périphérique périodiquement.
3. Lorsque le scanne est fini, la centrale repasse en *Standby*. Si dans ce mode celle-ci détecte la présence de son périphérique via les données reçues de l'état *Scanning*, celle-ci entre en mode *Initiating* afin d'initier une procédure de connexion avec le périphérique.
4. Si le périphérique accepte la connexion provenant de la centrale, les deux passent alors dans l'état *Connection*. Si la connexion échoue, les deux dispositifs repassent en *Standby*. Le périphérique peut se remettre à générer des *advertisements* s'il le souhaite.

3.1.5 Architecture d'une *stack* Bluetooth

L'architecture Bluetooth se subdivise en trois dénominations principales, illustrées à la Figure 3.7. Ces trois subdivisions se nomment `Controller`, `Host` et `Application`. Entre le `Controller` et l'`Host`, il existe un quatrième élément d'interfaçage nommé *Host Control Interface*. Le but de cette sous-section est d'explorer les différents composants de chacune de ces dénominations, afin d'avoir une vue plus complète du

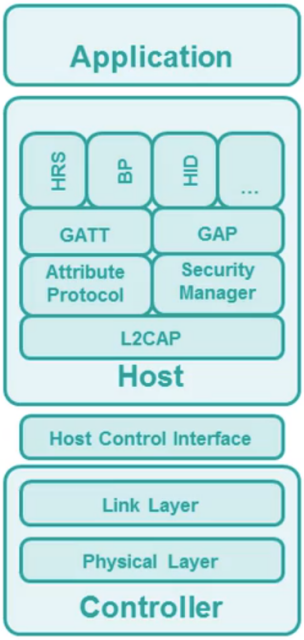


FIGURE 3.7 – Architecture complète de la stack BLE

protocole Bluetooth Low Energy. Une partie de ces explications proviennent de la vidéo nommée *Lesson 3 : Introduction to Bluetooth Low Energy*[1] par NXP.

Dans un OS, ces trois tâches consistent souvent en trois processus bien distincts qui communiquent entre elles avec des interfaces tenant compte des problèmes liés à la concurrence. Il est important de comprendre comment ces trois entités fonctionnent lorsque l'on programme sur un dispositif aussi bas niveau qu'un microcontrôleur.

3.1.5.1 *Controller*

Le contrôleur est la brique la plus basse du Bluetooth et a pour but de gérer l'interface RF avec les différentes spécificités liées à celle-ci. Elle est dépendante de l'implémentation matérielle du dispositif. Cette interface est différente pour chaque contrôleur utilisé, mais implémente des concepts similaires.

3.1.5.1.1 *Physical Layer*

La couche physique s'occupe de toute la gestion des différents canaux présentés précédemment sur la Figure 3.1. Cette couche effectue les décalages en fréquences (*Frequency Hopping Spread Spectrum*), mais ce n'est pas elle qui choisit les *timings* des sauts de fréquences. Son rôle est uniquement de générer la modulation demandée par la couche supérieure, avec les paramètres qui lui sont appliqués.

3.1.5.1.2 *Link Layer*

La couche `Link Layer` se charge de la gestion des connexions ainsi que de la qualité de celles-ci. Cette couche communique avec la `Physical Layer` lorsque le *Frequency Hopping Spread Spectrum* doit être adapté en raison d'une saturation du réseau. Cette couche gère également tous les aspects des connexions Bluetooth Low Energy.

1. https://www.nxp.com/video/lesson-3-introduction-to-bluetooth-low-energy:LESSON-3-KW41Z-WIRELESS-LAB

3.1.5.2 *Host Control Interface*

Le `Host Control Interface`, souvent abrégé par *HCI*, est une interface standardisée entre l'Host et le Controller. Cette interface implémente les mêmes concepts, quelleque soit la plateforme. L'avantage de cd type d'interface est de gagner du temps lorsque l'on change de contrôleur RF. Ainsi, seule la brique inférieure doit être modifiée sans influencer celles du dessus. L'interface HCI est également présente dans le Bluetooth *standard*. La base est similaire (entre les deux), mais les *advertisements* et les scans nécessitent de nouvelles commandes.

Dans certaines implémentations, le contrôleur est un périphérique externe au microcontrôleur et est connecté à celui-ci à l'aide d'une interface série (UART, SPI, etc.). Aussi, on retrouve le HCI dans ce type d'utilisations, bien que celle-ci soit du côté périphérique et convertisse ensuite les données en série en un format standardisé HCI.

3.1.5.3 *Host*

L'*Host* est l'élément le plus complexe du Bluetooth. Le but de cette section n'est pas d'explorer toutes les spécificités de cet élément, mais de comprendre ses rôles et ses composants principaux.

3.1.5.3.1 *L2CAP*

L2CAP signifie `Logical link control and adaptation protocol`. Il a les fonctions suivantes :

1. Transporter les données pour les couches de niveau supérieures, avec la gestion du *multiplexing* sur un même lien de données ;
2. Segmenter et réassembler les paquets de données ;
3. Concentrer les données provenant de plusieurs flux en une seule interface ;
4. Gérer la *Quality of Service* pour les couches supérieures. Il s'occupe de la gestion des erreurs et retransmet des données si celles-ci échouent.

3.1.5.3.2 *GAP*

GAP signifie `Generic Access Profile`. Il s'agit d'un service Bluetooth Low Energy qui vise à configurer l'accessibilité du périphérique. On trouve les informations suivantes dans ce service :

1. le périphérique peut-t-il être découvert et comment le découvrir ;
2. le périphérique est-il connectable et quelles sont les configurations nécessaires pour mettre en place cette connexion si elle est acceptée.
3. le périphérique est-il *bondable* (cet aspect sera exploré plus en détail avec la thématique de la sécurité Bluetooth Low Energy de la Section 4.1).
4. Une fois le *bond* effectué, le GAP s'occupe de la gestion des clés de sécurité et de l'autorisation des centrales qui peuvent se connecter au périphérique.

Le service GAP définit également les rôles implémentés par le dispositif. Les deux rôles "centrale" et "périphérique" ont été présentés précédemment, mais il existe deux *sous-rôles* :

1. `Broadcaster` : principe appliqué par les dispositifs de type *beacons* (cf. Section 3.1.6), c'est un périphérique qui émet des *advertisements,* mais qui n'est jamais connectable.
2. `Observer` : scanneur qui scrute en permanence le réseau Bluetooth Low Energy, mais qui n'initie pas de connexion avec un périphérique.

3.1.5.3.3 *Security Manager*

Le `Security Manager` porte, comme son nom l'indique, sur la gestion de tout l'aspect sécurité du périphérique. Il génère les clés lors des différentes connexions. Les clés temporaires sont stockées pour la session active, tandis que des *Long Term Key* (LTK) sont utilisées pour les périphériques *bonded* (cf. Section 4.1 pour plus de détails). Ce bloc s'occupe également de chiffrer les données lorsque le type de sécurité appliqué le demande.

3.1.5.3.4 *ATT*

ATT signifie `Attribute Protocol`. Ce bloc crée les deux interfaces Serveur et Client (cf. Section 3.1.2). Les éléments effectuant la mise à disposition de ces données sont appelés *Attributes* et présentent les caractéristiques suivantes :

— `Attribute Type` : Un attribut est défini par un identifiant unique. Celui-ci est un UUID [1] sur 128-bit. Cet identifiant peut être raccourci à 16 bits, lorsqu'il s'agit d'identifiants standardisés Bluetooth.
— `Attribute Handle` : définit où l'attribut est sauvegardé sur le serveur.
— `Attribute Value` : comme son nom l'indique, cette caractéristique définit la valeur de l'attribut. La taille maximale assignable à celui-ci est de 512 bytes.
— `Attribute Permissions` : Les permissions définissent les différentes opérations pouvant être effectuées sur l'attribut. Par exemple, *Read*, *Write* ou *Read/-Write* simultanément.

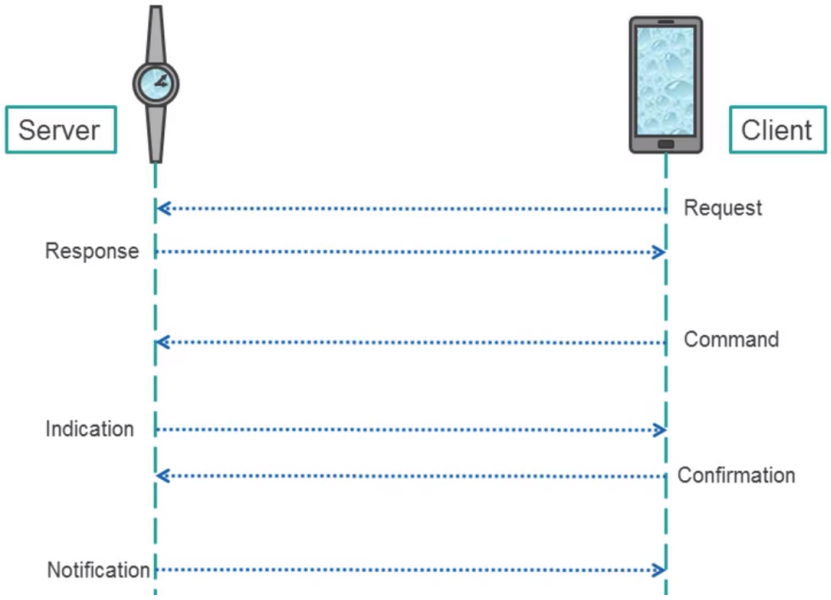


FIGURE 3.8 – Opérations possibles entre un serveur et un client Bluetooth Low Energy

1. https://en.wikipedia.org/wiki/Universally_unique_identifier

Une fois connecté, l'accès à un attribut entre un serveur et un client peut s'effectuer à l'aide de quatre opérations. Ces opérations, décrites sur la Figure 3.8, sont les suivantes :

1. `Request/Response` : Lorsque le client souhaite uniquement lire un attribut depuis le serveur, il effectue une opération de type *request*. Le serveur lui répond ensuite avec une *response*.
2. `Command` : Lorsque le client souhaite écrire un attribut, il effectue une opération de type *command*. Il existe une variante de la commande où il est possible de rajouter une confirmation lors de l'écriture.
3. `Indication` : Lorsque le serveur a une nouvelle donnée disponible sur l'attribut, il peut l'envoyer au client sous forme d'`indication`. Si le client récupère correctement cette donnée, un accusé de réception est envoyé avec une confirmation au serveur.
4. `Notification` : Lorsque le serveur a une nouvelle donnée disponible sur l'attribut, il peut l'envoyer au client sous forme de notification. Cependant, contrairement à l'indication, la réception de donnée par le client n'est pas confirmée. Cela permet ainsi d'économiser des échanges et d'augmenter le débit ou de réduire la consommation.

3.1.5.3.5 *GATT*

GATT signifie `Generic Attribute Profile`. C'est une couche au-dessus de l'ATT qui a pour but de hiérarchiser les différents groupes d'attributs en un concept nommé **`Services Bluetooth Low Energy`**.

Le GATT gère également la découverte de ces divers services en répondant aux requêtes de type `Discovery Services` d'un client. Le format pour la découverte est doté de diverses règles qui sont appliquées par le GATT.

3.1.5.3.6 *Profils et services*

Sur la Figure 3.7 on peut remarquer la présence de plusieurs blocs au-dessus du GATT. Il s'agit là de services GATT standardisés. Trois services sont listés :

— `HRS` : signifie *Heart Rate Service*
— `BP` : signifie *Blood Pressure Service*
— `HID` : signifie *Human Interface Device Service*, lequel offre ainsi la possibilité de connecter des périphériques à l'aide du Bluetooth Low Energy (ex. souris sans fil)

Ces services sont intégrés dans des **profils** standardisés par le protocole Bluetooth :

`https://www.bluetooth.com/specifications/gatt`

Le but de ces profils standardisés est d'augmenter l'interopérabilité des différents appareils avec les interfaces de données standards. Par exemple, dans le cas d'un HRS, l'interface de capture du système cardiaque est standard, ce qui permet à n'importe quel développeur de faire une application générique pour tous les capteurs cardiaques du marché.

Si un développeur souhaite créer son propre profil, il doit se plier à certaines restrictions quelques restrictions. Les UUIDs des attributs ne peuvent pas utiliser l'UUID attribué aux services Bluetooth standardisés, à savoir la valeur `0000XXXX-`

`0000-1000-8000-00805f9b34fb`. Les `XXXX` représentent le UUID 16 bits pouvant être utilisés au lieu des 128 bits, lorsque les services standards sont accédés. La liste complète de ces services, ainsi que leur UUID 16bit, est disponible à l'adresse suivante :

https://www.bluetooth.com/specifications/gatt/services

Table 3.1 – Paramètres d'un *Client Characteristic Configuration Descriptor*

Bit Field Client Characteristic Configuration (CCCD)			
Bit	**Size**	**Definition**	
		Key	**Value**
0	1	0	Notifications disabled
		1	Notifications enabled
1	1	0	Indications disabled
		1	Indications enabled
2 to 15	14	0	Reserved for future use
		1	Reserved for future use

La couche ATT indique que tout élément avec un UUID est un attribut. Au niveau GATT, on utilise toutefois des définitions différentes pour chaque élément. Un **service** ne produit pas de données, son unique rôle étant de définir une encapsulation pour des **caractéristiques**. Ce sont les caractéristiques qui contiennent toutes les données des capteurs. Une caractéristique peut ensuite être accompagnée de plusieurs **descripteurs**[1]. Les descripteurs ont pour rôle de configurer ou simplement d'informer une caractéristique. Le descripteur le plus connu et utilisé est le `Client Characteristic Configuration Descriptor` (CCCD). Si une caractéristique est autorisée à envoyer ses données en tant que notifications ou indications, l'utilisateur doit configurer ce descripteur pour activer ces fonctionnalités. Les paramètres possibles de ce descripteur sont visibles sur la Tableau 3.1.

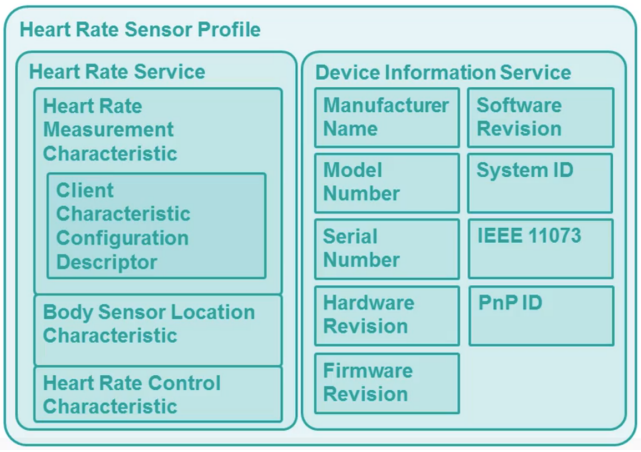


Figure 3.9 – Profil *Heart Rate Sensor* standardisé

Lorsqu'un développeur crée un profil, il doit parfaitement comprendre ses couches intérieures afin que son implémentation soit correcte. Si on reprend l'exemple du *HRS*, dont les éléments sont visibles sur la Figure 3.9, on y voit le contenu complet qui doit être implémenté pour qu'un dispositif soit compatible avec ce profil. On a tout

1. https://www.bluetooth.com/specifications/gatt/descriptors

d'abord deux services, le principal étant celui de la mesure cardiaque. Le deuxième est le `Device Information Service`, qui comme son nom l'indique, fournit des informations sur le périphérique. Il y a ensuite la présence de trois caractéristiques : la `Heart Rate Control` permet de configurer le temps entre chaque nouvelle acquisition de valeur ; La `Body Sensor Location`, qui informe sur l'emplacement du corps où le périphérique est situé ; la `Heart Rase Measurement`, caractéristique la plus importante qui fournit la fréquence capturée par le capteur cardiaque. Cette caractéristique est disponible en tant que notification ou indication, puisqu'elle est également constituée d'un CCCD. Si deux profils demandent l'implémentation du même serveur, par exemple le `Device Information Service`, celui-ci n'a besoin d'être implémenté qu'une seule fois.

3.1.5.4 *Application*

L'application est là où toutes les données des services sont acquisitionnées. Les communications avec les capteurs sont effectuées à ce niveau, puis une fois les données récupérées, elles sont inscrites dans les attributs ATT correspondants, à travers la base de données GATT.

3.1.6 Beacons

En principe, il est possible de savoir combien de dispositifs sont présents simplement en effectuant un scan. Si l'utilisateur a son Bluetooth activé, le système va périodiquement scanner ses environs et communiquer son adresse MAC lors des réponses aux *Scan Requests*. Cependant, depuis la norme 4.0 du Bluetooth Low Energy (améliorés avec la norme 4.2[36]), les périphériques doivent supporter le principe de Low Energy (LE) Privacy[36]. Il ne doit pas forcément être activé par défaut, mais cette option doit être implémentée sur tous les contrôleurs Bluetooth 4.2. Ce mode a pour but de modifier périodiquement l'adresse MAC du périphérique lorsque celui-ci envoi des *advertisements*. Cela a pour effet d'empêcher la traque de personnes à leur insu, au moyen d'une simple borne qui scan tous les périphériques à proximité. Le microcontrôleur KW41Z supporte ce mode, comme tous les smartphones iOS ou Android. Le but étant de scanner le nombre de personnes autour d'une borne, il faut pouvoir détecter ces personnes par un moyen sûr. Les systèmes d'exploitation des smartphones (iOS et Android) ont activé par défaut le mode privacy sans possibilité de le désactiver pour les programmeurs. La fréquence du changement d'adresse MAC enduite par le LE Privacy est de l'ordre de 3min sur un smartphone, mais celle-ci varie selon les modèles de *smartphones* et n'est pas configurable par le développeur.

Pour pallier à cette limitation, le principe de *beacons* s'est très vite répandu. Il s'agit alors de modifier le contenu d'un paquet *advertisement* afin d'y placer un *payload* personnalisable. La transmission d'un *beacon* est unidirectionnelle. Le périphérique qui scanne les *beacons* ne peut pas directement interagir avec eux. Normalement, le scanner a uniquement besoin de connaitre les périphériques qui l'entoure, ainsi que la distance qui les sépare (calculable avec la puissance du signal RSSI). Les *beacons* ont tous une taille de paquet maximale de 31 bytes (cf. Section 3.1.3), dont trois bytes réservés au champ *Advertisement Flag* (Adv Flag) obligatoire.

Il existe trois spécifications de *beacons* principalement utilisées [37] :

1. iBeacons : spécification de protocole développé par Apple [1]. Le contenu du paquet BLE est affiché sur la Figure 3.10. Le paquet contient un entête constitué

1. https://developer.apple.com/ibeacon/

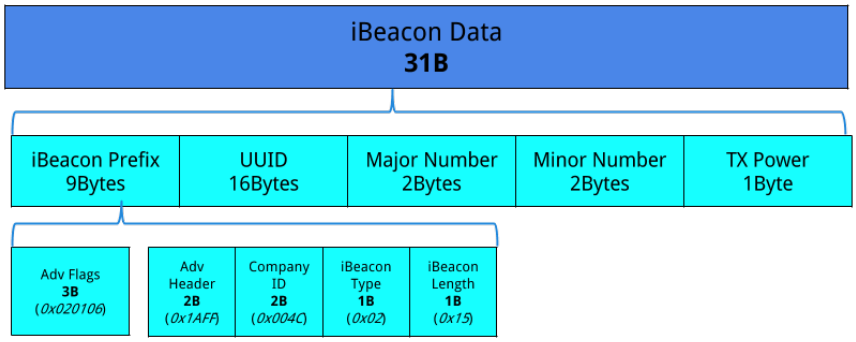


FIGURE 3.10 – Spécification des paquets iBeacons

d'un *Company ID*, lequel doit être délivré par Apple pour chaque fabricant de *beacons* afin que celui-ci soit conforme à la spécification. On a ensuite un UUID, unique pour chaque *beacon*, *Major* qui identifie un sous-réseau de plusieurs *beacons* et un *Minor* qui identifie un *beacon* précis. L'utilisateur peut ensuite utiliser ces trois identifiants pour récupérer les informations sur le *beacon* dans une base de données. En fin du paquet, il y a la puissance d'émission en dBm à 1m afin d'estimer la distance en fonction du RSSI mesuré par le scanneur. Le paquet fait au total 31 bytes, mais uniquement 20 bytes (UUID, *Major* et *Minor*) sont configurables par l'utilisateur, le reste étant fixé par Apple ou le fabricant certifié du *beacon*.

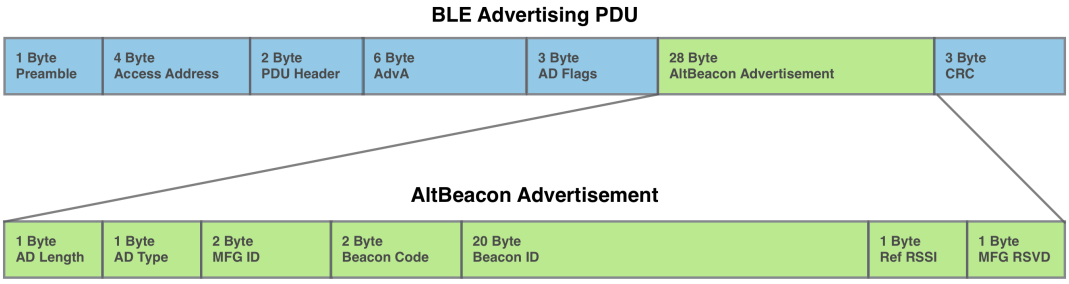


FIGURE 3.11 – Spécification des paquets AltBeacon

2. AltBeacon : spécification de protocole développé par Radius Networks [1]. Le contenu du paquet BLE est affiché sur la Figure 3.11. Le contenu du champ Beacon ID est identique à l'identifiant utilisé par le iBeacon, c'est-à-dire l'utilisation d'un UUID sur 16 bytes couplé à un *Major* et *Minor* sur deux bytes chacun. L'AltBeacon a 25 bytes modifiables par l'utilisateur, car les champs *Manufacturer ID* (MFG ID), *Beacon Code* et *Manufacturer Reserved Data* (MFG RSVD) sont tous configurables par l'utilisateur.
3. Eddystone : spécification de protocole développé par Google [2] et initialement appelée URIBeacon. Le contenu du paquet BLE est affiché sur la Figure 3.12. Cette spécification a un but différent des deux autres précédemment décrites. Celle-ci fait partie de l'initiative Physical Web [3], qui a pour objectif de faciliter l'interaction avec les objets qui entourent les utilisateurs sans nécessiter une connexion Bluetooth. Il existe trois types de paquets Eddystone :

1. http://altbeacon.org/
2. https://github.com/google/eddystone
3. http://google.github.io/physical-web/

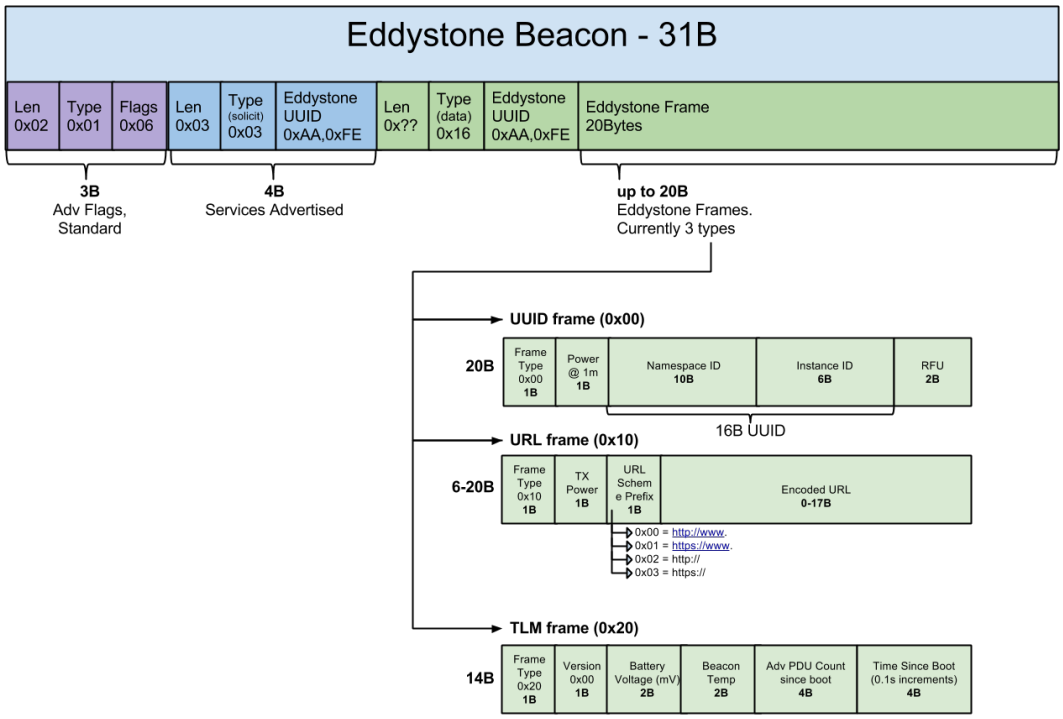


FIGURE 3.12 – Spécification des paquets Eddystone

(a) `UUID` : similaire aux deux technologies précédentes, son but est de diffuser une UUID de 16 bytes découpés en deux champs de 10 et 6 bytes.

(b) `URL` : Le *beacon* peut diffuser une URL qui peut être lue par un client avec un accès internet. Il peut par exemple diffuser l'URL `https://goo.gl/Aq18zF` et chaque client recevant ce paquet pourra visiter le site sur son téléphone. La navigateur Google Chrome sur Android offre ainsi la possibilité de scanner ses environs à la recherche de périphériques proposant des URL Eddystones.

(c) `TLM` : Offre la possibilité de diffuser des informations télémétriques sur le *beacon*. On peut ainsi connaitre la batterie du périphérique, la température ainsi que le nombre d'*advertisements* envoyés depuis qu'il a démarré.

La spécification UUID est la plus intéressante, mais elle reste plus limitée que le iBeacon ou l'AltBeacon. En effet, elle n'offre que 16 bytes modifiables.

Le but principal de ces *beacons* est d'informer l'utilisateur sur les événements survenant autour de lui, mais ceux-ci peuvent être détournés pour localiser les gens. En effet, si on place des *beacons* fixes, on peut ensuite remonter via Internet la liste des *beacons* à proximité et leurs puissances. Si un grand nombre de *beacons* sont à proximité, il est donc possible de traquer une personne à quelques mètres près. Pour ce faire, une application smartphone qui scan les *beacons* et qui remonte les informations à un serveur est nécessaire. Par contre, conscient de cette invasion de la privacité, les développeurs doivent maintenant demander les droits de localisation sur l'application (même principe pour iOS et Android) afin d'effectuer des scans lorsque le téléphone est en veille.

3.1.7 Consommation énergétique

Le Bluetooth Special Interest Group (SIG) affirme que le BLE consomme entre 1 et 50 %, de la consommation du Bluetooth Classic [38]. Ce gap oscille selon l'application souhaitée. La Figure 3.13 illustre la consommation des périphériques en mode *advertising* [39]. On constate que les facteurs qui influencent sur la consommation sont : la taille des paquets, l'intervalle entre chaque paquet et la puissance d'émission. Lorsque l'utilisateur le développeur souhaite optimiser la consommation il doit tenir compte de ses trois paramètres. Selon les systèmes utilisés, ces paramètres ne sont pas flexibles. Par exemple, sur Android, le développeur peut uniquement mettre trois modes de fréquences pour le paquet Bluetooth sans jamais pouvoir spécifier un intervalle temporel. Le temps ensuite appliqué varie selon le modèle du périphérique.

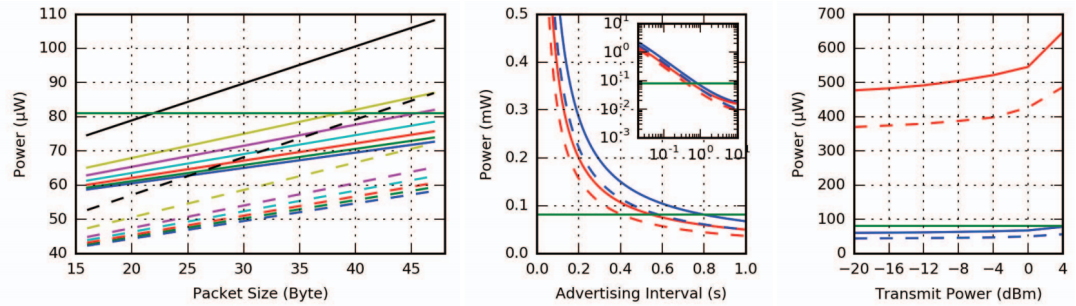


FIGURE 3.13 – Consommation des *advertisements* en Bluetooth Low Energy

3.2 LoRaWAN

Le protocole LoRaWAN a brièvement été introduit dans l'état de l'art de ce document en Section 2.3, mais il est particulièrement important de comprendre le fonctionnement de ce protocole afin d'en tirer la meilleure implémentation possible. LoRaWAN est développé et maintenu par la LoRa Alliance [1] dont les principaux membres du comité sont des grands noms du domaine [2]. On retrouve bien sûr Semtech qui fournit la technologie, mais également des grands noms de la télécommunication, tels que Orange ou Bouygues Telecom.

Le protocole LoRaWAN est un protocole définissant la sous-couche *Medium Access Control* (MAC), la couche liaison du modèle OSI [3], ainsi que la couche 3 (réseau). Le protocole LoRaWAN peut être implémenté sur une modulation LoRa ou FSK [40].

L'architecture complète du protocole LoRaWAN est visible en Figure 3.14 à sa base se trouve la couche LoRa comportant différentes fréquences en fonction de la région. Vient ensuite le concept de classe pour les périphériques. Ces classes définissent le comportement des périphériques vis-à-vis du réseau et seront développées plus en détail en Section 3.2.1. Vient ensuite toute la partie LoRa MAC avec l'implémentation du protocole LoRaWAN. Toute la formation du paquet avec ses entêtes et ses données se trouve dans ce dernier élément.

1. https://www.lora-alliance.org/
2. https://www.lora-alliance.org/board-officers
3. https://en.wikipedia.org/wiki/OSI_model

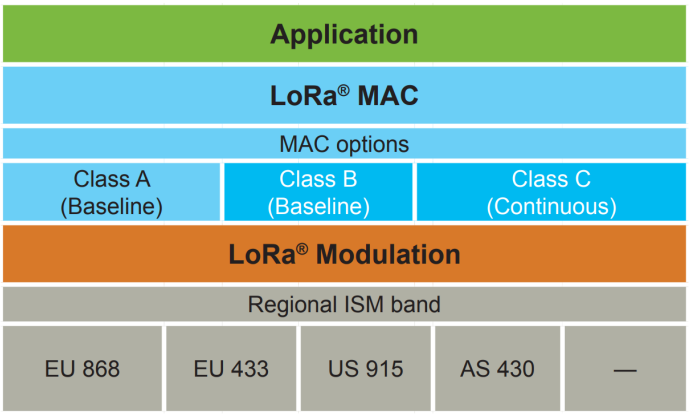


FIGURE 3.14 – Architecture du protocole LoRaWAN

3.2.1 Classes des périphériques

LoRaWAN est un protocole bidirectionnel, assorti de quelques conditions. Il existe trois classes de périphériques. LoRaWAN définit ces trois classes [41] comme ceci :

1. Classe A (*baseline*) : lorsqu'un périphérique envoi un message *uplink*, il se met en écoute durant un laps de temps défini sur deux fenêtres de réception. La séquence de réception pour cette classe est visible sur la Figure 3.15. C'est la classe qui consomme le moins d'énergie, car l'écoute ne se produit qu'après l'envoi d'un paquet. Quand le périphérique n'est pas ni en émission, ni en écoute de réponse, il peut être placé en veille profonde pour économiser un maximum d'énergie.

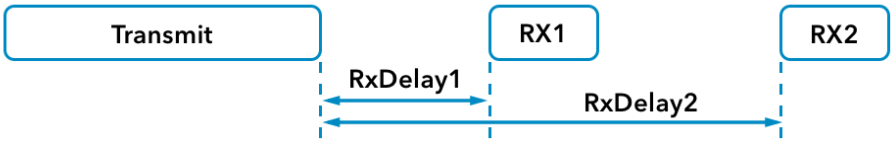


FIGURE 3.15 – Réception en tant que périphérique de type classe A

2. Classe B (*beacon*) : Cette classe est identique à la classe A, mais offre la possibilité supplémentaire de programmer des fenêtres d'écoutes régulières que le périphérique doit écouter. La synchronisation temporelle de ces fenêtres s'effectue à l'aide de *beacons* envoyés par la *gateway* [40]. L'implémentation de cette classe a été finalisée dans la spécification 1.1 de LoRaWAN (cf. Section 3.2.8). La Figure 3.16 illustre cette fenêtre de synchronisation.

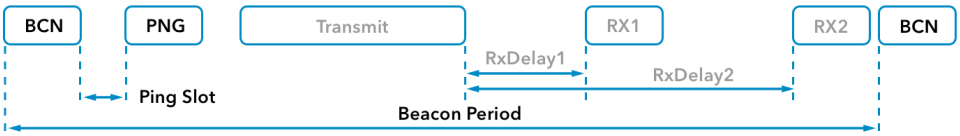


FIGURE 3.16 – Réception en tant que périphérique de type classe B

3. Classe C (*continuous*) : Cette classe est en écoute permanente sur une des fenêtres de réception, excepté lorsqu'un message est envoyé. On peut voir l'effet sur la Figure 3.17. Cela a pour avantage de réduire la latence des communications *downlink*, le périphérique n'ayant plus besoin d'un *uplink* pour recevoir le

message. En contrepartie, le périphérique consomme en permanence de l'énergie pour recevoir ces données. Cette option est uniquement envisageable si l'autonomie du périphérique ne constitue pas un problème.

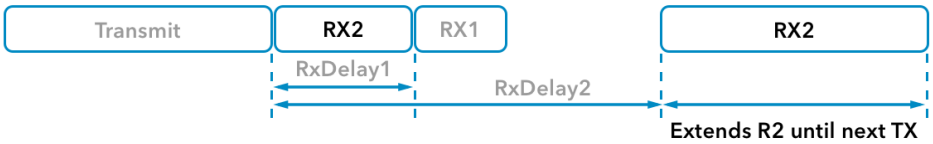


FIGURE 3.17 – Réception en tant que périphérique de type classe C

3.2.2 Architecture d'un réseau LoRaWAN

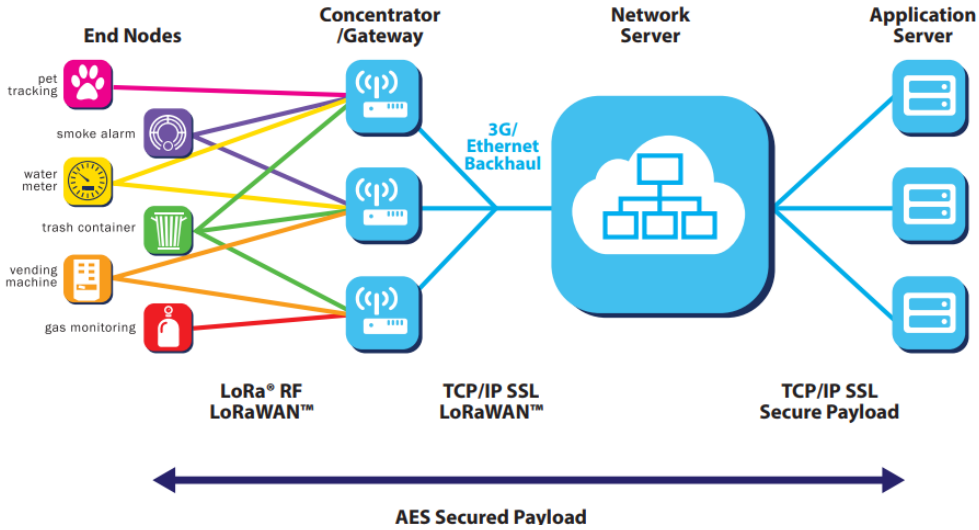


FIGURE 3.18 – Architecture d'un réseau LoRaWAN

La Figure 3.18 illustre l'architecture d'un réseau LoRaWAN avec ses divers composants. Les différents éléments de ce diagramme seront examinés plus en détail dans les sous-sections suivantes.

3.2.2.1 *End nodes*

Les *end nodes* sont les périphériques qui sont connectés au réseau. Ceux-ci sont tous équipés d'un module LoRa de la catégorie *transceivers.* [1]. La carte développée dans le cadre de ce projet entre dans cette catégorie. Un *end node* LoRaWAN est souvent pourvu d'un dispositif basse consommation qui effectue une acquisition et envoie périodiquement des données à une application à travers le réseau.

3.2.2.2 *Gateways*

Les concentrateurs sont le pont entre un réseau LoRa et un réseau TCP/IP. Ceux-ci sont tous équipés d'un ou plusieurs modules LoRa de type concentrateur [2]. Les *gateways* peuvent être achetées directement chez certains fabricants, mais il est également

1. https://www.semtech.com/products/wireless-rf/lora-transceivers
2. https://www.semtech.com/products/wireless-rf/lora-gateways

possible de créer sa propre *gateway* à l'aide d'un Raspberry Pi et d'un concentrateur LoRa connecté via une communication série [1]. L'une des *gateway* les plus utilisées est la Multitech Conduit, visible sur la Figure 3.19.

Le rôle de la *gateway* consiste à démoduler les signaux reçus dans un *buffer* afin de repérer la présence de trames LoRa. Tous les paquets sont transférés au *Network Server*, quand bien même ils ne sont pas du réseau LoRaWAN auquel appartient la *gateway*. Des métadonnées sont également ajoutées au paquet, tel que le *Receive Signal Strength Indicator* (RSSI), *Signal To Noise Ratio* (SNR), le canal de fréquence utilisé, le *data rate*, ainsi que le *Time of Arrival* (ToA). Les *gateways* n'ont aucune information concernant les périphériques connectés au réseau. Elles ne font que transférer les paquets au *Network Server*.

FIGURE 3.19 – Concentrateur LoRaWAN Multitech Conduit

Les fournisseurs de réseaux doivent offrir un *firmware* compatible avec les *gateways*. Les *firmwares* sont tous basés sur un code fournit par Semtech, nommé `Lora network packet forwarder` [2]. Comme son nom l'indique, ce code a pour but de transférer les données LoRa reçues vers un serveur avec un protocole d'échange spécifique. Le serveur répond ensuite à la *gateway* en fonction de l'action qui doit être effectuée sur le périphérique, par exemple, lorsqu'un objet est autorisé à rejoindre le réseau après une procédure de `Join`.

3.2.2.3 *Network Server*

Le *Network Serveur* est le gestionnaire du réseau. Il s'occupe de tous les paquets reçus sur le port 0. Les données reçues sur les autres ports sont transférées à la couche applicative gérée par l'*Application Server*. Les différentes tâches incombant au *Network Server* sont les suivantes [42] :

- Identification du périphérique qui souhaite communiquer avec le réseau. Les *gateways* redirigent tous les paquets, ceux-ci doivent ensuite être triés [42] ;
- Contrôler le *data rate* si le mode ADR (Adaptative Data Rate) est actif sur le périphérique ;
- Configurer les paramètres radio ainsi que les différents timings des fenêtres de réceptions [42] ;
- Gestion des commandes de type MAC [3] [40] ;
- Prévenir les attaques de type *replay* en analysant les *frame counters* des paquets ;
- Gestion d'une queue de messages *downlink* qui seront envoyés aux périphériques lors de la réception du prochain message *uplink* ;
- Lors de l'envoi d'un message *downlink*, le *Network Server* s'occupe de la sélection de la *gateway* qui doit envoyer le message. Celle-ci est sélectionnée en

1. https://github.com/ttn-zh/ic880a-gateway/wiki
2. https://github.com/Lora-net/packet_forwarder
3. http://www.rfwireless-world.com/Tutorials/LoRaWAN-MAC-layer-inside.html

fonction du RSSI et du SNR vis-à-vis du périphérique sur le message *uplink* reçu [42] ;
— Il s'occupe de la gestion du *duty cycle* des *gateways* afin de respecter les normes ETSI [42] ;
— Évite les conflits lors de l'envoi de messages *downlink* sur une *gateway*. Une *gateway* ne peut envoyer qu'un seul message à la fois [42].

Cette partie de l'architecture est traitée plus en détail en Section 2.10, via la présentation du projet SmartCanton.

3.2.2.4 Join Server

Le *Join Server* n'est pas illustré sur la Figure 3.18, bien qu'il soit une brique indispensable lors de l'utilisation du mode OTAA. Lors de la réception d'un message de type *Join Request*, le *Network Server* le transfère au *Join Server*.

Il existe deux modes en LoRaWAN pour rejoindre un réseau. Le premier mode se nomme *Over-the-Air Activation* (OTAA). Dans le mode OTAA, seules l'`App Key` et l'adresse `Dev EUI` (comparable à l'adresse MAC d'un périphérique) doivent être connues sur le périphérique. Le `Join Server` vérifie si le périphérique a été préalablement enregistré dans une base de données contenant une `App Key`, un `App EUI` (optionnel selon les cas), ainsi qu'un `Dev EUI`. Si tel est le cas, un `Join Accept` est retourné au périphérique qu'il a été accepté sur le réseau. Avec ce *Join Accept*, il reçoit son `Dev Address` qui lui permet d'émettre sur le réseau avec cette adresse, ainsi les éléments nécessaires à la `Network Session Key` et `App Session Key` pour la communication. La procédure du `join` du mode OTAA est observée en détail en Section 4.2 de ce document.

L'autre mode est *Activation by Personalization* (ABP). Dans ce mode, toutes les informations nécessaires doivent être prodiguées manuellement au périphérique. Il faut donc spécifier le *Device EUI* (normalement fixé par le fabricant), l'*Application EUI*, le *Device Address*, la *Network Session Key*, ainsi que l'*App Session Key*.

Le mode OTAA est préféré au mode ABP, non seulement parce qu'il simplifie l'ajout d'un périphérique au réseau, mais également parce qu'il ajoute un aspect de sécurité (cf. Section 4.2).

3.2.2.5 Applications Server

Le serveur d'application reçoit tous les paquets qui n'ont pas été envoyés sur le port 0. Toutes les *payload* sont déchiffrées par ce serveur. Les données sont ensuite mises à disposition de l'utilisateur. Certains fournisseurs de réseau proposent des interfaces Web complètes pour visualiser les données décodées avec la *App Session Key*. Ils offrent également la possibilité de rediriger ces données vers des fournisseurs d'application, par exemple Cayenne de MyDevices, qui accélèrent les prototypages rapides d'applications. Certains fournisseurs sont plus minimalistes et n'offrent qu'une interface de récolte de données à l'aide d'API diverses et variées.

L'application finale est quant à elle entièrement libre pour l'utilisateur, la seule contrainte étant l'interface de communication proposée par le fournisseur de réseau. Chaque fournisseur peut proposer l'interface qu'il désire. Par exemple, TTN propose une interface basée sur un flux MQTT [1] et développe actuellement une interface AMQP [2].

1. http://mqtt.org/
2. https://www.amqp.org/

3.2.3 *Spreading Factor* et *Data Rate*

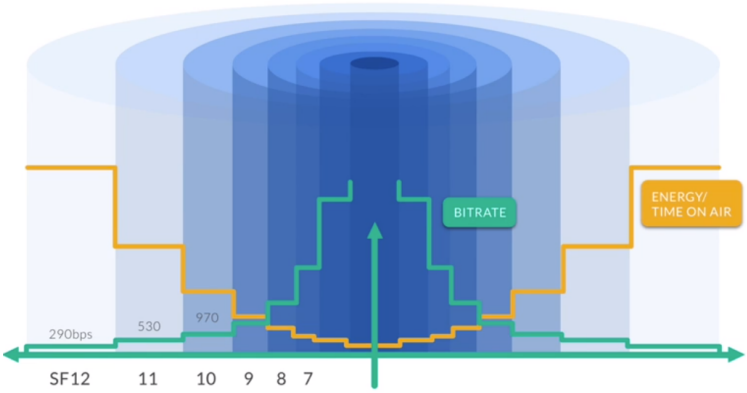


FIGURE 3.20 – Comparatif du temps d'émission d'une trame LoRaWAN en fonction de ses paramètres

Le *Spreading Factor* a d'ores et déjà été introduit en Section 2.3.3 lors de la présentation de la modulation LoRa. LoRaWAN utilise non seulement les SF pour l'émission, mais il ajoute également le concept de *Data Rate* (DR). Le Tableau 3.2 présente tous les *data rate* qui sont autorisés en Europe [15]. Chaque région possède des *data rate* différents en fonction des législations. Chaque DR a une configuration différente en se basant sur un SF et sur la bande passante assignée. On constate en observant le Tableau 3.2 que plus le DR est faible, plus le débit est faible. Avec un SF plus élevé, le temps d'émission est supérieur, ce qui a pour conséquence d'augmenter la portée en accentuant la sensibilité du récepteur. La taille du *payload* est également affectée par le temps d'émission,et ce, afin d'éviter qu'on ne sature trop le réseau lors de l'émission avec un SF élevé. Une illustration plus visuelle de l'effet du DR est visible sur la Figure 3.20.

TABLE 3.2 – Différents *data rate* LoRaWAN avec leurs débits pour l'Europe

Data Rate	**Configuration**	***bits/s***	***Max payload***
DR0	SF12/125kHz	250	59
DR1	SF11/125kHz	440	59
DR2	SF10/125kHz	980	59
DR3	SF9/125kHz	1 760	123
DR4	SF8/125kHz	3 125	230
DR5	SF7/125kHz	5 470	230
DR6	SF7/250kHz	11 000	230
DR7	FSK : 50kbps	50 000	230

Pour éviter de fixer une valeur qui ne sera pas efficace dans toutes les situations, il est possible d'utiliser l'*Adaptive Data Rate* (ADR) [40]. L'ADR est un mécanisme qui a pour but d'optimiser les débits de données, le temps d'émission et, par conséquent, la consommation énergétique. L'ADR ne doit être utilisé que par des nœuds statiques. Ce sont les nœuds qui décident si cette option est activée ou non. Lorsque le réseau est informé de l'utilisation de l'ADR, celui-ci sauvegarde les 20 transmissions les plus récentes avec le *frame counter*, *signal-to-noise ratio* (SNR) et le nombre de *gateways* qui ont reçu le paquet.

L'algorithme utilisé est décrit dans une *application note* de Semtech, nommée *Simple rate adaptation recommended algorithm*[1].

3.2.4 Paquets LoRaWAN

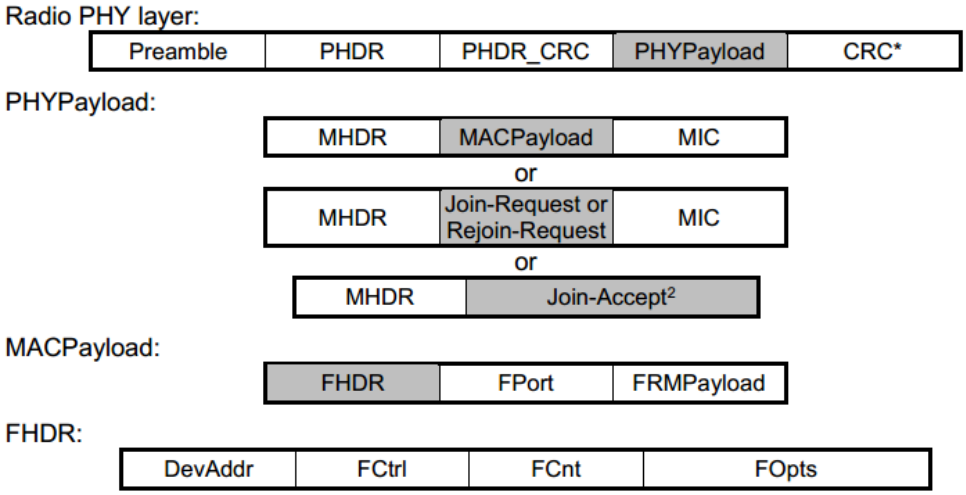


FIGURE 3.21 – Format des messages du protocole LoRaWAN

Les paquets LoRaWAN se basent sur les paquets LoRa pour la transmission. Le squelette d'un paquet LoRa a d'ores et déjà été présenté en Section 2.3. Tous les paquets de données disposent d'une `PHY` ***`payload`*** commençant par un octet de *MAC header* (`MHDR`), suivi d'un `MAC` ***`payload`*** (`MACPayload`). Le champ `PHYPayload` contient toute l'implémentation du protocole LoRaWAN. L'encapsulation des paquets au fil des différentes couches est visible sur la Figure 3.21.

TABLE 3.3 – Différents champs contenus dans un MHDR

Bit#	7..5	4..2	1..0
MHDR bits	MType	RFU	Major

TABLE 3.4 – Différents types de messages MAC possibles

MType	***Description***
000	Join-request
001	Join-accept
010	Unconfirmed Data Up
011	Unconfirmed Data Down
100	Confirmed Data Up
101	Confirmed Data Down
110	Rejoin-request
111	Proprietary

Les payload peuvent contenir différents types de paquets, ceux-ci étant définis dans l'entête MHDR dont les champs sont visibles sur la Tableau 3.3. Le type de message est codé sur 3 bits dans un champ nommé `MType`. LoRaWAN implémente 8

1. https://github.com/TheThingsNetwork/ttn/issues/265#issuecomment-255092765

types de paquets visibles sur le Tableau 3.4. Les messages envoyés par les périphériques sont de types `Data UP`. Ceux-ci peuvent être confirmés ou non selon le type d'utilisation désirée. Le champ RFU signifie *Reserved for Future Use* et le champ Major est utilisé pour spécifier la version de la spécification LoRaWAN utilisée (1.0 ou 1.1 actuellement).

La description complète des paquets est disponible dans le chapitre `MAC Message Formats` du document `LoRaWAN 1.1 Specification`[1] de la LoRa Alliance.

3.2.5 Clés de chiffrement

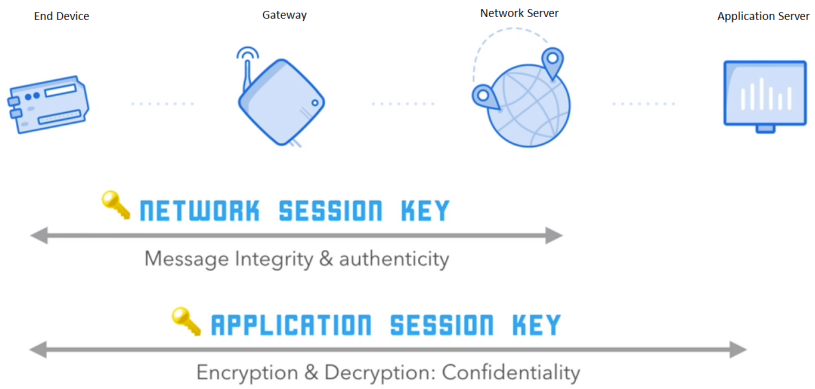


FIGURE 3.22 – Rôles de la *Network Session Key* et *Application Session Key*

En OTAA, lorsqu'un périphérique souhaite rejoindre un réseau il ne dispose que d'une seule et unique clé, nommée AppKey. Une fois le réseau LoRaWAN rejoint, deux autres clés sont créées. L'aspect sécurité de ces clés, ainsi que la génération de celles-ci est vue plus en détail en Section 4.2 de ce document. Pour l'heure, voici une liste des trois clés indispensables et leurs rôles respectifs :

1. `App Key` : doit être connue par le *Join Server* et par le périphérique. C'est un secret partagé qui doit être bien gardé, puisque toutes les clés sont générées grâce à lui.
2. `Network Session Key` : clé utilisée pour chiffrer les communications entre le périphérique et le *Network Server*. Cette clé est générée à l'aide de l'App Key.
3. `Application Session Key` : clé utilisée pour chiffrer les communications entre le périhpréqiue et l'*Application Server*. Elle est également générée à l'aide de l'App Key.

Sur la Figure 3.22, on peut visualiser l'utilisation des clés en fonction des types de messages envoyés.

3.2.6 Canaux d'émission

La spécification LoRaWAN définit 3 canaux communs de 125 kHz pour la bande des 868 MHz (868.10, 868.30 et 868.50 MHz)[40]. Ces canaux doivent être obligatoirement supportés par tous les réseaux LoRaWAN et par tous les périphériques. Ces 3 canaux sont ceux utilisés par tous les périphériques pour rejoindre un canal. Pendant la procédure de `Join`, le réseau peut demander aux périphériques d'ajouter de nouveaux canaux à sa liste. Ces canaux sont utilisés en `downlink`, mais également en `uplink`[40].

1. https://www.lora-alliance.org/lorawan-for-developers

3.2.7 Temps d'émission et nombre de paquets

Le *duty cycle* d'émission a été abordé en Section 2.3.6. Les pourcentages d'utilisation des bandes de fréquences limitent les utilisations qui peuvent être faites de ce protocole. Le temps d'émission est affecté par les paramètres choisis pour l'émission. Comme expliqué en Section 2.3.3 et en Section 3.2.3, le *spreading factor* et le *coding rate* augmentent considérablement le temps d'émission.

Outre ces restrictions, certains fournisseurs de réseau limitent encore plus drastiquement l'utilisation de la bande de fréquence. Par exemple, *The Things Network* (TTN) applique une *Fair Access Policy*[1] afin de réduire la charge d'accès sur leur réseau [43]. Les types de limitations sont les suivantes :

1. Une moyenne de 30 secondes en émission *uplink*, par jour et par périphérique ;
2. Maximum 10 *downlink* messages par jour, en incluant des confirmations pour les messages de type *confirmed uplink* ;
3. Les périphériques avec un SF de 11 ou 12 **fixe** ne sont pas autorisés à rejoindre le réseau (ceci ne s'applique pas aux périphériques avec l'ADR activé).

La limitation de 30 secondes d'émission est la plus restrictive. Par exemple, si un périphérique envoi un *payload* de 55 bytes en utilisant le *data rate* le plus élevé, cela crée une émission de 71.81 millisecondes[43]. En respectant uniquement les standards européens, ceci signifie qu'un périphérique doit attendre 71,09 secondes avant de pouvoir envoyer une nouvelle trame. Cependant, avec la limitation supplémentaire de TTN, le périphérique ne peut envoyer que 417 paquets de 71.81 par jours ; ce qui signifie en moyenne un paquet toutes les 207 secondes ainsi en appliquant ces limitations, une *gateway* TTN peut avoir jusqu'à 1000 nœuds connectés [43]. TTN propose un tableau sur Google Sheet[2] éditable pour calculer le nombre de paquets possibles sur le réseau en fonction de la configuration appliquée sur le périphérique.

3.2.8 Spécification LoRaWAN 1.1

La LoRa Alliance a finalisé la spécification V1.1 en octobre 2017[3]. Il faut à présent du temps pour qu'elle se démocratise sur les différents périphériques. Actuellement, le CMWX1ZZABZ ne propose pas de *firmware* pour le support de cette spécification. Celle-ci ne nécessite par ailleurs pas de nouveau matériel, son implémentation étant entièrement logicielle. Certaines nouveautés sur la sécurité sont exposées en Section 4.2.3.

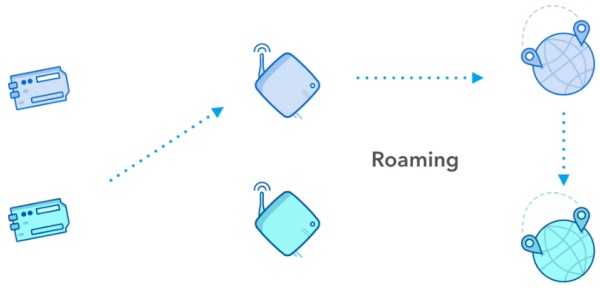


FIGURE 3.23 – Roaming LoRaWAN

1. https://www.thethingsnetwork.org/forum/t/limitations-data-rate-packet-size-30-seconds-uplink-and-10-messages-downlink-per-day-fair-access-policy/1300
2. https://docs.google.com/spreadsheets/d/1QvcKsGeTTPpr9icj4XkKXq4r2zTc2j0gsHLrnplzM3I
3. https://www.lora-alliance.org/lorawan-for-developers

LoRaWAN 1.1 ajoute le support du *roaming* entre différents opérateurs. Le concept est visible sur la Figure 3.23, où l'on peut voir un périphérique transmettant un paquet capturé par une *gateway* d'un réseau auquel il n'appartient pas. Si le *Network Server* ne reconnait pas le périphérique comme appartenant à son réseau, il *forward* les paquets vers le *Network Server* de l'opérateur d'origine. Le principe général est identique au *roaming* appliqué à la téléphonie mobile.

Chapitre 4

Sécurité

Ce chapitre a pour but d'examiner plus en détail les concepts de sécurité des protocoles qui peuvent être utilisés dans ce projet.

4.1 Bluetooth Low Energy

Bluetooth est l'un des protocoles les plus utilisés pour réaliser des communications à courtes distances. Il implémente plusieurs mécanismes afin de chiffrer l'échange de données entre deux dispositifs.

Note : La section de ce rapport n'a pas pour objectif de traiter la cryptographie même du protocole Bluetooth, ni la manière dont les clés sont générées algorithmiquement. Pour des informations complémentaires sur la génération des clés, il est proposé de lire deux articles. Le premier est élaboré par l'instance officielle du Bluetooth et le deuxième est un *whitepaper* rédigé par le très célèbre fabricant d'appareils de mesures électroniques Teledyne Lecroy :

`http://blog.bluetooth.com/bluetooth-pairing-part-1-pairing-feature-exchange`
`http://www.fte.com/webhelp/sodera/Content/Documentation/WhitePapers/BTLE/BluetoothSmartSecurity.htm`

La sécurité de la technologie Bluetooth a évolué au fil des diverses versions. Actuellement, les versions 4.2 et 5.0 ont les mêmes caractéristiques. Aucun élément de sécurité supplémentaire n'a été ajouté dans le Bluetooth 5.0 [44]. Cependant, beaucoup de changements sont survenus entre les versions 4.0, 4.1 et 4.2. Ces anciennes versions ne seront pas abordées en détail, puisque presque aucun dispositif Bluetooth moderne n'est distribué sans le support de la version 4.2 au minimum.

Lors de l'explication des différents blocs de la couche *Host* du Bluetooth, le bloc `Security Manager` a été abordé (cf. Section 3.1.5.3.3). Ce bloc s'occupe des générations et des sauvegardes de clés, de même que des méthodes et protocoles utilisés pour appairer deux dispositifs et partager les clés de connexion.

Lors de la mise en place d'une connexion sécurisée BLE, deux définitions sont utilisées :

- Initiateur (*initiator*) : correspond au maitre au niveau de la couche `Link Layer` et est aussi appelé centrale pour la couche GAP.
- Répondeur (*responder*) : correspond à l'esclave au niveau de la couche `Link Layer` et est aussi appelé périphérique pour la couche GAP.

4.1.1 Modes et niveaux de sécurité

Dans la Section 3.1.5.3.2, le bloc GAP a été défini comme étant l'élément s'occupant de l'authentification des centrales qui souhaitent se connecter auprès du périphérique. Il existe le concept de modes et niveaux de sécurité pour une connexion BLE. Voici la définition de ceux-ci :

1. Mode 1 :
   (a) Niveau 1 : Pas de sécurité (ni authentifiée, ni chiffrée) ;
   (b) Niveau 2 : Appairage non authentifié avec transfert de données chiffrées ;
   (c) Niveau 3 : Appairage authentifié avec transfert de données chiffrées ;
   (d) Niveau 4 : Appairage authentifié avec *LE Secure Connections* et transfert de données chiffrées.
2. Mode 2 :
   (a) Niveau 1 : Appairage non authentifié avec transfert de données signées ;
   (b) Niveau 2 : Appairage authentifié avec transfert de données signées.

Toutes les connexions commencent en **mode 1** et **niveau 1** [45].

4.1.2 Appairage et attachement

L'appairage Bluetooth (*pairing* en Anglais) est le processus qui définit l'échange des données de sécurité et la génération des clés [46][47]. Il y a au total trois phases lors de ce processus d'appairage, lesquelles sont visibles sur la Figure 4.1 et expliquées en détail dans les sous-sections qui suivent.

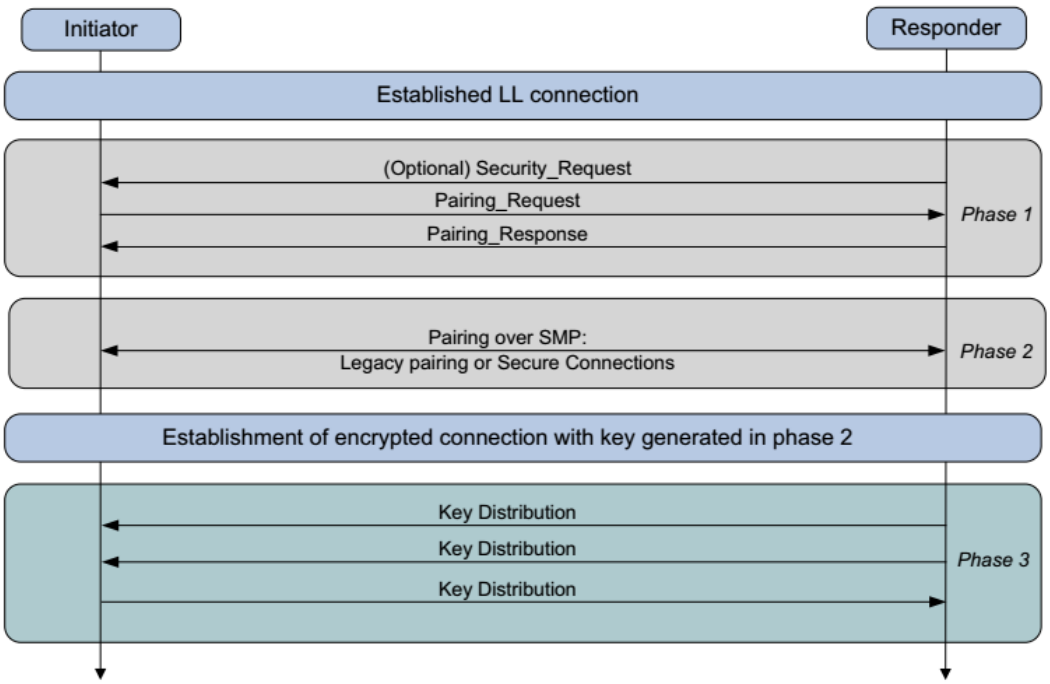


FIGURE 4.1 – Diagramme du processus d'appairage

Ce processus de sécurité n'est pas forcément déclenché directement à la connexion, car plusieurs niveaux de sécurité peuvent être appliqués pour différents attributs. Par exemple, si une périphérique souhaite uniquement protéger un seul et unique attribut, il peut placer tous les autres attributs en Mode 1 Niveau 1. Le processus d'appairage

ne sera déclenché que lorsque un attribut du mode 1 niveau 2 (au minimum) est accédé. L'appairage consiste à l'authentification de l'identité entre deux dispositifs et la création d'un canal chiffré utilisant une *Short-Term Key* (STK). Une fois ce canal, créé, il faut ensuite distribuer des *Long-Term Keys* (LTK), c'est à la fin de cette que la connexion est considéré comme attachée (*bonded*). Les LTK seront ensuite sauvegardées sur les deux dispositifs pour chiffrer les futures communications sans nécessiter un nouvel appairage.

Il est important de comprendre le fonctionnement de ce processus, car tous ces modes doivent être configurés manuellement sur un microcontrôleur.

4.1.2.1 Phase 1 : *Pairing Feature Exchange*

La phase 1 a pour objectif de partager les prérequis qui sont exigés pour la connexion. Parmi ces données, on trouve diverses informations, telles que les entrées/-sorties disponibles sur le périphérique, si l'utilisateur exige une protection *man-in-the-middle* (MITM), etc. Ceux-ci seront ensuite analysés en phase 2 afin de s'assurer que le mode et niveau choisi pour la connexion est possible avec les informations fournies dans la phase 1. Les dispositifs doivent indiquer les capabilités de leurs I/O à l'aide des types définis sur la Figure 4.2 pour les entrées et Figure 4.3 pour les sorties.

Capability	Description
No input	Device does not have the ability to indicate 'yes' or 'no'
Yes / No	Device has at least two buttons that can be easily mapped to 'yes' and 'no' or the device has a mechanism whereby the user can indicate either 'yes' or 'no' (see note below).
Keyboard	Device has a numeric keyboard that can input the numbers '0' through '9' and a confirmation. Device also has at least two buttons that can be easily mapped to 'yes' and 'no' or the device has a mechanism whereby the user can indicate either 'yes' or 'no' (see note below).

FIGURE 4.2 – Capacité des entrées disponibles sur un dispositif Bluetooth

Capability	Description
No output	Device does not have the ability to display or communicate a 6 digit decimal number
Numeric output	Device has the ability to display or communicate a 6 digit decimal number

FIGURE 4.3 – Capacité des sorties disponibles sur un dispositif Bluetooth

4.1.2.2 Phase 2 : *LE Legacy Pairing* ou *LE Secure Connections*

Après l'échange des *pairing features* de la phase 1, la centrale et le périphérique doivent s'accorder sur quelle méthode utiliser pour l'échange et la génération des clés.

Depuis l'introduction du Bluetooth 4.2, si les deux dispositifs sont compatibles avec l'option LE Secure Connections et Secure Simple Pairing (SSP), alors il est conseillé d'utiliser ceux-ci (mode 1 niveau 4). La nouveauté majeure dans ce type de

connexion est l'algorithme utilisé pour la génération des clés. Celui-ci est maintenant basé sur un échange et génération de clés publiques à l'aide de l'algorithme *Elliptic Curve Diffie Hellman*[1] (ECDH) [48]. La Figure 4.4 illustre la différence entre le mode *Legacy* (avant BLE 4.2) et les nouveaux algorithmes utilisés pour l'appairage et le *bonding* [49].

Security Mechanism	Legacy	Secure Simple Pairing	Secure Connections
Encryption	E0	E0	AES-CCM
Authentication	SAFER+	SAFER+	HMAC-SHA256
Key Generation	SAFER+	P-192 ECDH HMAC-SHA-256	P-256 ECDH HMAC-SHA-256

FIGURE 4.4 – Comparaison des algorithmes de générations de clés en fonction des versions BLE

En Bluetooth Low Energy, il existe quatre méthodes pour la génération et l'échange de clés, de même que l'authentification de l'utilisateur :

— ***Just Works*** : la STK est générée sur les deux dispositifs et est envoyée en *plain-text*. Une fois ces clés publiques échangées, le *responder* génère un *nonce* aléatoire avec lequel il génère une valeur de confirmation nommée `Cb`. Ces deux paramètres (nonce et Cb) sont ensuite transmis à l'initiateur. En parallèle, l'initiateur génère son propre *nonce* et le transfert au *responder*. L'initiateur utilise ensuite le *nonce* reçu de la part du *responder* pour générer la valeur de confirmation nommée `Ca`. Les valeurs `Cb` et `Ca` doivent être identiques, si c'est le cas, la connexion est acceptée[48]. Cette méthode ne fournit pas d'opportunité à l'utilisateur de vérifier la légitimité du périphérique connecté. Une attaque de type *man-in-the-middle* peut toujours prendre place[45].

— ***Passkey Display*** : un des deux dispositifs affiche un code PIN (*passkey*) de minimum 6 digits (en LE Secure Connection) et de préférence aléatoire. Le deuxième dispositif doit demander à l'utilisateur d'entrer le code affiché. Cette méthode Il faut donc au minimum un clavier sur un périphérique et un écran sur l'autre. Une protection *man-in-the-middle* est prodigué, sauf si le PIN généré est à la vue de tous.

— ***Out Of the Band (OOB)*** : lors de la phase 1, les deux dispositifs indiquent s'ils acceptent un échange de clé par OOB. Cela signifie que l'échange de clés ne s'effectue pas à l'aide de la bande de fréquence du Bluetooth (2.4 GHz), mais à l'aide d'un autre mécanisme. Les méthodes les plus connues sont l'échange par communication NFC ou la lecture d'un QR code. [45] [48].

— ***Numeric Comparison*** : cette méthode utilise la même procédure que `Just Works`, mais rajoute une nouvelle étape à la fin. Une fois que les périphériques confirment la réception des *nonces* utilisés pour la génération des clés, ils génèrent un nombre aléatoire avec ceux-ci. Les deux périphériques doivent ensuite l'afficher et si l'utilisateur voit que les deux valeurs sont identiques, alors il doit appuyer sur un bouton *OK* pour valider que les deux périphériques soient bien authentifiés entre eux. Si ce n'est pas un cas, un bouton *NOT OK* doit être pressé.

1. https://en.wikipedia.org/wiki/Elliptic-curve_Diffie%E2%80%93Hellman

Le choix de la méthode utilisée dépend ensuite des paramètres échangés à l'aide du *pairing request* et *pairing response*. Si l'option OOB est possible pour les deux périphériques, elle sera activée par défaut. Le reste des méthodes est choisi en fonction du tableau affiché sur la Figure 4.5. L'un des algorithmes présentés est ensuite choisi à la suite des paramètres échangé précédemment.

		Initiator				
	I/O Capabilites	DisplayOnly	Display YesNo	Keyboard	NoInput NoOutput	Keyboard Display
Responder	Display Only	**Just Works** **Unauthenticated**	**Just Works** **Unauthenticated**	**Passkey Display** (responder displays, initiator inputs) **Authenticated**	**Just Works** **Unauthenticated**	**Passkey Display** (responder displays, initiator inputs) **Authenticated**
	Display YesNo	**Just Works** **Unauthenticated**	**Just Works** [For LE Legacy Pairing] **Unauthenticated** **Numeric Comparison** (For LE Secure Connections Pairing) **Authenticated**	**Passkey Display** (responder displays, initiator inputs) **Authenticated**	**Just Works** **Unauthenticated**	**Passkey Display** (responder displays, initiator inputs) [For LE Legacy Pairing] **Authenticated** **Numeric Comparison** (For LE Secure Connections Pairing) **Authenticated**
	Keyboard Only	**Passkey Display** (initiator displays, responder inputs) **Authenticated**	**Passkey Display** (initiator displays, responder inputs) **Authenticated**	**Passkey Display** (initiator displays, responder inputs) **Authenticated**	**Just Works** **Unauthenticated**	**Passkey Display** (initiator displays, responder inputs) **Authenticated**
	NoInput No Output	**Just Works** **Unauthenticated**	**Just Works** **Unauthenticated**	**Just Works** **Unauthenticated**	**Just Works** **Unauthenticated**	**Just Works** **Unauthenticated**
	Keyboard Display	**Passkey Display** (initiator displays, responder inputs) **Authenticated**	**Passkey Display** [For LE Legacy Pairing] (initiator displays, responder inputs) **Authenticated** **Numeric Comparison** (For LE Secure Connections Pairing) **Authenticated**	**Passkey Display** (responder displays, initiator inputs) **Authenticated**	**Just Works** **Unauthenticated**	**Passkey Display** [For LE Legacy Pairing] (initiator displays, responder inputs) **Authenticated** **Numeric Comparison** (For LE Secure Connections Pairing) **Authenticated**

FIGURE 4.5 – Choix de la méthode de sécurité en fonction des entrées/sorties

Dans le cas d'un smartphone, celui-ci dispose à la fois d'un clavier et d'un écran. Il offre donc le niveau de sécurité maximal pour l'échange d'informations à l'aide du Bluetooth. Le tableau de la Figure 4.5 représente les recommandations pour les méthodes à utiliser, mais cela n'empêche pas le développeur de mettre en place, par exemple, un *passkey* sans avoir d'écran sur le *responder*. C'est l'une des raisons pour lesquelles beaucoup de périphériques dans le commerce utilisent des *passkeys* tels que 0000 ou 1234 pour la première connexion. Pour cela, les périphériques sont obligés de mentir lorsqu'il envoie le paquet de réponses à l'initiateur. Ils indiquent qu'ils sont de type ***display only*** alors que cette information est fausse. L'initiateur, la plupart du temps un smartphone ou ordinateur, choisi à l'aide du tableau la méthode qu'il doit appliqué et choisi *initiator* : *Keyboard* + *Display* et *responder* : *display only*. Le smartphone demande ainsi à l'utilisateur d'enter le *passkey*. Ce type d'approche

détruit le concept d'authentification, car le but initial du *passkey* est de s'assurer que la personne qui se connecte est bien celle qui est en face du périphérique, et donc qui a un accès physique à ce dernier avec la visibilité du *passkey* sur l'écran.

Dans le cadre de ce projet, c'est le choix du *passkey* qui a été retenu. Un *passkey* aléatoire est présent dans chaque DevBox et est uniquement accessible à une personne de confiance (cf. Section 6.4.7 pour plus de détail sur l'implémentation). Mais le PIN n'est pas connu à l'avance, contrairement à la plupart des périphériques du marché où celui-ci est identique sur tous les périphériques vendus.

4.1.2.3 Phase 3 : *Transport Specific Key Distribution*

Aussitôt que l'authentification est réussie, les deux dispositifs génèrent une Long Term Key (LTK) qui sera utilisée pour le chiffrement des futures connexions [50]. Toutes les prochaines reconnexions des périphériques utiliseront cette clé, sans avoir besoin de réitérer les deux phases précédentes.

4.1.3 Recommandations de sécurité

En mai 2017, le National Institute of Standards and Technology (NIST) a publié un document intitulé *Guide to Bluetooth Security*. Dans ce document de 67 pages, le NIST explique les vulnérabilités présentes sur des dispositifs Bluetooth, ainsi que les différentes failles qui peuvent y être associées [51]. Le document analyse la sécurité de tous les protocoles Bluetooth récents (Low Energy ou *Standard*). Il explique en outre les concepts de cryptographie qui sont utilisés pour la génération des clés. Voici le lien de la publication :

`http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-121r2.pdf`

À la fin dudit document, 37 recommandations sont proposées pour optimiser la sécurité sur un système utilisant le Bluetooth. Voici quatre exemples de recommandations extrêmement pertinentes pour le projet. La première indique le mode et le niveau de sécurité devant être utilisés sur les périphériques :

> *"15. Bluetooth 4.2 devices and services using low energy functionality should use Security Mode 1 Level 4 whenever possible. Low energy Security Mode 1 Level 4 implements Secure Connections mode and provides the highest security available for 4.2 low energy devices. If Security Mode 1 Level 4 is not available, recommend using Security Mode 1 Level 3 instead"*

Le KW41Z supporte tous les modes de sécurité et niveaux de sécurité. Cette recommandation peut ainsi être utilisée dans le projet.

Dans ce projet, un certain nombre de profils Bluetooth devant être implémentés, il est important de connaitre les recommandations y relatives. Voici la seule proposée pour les profils et services Bluetooth Low Energy dans le document :

> *"17. Unneeded and unapproved service and profiles should be disabled. Many Bluetooth stacks are designed to support multiple profiles and associated services. The Bluetooth stack on a device should be locked down to ensure only required and approved profiles and services are available for use"*

Par défaut, beaucoup de profils Bluetooth sont activés sur un même périphérique. Cependant, si l'utilisateur final n'a pas d'utilité à tous ces services, il convient de les désactiver afin qu'une éventuelle faille ne puisse être exploitée. Il est ici fait référence à des *unapproved services*, référence qui peut être interprétée de deux façons. En premier lieu, il pourrait s'agir des services standards Bluetooth et fournis par la spécification Bluetooth. Néanmoins, cela semble beaucoup trop restrictif comme approche. Si un programmeur ne peut pas développer son propre service, il lui est impossible de faire certaines applications, car elles n'ont pas toutes été prévues par la spécification. L'on pourrait considérer, en second lieu, qu'aucun service non autorisé au sein du développement de l'application (par exemple un service de débogage) ne devrait être intégré dans le produit final. La seconde interprétation parait plus sensée avec les besoins des développeurs et des applications.

Lorsqu'une entreprise ou entité dispose de plusieurs périphériques Bluetooth Low Energy, il est important de garder une trace de ceux-ci, de sorte à créer une *whitelist*. C'est pour cela que le NIST adresse la recommandation suivante :

> *"6. Maintain a complete inventory of all Bluetooth-enabled wireless devices and addresses (BD_ADDRs)."*

Les dispositifs sont bien souvent protégés par des codes PIN, afin de ne pas autoriser n'importe quelle centrale à s'appairer avec celui-ci. NIST propose les recommandations suivantes pour la génération de ces codes PIN :

> *"9. Choose PIN codes that are sufficiently random, long and private. Avoid static and weak PINs, such as all zeroes. PIN codes should be random so that malicious users cannot easily guess them. Longer PIN codes are more resistant to brute force attacks. For Bluetooth 2.0 (or earlier) devices, an eight-character alphanumeric PIN should be used, if possible. The use of a fixed PIN is not acceptable."*

Toutes ces recommandations sont parfois difficiles, voire impossibles à implémenter selon le type d'équipement utilisé. Si l'équipement le permet, il est toutefois préférable de les appliquer.

4.2 LoRaWAN

La modulation LoRa en tant que telle ne comprend pas d'aspect lié à la sécurité, car il ne s'agit que de la couche 1 du modèle OSI. La sécurité est gérée par les couches supérieures, ce qui relève du protocole que l'utilisateur souhaite appliquer au-dessus de la modulation. Cette section sécurité se concentre principalement sur le mode OTAA pour l'enregistrement sur un réseau. Celui-ci est considéré comme étant le plus sûr et le mieux adapté à des dispositifs LoRaWAN [52][53].

Le 11 octobre 2017, la LoRa Alliance a dévoilé la version définitive du protocole LoRaWAN 1.1. Les explications prodiguées dans les sous-sections qui suivent décrivent la version 1.0.2 de LoRaWAN. Peu d'opérateurs proposent un support pour la spécification 1.1. Par exemple, le réseau The Things Network a pour objectif d'apporter un support uniquement pour avril ou mai 2018. Le module LoRaWAN choisi n'est actuellement pas encore compatible avec cette dernière version. Toutefois, la mise à jour des périphériques ne nécessite qu'une mise à niveau logicielle et non matérielle. Un résumé des nouveautés de LoRaWAN 1.1 est proposé en Section 4.2.3.

Le 23 mars 2016, la société MWR InfoSecurity [1] spécialisée dans la recherche et les conseils en sécurité informatique a publié un *whitepaper* sur la sécurité d'un réseau LoRa. Le document est intitulé *LoRa Security : Building a secure LoRa solution* et écrit par Robert Miller [53]. Les explications du présent chapitre ont été en partie reprises de ce *whitepaper*, lequel contient un résumé des points essentiels présentés dans la spécification LoRaWAN [2].

4.2.1 Rejoindre un réseau à l'aide d'OTAA

La structure des paquets LoRaWAN a été présentée en Section 3.2. Il est important d'avoir en tête les différents champs de ces paquets pour comprendre la façon dont la sécurité est appliquée aux messages LoRaWAN.

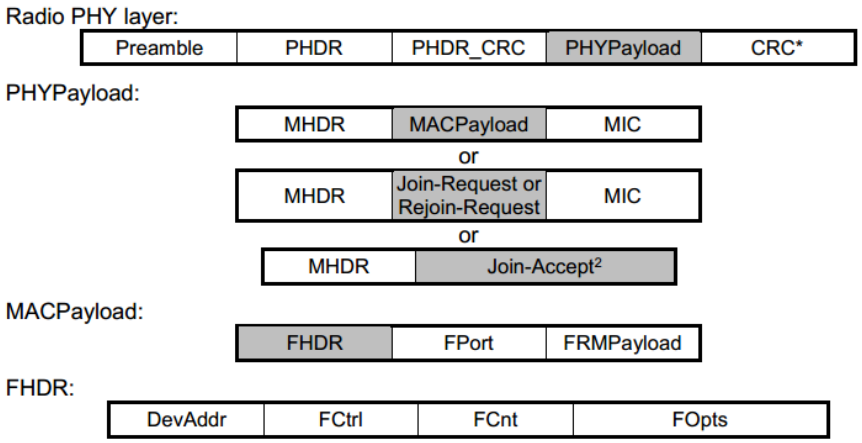


FIGURE 4.6 – Format des messages du protocole LoRaWAN

Chaque périphérique du réseau est déployé avec une `AppKey` et une `App EUI`. Cette première clé est indispensable lors de la procédure raccordement au réseau LoRaWAN pour authentifier le périphérique. L'`App EUI` est un identifiant qui est normalement unique pour l'utilisateur du périphérique et qui permet de déterminer

1. https://www.mwrinfosecurity.com/
2. https://www.lora-alliance.org/lorawan-for-developers

à quelle application ce périphérique appartient. Pour initialiser cette procédure, le périphérique construit un message de type `Join Request` (cf. Figure 4.6) composé de l'`App EUI` de son `Dev EUI` de même que d'une valeur générée aléatoirement sur 2 bytes nommé DevNonce [53]. Le `Dev EUI` est l'identifiant unique du périphérique. Ces trois éléments sont ensuite signés à l'aide d'une valeur sur 4 bytes nommée *Message Integrity Code* (MIC) et calculée comme suit :

```
mac=aes128_cmac(AppKey, MHDR | AppEUI | DevEUI | DevNonce)
MIC = mac[0..3]
```

A la réception du message, le *Network Server* détecte qu'il s'agit là d'un *Join Request* et transfère la demande au *Join Server* couplé à une valeur aléatoire (*AppNonce*) qui sera utilisée pour la génération des clés. Le *Join Server*vérifie les informations reçues afin de déterminer si le périphérique est ou non autorisé à rejoindre le réseau, puis il recalcule le MIC de son côté afin de s'assurer que le message n'a pas pu être altéré par une tierce personne. Si tout est validé, le *Join Server* informe le *Network Server* qu'une réponse de type *Join Accept* doit être transmisse au périphérique. Cette réponse contient l'AppNonce précédemment généré par le *Network Server*. Avec toutes ces informations, le serveur peut générer les deux clés de sessions (*Network Session Key* et *Application Session Key*) :

```
NwkSKey = aes128_encrypt(AppKey, 0x01 | AppNonce | NetID | DevNonce | pad16)
AppSKey = aes128_encrypt(AppKey, 0x02 | AppNonce | NetID | DevNonce | pad16)
```

Le *Join Accept* contient également les informations concernant les délais RF (RxDelay), le *Dev Address* et la liste des canaux à utiliser (CFList). Un MIC doit être généré pour être ajouté au paquet de réponse :

```
mac = aes128_cmac(AppKey, MHDR | AppNonce | NetID | DevAddr | RFU | RxDelay |
 ↪  CFList)
MIC = mac[0..3]
```

Les données sont retournées en utilisant l'*AppKey* pour le chiffrement. A réception du paquet, le nœud déchiffre les données et calcule à son tour les clés *Network Session Key* et *Application Session Key*. Ces deux clés sont maintenant utilisées pour chiffrer toutes les nouvelles données transmisses sur le réseau. La première pour les paquets qui sont envoyés sur le port 0 (*Network Server*) ; la deuxième est quant à elle utilisée pour les données sur le port 1 à 223 impliquant l'*Application Server*.

4.2.2 Protection des données

Une fois un réseau LoRaWAN rejoint, toutes les données sont chiffrées en utilisant les deux clés de sessions.

4.2.2.1 Chiffrement des données

Le chiffrement des messages utilise AES-128 [1] en *Counter mode* (CTR) [53]. Une composante importante de la sécurité LoRaWAN réside dans les compteurs de messages *uplink* (FCntUp) et *downlink* (FCntDown) maintenus par le périphérique et le Network Serveur. Cette technique évite ainsi la possibilité de faire du *replay attack* [2] sur en copiant l'intégralité d'un paquet. Pour le chiffrement et le déchiffrement, un *keystream* S est généré à l'aide des informations suivantes :

1. https://en.wikipedia.org/wiki/Advanced_Encryption_Standard
2. https://en.wikipedia.org/wiki/Replay_attack

```
i = 1..k where
k = ceil(len(FRMPayload) / 16)
Ai = (0x01 | (0x00 * 4) | Dir | DevAddr | FCntUp or FCntDown | 0x00 | i)
Si = aes128_encrypt(K,Ai), for i = 1..k
S = S1|S2|..|Sk
```

Une opération XOR est ensuite appliquée sur le champ `FRMPayload` avec le *keystream* S pour chiffrer ou déchiffrer les données. Certaines données telles que le `FPort` ou `FCntUp` sont envoyées en clair.

4.2.2.2 Signature des messages

Dans le champ *MAC Payload* (cf. Figure 4.6), les données sont singées afin d'éviter des modifications du *cipher-text* ou d'autres valeurs comme le Dev Addr, FCnt ou FCntDown. De nouveau un champ MIC est utilisé comme signature :

```
Msg = MHDR | FHDR | FPort | FRMPayload
B0 = (0x49 | 4*0x00 | Dir | DevAddr | FCntUp or FCntDown | 0x00 | len(msg) )
mac = aes128_cmac(NwkSKey, B0 | msg)
MIC = mac[0..3]
```

4.2.2.3 Infrastructure des opérateurs LoRaWAN

L'architecture LoRaWAN a déjà été abordée en Section 3.2.2, avec la description des cinq composants principaux (périphérique, *gateway*, *Network Server*, *Join Server* et *Application Server*).

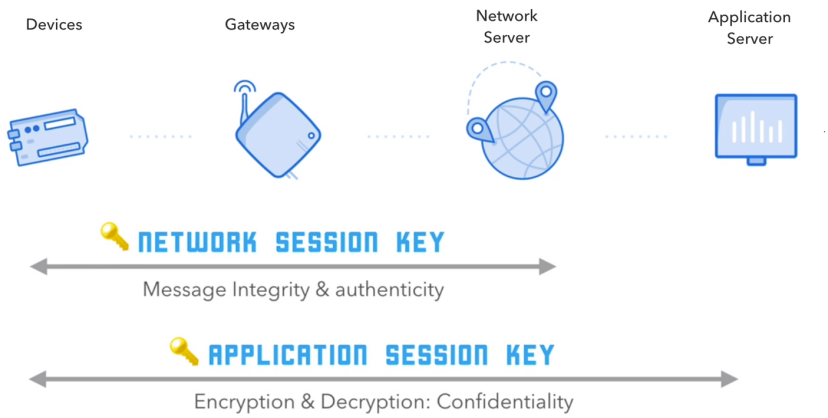


FIGURE 4.7 – Infrastructure d'un réseau LoRaWAN comme imaginée dans la spécification

Le schéma de la Figure 4.7 expose l'utilisation des clés dans un réseau LoRaWAN. L'infrastructure y étant illustrée affiche une configuration pensée dans la spécification LoRaWAN : une séparation entre le *Network Server* et l'*Application Server*.

La réalité est quelque peu différente de cette implémentation. En effet, la plupart des fournisseurs de réseau LoRaWAN (The Things Network, Swisscom, Loriot, etc.) n'offrent pas la possibilité de séparer ces deux entités. L'architecture pratiquée par ces opérateurs est visible sur la Figure 4.8. L'opérateur possède les deux serveurs et fournit un accès au données au travers d'API propres à chacun. Cette architecture

présente une problématique pour la sécurité des données. En effet, l'opérateur a à sa disposition la clé la plus importante, l'AppKey. Avec celle-ci, il doit générer la Network Session Key, indispensable à l'opérateur, et l'Application Session Key pour déchiffrer les données fournies à l'utilisateur. La plupart des opérateurs ont choisi cette approche puisque les clients ne souhaitent pas nécessairement mettre en place un *Application Server* et un *Join Server* en interne. Le SmartCanton a quant à lui décidé d'utiliser cette approche afin de garantir une sécurité et une maitrise totale sur les données tout en utilisant les infrastructures d'un opérateur existant pour le transport des paquets (cf. Section 4.2.5).

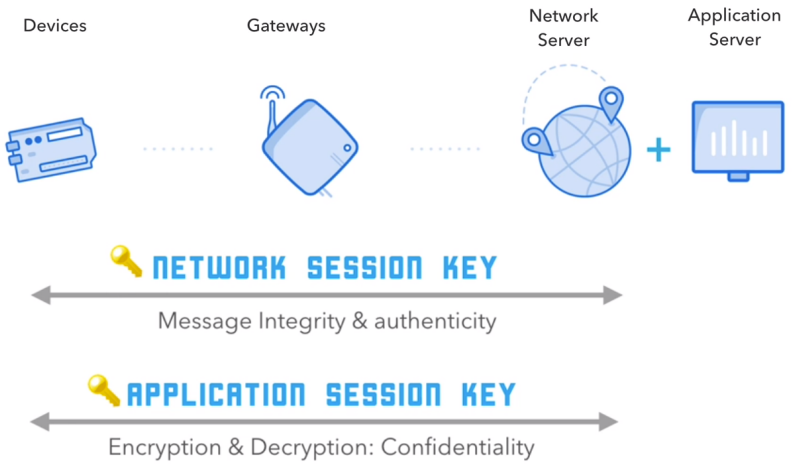


FIGURE 4.8 – Infrastructure d'un réseau LoRaWAN réellement implémenté avec les opérateurs du marché

4.2.3 LoRaWAN spécification 1.1

La spécification LoRaWAN 1.1 apporte de nouvelles améliorations dans le domaine de la sécurité avec l'ajout de nouveaux concepts et des changements de mécanismes [54]. Voici un résumé des changements principaux :

- le compteur du nombre de paquets envoyés doit être stocké dans une mémoire non volatile et le même compteur ne peut pas être utilisable dans une même session. Un périphérique ABP ne peut ainsi plus être remis à zéro ;
- DevNonce et JoinNonce (précédemment nommé AppNonce) ne sont plus aléatoire. Ce sont maintenant des valeurs incrémentales qui ne peuvent être utilisées qu'une seule fois et doivent être stockées en mémoire non volatile sur le périphérique ;
- compteurs de message sur 32 bits (il s'agissait précédemment d'une recommandation, la version 16 bits était autorisée) ;
- deux compteurs de messages *downlink* séparés ; un pour le réseau (NFCntDown), l'autre pour l'application (AFCntDown) ;
- le compteur d'*uplink* (FCntUp) est maintenant utilisé pour générer le MIC de l'*acknowledge*.

Tous les périphériques doivent obligatoirement avoir la possibilité de stocker en mémoire non volatile les différentes informations. C'est la seule obligation matérielle impérative sur les périphériques 1.1.

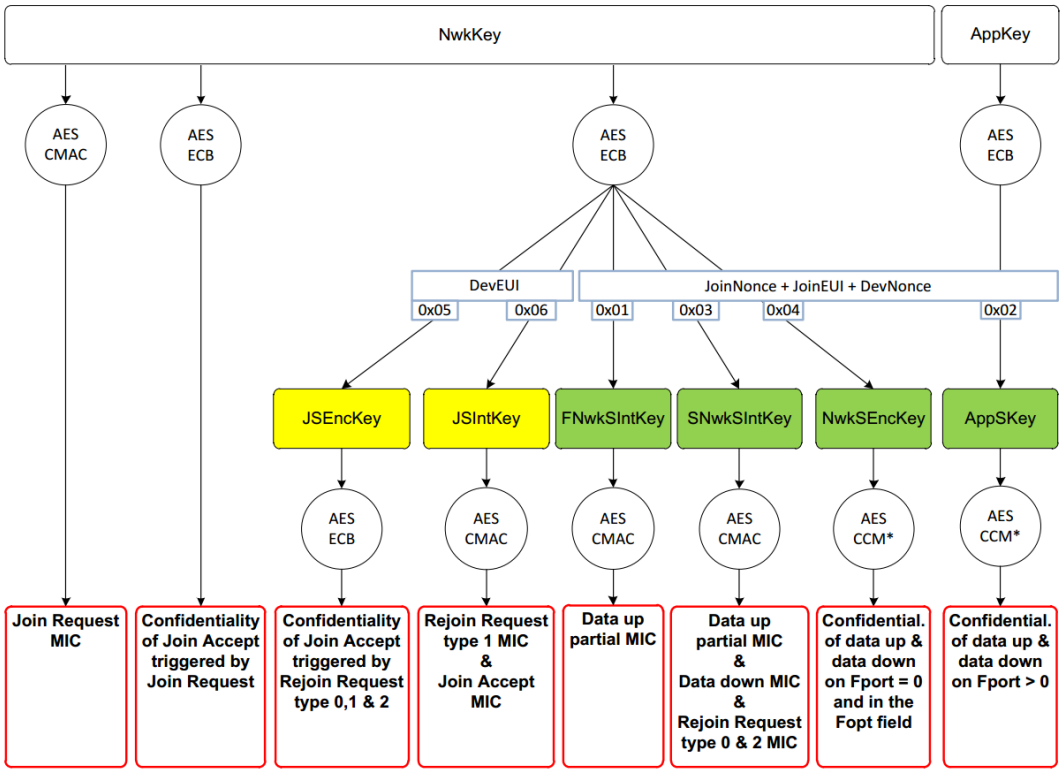


FIGURE 4.9 – Clés utilisées dans un réseau LoRaWAN avec la nouvelle spécification 1.1

Le changement le plus important de la spécification 1.1 est l'introduction d'un nouveau secret partagé, nommé Network Key (**NwkKey**). Cette clé est utilisée pour créer trois nouvelles clés :

1. NwkSEncKey : clé utilisée pour chiffrer les commandes MAC ;
2. SNwkSIntKey : utilisée par un *Serving Network Server* pour le calcul de la première moitié du MIC ;
3. FNwkSIntKey : utilisée par le *Forwarding Network Server* pour le calcul de l'autre moitié du MIC.

Une meilleure vue d'ensemble de toutes les clés et leur utilité est visible à l'aide de la Figure 4.9. Si un périphérique 1.1 souhaite rejoindre un réseau gouverné par un Network Serveur 1.0, toutes les clés seront calculées à l'aide de la NwkKey. Les clés de sessions seront identiques et le périphérique n'utilisera plus sa AppKey.

Conjointement avec la parution de la spécification LoRaWAN 1.1, une nouvelle spécification pour la *backend* des réseaux LoRaWAN nommée *LoRaWAN Backend Interfaces 1.0 Specification* est disponible [2]. Les informations de celles-ci affectent principalement les opérateurs de réseau LoRaWAN.

4.2.4 Vulnérabilités et recommandations de sécurité

Les sous-sections qui suivent proposent quelques recommandations aux vulnérabilités sur les réseaux LoRaWAN [52] [53]. La spécification 1.1 apporte de nouvelles améliorations, toutefois certaines recommandations restent d'actualités [54].

4.2.4.1 Infrastructure du réseau

L'infrastructure des opérateurs d'un réseau LoRaWAN a été étudiée en Section 4.2.2.3. Laquelle a permis d'exposer la problématique sur le contrôle des serveurs. La plupart des opérateurs offrent une infrastructure complète avec l'intégration des trois serveurs. Il existe deux solutions pour pallier à cette problématique :

1. Utiliser un opérateur autorisant la connexion avec un *Join Server* et un *Application Server* externe. Cette approche est visible sur la Figure 4.10. Il s'agit là d'une réelle séparation des données et du réseau, comme la spécification LoRaWAN l'a prévu.

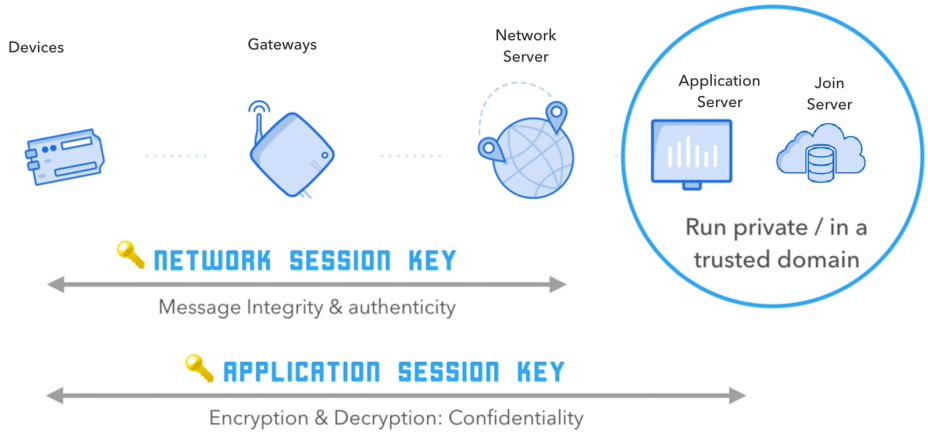


FIGURE 4.10 – Recommandation pour l'implémentation d'une infrastructure LoRaWAN sécurisée

2. Ajouter une nouvelle étape de chiffrement asymétrique, exploré sur la Figure 4.11. Lorsque cette option est utilisée, un opérateur de réseau LoRaWAN ne peut pas déchiffrer. Le déchiffrement peut ensuite être effectué par l'application finale qui se connecte au flux provenant de l'*Application Server*.

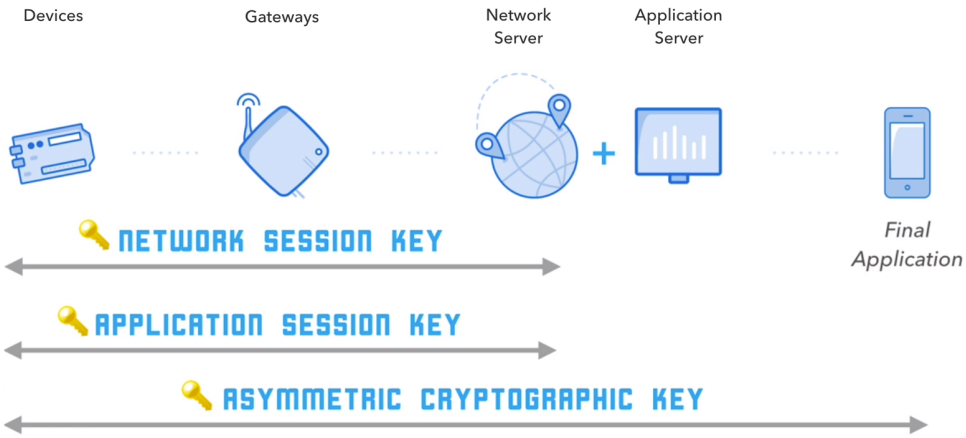


FIGURE 4.11 – Clé asymétrique utilisée pour chiffrer les données

4.2.4.2 Gestion des clés pour les périphériques

Le mode de connexion le plus sécurisé est l'OTAA, car il ne nécessite que la connaissance d'un seul et unique secret partagé entre le *Join Server* et le périphérique, l'AppKey. Lorsque l'une des deux clés de sessions est compromise, le périphérique peut être forcé à se joindre à nouveau au réseau et ainsi régénéré de nouvelles clés de session. La programmation des clés en ABP n'offre pas cette possibilité.

Un attaquant peut toujours récupérer des clés de chiffrements lorsqu'un accès direct au périphérique est possible. Par exemple, les attaques de type *side channel analysis* qui consistent à mesurer la variation de courant ou la mesure des radiations électromagnétiques lorsque le *transceiver* effectue le chiffrement AES pour déterminer la clé utilisée [53].

Toutes les clés stockées sur les périphériques doivent être uniques. Il est possible d'assigner la même AppKey pour tous les périphériques, puisque l'identifiant unique du périphérique est le Dev EUI. Toutefois, ceci ne devrait jamais être le cas, car il suffit qu'un seul périphérique soit compromis pour que tous les périphériques d'un réseau doivent être reprogrammés.

La programmation des clés sur les périphériques ne doit pas se faire à l'aide de la même plage de fréquence que le LoRa [52]. Il est conseillé d'utiliser un nouveau type accès et ainsi appliquer un concept de *out of the band*[1] pour l'authentification, par exemple, en utilisant une technologie en 2.4 GHz.

Il est parfois possible d'envoyer un fichier binaire aux fabricants de cartes électroniques pour qu'ils programment directement les microcontrôleurs à la suite du montage des cartes électroniques [52]. Cette méthode est fortement déconseillée si le fabricant n'est pas de confiance.

Les clés ne doivent pas être stockées en tant que constantes dans le *firmware* du microcontrôleur et la mémoire du microcontrôleur doit être sécurisée (cf. Section 4.4 pour plus d'informations).

4.2.4.3 Gestion des clés pour les serveurs

Les serveurs doivent avoir accès aux différentes clés afin d'effectuer leurs tâches, mais celles-ci ne doivent être accessibles qu'à des personnes autorisées, que ce soit en lecture ou en écriture.

La génération de ces clés (cf. Section 4.2.1) doit être aléatoire, par exemple, en utilisant des *true random number generators*.

4.2.4.4 Compteur de paquets

La vérification des compteurs de paquets doit être correctement effectuée par le *Network Server*. Certains opérateurs allouent à l'utilisateur la possibilité de désactiver cette option pour effectuer des tests, mais ceci n'est pas acceptable lors d'une implémentation définitive.

1. https://en.wikipedia.org/wiki/Out-of-band_data

4.2.4.5 Renouvellement des clés OTAA

Lorsqu'un périphérique rejoint un réseau en OTAA, les clés de sessions générées sont utilisables indéfiniment [52]. Si le périphérique accepte les *uplinks*, une commande applicative peut être envoyée pour demander au périphérique de relancer une nouvelle session à l'aide d'un *join request*. Si les paquets *downlink* ne sont pas implémentés, cette procédure peut être automatisée en local après l'envoi d'un nombre défini de paquets *uplink*. Cette problématique a été corrigée avec le LoRaWAN 1.1 avec l'ajout de nouvelles commandes pour forcer un *rejoin* du réseau en OTAA.

4.2.4.6 Usurpation du microcontrôleur

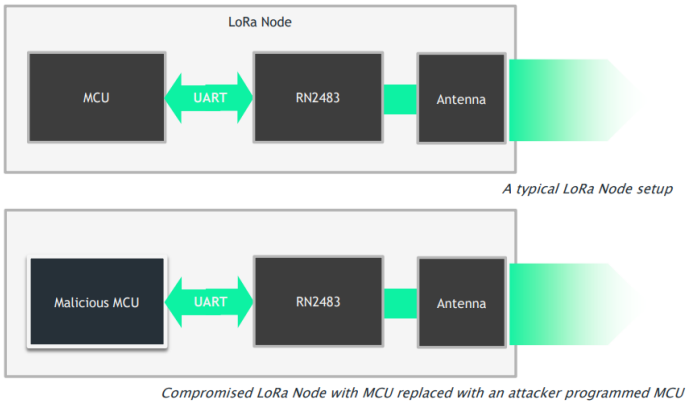


FIGURE 4.12 – Vulnérabilité des modules LoRaWAN avec le remplacement du microcontrôleur

Lorsque l'attaquant dispose d'un accès direct sur le matériel, il lui est possible d'usurper le microcontrôleur principal communiquant. Par exemple, le RN2483 de Microchip [1] offre une interface UART pour la configuration. La Figure 4.12 expose la situation. Une solution pour pallier à ce type d'attaque consiste à utiliser un module qui ne peut pas être lu depuis l'extérieur. Une fois celui-ci programmé pour rejoindre un réseau, il sauvegarde dans sa mémoire non volatile les clés sans les redistribuer. Il est possible de ne pas utiliser de coprocesseur LoRaWAN et d'implémenter la *stack* LoRaWAN directement dans le processeur principal. On peut ainsi utiliser un circuit intégré radio de Semtech. Les inconvénients de ce type d'implémentation est la complexité logicielle ajoutée au processeur principal. En Section 4.4 divers conseils relatifs au stockage des clés dans des mémoires non volatiles sont présentés.

4.2.4.7 Accès aux *gateways*

Si un attaquant a accès à une *gateway*, il ne peut pas accéder aux données, car celles-ci sont toujours chiffrées avec les différentes clés. Toutefois, il a la possibilité de la rendre inopérationnelle. L'attaquant peut également corrompre les données, mais dans ce cas, le *Network Server* détectera cette corruption et les ignorera. Les *gateways* sont le plus souvent connectées à Internet pour créer le pont entre le protocole LoRaWAN et les serveurs. Il est important de les mettre à jour régulièrement, et tout particulièrement lorsqu'une vulnérabilité est découverte ou lorsqu'une nouvelle version du protocole est disponible. Cette mise à jour peut se faire en se déplaçant physiquement auprès de celle-ci ou à distance. La deuxième option est indispensable

1. https://www.microchip.com/wwwproducts/en/RN2483

lorsqu'un grand parc de *gateways* est maintenu. La connexion à celle-ci doit donc être sécurisée du mieux possible à l'aide mots de passe uniques pour chaque *gateway* ou encore mieux, des clés d'authentifications (ex. clés SSH) gardées secrètes.

4.2.5 SmartCanton

Le POC SmartCanton a été présenté brièvement en Section 2.10. Il en est ressorti la forte sécurité qui a été implémentée et pensée dans ce projet. À la Section 4.2, la sécurité LoRaWAN a été étudiée de façon générale. En fin de la Section 4.2, des recommandations pour des réseaux et périphériques LoRaWAN ont été présentées. Le POC SmartCanton a pris très au sérieux les problématiques liées à la sécurité dans l'implémentation du projet. Les buts des sous-sections qui suivent sont d'explorer quels mécanismes ont été appliqués et ainsi mieux comprendre l'intérêt de la mise en place de ce travail de Master.

4.2.5.1 *Hardware Security Module*

Le SmartCanton s'est équipé d'un Hardware Security Module (HSM) de la série Nshield [1] de l'entreprise Thales. Ce composant est utilisé pour générer les clés, mais également pour les stocker. Il s'agit là d'un coffre-fort inviolable et hautement certifié. Une donnée chiffrée peut lui être envoyée pour être déchiffrée, comme l'exemple avec le *Join Server* sur la Figure 4.13.

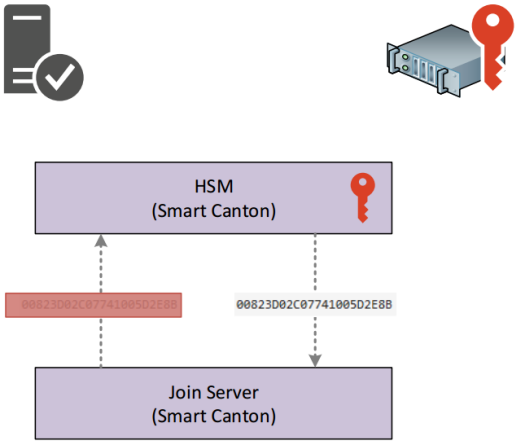


FIGURE 4.13 – Thales Nshield

4.2.5.2 *Join Server*

Pour faciliter la lecture des diagrammes du SmartCanton qui suivent, voici la légende des couleurs des clés utilisées :

- **Bleu** : Application Key (AppKey) ;
- **Rouge** : Application Session Key (AppSKey) ;
- **Orange** : Network Session Key (NwkSKey).

La fédération des opérateurs implémentée dans le POC est présentée à la Figure 2.19, celle-ci est une composante primordiale pour la couverture en réseau du

1. https://www.thalesesecurity.com/products/general-purpose-hsms/nshield-connect

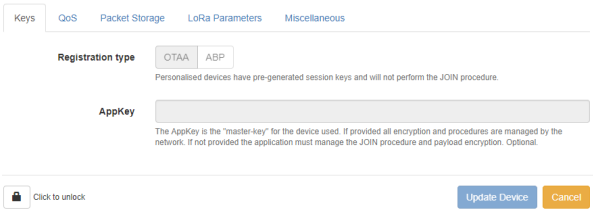


FIGURE 4.14 – Périphérique enregistré sur un réseau OrbiWise sans AppKey

canton et de la ville de Genève. Le réseau d'OrbiWise autorise la création d'un périphérique sans spécification d'AppKey (cf. Figure 4.14, contrairement à TTN et Swisscom où cela est impossible. Cette fédération nécessite la redirection des requêtes vers un *Join Server* personnalisé. Ce *Join Server* est directement connecté au HSM pour la gestion des clés (cf. Figure 4.13).

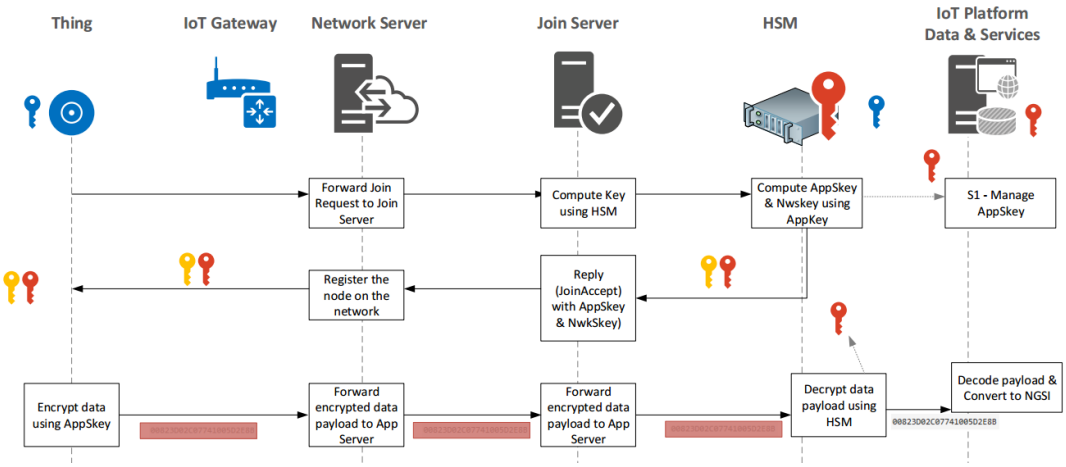


FIGURE 4.15 – Déchiffrement des données par le HSM

OrbiWise équipe son infrastructure d'origine avec deux composants : un *Network Server* (NST) pour la gestion du réseau et un Data Access Sub-System (DASS) qui équivaut à un *Application Server* pour la récupération des données. Le *Join Server* du SmartCanton travaille en tandem avec le Network Server d'OrbiWise pour récupérer les paquets qui sont envoyés par des périphériques inconnus au réseau d'OrbiWise. La Figure 4.15 décrit la séquence complète de connexion d'un nouveau périphérique sur le réseau. Voici les points clés à retenir lors de la séquence de connexion à un partenaire du POC SmartCanton :

1. Le périphérique envoie un *Join Request* sur le réseau.
2. La *gateway* de l'opérateur partenaire le récupère et l'envoi au *Network Server* de l'opérateur.
3. Le *Network Server*, génère un AppNonce et l'envoi, conjointement à la demande de *Join Request*, au *Join Server* du SmartCanton.
4. Le *Join Server* identifie le périphérique comme étant autorisé à rejoindre le réseau et demande au HSM de créer les clés de sessions pour ce dernier. Une copie des clés est conservée à l'intérieur du HSM.
5. Le Join Server construit et chiffre le Join Accept et le renvoi au Network Server accompagné de la NwkSKey.

6. Le *Network Server forward* le *Join Accept* vers le périphérique et sauvegarde en local la NwkSKey pour les futures communications avec le périphérique.

4.2.5.3 Cheminement des données et répartition des clés

Le POC SmartCanton a pensé à deux implémentations de déchiffrement en utilisant les clés :

1. le déchiffrement des données est géré directement par le HSM. Cette implémentation est celle visible sur la Figure 4.15.
2. le déchiffrement des données est géré par la plateforme IoT, par exemple, dans un module FIWARE. Cette implémentation est quant à elle visible sur la Figure 4.15.

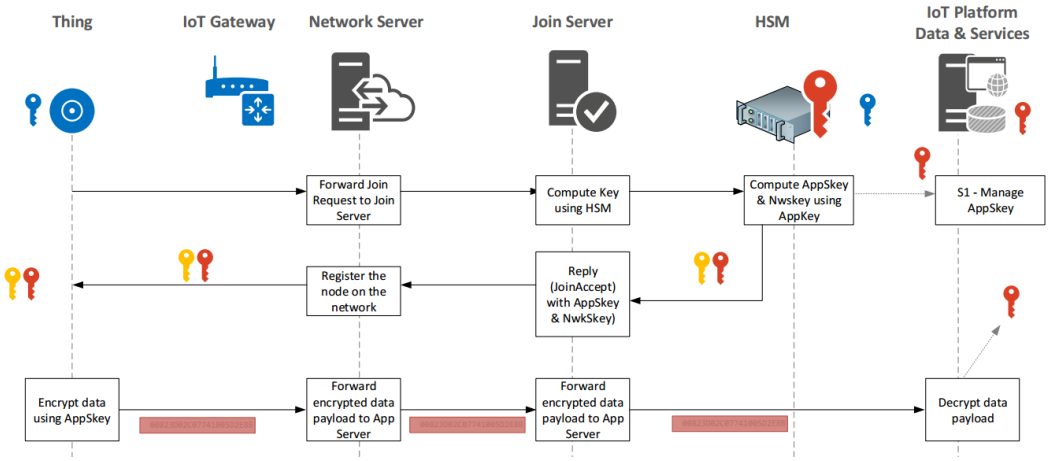


FIGURE 4.16 – Déchiffrement des données par la plateforme IoT (FIWARE)

La Figure 4.17 expose le cheminement complet d'un paquet de données produit par un périphérique (*Thing*) jusqu'à l'application. On peut ainsi voir la répartition des différentes clés sur le schéma. La clé AppKey, la plus importante dans les connexions OTAA, est stockée en sécurité dans le HSM. La seule autre copie de cette clé est sur le périphérique. Le *Network Server* de l'opérateur n'a accès qu'à la Network Session Key, il ne peut ainsi jamais déchiffrer les paquets de données. Cette implémentation reflète le principe qui est conseillé dans les recommandations énoncées à la Section 4.2.4.

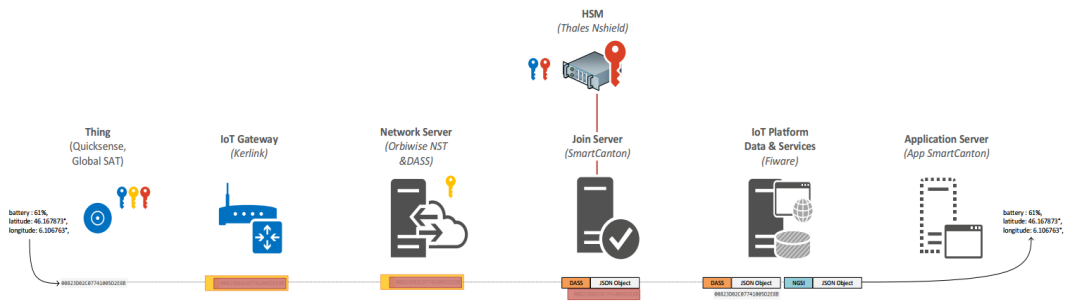


FIGURE 4.17 – Cheminement des données depuis le périphérique jusqu'à l'application en utilisant le HSM pour le déchiffrement

4.3 Serveur Web

Dans le cadre du présent projet, un serveur web a été créé. S'est posée la question de savoir comment sécuriser l'accès à celui-ci. Ci-dessous, une brève explication des deux mécanismes retenus.

4.3.1 HTTPS

Le protocole *HyperText Transfer Protocol Secure* (HTTPS) consiste, de manière résumée, en l'utilisation de la couche HTTP couplée avec un chiffrement à l'aide de SSL ou TLS [55]. HTTPS a pour but d'identifier (l'identité) le site web auquel le client accède. Pour cela, un certificat d'authentification doit être émis par une autorité de certification tierce, reconnue comme fiable par le navigateur ou système d'exploitation effectuant la requête. Toutes les données communiquées entre le client et le serveur sont chiffrées à l'aide d'une paire de clé publique et clé privée (chiffrement asymétrique) [55].

4.3.2 JWT

Les *JSON Web Tokens* (JWT) sont définis dans un standard *open source* (RFC 7519 [1] [56]. Ils offrent la possibilité d'échanger de l'information de manière sécurisée à l'aide d'un jeton signé. Le processus d'identification est illustré à l'aide de la Figure 4.18. Il y a deux étapes :

1. L'utilisateur se connecte à un serveur d'authentification afin de récupérer un JWT. Le processus d'identification n'est pas standardisé, le développeur peut ainsi utiliser une *basic authentification* ou un champ clé/valeur dans une requête POST.
2. L'utilisateur envoie ce JWT dans chacune de ses requêtes vers le serveur applicatif.

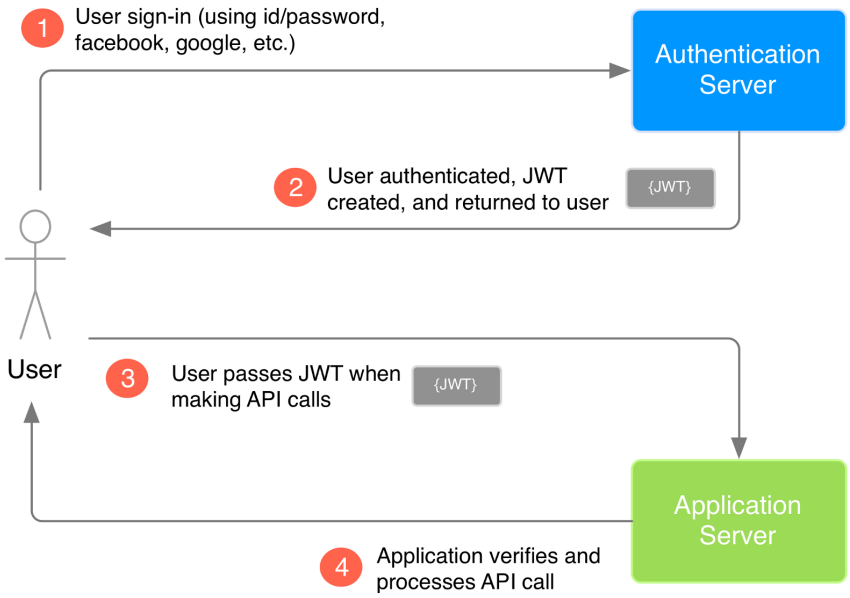


FIGURE 4.18 – Génération et utilisation d'un *token* JWT

1. https://tools.ietf.org/html/rfc7519

Un JWT est composé de trois champs codés en base64 et séparés par un point. Le site officiel de JWT [1] offre la possibilité de déchiffre un *token* directement en ligne. Ce déchiffrement est visible sur la Figure 4.19 et se décompose comme suit :

1. *Header* : entête contenant uniquement le nom de l'algorithme utilisé pour générer la signature du paquet ainsi que le format du *token*.
2. *Payload* : champ modifiable par l'utilisateur comme il le souhaite. Il y a toutefois des clés qui sont standardisées par la norme RFC 7519. Par exemple, la clé `exp` signifie *Expiration Time* exprimée en heure POSIX [2].
3. *Signature* : champ dérivé de la concaténation des deux champs précédents en base64, sur lequel un chiffrement est appliqué à l'aide de la clé privée basée sur l'algorithme spécifié dans le champ `Header`. Quand le serveur reçoit un JWT, ce champ est utilisé pour confirmer que le *token* n'a pas été forgé. La clé privée ne quitte jamais le serveur et doit être gardée en lieu sûr.

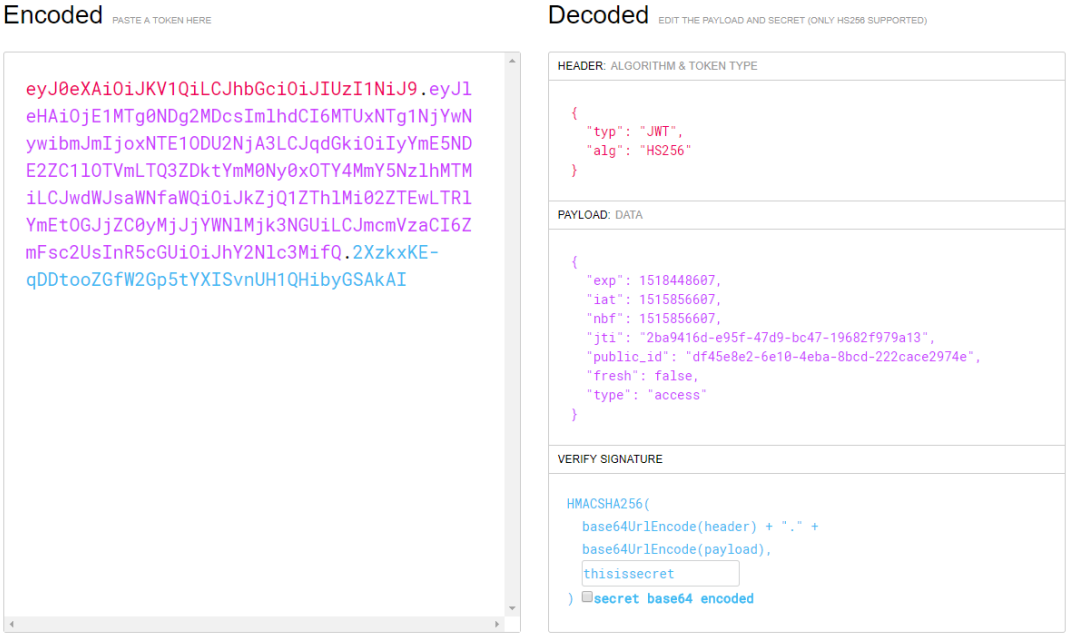


FIGURE 4.19 – Déchiffrement d'un JWT

Il faut être conscient que les *tokens* JWT n'ajoutent aucune couche de sécurité sur les données. Cela permet d'authentifier l'utilisateur à l'aide d'un *token* temporaire. Cela évite d'envoyer en permanence son *login* et mot de passe dans chacune de ses requêtes. Le développeur peut ainsi gérer automatiquement le concept de sessions et, s'il le souhaite, invalider des *tokens* sans demander à l'utilisateur de changer son mot de passe. Si une personne malveillante s'approprie ce *token*, il pourra toujours l'utiliser pour forcer ses propres requêtes. Pour éviter cela, il convient d'utiliser un protocole qui encapsule les trames avec une couche de sécurité, telle que HTTPS.

4.4 Protections matérielles

La sécurité est d'autant plus importante lorsqu'un produit est commercialisé. La question n'est pas de savoir si le système est infaillible, car chaque système est

1. http://jwt.io
2. https://en.wikipedia.org/wiki/Unix_time

accessible d'une façon ou d'une autre, ce n'est souvent qu'une question de temps et de détermination. La réelle question est de savoir quels sont les dangers une fois qu'une personne malveillante a accès à un périphérique et quelles sont les meilleures protections disponibles contre les risques engendrés par ces attaques. Il y a toutefois des mécanismes très simples et peu couteux qui peuvent rendre le travail de personnes malveillantes beaucoup plus difficile.

4.4.1 Détection d'ouverture de boitier

Il est possible de détecter si un périphérique a été ouvert par une personne, par exemple avec le placement d'un capteur de luminosité à l'intérieur d'un boitier ou de tout autre mécanisme de détection de mouvement. On peut imaginer envoyer un message à la centrale pour notifier un problème et demander à une personne physique de se déplacer pour analyser l'état de la situation. Outrepasser ce mécanisme est néanmoins très simple ; il est par exemple possible de se placer dans un milieu qui empêche l'accès au réseau de communications (ex. cage de faraday ou autre). Ce type de mécanismes sont toutefois inutiles si la batterie du dispositif est vide, car les capteurs ne peuvent rien détecter. Il faut ainsi éviter de stocker des informations critiques dans des mémoires non volatiles mal protégées.

4.4.2 Protection de la mémoire flash

Beaucoup de microcontrôleurs proposent des options pour verrouiller les accès à la mémoire flash bloquant ainsi la possibilité de relire le code et de le décompiler pour tenter de trouver des failles dans les implémentations. Le KW41Z de NXP propose ce mode de sécurité. Celui-ci est décrit dans la note d'application AN4507 intitulée *Using the Kinetis Security and Flash Protection Features*[1]. Voici quelques exemples d'options qui peuvent être activées pour protéger la mémoire flash :

1. *Flash Security* (SEC) : ce mode par défaut est désactivé. La mémoire peut donc être lue et écrite à volonté.
2. *Mass Erase Enable* (MEEN) : cela permet d'empêcher que le dispositif ne soit reprogrammable à l'aide d'une nouvelle application.
3. *Freescale Failure Analysis Code* (FSLACC) : ce mode est désactivé par défaut. S'il est activé et que le microcontrôleur est retourné à l'usine de Freescale à cause d'un problème lié à la construction du microcontrôleur, Freescale aura la possibilité d'accéder au contenu de la Flash, et ce, même si la sécurité est activée. La note d'application de Freescale ne fournit pas d'informations supplémentaires sur ce mode.
4. *Backdoor Key Security* : si l'application a besoin d'écrire dans la mémoire flash, il est possible de forcer la création d'une *backdoor key* offrant la possibilité de désactiver temporairement la sécurité de la mémoire. Ce type d'accès est nécessaire, par exemple, lorsque le microcontrôleur doit mettre à jour son *firmware*. Lorsque le microcontrôleur active cette option, une clé est générée et celle-ci doit être sauvegardée à l'extérieur du microcontrôleur. Lorsque la sécurité devra être désactivée, cette clé sera demandée à l'utilisateur qui devra la fournir à l'aide d'une interface de communication (UART, SPI, I2C, etc.).

Selon le type de sécurité choisie, le JTAG reste accessible, mais uniquement les registres qui lui sont propres en revanche. Si la lecture de la mémoire flash est désactivée, le JTAG ne pourra pas y avoir accès.

1. https://www.nxp.com/docs/en/application-note/AN4507.pdf

4.4.3 Coffres-forts matériels

Le CMWX1ZZABZ de Murata dispose d'un coffre fort matériel (cf. Section 5.3.3.1) pouvant stocker des informations et être utilisé comme coprocesseur cryptographie. Ce circuit intégré est nommé *STSAFE-A100.*

Voici une liste des quelques fonctionnalités mises en avant par le fabricant [1] :

- Service de vérification de signature avec possibilité de contrôler un *secure boot* et mise à jour du firmware signée.
- Cryptographie asymétrique avancée :
  - *Elliptic curve cryptography* (ECC) avec NIST ou Brainpool 256-bit et 384-bit. *curves* ;
  - *Elliptic curve digital signature algorithm* (ECDSA) avec SHA-256 et SHA-384 pour la génération et vérification de signatures numériques ;
  - *Elliptic curve Diffie-Hellman* (ECDH) pour l'établissement des clés.
- Cryptographie symétrique avancée :
  - *Key wrapping* et *unwrapping* en utilisant AES-128/AES-256 ;
  - Création de canaux sécurisés basés sur AES-128.
- Système d'exploitation sécurisé avec protection contre les attaques logicielles et physiques.

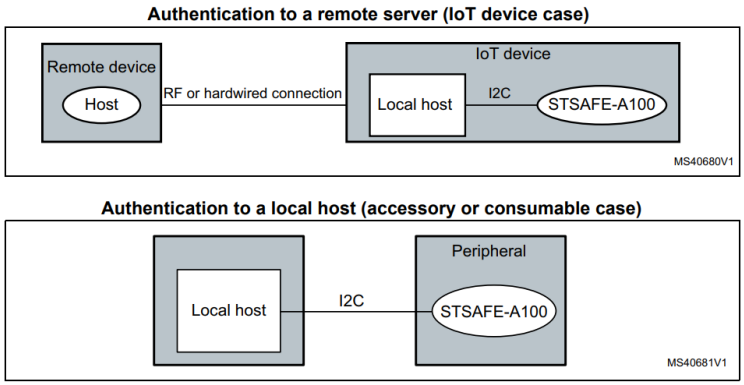


FIGURE 4.20 – Authentifications possibles avec un STSAFE

La communication avec ce module s'effectue à l'aide d'une interface I2C. Malheureusement, la documentation de ce coffre n'est pas disponible en ligne, à l'exception d'un *databrief*[2] de 15 pages expliquant les caractéristiques principales du circuit et les descriptions matérielles du boitier électronique. Il faut directement s'adresser à STMicroelectronics pour recevoir informations précises sur le module et la programmation (registres I2C). Les fonctionnalités listées par le fabricant demeurent toutefois vagues et, à défaut de documentation, il est difficile d'en apprendre davantage. Cette même *databrief*, présente le schéma illustré en Figure 4.20 afin d'expliquer les deux types d'authentification pouvant être utilisé avec ce module. On constate ainsi que ce module peut également authentifier une connexion externe sans fil. Celui-ci peut, par exemple, être couplé à du Bluetooth pour garantir l'identité du maitre initiant la communication sans utiliser de *passkey*. Dans le cadre du présent projet, la première option de ce schéma qui est celle qui présente un réel intérêt.

1. http://www.st.com/en/secure-mcus/stsafe-a100.html
2. http://www.st.com/resource/en/data_brief/stsafe-a100.pdf

Chapitre 5

Conception matérielle

Ce chapitre présente toute l'implémentation de la carte électronique qui a été réalisée dans le cadre de ce projet. La description des composants, de même que le choix de ceux-ci y sont évoqués.

5.1 Introduction

Le présent projet avait pour objectif premier la réalisation d'une carte électronique. Les différents composants à placer sur cette carte ont été choisis en fonction des besoins de certains *use cases* exposés à la DGSI. Voici une liste contenant quelques exemples :

- Un traqueur GPS transmettant la position par LoRa ;
- Un détecteur de vol avec détection de mouvement transmettant ensuite la position par LoRa une fois l'objet en mouvement ;
- Un scanneur Bluetooth permettant de détecter un certain nombre de *beacons* autour d'une borne ;
- Une station météorologique avec la possibilité de mesurer la qualité de l'air ambiant.

Ces exemples ont ainsi permis la mise en place d'une liste de périphériques indispensables à la carte.

Un nouveau concept de partage de clé a également été imaginé, avec la possibilité de changer l'App Key LoRaWAN présente sur le périphérique via l'utilisation d'un simple smartphone. Il fallait dès lors choisir une méthode d'accès au périphérique. La quantité de données transférées étant faible (de l'ordre de quelques centaines de bytes par secondes, au grand maximum) et le dispositif devant être utilisable sur batterie, le Bluetooth Low Energy a été retenu. Le Bluetooth est une technologie qui avait déjà été retenue pour l'implémentation du scanneur de périphériques.

Tous les documents liés au circuit électronique (schémas, modèles 3D et footprints) sont disponibles dans l'Annexe E.

5.2 Architecture

Pour une plus grande flexibilité de la carte de développement, il est judicieux d'utiliser deux processeurs sur la carte électronique, ce qui implique deux implémentations possibles. Les Figure 5.1 et Figure 5.2 illustrent ces deux options. La première consiste à utilise un processeur LoRa comme processeur principal du système ; soit un processeur sur lequel la communication LoRa est gérée en plus de la gestion des différents périphériques de la carte. Par exemple, un GPS, un accéléromètre, etc. Le

processeur gérant le Bluetooth serait ainsi un simple périphérique pour le processeur LoRa. Une API doit être mise en place pour faciliter l'échange d'informations entre les deux. La deuxième possibilité est d'inverser les rôles des processeurs. Le processeur principal serait celui qui gère les protocoles en 2.4 GHz et communique périodiquement avec le processeur LoRa afin d'envoyer et recevoir des données provenant du réseau.

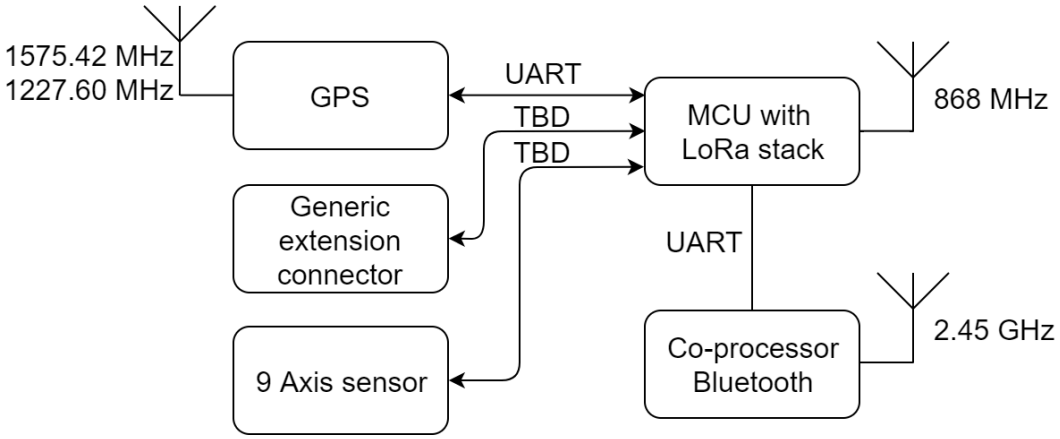


Figure 5.1 – Architecture option numéro 1

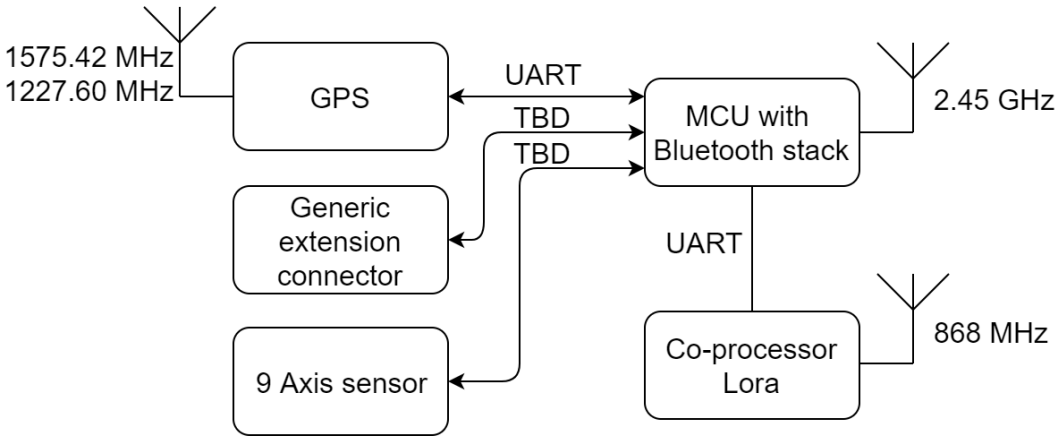


Figure 5.2 – Architecture option numéro 2

L'architecture choisie est l'option numéro 2. Nous avons donc tous les périphériques qui sont connectés au processeur s'occupant du Bluetooth. Le processeur avec l'interface LoRa, lui ne s'occupe, quant à lui, que de communiquer les évènements de réception de données et de transférer les messages provenant depuis le processeur Bluetooth. Il est également plus simple de faire une API pour le LoRa, le nombre de données et paramètres configurables étant moins important que dans le cas des communications Bluetooth. Si on souhaite faire une API entre les deux processeurs, il est ainsi nécessaire de tenir compte de la complexité de ceux-ci. Les accès LoRa sont également plus ponctuels. Le nombre de paquets envoyés par heures est très faible, en comparaison avec ce qui est réalisable en Bluetooth. On peut ainsi faire en sorte que le processeur soit en mode très basse consommation en attendant une commande de la part du processeur principal ou la réception d'un paquet LoRa sur son antenne.

5.3 Choix des composants et leurs caractéristiques

Cette section a pour but d'une part d'expliquer les différents choix effectués pour les composants présents sur la DevBox et, d'autre part, de présenter les caractéristiques principales des composants les plus importants.

5.3.1 Microcontrôleur

Le microcontrôleur est le cœur de ce projet ; c'est dans celui-ci que la plus grande partie du développement va se produire. Il est dès lors important de sélectionner le composant idéal pour notre application, car une fois la sélection faite et les cartes électroniques réalisées, il est impossible de le changer sans générer des changements conséquents. Pour éviter cela, une étude des divers microcontrôleurs disponibles sur le marché a été réalisée.

5.3.1.1 Critères de comparaison

Le microcontrôleur a dû être choisi judicieusement en fonction du cahier des charges. Un comparatif, visible sur le Tableau 5.1, a donc été effectué. L'un des premiers points retenus est le type de protocoles qui sont supportés par le microcontrôleur. Le Bluetooth Low Energy étant un prérequis, il nous a été possible de procéder à un premier filtre, ce qui explique également que tous les microcontrôleurs affichés disposent de cette fonctionnalité. Néanmoins, le Bluetooth Low Energy s'accompagne de la norme 4.2, annoncée le 2 décembre 2014 [57], et de la norme 5.0, annoncée quant à elle le 16 juin 2016. Cette dernière n'était malheureusement pas encore parfaitement supportée par tous les fabricants de microcontrôleurs. Comme précédemment expliqué en Section 2.5, une simple mise à jour logicielle permet de promouvoir un microcontrôleur en Bluetooth 5.0, du moins, en garantissant les fonctionnalités primordiales du protocole. Les microcontrôleurs affichés dans le tableau disposaient, en date du 1er octobre 2017, uniquement des fonctionnalités listées, ce qui ne signifie toutefois pas qu'elles ne peuvent pas s'agrandir avec le temps.

Après le Bluetooth, le deuxième type de protocole est le IEEE 802.15.4 [23]. Ce standard IEEE n'est pas un protocole en tant que tel, mais un modèle qui est respecté par plusieurs protocoles. Parmi les protocoles qui utilisent ce standard, on retrouve le ZigBee, WirelessHART, MiWi, SNAP, ou encore le Thread. Ce standard est de plus en plus utilisé pour la domotique. En effet, il permet de facilement créer des réseaux maillés permettant ainsi une grande couverture de l'habitation.

Enfin, un moyen de communication en dessous d'un gigahertz était également envisageable. Si le processeur supporte cela, ce peut être un avantage pour la réalisation de futurs *use cases*. Néanmoins, la communication dans cette plage de fréquence a quelques désavantages, en commençant par la taille de l'antenne qu'il faut utiliser. La consommation du processeur est également affectée, pouvant atteindre jusqu'à 100 mA (cf. EFR32M datasheet). LoRa entre dans cette catégorie, mais seuls les processeurs fournis par Semtech peuvent implémenter la modulation. Ils ne sont, à l'heure actuelle, pas assez développés pour pouvoir être utilisés comme unique processeur gérant du 2.4 GHz avec en parallèle une interface LoRa (868 ou 433 MHz).

On a ensuite l'aspect consommation lors de l'émission qui entre en jeu. Afin d'être le plus proche possible en termes de comparaison entre les différents modèles, la consommation en émission a été retenue pour la fréquence de 2.4 GHz avec une

puissance de 0 dBm en sortie. Plus cette consommation est faible, plus notre système aura la possibilité d'être autonome grâce à sa batterie. Finalement, les deux derniers points à retenir sont la taille des mémoires (RAM et Flash), qui selon leur taille nous permettent de faire un programme plus complexe, puis le coût à l'unité du microcontrôleur. Les prix ont été sélectionnés pour une seule unité sur le site d'achat Mouser [1] Suisse.

Davantage d'éléments auraient pu être pris en compte pour la comparaison, à l'instar des périphériques disponibles ou la possibilité d'avoir une matrice d'assignation des pins, utile lorsque l'on souhaite avoir des cartes modulables. Ce qui est primordial, c'est d'avoir au minimum une interface de communication série (SPI, I2C ou UART), étant précisé que tous ces microcontrôleurs en ont plusieurs de disponibles. Il convient également de prendre en considération la consommation du microcontrôleur, surtout dans les modes de sommeil. Celle-ci se trouve dans la même plage pour tous les microcontrôleurs listés, soit aux environs des 200 µA avec une vitesse du microcontrôleur réduite au minimum, tous les périphériques désactivés et la persistance des données en RAM. Pour sortir de ces modes, le réveil émane habituellement d'une interruption matérielle, de sorte à pouvoir continuer à exécuter le code une fois sorti de veille. Vu que la carte de développement finale sera équipée d'un GPS, lequel sera le point principal de la consommation, cette donnée n'a pas été intégrée dans le tableau, bien qu'elle soit observée sur chaque microcontrôleur.

5.3.1.2 Comparaisons et choix

Pour une comparaison, il est nécessaire d'analyser les microcontrôleurs listés dans le Tableau 5.1. Tout d'abord, le seul qui intègre tous les protocoles de communication est le Silicon Labs EFR32MG12P433F1024 [2]. En effet, celui-ci dispose de multiples interfaces et le support pour la dernière technologie Bluetooth. Il a tout pour son avantage excepté deux choses : son prix et sa consommation. Il est presque trois fois plus cher qu'un NRF52832, bien qu'au final ce prix soit compréhensif, puisque celui-ci supporte 2 protocoles supplémentaires. Vient ensuite la problématique de la consommation. Si l'on compare sa consommation à 2.4 GHz, elle est plus élevée que tous les microcontrôleurs présents sur le Tableau 5.1. Si le projet nécessitait absolument l'utilisation de tous ces protocoles en parallèle, ce microcontrôleur serait alors le meilleur choix. Cela étant, c'est surtout le Bluetooth qui présente un intérêt pour le projet.

Le CC2650 [3] du constructeur Texas Instruments est un microcontrôleur très utilisé dans le commerce, mais qui présente un défaut majeur, la taille de sa mémoire. En effet, ses 128 KB de flash et, surtout ses 20 kB de RAM, sont très limitatifs en termes de performances. Dans les besoins du projet, plusieurs tâches communiqueront en parallèle les unes avec les autres, avec des transferts de données qui engendreront des allocations mémoire diverses et conséquentes. Vient ensuite la création des données et l'utilisation de bibliothèques externes pour certains capteurs. Cela implique beaucoup de codes, en particulier si l'on se trouve dans une approche de prototypage où l'optimisation logicielle n'est pas forcément une priorité dès le départ. Le CC2650 et le CC2640R2F [4] présentent tous deux cette problématique ; ils sont équipés d'un

1. http://www.mouser.ch/
2. https://www.silabs.com/products/wireless/mesh-networking/efr32mg-mighty-gecko-zigbee-thread-soc/device.EFR32MG12P433F1024
3. http://www.ti.com/product/CC2650
4. http://www.ti.com/product/CC2640R2F

dual processeur, d'un M3 pour l'application et d'un M0 pour la gestion des communications Bluetooth. L'avantage du CC2640R2F est qu'il supporte également le BLE 5.0 au détriment de l'IEEE 802.15.4. TI a d'ores et déjà annoncé qu'il ne prévoit pas de développement pour le BLE 5.0 sur le CC2650 [58].

Il reste donc quatre microcontrôleurs qui semblent intéressants dans cette liste. Deux du fabricant NXP et deux du fabricant Nordic Semiconductor. Parmi ceux-ci, il y a le NRF52840, soit le microcontrôleur parfait pour notre application. Il est dans la même gamme de prix que ses concurrents et possède deux très grandes mémoires (flash et RAM). Il a comme avantage de supporter le Bluetooth et l'IEEE 802.15.4. Il est équipé d'un Cortex M4F à 64 MHz, ce qui fait de lui le processeur le plus performant parmi les sept du Tableau 5.1 (à égalité avec le NRF52832). Sa consommation n'étant pas encore connue au démarrage du projet, celle-ci n'a pas été spécifiée dans le tableau. Malheureusement, lorsque ce projet a commencé, le NRF52840 n'était disponible à la vente chez aucun fournisseur. La documentation du fabricant ne se résumait, à l'époque, qu'à une simple *datasheet* minimaliste, sans aucun guide utilisateur complet. À l'heure de la rédaction de ce rapport (janvier 2018), une version BGA 125 pins est disponible, malheureusement avec quelques mois de retard pour notre application. Malgré le fait que ce microcontrôleur soit en BGA, il s'agit là certainement de l'un des meilleurs choix pour une application Bluetooth. La version NRF52832 a des caractéristiques identiques au NRF52840, excepté le fait qu'il ne supporte que du Bluetooth Low Energy.

NXP offre également plusieurs microcontrôleurs dans la gamme de fréquences 2.4GHz qui sont orientés basse consommation. Les deux plus intéressants sont le QN908x et le KW41Z. Le KW41Z est équipé d'un CPU Cortex-M0+ de ARM alors que le QN908x est quant à lui pourvu d'un M4F. Malgré un M0+, le KW41Z a comme atout le fait de pouvoir supporter la spécification IEEE 802.15.4 et, surtout, le protocole Thread précédemment présenté dans la Section 2.7.2. Ces deux protocoles peuvent être utilisés en parallèle, ouvrant ainsi la possibilité de créer des ponts entre ces deux mondes.

En somme, le choix s'articule autour du NRF52832 et du KW41Z. Finalement, le microcontrôleur retenu a été le KW41Z de NXP. Celui-ci a été initialement développé par l'entreprise Freescale, qui depuis le développement de ce produit a été rachetée par NXP. Un argument intéressant est le fait que le KW41Z puisse être connecté à la fois un réseau Bluetooth LE et à un réseau IEEE 802.15.4. Sa connectivité à un réseau Thread est également un avantage si l'on souhaite utiliser la carte dans un milieu plus *SmartHome* que *SmartCities*. Bien qu'il n'utilise qu'un Cortex-M0 de ARM, pour le type d'application que l'on souhaite réaliser, la puissance de calcul n'est pas le prérequis le plus important. C'est donc le KW41Z qui a été retenu pour ce projet.

Table 5.1 – Comparaison des microcontrôleurs envisagés pour le projet

Manufacturer	**Model**	**CPU**	**BLE 4.2**	**BLE 5.0**	**IEEE 802.15.4**	**Sub 1GHz**	**Receive Current**	**Transmit Current (@ 2.4Ghz, @ 0dBm)**	**Flash/ SRAM**	**Disponi-bility**	**Price**
NXP	KW41Z	Cortex M0+ @ 48 MHz	Yes	No	Yes	No	6.8 mA	6.1 mA	512 kB / 128 kB	Yes	6.34 CHF
NXP	QN908x	Cortex M4F @ 32 MHz	Yes	Yes	No	No	3.4 mA	3.4 mA	512 kB / 128 kB	Yes	5.87 CHF
Texas Instrument	CC2650	Cortex M3 @ 48 MHz + M0 for RF	Yes	No	Yes	No	6.2 mA	6.8 mA	128 kB / 20 kB	Yes	6.34 CHF
Texas Instrument	CC2640R2F	Cortex M3 @ 48 MHz + M0 for RF	Yes	Yes	No	No	6.1 mA	7.0 mA	128 kB / 20 kB	Yes	6.34 CHF
Nordic Semiconductor	NRF52832	Cortex M4F @ 64 MHz	Yes	Yes	No	No	5.4 mA	5.3 mA	512 kB / 64 kB	Yes	5.51 CHF
Nordic Semiconductor	NRF52840	Cortex M4F @ 64 MHz	Yes	Yes	Yes	No	N/A	N/A	1MB / 256 kB	No	N/A
Silicon Labs	EFR32M G12P433 F1024	Cortex M4F @ 38 MHz	Yes	Yes	Yes	Yes	9.3 mA	9.5 mA	1MB / 256 kB	Yes	13.5 CHF

5.3.1.3 NXP Kinetis KW41Z

La Figure 5.3 illustre le contenu du KW41Z avec tous ses composants internes. On remarque des éléments intéressants dans la section *security*, soit un support de chiffrement AES-128 et la possibilité de générer des nombres aléatoires avec un *True Random Number Generator* (TRNG). Ces deux éléments ne visent pas un but précis dans le cadre des applications présentées dans ce projet, mais si l'utilisateur souhaite un grand niveau de sécurité, il a la possibilité d'accéder à ces deux éléments.

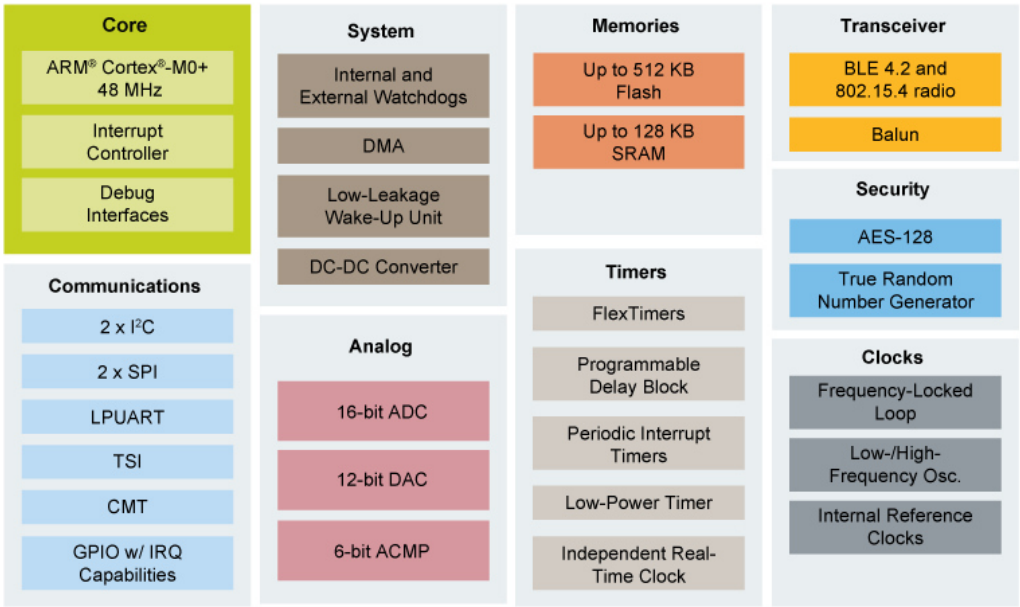


FIGURE 5.3 – Diagramme bloc interne du KW41Z

Une fois le microcontrôleur choisi, il était nécessaire de déterminer, les types d'interfaces allant être utilisées pour les différentes communications sur la DevBox. On peut voir sur Figure 5.3 la présence de deux interfaces I2C, de deux SPI et d'une LPUART. Celle en UART a été réservée pour la communication entre le microcontrôleur Bluetooth et le coprocesseur LoRa. L'UART offre l'implémentation la plus versatile dans ce type de communication. Il peut y avoir des messages asynchrones provenant du coprocesseur selon le type de protocole de communication que l'utilisateur souhaite utiliser entre les deux éléments. Des interfaces SPI et I2C sont utilisées par les composants inclus sur la carte électronique. Cette même interface SPI est partagée avec un connecteur PMOD installé sur la DevBox. Une deuxième interface I2C est quant à elle réservée au port PMOD. Pour une vue d'ensemble des interfaces de communication et leurs connexions sur la carte DevBox, il est conseillé de visualiser le schéma Altium en Annexe E, sur le schéma nommé `Top View`.

NXP propose deux cartes de développement avec le KW41Z visibles sur la Figure 5.4. La première est un simple dongle USB avec quelques boutons et une LED qui peut s'avérer utile pour faire des tests, par exemple, si on a besoin de scanner des paquets Bluetooth Low Energy. La FRDM-KW41Z est beaucoup plus complète. Elle propose en premier lieu des connecteurs compatibles avec les cartes d'extension Arduino. Elle contient en outre une meilleure antenne sur le PCB avec la possibilité de faciliter la connexion d'une antenne externe sur le connecteur prévu à cet effet.

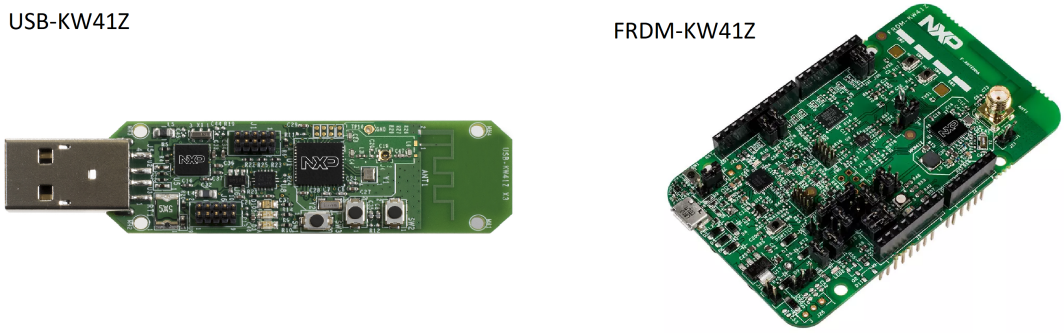


FIGURE 5.4 – Cartes de développement pour KW41Z

La carte est également équipée d'une vingtaine de *jumpers* qui offrent la possibilité de tester les différents modes de consommation du circuit, ainsi que plusieurs confirmations des sorties. Elle possède également une mémoire flash externe qui peut être utilisée pour les mises à jour OTAA, ainsi qu'un accéléromètre/gyroscope via l'intégration d'un FXOS8700CQ. Cette deuxième carte a été utilisée dans le cadre de ce projet pour avoir une première approche avec les outils de programmation et ainsi pouvoir commencer à programmer parallèlement à la réalisation du PCB.

5.3.2 Gestion de la batterie

Le système doit pouvoir être autonome. Afin de garantir cela, une batterie peut être connectée. La capacité de cette batterie reste au choix de l'utilisateur. L'utilisateur peut recharger celle-ci à l'aide d'un connecteur micro-USB présent sur la carte. L'aspect autonomie peut être encore plus poussé en connectant un panneau solaire directement sur la carte. Celui-ci a son propre connecteur dédié en entrée. Ce panneau permet ainsi de directement charger la batterie qui fournira à son tour la tension pour le système.

Pour optimiser le rendement lors de l'utilisation d'un panneau solaire, un circuit intégré nommé BQ24210 [1] a été utilisé. Ce dernier s'occupe directement de la recharge de la batterie supportant en entrée jusqu'à 20V de tension. Il est capable de fournir jusqu'à 800 mA à la batterie, tout en protégeant celle-ci en cas de surtension. Il accepte également la possibilité de recharger avec une tension de 5V, permettant ainsi la recharge directement depuis une prise USB.

Une batterie est couplée au système. Celle-ci n'a pas de restriction de taille, elle doit simplement être de type Li-Ion en raison de la tension et de la courbe de charge fournie par le BQ24210. La batterie n'a pas besoin d'offrir un circuit de protection contre les surcharges ni les décharges profondes. Comme expliqué précédemment, le circuit BQ24210 interrompt la charge lorsque la batterie est pleine et protège la batterie des décharges profondes.

L'ensemble de la carte est alimentée en 3.3V à l'aide d'un régulateur TPS62291. Celui-ci peut fournir un courant maximum de 1A. Ce haut courant est une précaution selon le type de cartes d'extensions que l'utilisateur souhaite ajouter à la DevBox. Il dispose d'un courant typique de fonctionnement très faible de 15 µA, pouvant ainsi consommer un minimum si l'on souhaite mettre toute la carte en basse consommation.

1. http://www.ti.com/product/BQ24210

5.3.3 Coprocesseur LoRa

La communication via LoRa est uniquement possible en utilisant les circuits proposés par l'entreprise Semtech, laquelle propose une gamme de circuits qui est visible à l'adresse suivante :

http://www.semtech.com/wireless-rf/lora.html

Tous les circuits intégrés proposés sur le site de Semtech sont des périphériques qu'il faut ensuite relier avec un microcontrôleur via une communication SPI. Afin de pallier à cela, plusieurs fabricants ont développé des circuits intégrés composés d'un processeur couplé à un module de Semtech. Cela permet ainsi d'économiser du temps de conception. En effet, le plus difficile dans l'intégration de ces modules est la gestion de la partie radiofréquence. Celle-ci nécessite souvent des composants spécifiques qui supportent des fréquences plus élevées que des composants traditionnels. C'est le cas des condensateurs et inductances utilisées, ceux-ci devant être indiqués par le fabricant comme étant utilisables pour des fréquences supérieures à plusieurs gigahertz. Le plus souvent, les fabricants proposent des gammes complètes de ces composants pour ces cas d'utilisations, à l'instar des fabricants Murata [1] ou Vishay [2].

Specifications	SX1272	SX1276
Frequency bands of operation	850 MHz to 1 GHz	169 MHz, 433 MHz, 850 MHz to 1 GHz
Programmable Bandwidth	125 KHz , 250 KHz, 500 KHz	7.8 KHz, 10.4 KHz, 15.6 KHz, 20.8 KHz, 31.2 KHz, 41.7 KHz, 62.5 KHz. 125 KHz, 250 KHz, 500 KHz
Receiver Sensitivity	Good (down up to -137dBm)	Better than SX1272, (down up to -148 dBm)
Max. Link Budget	157 dB	168 dB
IIP3	-12.5 dBm	-11 dBm
Rx current	10 mA, 100nA register retention	9.9 mA, 200nA register retention
Modulation types supported	GFSK, FSK, MSK, GMSK, LoRa™ as well as OOK.	Same as SX1272
Dynamic Range (RSSI)	127 dB	127 dB
Temperature sensor and low battery indication	YES	YES
packet engine size	256 bytes	256 bytes
Blocking Immunity	89 dB	Excellent
Bit Synchronizer	YES	YES
Preamble detection	YES	YES

FIGURE 5.5 – Comparatif SX1272 et SX1276

Tous les modules sont équipés d'un processeur connecté via SPI avec un circuit de Semtech. Parmi ces circuits se trouvent le SX1272 et le SX1276. Un comparatif sur le

1. https://www.murata.com/tool/netlist/mlcc
2. https://www.vishay.com/capacitors/high-frequency/

site `rfwireless-world`[1] offre une vue d'ensemble des différences entre ces deux circuits. La Figure 5.5 illustre cette comparaison. Ils sont similaires, mais le SX1276 offre de meilleures performances. En effet, on constate que le SX1276 propose davantage de bandes fréquence ainsi qu'un meilleur *link budget* (correspondant à la puissance de réception[2]). Le SX1272 ne propose que la fréquence 868 et 915 MHz de la bande de fréquence LoRa, soit celle respectivement pour l'Europe et l'Amérique du Nord respectivement. L'Asie avec ses 433 MHz n'est donc pas supportée. Le SX1276 est actuellement l'une des meilleures options disponibles par Semtech pour connecter un nœud à un réseau LoRaWAN. Néanmoins, pour notre application, le SX1272 est parfaitement suffisant.

En termes de consommation, les deux circuits sont similaires. Preuve en est la comparaison des données présentes dans leurs *datasheets* respectives (cf. Figure 5.6). Dans le cadre d'applications low power, les consommations en veille et au repos sont les plus importantes. Celles-ci sont de 0.1 ou 0.2 µA pour le mode *sleep* et de 1.5 µA pour le mode *idle*. Ils constituent des circuits parfaits pour des applications basse consommation. Qui plus est, si on émet à la puissance maximale de 20 dBm, la consommation peut monter à 128 mA dans le cas du SX1272, constat qui doit être pris en compte lors du dimensionnement de l'alimentation du système.

		SX1276			SX1272			
Description	Conditions	Min	Typ	Max	Min	Typ	Max	Unit
Supply current in Sleep mode		-	0.2	1	-	0.1	1	uA
Supply current in Idle mode	RC oscillator enabled	-	1.5	-	-	1.5	-	uA
Supply current in Standby mode	Crystal oscillator enabled	-	1.6	1.8	-	1.4	1.6	mA
Supply current in Synthesizer mode	FSRx	-	5.8	-	-	4.5	-	mA
Supply current in Receive mode	*LnaBoost* Off, band 1 *LnaBoost* On, band 1 Bands 2&3	- - -	10.8 11.5 12.0	- - -	- -	10.5 11.2	- -	mA
Supply current in Transmit mode with impedance matching	RFOP = +20 dBm, on PA_BOOST RFOP = +17 dBm, on PA_BOOST RFOP = +13 dBm, on RFO_LF/HF pin RFOP = + 7 dBm, on RFO_LF/HF pin	- - - -	120 87 29 20	- - - -	- - - -	125 90 28 18	- - - -	mA mA mA mA

FIGURE 5.6 – Comparaison de la consommation de courant entre le sx1272 et sx1278

5.3.3.1 Murata CMWX1ZZABZ-078

FIGURE 5.7 – Module LoRa CMWX1ZZABZ-078

1. http://www.rfwireless-world.com/Terminology/SX1272-vs-SX1276-Semtech-LoRa-RF-Transceivers.html
2. https://en.wikipedia.org/wiki/Link_budget

Le célèbre fabricant de semiconducteurs Murata [1] a commercialisé, en juillet 2016, un module LoRa intégrant un SX1276, ainsi qu'un processeur de STMicroelectronics [59]. Le module est visible sur la Figure 5.7. Murata est renommé pour ses composants passifs de très haute précision, comme ses résistances, condensateurs ou inductances. Comme expliqué dans la section précédente, c'est toujours un challenge d'optimiser des circuits lorsque l'on utilise de hautes fréquences. Murata a développé ce circuit en y ajoutant ses composants afin d'offrir les meilleures performances en radio fréquence tout en minimisant l'espace occupé sur un PCB par le module.

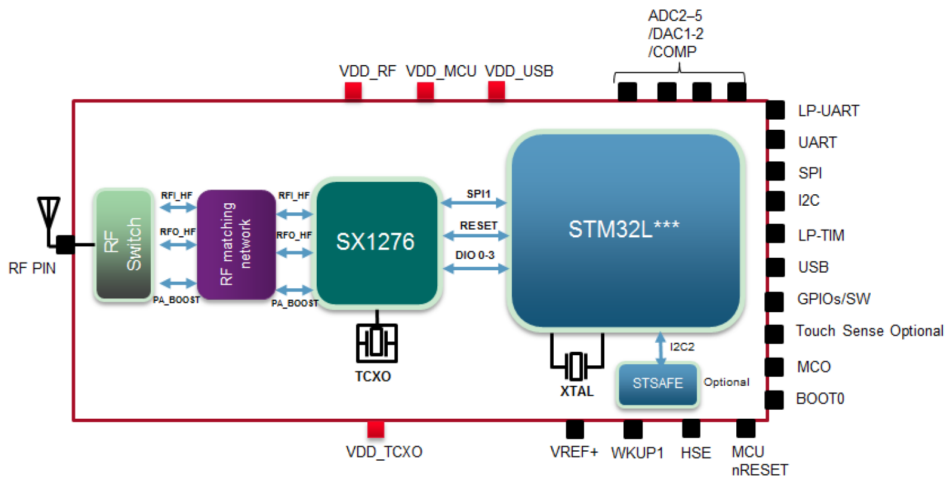


FIGURE 5.8 – Contenu du module LoRa CMWX1ZZABZ-078

La Figure 5.8 illustre le contenu interne du module. On peut y voir un processeur ainsi que le SX1276. Le point fort de ce module est toutefois l'électronique placée entre le SX1276 et l'antenne. En effet, c'est la partie la plus difficile à réaliser lorsque l'on fait du routage haute fréquence, puisqu'il faut adapter les impédances et le dimensionnement des composants. Il existe en réalité deux modèles du CMWX1ZZABZ, la version 078 et la 091. La différence entre les deux est le processeur ST qui est placé en interne. Le 078 contient un STM32L082 alors que le 091 est équipé du STM32L072. C'est de là d'où proviennent les astérisques sur la Figure 5.8 avec le bloc bleu représentant le processeur. Cependant, à l'heure actuelle, le CMWX1ZZABZ-078 est le seul disponible sur le marché, aucun revendeur ne propose l'autre alternative. La Figure 5.8 nous indique également la présence d'un composant nommé Section 4.4.3. Son nom complet est *State-of-the-art security for peripherals and IoT devices*. Celui-ci est développé par STMicroelectronics et propose plusieurs mécaniques d'authentification, afin de garantir que le périphérique fournissant des données à un client n'a pas été usurpé par une entité tierce. Ce périphérique est exploré plus en détail dans la Section 4.4.3.

Ses dimensions sont de 12.5 mm x 11.6 mm x 1.76 mm ; il est le plus petit des modules présentés ici. Ceci vient surtout du fait que Murata a utilisé sur son circuit des composants passifs avec des boitiers 0201 et 01005, ce qui lui a permis de créer le module le plus compact possible. C'est le seul des modules listé dans ce document qui propose un SX1276, contrairement à ses concurrents qui utilisent un SX1272.

1. https://www.murata.com/

La *datasheet* du module ne spécifie pas toutes les consommations en courant. La Figure 5.9 contient le peux d'informations qui sont disponibles sur ses consommations. On y retrouve la consommation maximale du composant lors de l'émission à 20dBm. La *datasheet* indique l'ajout de 3 mA en comparaison avec à la consommation seule du SX1276 (cf. Figure 5.6 pour les diverses consommations). Concernant la consommation en *sleep*, on peut observer celle du STM32L082 pour avoir une idée, puisqu'aucune estimation n'est fournie par Murata. La Figure 5.10 montre la consommation d'un STM32L082 en mode *low power* avec tous les périphériques éteints. Ce mode serait envisageable avec la DevBox, car le microcontrôleur Bluetooth est capable d'activer ce mode à l'aide, par exemple, d'un GPIO. On observe des consommations très faibles en fonction des horloges choisies. Dans l'ensemble, la consommation du processeur varie de 4.7 µA à 73 µA selon la fréquence désirée et de la température d'opération. Il faut ensuite ajouter les consommations du SX1276 dans un mode similaire. Les consommations du SX1276 ont été précédemment présentées dans la Figure 5.6, laquelle indiquait une consommation en mode *sleep* de 0.2 µA. Ce sont des courants tout à fait adaptés à des applications basse consommation.

FSK/OOK Receiver Specification

Symbol	Description	Conditions	Min.	Typ	Max	Unit
RFS_F_HF	LnaBoost is turned on	FDA = 5 kHz, BR = 4.8 kb/s		-117.5		dBm
IDDR	Supply current in Receive mode	LnaBoost Off, band 1		22		mA
		LnaBoost On, band 1		23		mA

FSK/OOK Transmitter Specification

Symbol	Description	Conditions	Min.	Typ	Max	Unit
IDDT	Supply current in Transmit mode with impedance matching	RFOP = +20 dBm, on PA_BOOST		128		mA
		RFOP = +17 dBm, on PA_BOOST		106		mA
		RFOP = +14 dBm, on RFO_HF pin		47		mA
		RFOP = + 7 dBm, on RFO_HF pin		34		mA

FIGURE 5.9 – Consommation en courant du module CMWX1ZZABZ-078

Current consumption in Low-power sleep mode

Symbol	Parameter	Condition			Typ	Max (1)	Unit
I_{DD} (LP Sleep)	Supply current in Low-power sleep mode	All peripherals OFF, code executed from Flash memory, V_{DD} from 1.65 to 3.6 V	MSI clock = 65 kHz, f_{HCLK}= 32 kHz, Flash memory OFF	T_A = −40 to 25°C	4,7	-	µA
			MSI clock = 65 kHz, f_{HCLK}= 32 kHz	T_A = −40 to 25°C	17	24	
				T_A = 85°C	19,5	30	
				T_A= 105°C	23	47	
				T_A= 125°C	32,5	70	
			MSI clock = 65 kHz, f_{HCLK}= 65 kHz	T_A= − 40 to 25°C	17	24	
				T_A= 85°C	20	31	
				T_A = 105°C	23,5	47	
				T_A = 125°C	32,5	70	
			MSI clock = 131kHz, f_{HCLK}= 131 kHz	T_A= − 40 to 25°C	19,5	27	
				T_A = 55°C	20,5	28	
				T_A = 85°C	22,5	33	
				T_A = 105°C	26	50	
				T_A= 125°C	35	73	

FIGURE 5.10 – Consommation en courant du microcontrôleur STM32L082

5.3.3.2 IMST iM880B

FIGURE 5.11 – Module LoRa iM880B du fabricant IMST

Le module iM880B, visible sur la Figure 5.11, est un produit de l'entreprise IMST. Celle-ci est renommée pour ses concentrateurs LoRa nommés iC880A [1] qui, dotés d'un prix très accessible, facilitant la mise en place d'une gateway LoRaWAN.

Sa dimension étant de 25 x 20 mm, il presque quatre fois plus grand que le module de Murata. Il dispose toutefois d'un processeur plus puissant que le Murata. En effet, il s'agit là de la gamme STM32L1, contrairement au STM32L0 du CMWX1ZZABZ. Le STM32L1 est basé sur un Cortex M3 cadencé à 16 MHz. Un processeur puissant n'a pas vraiment de sens dans ce projet, car le processeur LoRa restera le plus clair de son temps en veille en attendant des commandes du processeur principal.

La *datasheet* de ce module nous fournit plus d'informations sur sa consommation. Les consommations sur les multiples modes sont visibles sur la Figure 5.13. On a la même consommation en émission maximale que le Murata à 128 mA à +20

1. https://wireless-solutions.de/products/radiomodules/ic880a.html

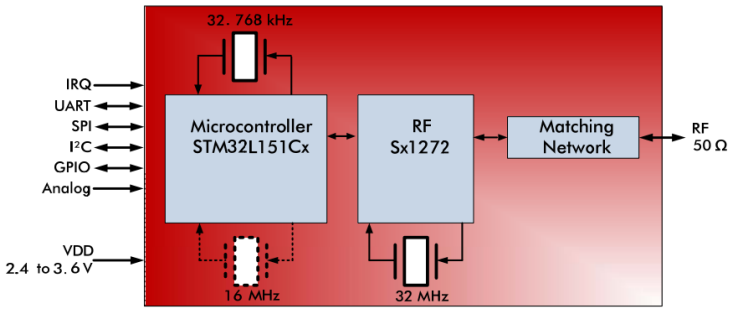


FIGURE 5.12 – Contenu du module LoRa iM880B

dBm. Cette *datasheet*, présente cette fois la consommation lorsque le périphérique est en mode *low power*. Celle-ci s'élève à 1.85 µA avec un RTC activé, ou 800 nA sans celui-ci. La consommation du STM32L082 est donc similaire à celle du Murata. Malgré l'utilisation d'un processeur plus puissant (STM32L1), la consommation est donc inférieure à celle du CMWX1ZZABZ (4.9 µA). Il faudrait examiner quel mode du microcontrôleur est activé dans cette configuration pour pouvoir faire une comparaison exacte entre les deux configurations. Dans le cas du Murata, il s'agissait du mode *sleep* du processeur, mais cette information n'est pas disponible pour le module IMST.

T = 25°C, VDD = 3.0 V (typ.) if nothing else stated

Parameter	Condition	Min	Typ.	Max	Unit
Supply Voltage (VDD)		2.4	3.0	3.6	V
Current Consumption Low Power Mode	RTC off		800		nA
	RTC on		1.85		µA
Current Consumption System IDLE	TRX idle mode, µC idle mode		5		mA
Current Consumption RECEIVE LoRa	TRX receive mode, µC sleep mode		11.2		mA
Current Consumption TRANSMIT (Note 3)	TRX transmit mode, µC sleep mode, all µC units off, max. RF power level		128		mA
MCU operation frequency			32		MHz
			32.768		kHz

FIGURE 5.13 – Consommation en courant du module iM880B

5.3.3.3 USI WM-SG-SM-42

Le fabricant chinois USI a annoncé la production, en décembre 2016, d'un module similaire à celui de Murata. Celui-ci est nommé WM-SG-SM-42 et est visible sur la Figure 5.14. Celui-ci a été adopté par STMicroelectronics pour la création d'un *shield* pouvant être placé sur des plateformes Arduino ou, directement, sur certaines de leur cartes de développement. Le contenu du module est illustré sur la Figure 5.15.

Néanmoins, lorsque ce projet a commencé (septembre 2017), très peu d'informations étaient disponibles sur ce module, la seule référence postée sur le site du fabricant étant le communiqué de presse suivant :

http://www.usish.com/english/pressroom_more.php?n_sn=23

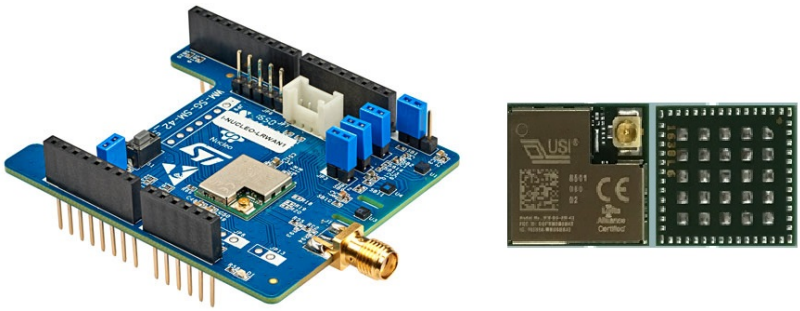


FIGURE 5.14 – *Shield* I-NUCLEO-LRWAN1 de ST et module USI WM-SG-SM-42

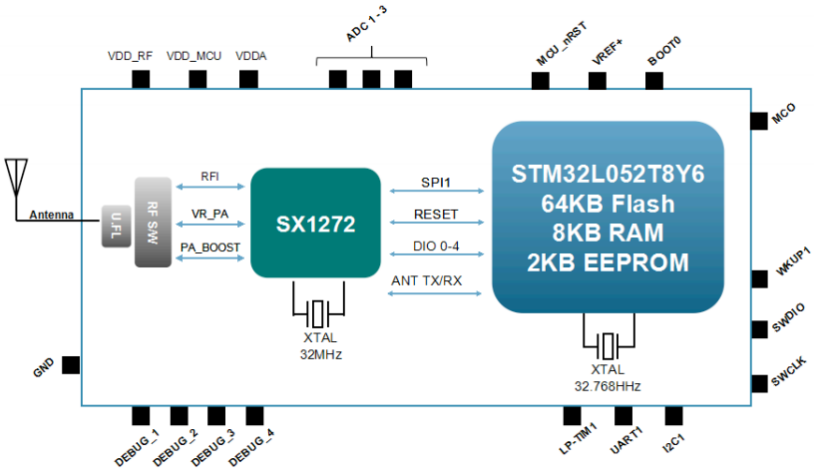


FIGURE 5.15 – Contenu du module WM-SG-SM-42

Il n'y avait pas de description du produit ni de *datasheet*, et le site n'était pas encore entièrement traduit en anglais. Il semble que tout cela se soit amélioré à la date de la rédaction de ce rapport (janvier 2018). Une page du produit est maintenant disponible [1], de même qu'un PDF contenant une *datasheet* du module [2]. Cependant, un problème majeur est toujours présent : ce module ne semble être disponible chez aucun revendeur ; même sur des sites de vente en ligne comme Alibaba ou Aliexpress. Contrairement au module de Murata, qui est très facilement accessible. En revanche, le *shield* produit par STMicroelectronics, équipé du module USI, est bien commercialisé en Europe et disponible chez plusieurs fournisseurs [3].

Le module a une dimension de 12 mm x 13 mm, similaire au module de Murata. Il possède en outre un connecteur U.FL placé sur le module. Ceci est un avantage, puisqu'il permet d'économiser de la place sur le circuit si le but final est d'utiliser une antenne externe au PCB. Dans le cadre de ce projet, les deux options ont été retenues. L'utilisation de ce module aurait présenté un avantage en termes de gain de place.

1. http://www.usish.com/english/products_WM_SG_SM_42.php
2. http://www.usish.com/pdf/WM-SG-SM-42_Product\protect\kern+.1667em\relax%20SPEC.pdf
3. https://www.findchips.com/search/I-NUCLEO-LRWAN1

Mode	TYP.	Max.
Standby mode	7mA	10mA
RX	17mA	25mA

FIGURE 5.16 – Consommation en courant du module WM-SG-SM-42

En ce qui concerne la consommation, la *datasheet* reste très vague sur les informations. On peut voir uniquement les deux valeurs affichées sur la Figure 5.16 sur une configuration LoRa à 3.3V, 25°C et BW 125kHz. La consommation en mode *standby* est trop élevée pour qu'elle soit celle de la consommation définitive du circuit. Il est possible de le mettre en mode *low power*, mais la consommation de ce mode n'est pas documentée dans la *datasheet*. Les consommations en émissions sont listées dans la *datasheet* en fonction de la puissance d'émission. Elles varient de 47 mA à 126 mA à 868 MHz, avec des puissances d'émission allant de 5.6 à 19 dBm. Il faut être conscient que ces consommations ne sont présentes que lorsque l'utilisateur émet des paquets sur le réseau, sont durant quelques centaines de millisecondes au maximum. Pour la consommation en veille, on peut estimer que celle-ci est identique au Murata, puisque toute la gamme STM32L0 a des consommations similaires.

Comme pour les deux autres modules, la partie RF est intégrée dans le module, ce qui simplifie le développement matériel final. Il suffit de raccorder une antenne avec une impédance correcte de 50 Ohms.

5.3.3.4 Module choisi

Le module de Murata est celui qui a été retenu pour ce projet. Il a l'avantage d'être extrêmement compact et simple à intégrer. Il est également disponible chez tous les revendeurs en Europe, contrairement à son rival, celui du fabricant USI. Au niveau logiciel, plusieurs codes exemples sont directement fournis par ST sur leur site Internet, ce qui offre un support en cas de problèmes au fil du développement.

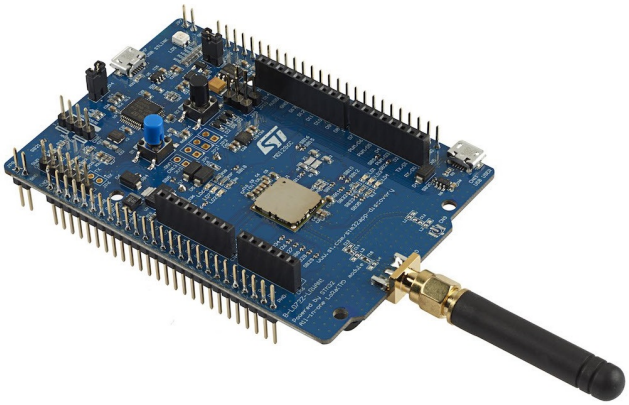


FIGURE 5.17 – Carte de développement ST B-L072Z-LRWAN1

La carte de développement B-L072Z-LRWAN1 de ST Microelectronics, visible sur la Figure 5.17, offre la possibilité directement commencer à travailler sur le matériel sans que la carte électronique ne soit finalisée. Une partie du code de cette carte peut ensuite être facilement adapté sur la DevBox.

5.3.4 Périphériques

La carte DevBox est équipée de certains périphériques capteurs de données. Cette sous-section a pour but de les énumérer et de présenter leurs principales caractéristiques. Le schéma contenant ces périphériques est disponible dans l'Annexe E de ce document. Ils sont principalement présentés dans les schémas nommés `Embedded Sensors`, `GPS` et `PMOD`.

5.3.4.1 LED

Chaque microcontrôleur a à sa disposition quatre LED. Une bleu, une verte, une rouge ainsi qu'une jaune. Ces LED sont contrôlables via des GPIOs avec un état bas pour les allumer. Elles sont programmables selon la volonté de l'utilisateur.

5.3.4.2 Buttons

Chaque microcontrôleur a à sa disposition 2 boutons poussoirs. Ceux-ci produisent un état bas sur les GPIOs lorsque le bouton est pressé. Ces boutons sont programmables selon la volonté de l'utilisateur.

5.3.4.3 Plateforme inertielle 9 axes

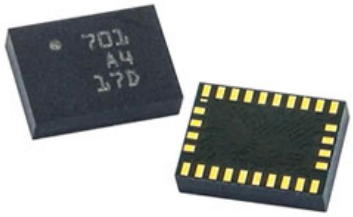


FIGURE 5.18 – Plateforme 9 axes BNO055

La DevBox dispose d'une plateforme inertielle 9 axes (14-bit accéléromètre, 16-bit gyroscope et 16-bit magnétomètre) intégrés dans un même circuit électronique. Il s'agit du BNO055 fabriqué par Bosch Sensortec, visible sur la Figure 5.18, dont la communication s'effectue via un bus I2C relié au processeur principal (KW41Z). L'adresse I2C est configurable en fonction de l'état de la pin COM3 du module. Deux résistances (R11 et R13, une seule doit être placée) permettent de choisir le dernier bit de l'adresse. Si la résistance R11 est placée, le périphérique comporte alors l'adresse 0x29. Si c'est la résistance R13, cette adresse est 0x28.

À l'intérieur de ce composant, on trouve un ARM Cortex-M0+ avec un *firmware* développé par Bosch, afin de pouvoir délocaliser le calcul de certaines données capturées. Cela permet, par exemple, de demander au composant de retourner la valeur de la gravité en m/s^2 ou de calculer de quaternions, réduisant ainsi le nombre d'opérations de lecture et surtout, économisant le temps de calcul du processeur principal sur un circuit embarqué. Le BNO055 a été créé avec la problématique de la consommation en tête. Dans son mode le plus bas, nommé *Suspend Mode*, il ne consomme que 40 uA.

Bosch Sensortec propose un pilote pour la plupart de ses composants et celui-ci ne fait pas exception. Cela permet d'accélérer le développement, ce qui est parfait dans le cadre de ce projet, puisque l'utilisation de l'accéléromètre n'est pas l'objectif principal.

5.3.4.4 Capteur environnemental et qualité de l'air ambiant

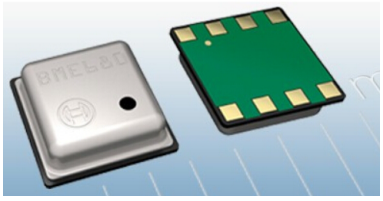


FIGURE 5.19 – Capteur environnemental et qualité de l'air BME680

Un capteur de qualité de l'air a également été intégré sur la DevBox. Il s'agit du BME680 de Bosch Sensortec visible sur la Figure 5.19. Celui-ci ne mesure pas seulement la qualité de l'air, il mesure également la pression ambiante, la température, l'humidité et le gaz. Pour la mesure du gaz, il s'agit des *Volatile Organic Compounds*[1] (VOC), avec lesquels ont évalue la qualité globale de l'air ambiant. La liste des VOC utilisés pour l'index de la qualité de l'air ainsi que leur taux de précision est disponible sur la Figure 5.20. Le capteur contient en interne une zone chauffante, laquelle s'active un bref instant pour pouvoir ensuite extraire la résistance de l'air. Cette résistance est la seule valeur fournie par ce capteur. Il faut ensuite faire des corrélations afin d'extraire les informations. La température, l'humidité et la pression sont elles directement retournées dans leurs unités respectives (°C, % et Pa).

Molar fraction	Compound	Production tolerance	Certified accuracy
5 ppm	Ethane	20 %	5 %
10 ppm	Isoprene /2-methyl-1,3 Butadiene	20 %	5 %
10 ppm	Ethanol	20 %	5 %
50 ppm	Acetone	20 %	5 %
15 ppm	Carbon Monoxide	10 %	2 %

FIGURE 5.20 – VOC mesurés par le BME680

Comme pour le BNO055, Bosch Sensortec met a disposition un driver en C[2]. Le driver est beaucoup plus sommaire que celui du BNO055, car Bosch propose également une bibliothèque nommée BSEC pour *Bosch Sensortec Environmental Cluster*[3]. Cette dernière n'est pas open source, il faut se contenter d'un binaire précompilé pour l'architecture utilisée.

5.3.4.5 GPS

La DevBox peut également être équipée d'un GPS au besoin. Celui-ci étant assez couteux, il ne sera pas forcément monté sur toutes les cartes. Le GPS peut être utilisé si l'on souhaite traquer un objet en déplacement avec une très grande précision (± 8m, 95 % du temps [60]). Il peut également être utilisé pour corréler la localisation obtenue via LoRa avec une position réelle. Cette géolocalisation par LoRa est expliquée dans l'introduction au LoRa de sous la Section 2.3.

Le choix du GPS, a été opéré parmi ceux du fabricant suisse U-blox, spécialisé dans les semiconducteurs avec communication sans fil. Ils ont un large choix en termes de périphériques GPS. Ils proposent des GPS assortis d'antennes externes ou d'antennes céramiques placées directement sur un module. Ces GPS ont également la possibilité

1. https://en.wikipedia.org/wiki/Volatile_organic_compound
2. https://github.com/BoschSensortec/BME680_driver
3. https://www.bosch-sensortec.com/bst/products/all_products/bsec

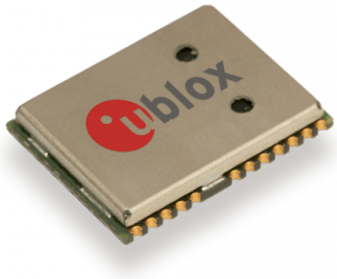


FIGURE 5.21 – GPS U-blox NEO-M8

d'incorporer un *low-noise amplifier* (LNA [1]) pour l'amplification du signal. On peut également y trouver un filtre de type *surface acoustic wave* (SAW [2]) pour améliorer le signal et ainsi améliorer la réception. Lorsque ces deux éléments sont directement intégrés dans le module, cela facilite considérablement la complexité de routage d'un circuit équipé d'un GPS. En effet, s'il est nécessaire de créer ces deux éléments à l'extérieur du GPS, il faut faire particulièrement attention au type de composants utilisés, car le GPS communique à l'aide de deux hautes fréquences de 1575.42 MHz et 1227.60 MHz [61]. Il y a également l'adaptation d'impédance sur la connexion avec l'antenne qui doit être correctement calculée en considérant l'épaisseur du circuit imprimé.

Le GPS recherché devait également être capable d'accepter la lecture par une interface de type I2C ou SPI, puisque ce sont les seules interfaces encore disponibles sur le KW41Z, puisque l'UART a été utilisé pour la communication entre le KW41Z et le module LoRa. Malheureusement, l'UART est l'interface la plus utilisée en GPS, donc certains GPS ne proposent que ce type de protocole de communication.

Model	Category				GNSS					Supply		Interfaces				Features									Grade		
	Standard Precision GNSS	High Precision GNSS	Dead Reckoning	Timing	GPS / QZSS	GLONASS	Galileo	BeiDou	Number of Concurrent GNSS	1.65 V – 3.6 V	2.7 V – 3.6 V	UART	USB	SPI	DDC (I²C compliant)	Programmable (Flash)	Data logging	Additional SAW	Additional LNA	RTC crystal	Oscillator	Built-in antenna	Built-in antenna supply and supervisor	Timepulse	Standard	Professional	Automotive
NEO-M8N	•				•	•	•	•	3		•	•	•	•	•	•	•	•	•	•	T			1			
NEO-M8Q	•				•	•	•	•	3		•	•	•	•	•			•	•	•	T			1			
NEO-M8M	•				•	•	•	•	3	•		•	•	•	•					•	C			1			

C = Crystal / T = TCXO

FIGURE 5.22 – Comparaison de la gamme de GPS U-blox NEO-M8

Le GPS retenu a été le NEO-M8 dont la gamme complète est visible sur la Figure 5.22. Celui-ci offre tout ce dont on a besoin, à savoir la possibilité d'une connexion avec une interface I2C ou SPI, ainsi que l'intégration d'un filtre SAW et d'un LNA. L'interface retenue pour le GPS a été le SPI, laquelle doit être partagée avec les composants qui sont connectés sur le connecteur PMOD (cf. Section 5.3.4.6). Sa taille est importante par rapport à celle des deux processeurs utilisés sur la carte (12.2 x 16 x 2.4 mm pour le NEO-M8 contre 12 x 13 mm pour le module LoRa). Il est entouré

1. https://en.wikipedia.org/wiki/Low-noise_amplifier
2. https://www.vectron.com/products/saw/gps.htm

d'un boitier métallique, visible sur la Figure 5.21, afin d'atténuer les perturbations externes. Cependant, il ne nécessite aucun périphérique externe, à l'exception d'un connecteur U.FL pour la connexion d'une antenne externe.

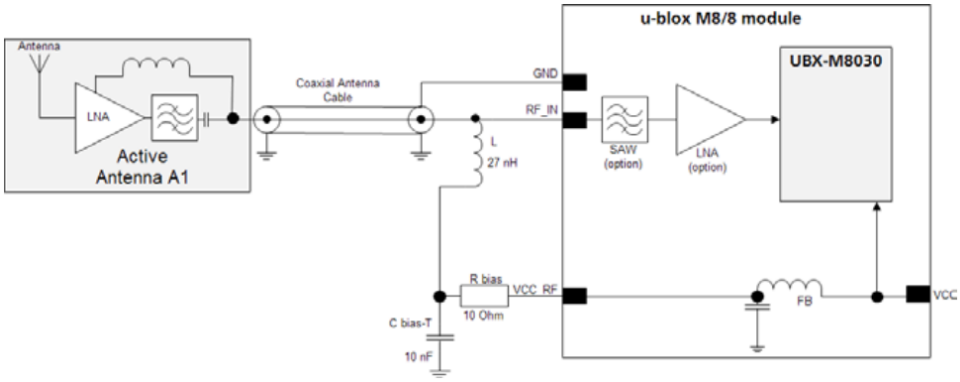


FIGURE 5.23 – Schéma de connexion d'un NEO-M8 avec une antenne active

Le GPS a été routé sur la DevBox pour pouvoir utiliser des antennes non actives. Si l'utilisateur souhaite utiliser des antennes actives, il est possible de connecter l'alimentation directement le signal RF en entrée de l'antenne. Ceci n'est pas la connexion recommandée, mais elle est fonctionnelle. Selon la *datasheet* du constructeur, si l'on souhaite convertir le signal RF actif, il faut normalement filtrer le signal d'alimentation afin d'éviter l'injection de bruit dans la radio. La Figure 5.23 illustre la connexion recommandée par U-blox. Le NEO-M8 propose également une sortie nommée *Time-pulse* qui génère une pulse à chaque seconde (l'intervalle est configurable). Cette pulsation peut être utilisée pour synchroniser une horloge sur le microcontrôleur, de même que pour savoir si le GPS est prêt à fournir un nouveau paquet.

Ce module GPS entre directement dans la gamme professionnelle. La gamme supérieure est l'*automotive* pour l'industrie des véhicules, mais il n'a pas été jugé nécessaire de prendre des composants de ce type pour le projet réalisé. Le prix du GPS s'élève à 9.24 euros à partir de 250 pièces achetées (prix visible en janvier 2018).

Parameter	Symbol	Module	Typ GPS & GLONASS	Typ GPS	Max	Units	Condition
Max. supply current [14]	Iccp	All			67	mA	
Average supply current [15, 16]	Icc Acquisition[17]	NEO-M8N	32	25		mA	Estimated at 3 V
		NEO-M8M	24	19		mA	
		NEO-M8Q	30	24		mA	
	Icc Tracking (Continuous mode)	NEO-M8N	30	23		mA	Estimated at 3 V
		NEO-M8M	21	17		mA	
		NEO-M8Q	28	23		mA	
	Icc Tracking (Power Save mode / 1 Hz)	NEO-M8N	13	12		mA	Estimated at 3 V
		NEO-M8M	5.3	4.8		mA	
		NEO-M8Q	11.5	11.1		mA	

FIGURE 5.24 – Consommation du GPS NEO-M8

En termes de consommation, tous les GPS sont de gourmands en énergie, car ils doivent continuer à écouter les signaux GPS. La Figure 5.24 illustre combien le circuit consomme en acquisition. Cela dépend du mode choisi, le mode normal étant nommé *ICC Acquisition*. Pour le NEO-M8Q, la consommation est en moyenne de 30mA avec GLONASS et GPS utilisés. À cette consommation, il faut encore ajouter celle d'une antenne active, dans la mesure où ce type d'antenne est raccordée. Les antennes actives consomment entre 3mA et 25mA selon les modèles utilisés. La consommation

finale oscille ainsi autour de 50 mA, ce qui est problématique pour beaucoup de systèmes embarqués autonomes. Heureusement, le NEO-M8 peut être placé en mode *low-power* via l'utilisation de commandes UBX ; il est toutefois impossible de trouver la valeur exacte de la consommation de ce mode parmi les documents de U-blox. La seule référence à la consommation indique ce qui suit :

> *During operation, the current drawn by the module can vary by some orders of magnitude, especially if enabling low-power operation modes.*

Aucune valeur n'est fournie pour la consommation dans ce mode. Il est également possible de placer le GPS en mode *backup*. C'est un mode où le GPS est désactivé, mais l'éphéméride des satellites est sauvegardée dans une mémoire volatile pour accélérer la synchronisation lorsque le mode normal est resélectionné. Pour sauver l'état de cette mémoire, une tension doit être appliquée au circuit sur une pin spécifique. Cette sauvegarde consomme 15µA si la tension de l'alimentation est de 1.8V. Ce type de mémoires peuvent être alimentées par des piles boutons, puisque le seuil minimum de tension est de 1.4V. Dans le cadre de ce projet, cette pin a été directement connectée à la tension VCC du circuit, celui-ci étant régulé par une batterie.

5.3.4.6 Connecteur PMOD

La carte DevBox est également compatible avec les connecteurs PMOD. Ces connecteurs sont basés sur un standard d'interface [1] du même nom développé par l'entreprise Digilent [2]. Le pinout des différents types d'interfaces est disponible sur la Figure 5.25. Dans le cas de la DevBox, les standards n'ont malheureusement pas pu être intégralement respectés, en raison de l'absence de GPIO disponibles sur le module Bluetooth. Les connecteurs sont compatibles avec les PMOD de type I2C, à l'exception des deux IO et des signaux. L'interface SPI est supportée à l'exception de INT et RST. Le pinout final de la DevBox est visible sur le schéma `PMOD` de l'Annexe E.

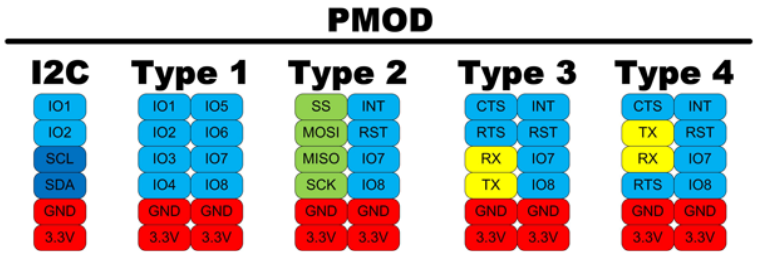


FIGURE 5.25 – Pinout du standard PMOD en fonction des types choisis

Digilent commercialise des modules compatibles avec ces interfaces, ce qui facilite l'ajout d'un type de capteur qui n'aurait pas , par défaut, été rajouté sur la carte de développement. Le problème lié aux processeurs utilisés est le nombre de connexions disponibles vers l'extérieur. Nos deux microcontrôleurs ne sont équipés que de 48 pins au total, dont la moitié sont utilisés pour les alimentations et divers contrôles internes.

1. https://en.wikipedia.org/wiki/Pmod_Interface
2. http://store.digilentinc.com/

5.4 Schéma bloc général final

La Figure 5.26 illustre le schéma bloc final sur Altium de la carte DevBox. Il représente toutes les interfaces de communications entre les différents blocs. Cette image est également disponible sur l'Annexe E, tout comme les autres schémas électroniques du projet.

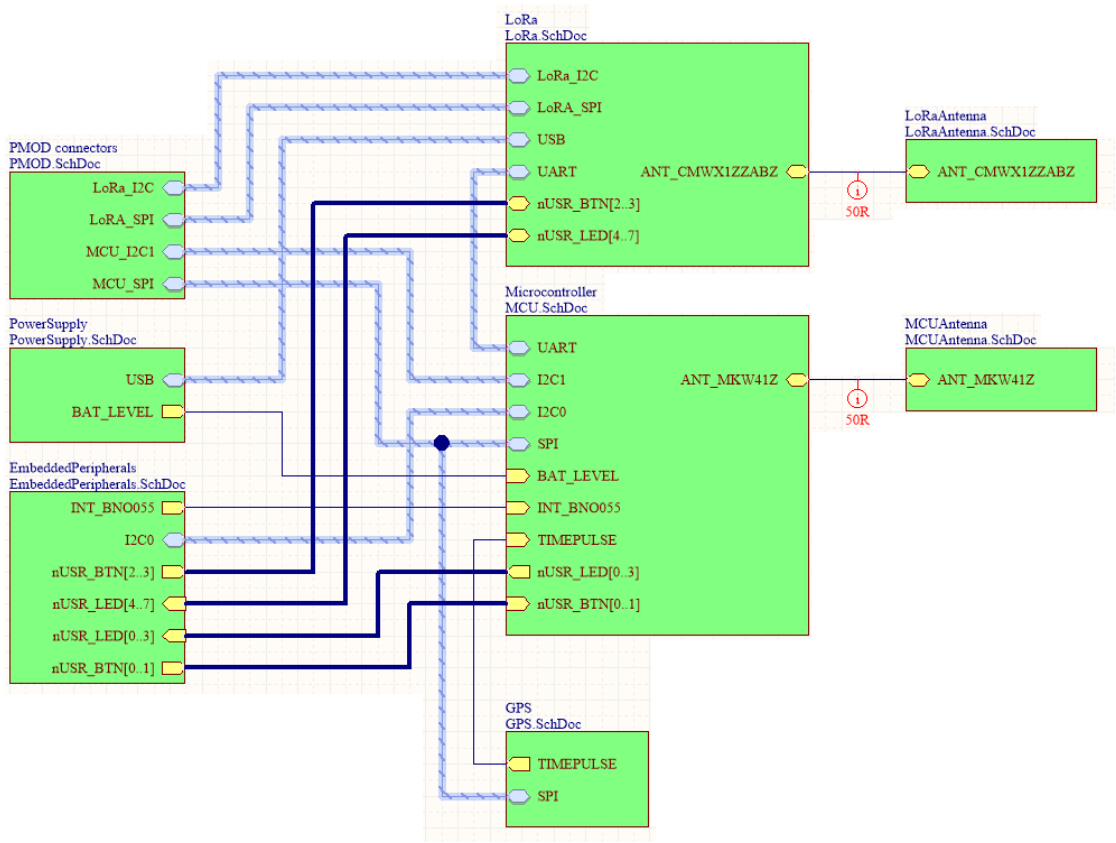


FIGURE 5.26 – Schéma bloc général de la DevBox

5.5 Réalisation et production du circuit électronique

Le PCB réalisé a été routé selon les contraintes du fabricant de PCB [1] choisi, soit PcbWay. L'épaisseur choisie est de 1.6mm et les largeurs de piste minimales de 4mil. Les plus petits trous autorisés sont de 0.2mm. Afin de facilité la soudure, une finition à l'or a été demandée. Le temps de production est d'environ une semaine et demie à partir du moment où la transaction est validée.

Les composants électroniques ont principalement été commandés auprès de trois fournisseurs : Mouser, Farnell et Digikey. Mouser a été privilégié pour les composants, les deux autres n'ayant été utilisés qu'en cas de rupture de stock chez Mouser. La liste complète des composants avec leurs numéros de commande sont indiqués dans le projet Altium présent sur GitHub.

1. https://www.pcbway.com/

Un masque de dépose de pâte a dû être commandé à l'entreprise DB Products [1]. Une fois celui-ci reçu la production a pu être effectuée avec une machine de dépose de composants manuelle.

5.6 Erreurs de conception et corrections possibles

Une fois le PCB réalisé, plusieurs tests ont été effectués pour vérifier son fonctionnement. Quelques erreurs critiques ont du être corrigées :

- **Module LoRa** : VDD_USB doit être connecté au 3.3V et non au 5V ;
- **Module LoRa/Bluetooth** : RX et TX sont inversés sur la ligne UART ;
- **Module LoRa** : VDD_TCXO doit être connecté à la pin PA12 afin de pouvoir placer le SX1276 en mode *sleep*.

Quelques améliorations peuvent également être envisagées pour la prochaine version, celles-ci n'étant toutefois pas un obstacle au fonctionnement de la version actuelle de la DevBox :

- **Module LoRa** : Connecter la mesure de la batterie (diviseur de tension) sur une pin ADC du module LoRa. Pour ce faire, il faut également relier la pin VREF à une référence externe ou au VCC. A l'heure actuelle, uniquement le KW41Z a accès à cette mesure.
- **GPS** : Ajouter le support d'une antenne active sur le PCB avec l'ajout de la tension d'alimentation filtrée sur le signal RF. Il est également possible d'ajouter un autre LNA à la sortie du signal RF (cf. `Hardware Integration` PDF aux pages 15 et 16).
- **GPS** : Si l'on souhaite mettre en veille le GPS, il est possible de configurer une pin afin que celle-ci le place en mode *sleep* lorsqu'elle est placée à un état 0. Il s'agit là de la pin EXTINT, mais celle-ci doit préalablement être programmée au moyen du protocole UBX.
- **GPS** : Connecter la pin GPS_nCS sur une pin contrôlable par le DSPI en tant que *chip select*. Actuellement, cette pin est contrôlée par un GPIO.
- **PCB layout** : Déplacer le bouton reset du KW41Z afin d'éviter les *resets* non souhaités lors de la connexion sur câble JTAG.
- **PCB layout** : Placer les boutons du KW41Z en face des LED du même microcontrôleur.
- **PCB layout** : Les LED de la charge de la batterie peuvent être placées sur le haut de la carte.
- **BME680** : Le BME680 pourrait être placé dans un coin plus proche du bord de la carte. On peut également découper un sillage autour du capteur afin de limiter la chauffe du PCB.

5.7 Boitier en plastique

Un petit boitier en plastique, visible sur la Figure 5.27, a été créé afin d'y loger la carte. Ce boitier a été réalisé à l'aide du logiciel freecad [2] et est cité dans les sources du projet sur GitHub. Son impression a été réalisée à l'aide d'une imprimante 3D.

1. https://www.dbproducts.fr/
2. https://www.freecadweb.org/

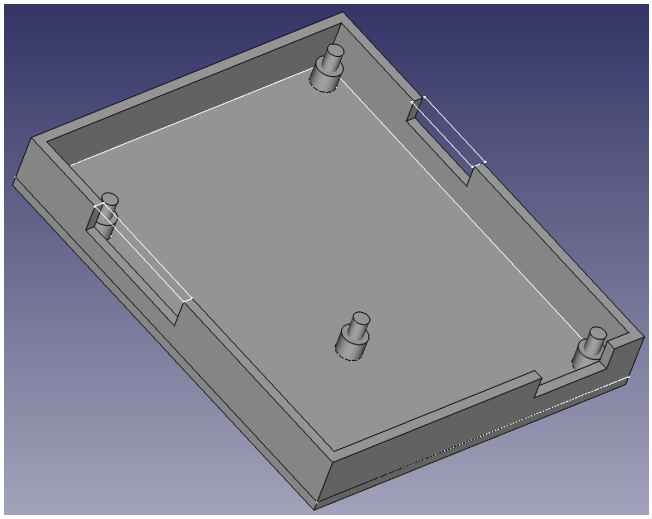


FIGURE 5.27 – Boitier imprimé en 3D pour y loger le circuit électronique

Chapitre 6

Développement logiciel

Ce chapitre a pour but d'expliquer l'implémentation logicielle sur les 3 plateformes utilisées (DevBox, Android et le serveur en Python). L'architecture générale y est présentée, mais également les implémentations spécifiques à chaque sous-catégorie.

6.1 Architecture générale

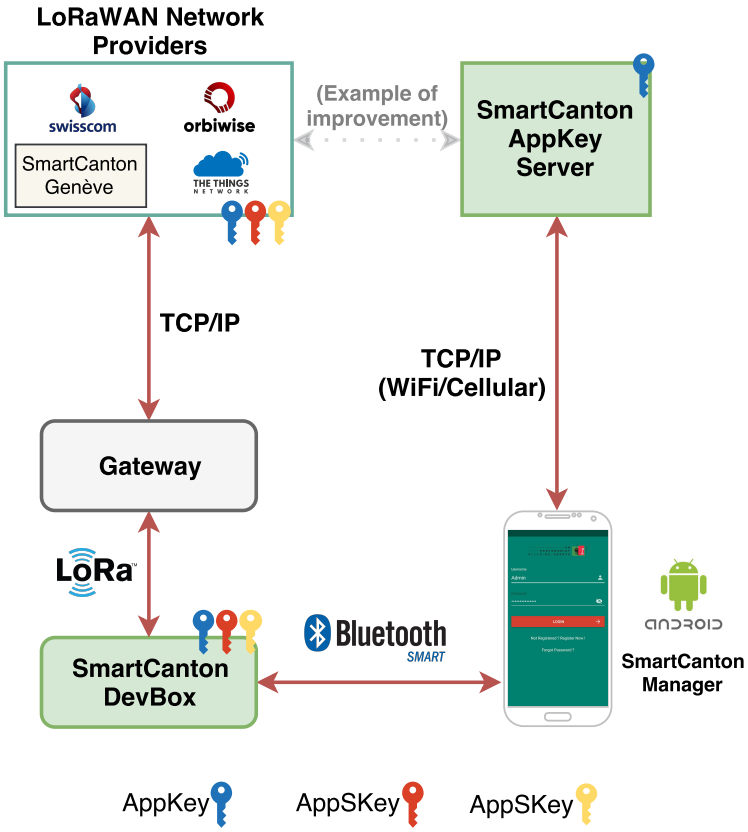


FIGURE 6.1 – Schéma général du projet avec les différentes technologies utilisées

L'architecture logicielle avec les différents protocoles de communications est visible sur la Figure 6.1. Au centre se situe la DevBox qui a deux interfaces de communication avec le LoRa et le Bluetooth Low Energy. Le Bluetooth sera accessible par une application développée pour Android dans un premier temps. La DevBox implémente une table GATT pour les services liés à la DevBox (batterie, LoRaWAN, GPS, BME680, etc.). Ceux-ci sont explorés en détail en Section 6.4.6.

6.2 Scanneur de périphériques Bluetooth Low Energy

L'implémentation du scanner de périphériques Bluetooth Low Energy est à cheval sur deux plateformes (Android et DevBox). Les spécifications liées à l'implémentation sur les plateformes sont présentées dans leurs sections respectives (Section 6.3 et Section 6.5).

Les spécificités des *advertisements* BLE ont été présentées dans l'état de l'art de ce document (Section 2.6). Lorsqu'un périphérique Bluetooth Low Energy souhaite être visible, il doit diffuser un paquet de données. Ce paquet est entièrement personnalisable par l'utilisateur. Trois types de *beacons* ont été présentés, parmi ceux-là, l'AltBeacon a été l'implémentation choisie. Principalement parce qu'elle est plus libre que le iBeacon. En effet, les *Manufacturer ID* ne sont pas délivrés par une entité, contrairement aux iBeacons (si on souhaite être certifiée par Apple). L'Eddystone est aussi très personnalisable, mais son grand avantage provient de ses URLs, qui dans notre projet ne sont pas nécessaires.

Le Tableau 6.1 explique le contenu du format choisi pour diffuser la position. L'UUID de 16bit a un *pattern* similaire à celui choisi pour les services Bluetooth. Ce qui est intéressant de voir c'est les bytes pour le Major et le Minor. Ceux-ci sont générés une seule fois aléatoirement pour chaque client au lancement du diffuseur d'*advertisements*. Cela permet ainsi de traquer un utilisateur durant un laps de temps contrôlable, c'est-à-dire, tant que l'application n'est pas arrêtée sur le périphérique. L'adresse MAC du Bluetooth peut donc changer, mais on pourra suivre un utilisateur simplement en sauvegardant ce paquet.

Table 6.1 – Format du AltBeacon utilisé par le scanner Bluetooth

AltBeacon Template				
Byte	**0**	**1**	**2**	**3**
0	AD Length	AD Type	MFG ID	
32	BEACON Code		BEACON ID	
64	BEACON ID		BEACON ID	
96	BEACON ID		BEACON ID	
128	BEACON ID		BEACON ID	
160	BEACON ID		BEACON ID	
192	BEACON ID		RSSI TX	MFG RSV
AltBeacon SmartCanton				
Byte	**0**	**1**	**2**	**3**
0	0x1B	0xFF	0xFF	0xFF
32	0xBE	0xAC	0x00	0x00
64	0x00	0x00	0xAA	0xAA
96	0xBB	0xBB	0xCC	0xCC
128	0xDD	0xDD	0xDD	0xDD
160	0xDD	0xDD	RANDOM	
192	RANDOM		0xBB	0x00

6.3 DevBox : module LoRaWAN

Le module LoRaWAN est interconnecté avec le microcontrôleur Bluetooth via une interface UART. Le débit final de cette interface a été fixé à 115200 bits par seconde.

6.3.1 Architecture logicielle

STMicroelectronics propose une suite de bibliothèques pour ses divers microcontrôleurs afin de pouvoir communiquer avec des modules LoRa. Cette suite de bibliothèques est basée sur la très connue suite de développement STM32Cube [1] et nommé `STM32 LoRa software expansion` [62]. La Figure 6.2 illustre l'architecture logicielle de cette suite de bibliothèques. Elle est composée d'un Hardware Abstraction Layer avec tous *drivers* des périphériques du microcontrôleur STM32. Pour le *middleware*, les blocs en vert sont repris de Semtech avec leur projet LoRaMac-node [2], alors que les blocs en bleu sont les *drivers* fournis par ST. Il s'agit là de *drivers* qui sont uniquement utilisés sur les différents capteurs présents sur les cartes de développement, telles que la B-L072Z-LRWAN1 et donc non utilisée dans ce projet. Il a un bloc nommé *utilities*, créé par Semtech, mais modifié par ST. Cette modification permet d'utiliser au mieux les ressources matérielles du STM32L0, telles que les *timers*, le mode *Low-Power*, ainsi que le *True Random Number Generator* (TRNG). L'architecture complète de cette suite de bibliothèques, ainsi que la description des diverses fonctions sont décrites dans le manuel d'utilisateur numéro **UM2073** de ST [62].

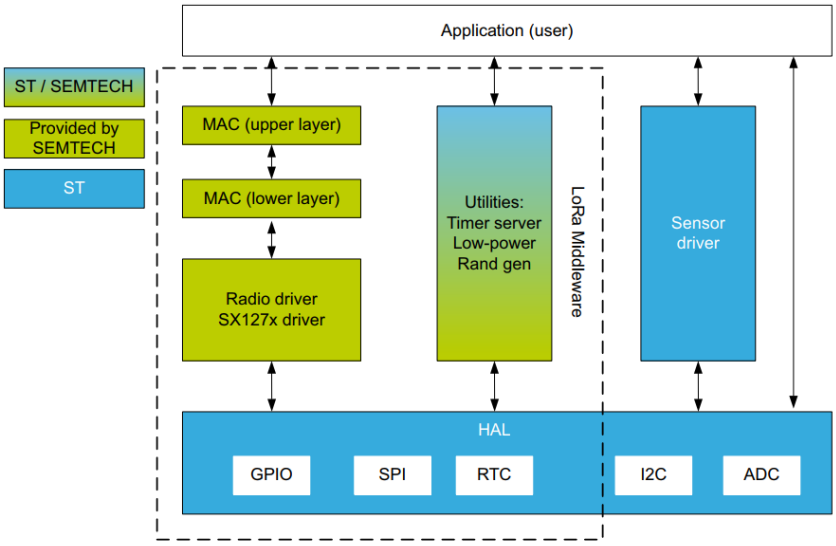


FIGURE 6.2 – Architecture logicielle du CMWX1ZZABZ

6.3.2 Exemples et modifications

STMicroelectronics propose trois exemples de codes pour le CMWX1ZZABZ adaptés pour la carte de développement B-L072Z-LRWAN. Voici la description de ces projets :

1. `End Node` : projet ayant pour but de créer un nœud LoRaWAN autonome avec la possibilité d'envoyer des données à des intervalles réguliers ou lors de la pression d'un bouton.

1. http://www.st.com/en/embedded-software/stm32cube-mcu-packages.html
2. https://github.com/Lora-net/LoRaMac-node

2. `AT Master` : offre la possibilité de communiquer avec un nouveau coprocesseur LoRa, implémentant un AT Slave interface. Cet exemple est très complet et est très utile pour comprendre comment la communication avec un AT Slave fonctionne. La Figure 6.3 illustre la connexion entre un microcontrôleur utilisant l'interface *Master* et un *Slave*. Néanmoins, ce projet repose sur beaucoup de bibliothèques de ST, il est donc difficile de l'utiliser directement sur un microcontrôleur d'un autre fabricant.
3. `AT Slave` : est utilisé pour offrir une interface de programmation du module à un utilisateur. Cet utilisateur peut utiliser un simple terminal série afin d'envoyer des commandes AT ou directement connecter un microcontrôleur (cf. Figure 6.3). La liste de toutes les commandes AT est disponible dans la note d'application nommée AN4967 [1], avec les paramètres, ainsi que les réponses.

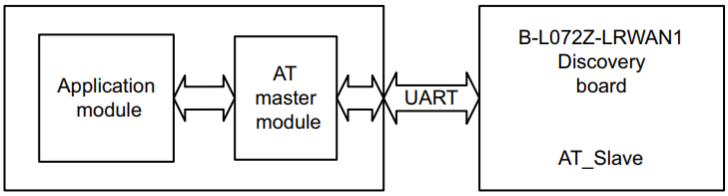


FIGURE 6.3 – Connexion entre un AT *Slave* et un AT *Master* utilisant la suite de bibliothèque STM32Cube LoRa

Une interface entre le KW41Z et le CMWX1ZZABZ doit être implémentée, il a donc été décidé de reprendre le projet AT Slave pour le CMWX1ZZABZ et de l'adapter au besoin de la carte. Cet exemple implémente déjà toutes les opérations nécessaires pour rejoindre un réseau LoRa et envoyer des données. Le seul inconvénient de cette interface est l'utilisation d'un protocole basé sur des strings qui demande plus de stockage en mémoire et de temps pour transformer les requêtes. Mais le KW41Z a 128 kB de RAM, qui sont amplement suffisants pour cela. Toutefois, le code a dû être modifié avant d'enlever quelques périphériques qui ne sont pas présents sur la DevBox. Il faut également changer l'interface de communication afin d'utiliser l'UART0 de la carte qui n'est pas sur les mêmes pins que le LPUART initialement connecté.

6.4 DevBox : NXP Kinetis KW41Z

Le développement logiciel du microcontrôleur NXP Kinetis KW41Z est l'élément central de ce projet. Il implique l'interfaçage de divers périphériques avec plusieurs interfaces distinctes et tout cela sur un système temps réel.

6.4.1 *OS Abstraction Layer* et *Connectivity Framework*

Les exemples de codes fournis par NXP pour ses microcontrôleurs implémentent tous FreeRTOS. Comme présenté en Section 2.9.3.1, il s'agit actuellement du RTOS le plus populaire et le plus actif en termes de communauté, ce qui apporte beaucoup d'avantages lors du développement. Lors de la présentation des différents RTOS, il a été fait référence aux couches d'abstraction de RTOS, en prenant l'exemple de l'OS Abstraction Layer (cf. Section 2.9.3.4). Celui-ci est visible sur la Figure 6.4. Il offre ainsi la possibilité de changer facilement de RTOS en fonction des besoins du projet

1. www.st.com/resource/en/application_note/dm00346311.pdf

pour les utilisateurs. NXP propose un framework nommé *Connectivity Framework* qui s'occupe principalement des connexions avec le monde extérieur (Bluetooth, Wifi, etc.), mais également qui incluent différents autres blocs qui sont tous affichés sur la Figure 6.4.

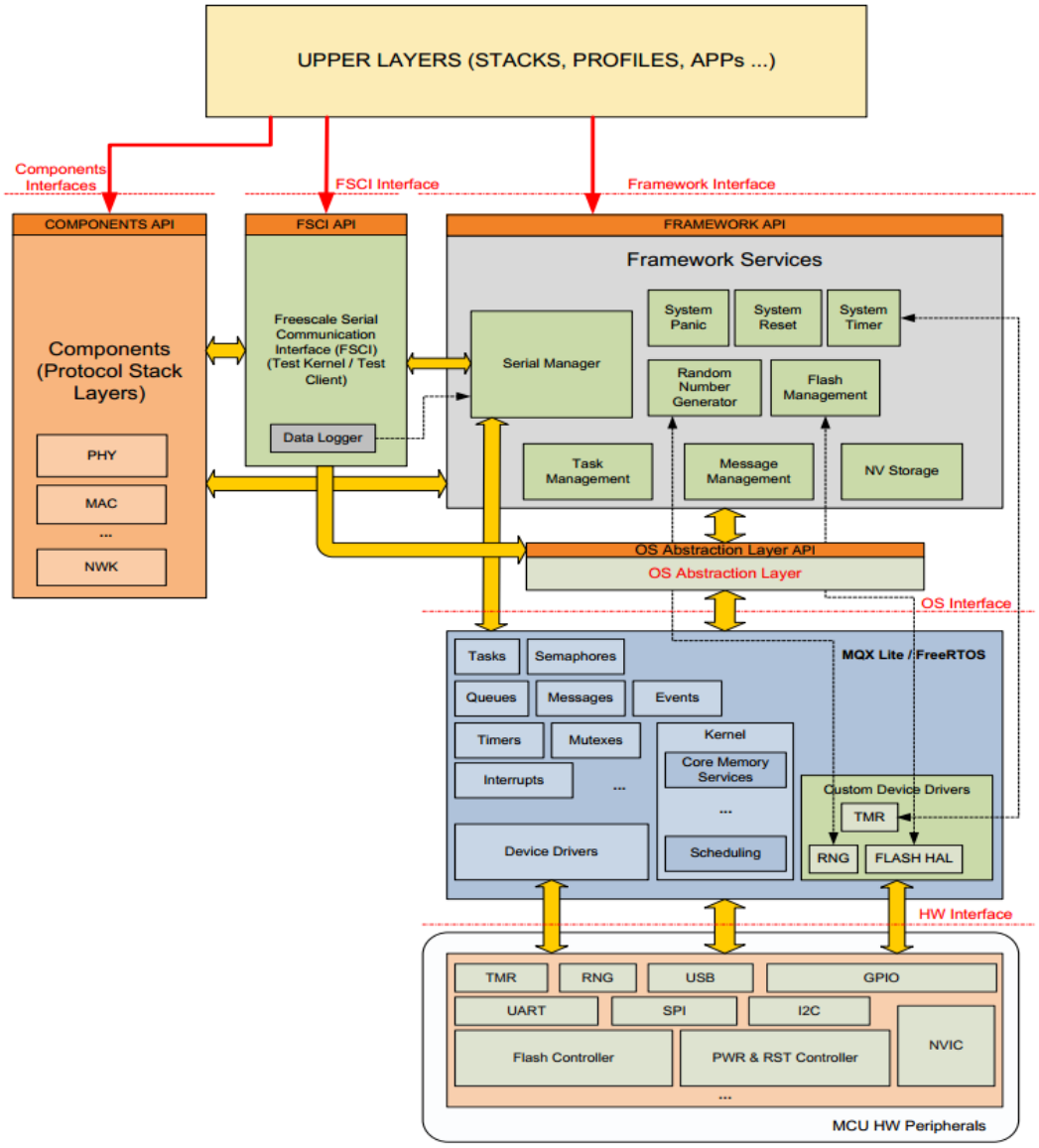


FIGURE 6.4 – *Connectivity Framework* de Freescale (NXP)

Tout d'abord, en commençant par les différents blocs en bas de la Figure 6.4 , la présence de plusieurs *drivers* pour les périphériques peut être constatée. Ceux-ci couvrent toutes les interfaces de communications présentes sur les microcontrôleurs et sont tous très complets. On a ensuite la présence d'un RTOS qui est composé des différents composants standards d'un RTOS tel que des *queues*, sémaphores et mutexes. Cette couche de RTOS repose directement sur l'interface matérielle, car il est parfois nécessaire d'adapter les drivers pour qu'ils soient *thread safe* en fonction des différents ordonnancements utilisés par les RTOS et ainsi garantir l'intégrité des données. Vient ensuite l'interface OS Abstraction Layer pour uniformiser les différents RTOS. Un certain nombre de services sont disponibles, comme par exemple, le *Serial Manager* qui a été utilisé dans ce projet pour gérer les connexions UART avec le coprocesseur LoRa. Celui-ci peut gérer des flux asynchrones provenant de plusieurs

types d'interfaces (UART, USB, I2C et SPI). Pour les interfaces I2C ou SPI, celles-ci doivent être configurées avec une interruption sur un GPIO pour informer le *Serial Manager* qu'une nouvelle donnée est disponible. La plupart de ces frameworks sont des tâches RTOS propre et utilisent des interfaces de communications *thread-safe* afin d'être programmés.

Ensuite, on peut voir l'élément qui simplifie l'intégration des communications externes (Bluetooth, WiFi, etc.) qui se nomme *Components API*. Ce bloc a pour but de faciliter la programmation des diverses *stacks* présentes sur les microcontrôleurs en fonctions des différentes gammes. On retrouve ainsi le Bluetooth, mais également des stacks pour le TCP/IP ou Thread. C'est souvent ce dernier bloc qui n'est pas open source.

Freescale Serial Communication Interface (FSCI) est à la fois un module, mais également un protocole de communication utilisé entre les interfaces de communications externes (*Components API*) et les services. On peut voir sur la Figure 6.4 qu'il n'est pas forcément nécessaire de passer par le FSCI pour accéder aux composants depuis les composants applicatifs. Le FSCI ainsi que les *frameworks* Services sont documentés dans un manuel de référence de NXP nommé `CONNFWKRM`. Toutefois, la documentation est légère, il est souvent plus facile de regarder le code source ou simplement les fichiers d'entêtes de fonctions pour les parties non open sources.

Le bloc, le plus haut niveau du framework, nommé *upper layers*, est là où l'utilisateur doit normalement implémenter la plus grande partie de son code. Il a ensuite accès à ses trois interfaces pour utiliser toutes les ressources à sa disposition sur le microcontrôleur.

6.4.2 Interfaces de programmation

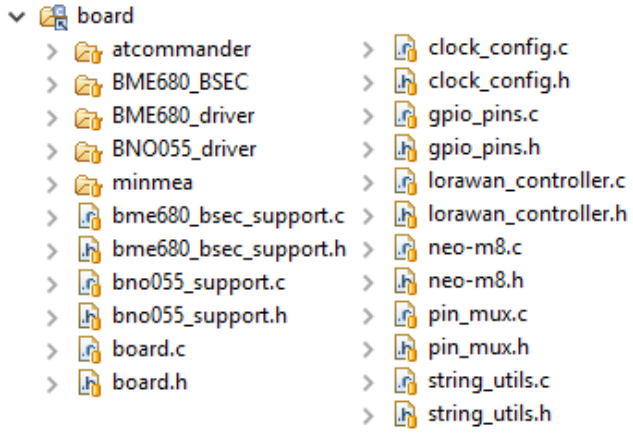


FIGURE 6.5 – Bibliothèques et *drivers* utilisés sur la DevBox

Lors du développement de ce projet, plusieurs éléments ont été utilisés afin de simplifier la programmation de la DevBox. Un répertoire nommé `smartcanton_devbox_board` présent dans le dépôt GitHub au chemin d'accès, `dev/MKW41Z` contient des interfaces qui ont pour but d'être utilisées sur tous les projets qui utiliseront la DevBox à l'avenir. Ce répertoire est ensuite ajouté à l'aide d'un lien symbolique dans le projet MCUXpresso, comme illustré à l'aide de la Figure 6.5. Certains de ces éléments sont des bibliothèques ou *drivers* qui ont été développés par la communauté ou directement par les fabricants des composants électroniques utilisés.

6.4.2.1 Bibliothèques et *drivers* repris

En plus du SDK fourni par NXP avec les différents *drivers* pour le KW41Z, un certain nombre de bibliothèques ont été reprises et utilisées afin d'accélérer le développement du projet.

6.4.2.1.1 AT Commander

La communication avec le module LoRaWAN s'effectue avec des commandes AT. Pour cela, une bibliothèque nommée AT Commander a été utilisée :

`https://github.com/openxc/AT-commander`

Elle a toutefois du être modifiée à cause du format des requêtes qui sont transmissent par le module LoRaWAN. Cette bibliothèque est par défaut utilisable avec des modules AT de type RN42 ou XBEE, dont le format de trames diffère de celles du projet STM32 LoRa. Pour utiliser cette bibliothèque, l'utilisateur doit simplement spécifier deux fonctions pour lire et écrire sur un port UART souhaité.

6.4.2.1.2 minmea

Le *parsing* des trames GPS est fastidieux et peut être complexe en fonction des différents GPS et du type de données contenues dans une trame GPS. Il existe beaucoup de bibliothèques pour les GPS, mais souvent celles-ci ne sont pas adaptées aux systèmes embarqués. Cela s'explique par les fonctions de traitement de *strings* qui demandent trop de ressources au processeur. Heureusement, il existe une bibliothèque nommée `minmea` qui implémente un parseur pour la spécification NMEA 0183 [63]. Celle-ci est pensée pour les environnements avec des capacités restreintes, comme les microcontrôleurs. Elle est disponible à l'adresse suivante :

`https://github.com/kosma/minmea`

`Minmea` est composée de deux fichiers et très simple d'utilisation. La trame récupérée depuis le GPS lui est directement fournie et celle-ci s'occupe d'extraire toutes les informations souhaitées. Elle supporte les données de type RMC, GGA, GSA, GLL, GST, GSV, VTG et ZDA [64]. Toutes les données sont ensuite accessibles via des structures selon le type de données.

6.4.2.1.3 BNO055 *driver*

Bosch Sensortec fournit différents drivers pour la plupart de ses composants. Un *driver* est disponible pour le BNO055, avec lesquels, il est possible de communiquer en I2C ou SPI. Selon le mode choisi, des fonctions pour la lecture et écriture doivent être fournies à la bibliothèque. L'utilisateur peut ensuite accéder aux différents registres du capteur. Le code source de la bibliothèque contient plus de 17000 lignes de codes, car le BNO055 est un capteur à la fois complet, mais également complexe à programmer. Ce *driver* est disponible à l'adresse suivante :

`https://github.com/BoschSensortec/BNO055_driver`

6.4.2.1.4 BME680 *driver*

Le BME680 dispose également d'un driver pour récupérer les différentes informations du capteur comme pour le BNO055. Cependant, une grande partie de la correction des mesures s'effectue en logiciel, ce qui a pour conséquence de créer un *driver* minimal. Il est capable de fournir uniquement la valeur brute de la température, de l'humidité et de la pression. Pour la mesure de la qualité de l'air, celle-ci est uniquement possible à l'aide d'un traitement logiciel. La seule information sur le gaz est la valeur ohmique, mais cela n'indique en rien la qualité de l'air. La bibliothèque est disponible sur le dépôt GitHub suivant :

`https://github.com/BoschSensortec/BME680_driver`

6.4.2.1.5 Bosch Sensortec Environmental Cluster

Comme expliqué dans la section précédente, ainsi qu'en Section 5.3.4.4, la grande partie du traitement des données s'effectue en software. Le fabricant propose une bibliothèque nommée Bosch Sensortec Environmental Cluster (BSEC). BSEC n'est pas open source, elle est fournie précompilée pour diverses architectures, limitant le débogage du système, mais permettant ainsi de garder secret l'algorithme utilisé. La bibliothèque à l'adresse suivante :

`https://www.bosch-sensortec.com/bst/products/all_products/bsec`

IAQ Index	Air Quality
0 – 50	good[10]
51 – 100	average
101 – 150	little bad
151 – 200	bad
201 – 300	worse[2]
301 – 500	very bad

FIGURE 6.6 – Niveaux de l'indice IAQ

Name	Value	Accuracy description
UNRELIABLE	0	Sensor data is unreliable, the sensor must be calibrated
LOW_ACCURACY	1	Low accuracy, sensor should be calibrated
MEDIUM_ACCURACY	2	Medium accuracy, sensor calibration may improve performance
HIGH_ACCURACY	3	High accuracy

FIGURE 6.7 – Niveaux de l'indice IAQ Accuracy

Pour utiliser cette bibliothèque, le *driver* BME680 doit être également utilisé et lié au projet. La qualité de l'air est mise à disposition à l'aide de l'indicateur *Indoor Air Quality* (IAQ). Celle-ci est accompagnée d'un niveau de fiabilité nommé IAQ *Accuracy*. L'IAQ est calculé à l'aide d'un algorithme propriétaire dont la valeur finale se situe entre 0 et 500. Les différents paliers de cet indice sont visibles sur la Figure 6.6. Selon le guide d'intégration de la bibliothèque BSEC [65], le périphérique nécessite 4 à 28 jours de calibration. Sur cette période, le composant doit être exposé 30min au milieu le moins pollué possible et 30min dans un milieu très pollué, afin de permettre une calibration plus correcte. La température et l'humidité sont également corrigées en fonction de l'environnement et la bibliothèque offre même la possibilité de placer un offset sur les valeurs pour prendre en compte l'échauffement des composants électroniques à proximité.

6.4.2.2 Bibliothèques et *drivers* développés

Tous les fichiers présents sur la Figure 6.5 illustrant le répertoire `board` qui ne sont pas dans un sous répertoire ont été développés pour ce projet. Les sous-sections qui suivent ont pour but d'expliquer l'utilité des fichiers les plus importants.

6.4.2.2.1 `pin_mux`

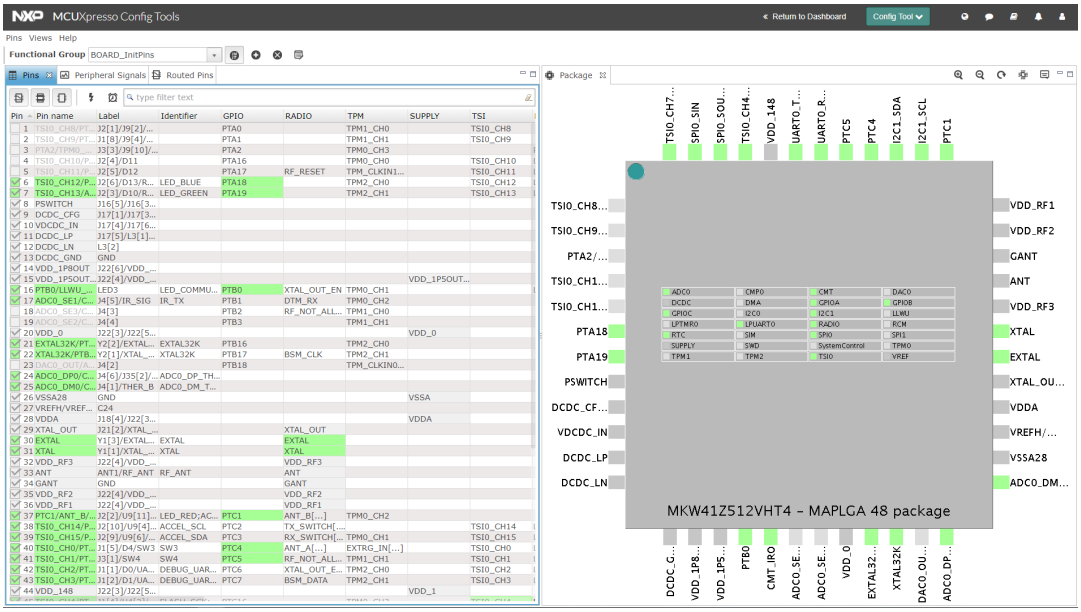


FIGURE 6.8 – Utilitaire de configuration du multiplexeur de pins des microcontrôleurs NXP

Le KW41Z est équipé d'un multiplexeur de pins, cela signifie que chaque pin peut avoir plusieurs rôles qui peuvent se modifier en cours d'exécutions. Pour aider les programmeurs, NXP propose dans le cadre de son SDK, un utilitaire nommé MCUXpresso Config Tools [1]. Cet utilitaire est visible sur la Figure 6.8. Une fois les pins connectés sur l'utilitaire, il est possible d'exporter la configuration sous forme de fichier source afin d'être intégré au projet.

6.4.2.2.2 `board`

Une partie de ces deux fichiers est une reprise des divers exemples prodigués par NXP. Leur rôle est de fournir des accès génériques sans tenir compte du pinout appliqué au microcontrôleur. Par exemple, on peut y trouver une fonction nommée `BOARD_EnterLowPowerCb` offrant ainsi la possibilité de mettre le processeur en mode basse consommation.

6.4.2.2.3 `bme680 et bno055 support`

Les drivers et bibliothèques du BME680 et BNO055 sont très bien fournis (cf. Section 6.4.2.1.3 et Section 6.4.2.1.4), mais il faut leur spécifier les interfaces matérielles qui communiquent avec le périphérique. Dans les deux cas, il s'agit d'un BUS I2C. Pour éviter tout problème de concurrence entre les tâches qui utilisent la

1. https://mcuxpresso.nxp.com/en/pins

même interface, le driver `fsl_i2c_freertos` a été utilisé dans ces deux bibliothèques.

La bibliothèque BSEC enregistre ponctuellement l'état de son algorithme d'apprentissage. Pour cela, il est nécessaire de fournir une fonction de sauvegarde en mémoire non volatile. Un secteur en flash a donc été réservé pour sauvegarder toutes les 2 heures l'état de la bibliothèque (fréquence de sauvegarde configurable). Une fonction de relecture de cette mémoire est également nécessaire, ainsi qu'une fonction pour la configuration utilisée (cf. le document *Integration Guide Bosch Software Environmental Cluster* (BSEC) [65] pour toutes les configurations possibles).

Dans le cas du BME680, une fonction nommée `bme680_bsec_kw41z_iot_loop` s'occupe de réaliser l'acquisition en continu du capteur avec toute la gestion de la bibliothèque BSEC (capture, attente, sauvegarde de l'état de l'algorithme, etc.). L'utilisateur doit simplement fournir une fonction de *callback* afin d'être notifié de la mesure d'un nouveau set de données, ainsi que divers paramètres concernant cette fréquence d'acquisition.

6.4.2.2.4 `lorawan_controller`

Le contrôleur LoRaWAN est l'un des éléments centraux de ce travail, c'est grâce à celui-ci qu'il est possible de configurer et de communiquer à travers un réseau LoRaWAN. Pour la communication avec le module LoRaWAN, les commandes AT sont envoyées à l'aide de la bibliothèque AT Commander (cf. Section 6.4.2.1.1). Le contrôleur supervise également un espace mémoire en flash afin de stocker la configuration du module LoRaWAN dans but de se reconnecter au réseau lorsque le processeur est redémarré. Le contrôleur stocke en interne la configuration appliquée au module, ainsi que la nouvelle configuration que l'utilisateur souhaite appliquer à celui-ci. La structure `lorawanControllerConfiguration_t` a pour but de contenir toutes les informations de la configuration du module. À l'heure actuelle, l'AppKey est stockée sur le KW41Z, mais dans une version finale du système, celle-ci doit être stockée directement dans le module pour éviter que quelqu'un ne puisse intercepter la configuration lors de l'initialisation. Les attributs de cette structure sont listés ci-dessous :

lorawan_controller.h

```
30  /**
31   * @brief Structure defining the configuration that will be saved in flash.
32   * Every time we restart, this configuration is read and reapplied to the
33   * LoRaWAN module.
34   */
35  typedef struct lorawanControllerConfiguration_tag
36  {
37          // Used to make the lorawanControllerConfiguration_tag a
38          // multiple of 4 and to verify validity
39          uint16_t crc;
40
41          uint32_t magicWord;
42
43          char appEui[24];
44          char appKey[48];
45          char devEui[24];
46
47          char devAddr[12];
48          char nwkSessionKey[48];
49          char appSessionKey[48];
50
51          char networkJoinMode[2];
52          char confirmMode[2];
53          char etsiDutyCycleEnable[2];
54  } lorawanControllerConfiguration_t;
```

Ces champs sont stockés sous forme de chaine de caractères. Cela est dû au fait que la communication avec le module LoRaWAN est basée sur des commandes AT, et donc les paramètres de ces commandes sont tous des *strings*. Afin d'économiser du temps CPU dans la conversion des données lors de l'envoi de chaque commande au module,

il a été décidé de garder ce format tout au long du processus, au risque d'occuper plus d'espace mémoire. Un CRC codé sur 16bits a été ajouté à la configuration, à la suite de plusieurs problèmes liés à l'écriture et à la lecture de la mémoire flash. Ces problèmes ont été résolus, mais ce CRC a été maintenu puisque si la carte s'éteint lors de l'écriture en flash, les données peuvent toujours être corrompues. Si le CRC est erroné lors de la lecture d'une configuration, la configuration par défaut est appliquée.

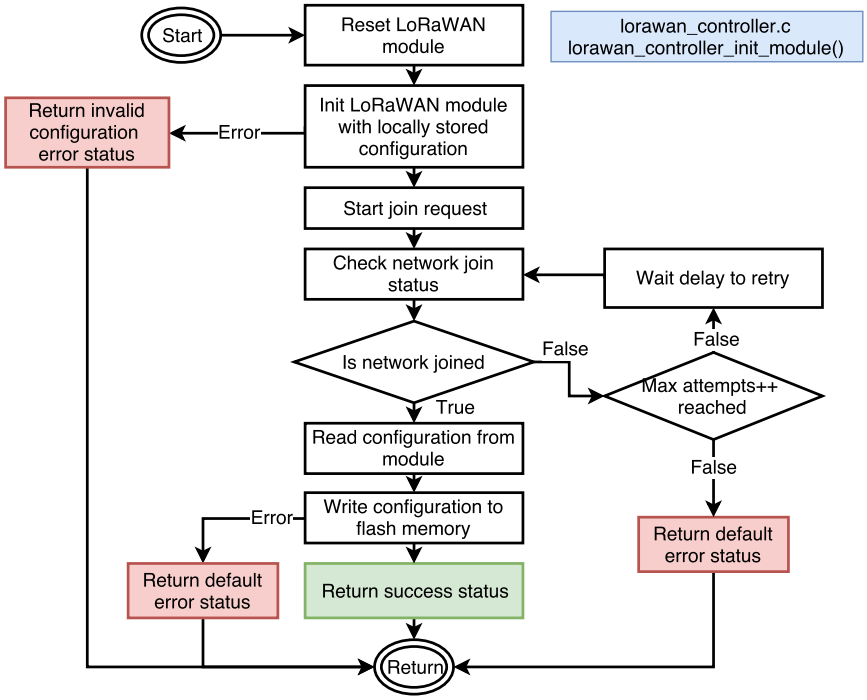


FIGURE 6.9 – Diagramme de la fonction d'initialisation du module LoRaWAN

L'une des étapes les plus importantes du module est son initialisation. L'initialisation s'effectue à l'aide de la fonction `lorawan_controller_init_module`. Un diagramme représentant l'algorithme utilisé est visible sur la Figure 6.9. En observant ce diagramme, on constate que la configuration du module est sauvegardée uniquement lorsque le réseau a pu être correctement rejoint à l'aide d'un *Join Request*. Tant que la demande de *join* n'est pas confirmée, le contrôleur est bloqué. Un temps d'attente maximum est configurable dans le contrôleur en spécifiant le nombre de vérifications du statut de *join* avec le temps d'attente entre chacune de ces vérifications.

6.4.2.2.5 `neo-m8`

Ces fichiers contiennent toutes les fonctions pour récupérer les données provenant du NEO M8 avec le support de l'interruption du signal `timepulse` pour démarrer l'acquisition. Une fois les données récupérées, il est possible de spécifier une fonction de *callback* à l'aide d'un pointeur pour informer l'application de la réception de nouvelles mesures. Ces mesures sont ensuite accessibles par l'application en utilisant le parseur minmea (cf. Section 6.4.2.1.2). Cette bibliothèque est principalement utilisée par la tâche GPS présentée en Section 6.4.4.2.

Le NEO M8 supporte le protocole propriétaire UBX afin de communiquer avec le GPS pour le configurer. Cette configuration n'a pas été mise en place à l'heure actuelle, principalement par manque de temps. Toutefois, si l'on souhaite avoir un

système base consommation, le support de ce protocole doit être fait. Plusieurs bibliothèques sont disponibles pour le protocole UBX [1], cependant peu d'entre elles sont orientées systèmes embarqués avec de faibles ressources. Le protocole UBX est très conséquent et nécessite beaucoup de place en mémoire. Par défaut le GPS envoie des trames GPS NMEA avec un intervalle de 1 seconde et sans la possibilité d'être mis en mode *sleep*. Ces paramètres sont uniquement modifiables à l'aide du protocole UBX.

6.4.2.2.6 `string_utils`

Dans ce projet, il est souvent nécessaire de faire des conversions entre plusieurs formats de données. Le type de conversion le plus utilisé est le tableau de *bytes* en *string* et vice versa. Des fonctions ont donc été développées et placées à l'intérieur de ces fichiers.

6.4.3 Architecture des tâches

Dans ce projet, il a été décidé de faire en sorte que la communication entre les tâches soit le plus uniforme possible. L'idée est de permettre à l'utilisateur de désactiver simplement une tâche, s'il le souhaite.

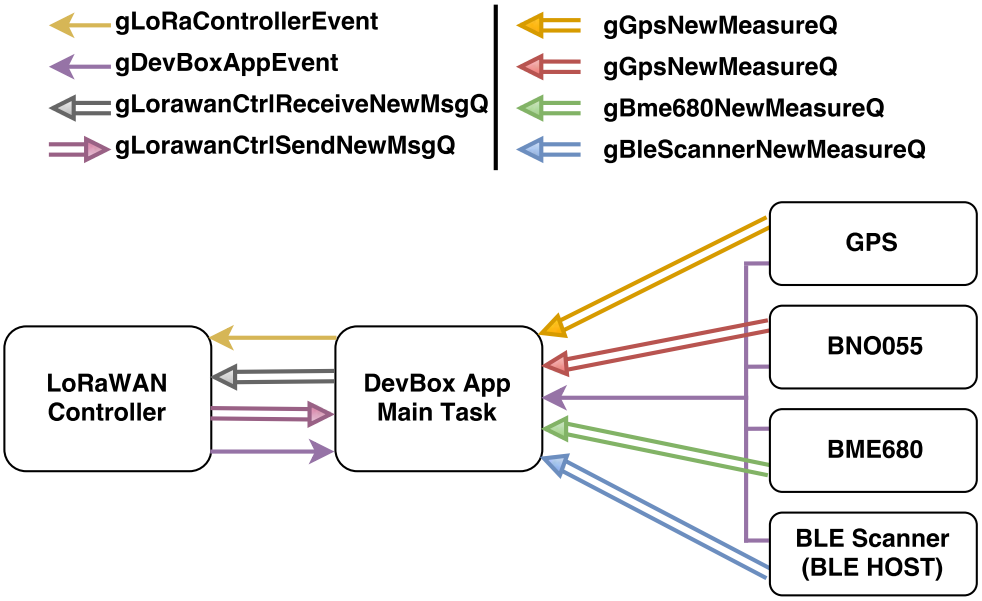


FIGURE 6.10 – Vue générale des communications entre les tâches

À l'aide de la Figure 6.10, il est possible de voir les différentes tâches implémentées ainsi que les mécanismes de communication mis en place. Chaque tâche produisant des données de ses capteurs dispose d'une *queue* (file) dans laquelle elle va alimenter les données des captures. Une fois ces données dans cette file, une notification est envoyée à la tâche principale à l'aide d'un événement. La tâche principale attend en permanence sur cet événement pour agir en conséquence. Dans les RTOS, il est possible d'attendre sur l'arrivée de données provenant des *queues*, mais il n'est pas envisageable d'écouter plusieurs flux en même temps, du moins, en utilisant l'OSA. Toutes les tâches utilisent des attentes sur des signaux, ce qui permet d'économiser le temps du processeur contrairement aux attentes actives (scrutations). S'agissant là

1. https://github.com/arobenko/ublox

d'un système embarqué, la consommation liée à ces ressources est un facteur important.

6.4.4 Algorithmes des tâches

Le but de cette section est de présenter les différentes étapes qui se déroulent dans les différentes tâches d'acquisition des données. Le but n'est pas d'entrer dans les détails, mais d'avoir un aperçu global du *workflow* implémenté dans chaque tâche.

6.4.4.1 DevBox Main

La Figure 6.10 explique la synchronisation à l'aide de l'envoi d'événements avec la tâche principale. Le nom de l'événement est `gDevBoxAppEvent` et peut transmettre les types d'événements suivants :

dev_box_app_task.h

```
63  /* Event ID to sent new data to task or to request a new LoRaWAN to be sent
64   * cf. below to see the events possible */
65  extern osaEventId_t gDevBoxAppEvent;
66
67  /* Task Events */
68  /* <Set by the LoRaWAN controller task when a network as been successfully joined> */
69  #define gDevBoxTaskEvtNewLoRaWANConfig_c        (1 << 0)
70
71  /* <Set by the GPS task when need data are ready to be process> */
72  /* <Data can be read from the gGpsNewMessageMeasureQ Queue> */
73  #define gDevBoxTaskEvtNewGgpMeasureAvailable_c        (1 << 1)
74
75  /* <Set by the BME680 task when need data are ready to be process> */
76  /* <Data can be read from the gBme680NewMessageMeasureQ Queue> */
77  #define gDevBoxTaskEvtNewBme680MeasureAvailable_c        (1 << 2)
78
79  /* <Set by the BNO055 task when need data are ready to be process> */
80  /* <Data can be read from the gBno055NewMessageMeasureQ Queue> */
81  #define gDevBoxTaskEvtNewBno055MeasureAvailable_c        (1 << 3)
82
83  /* <Set by the BLE Scanner when need data are ready to be process> */
84  /* <Data can be read from the gBleScannerNewMessageMeasureQ Queue> */
85  #define gDevBoxTaskEvtNewBleScannerMeasureAvailable_c        (1 << 4)
86
87  /* <Set when the user want to send new data to the LoRaWAN network> */
88  /* <Data should be sent to the gLorawanCtrlSendNewMessageQ Queue> */
89  #define gDevBoxEvtSendNewLoRaData_c        (1 << 5)
90
91  /* <Set by the LoRaWAN controller when new data have been received> */
92  /* <Data can be read from the gLorawanCtrlReceiveNewMessageQ Queue> */
93  #define gDevBoxEvtNewLoRaDataReceived_c        (1 << 6)
```

La tâche DevBox Main est la tâche principale de toute l'application. Elle a comme rôle de connecter les données des capteurs avec les différentes interfaces de communications disponibles. Voici la liste des opérations qui lui incombent :

1. Son rôle principal est de récupérer les données des divers capteurs présents sur la carte. Ces données sont ensuite stockées en local dans un historique temporaire.
2. De manière générale, tous les services Bluetooth sont actualisés par cette tâche afin de garder une certaine structure dans le code et ainsi éviter des problèmes de fragmentations. Par exemple, lorsqu'un réseau LoRaWAN est rejoint, le contrôleur informe la tâche principale que les services doivent être mis à jours. De même pour les capteurs, les mesures reçues des tâches adjacentes sont également mises à jour dans les services Bluetooth Low Energy correspondants (cf. Section 6.4.6.2).
3. Périodiquement, la tâche envoie des données sur le réseau LoRaWAN. L'envoi de nouvelles données s'effectue à l'aide d'un *timer* qui définit l'intervalle entre chaque paquet LoRaWAN. Le paquet contient toujours les dernières données qui ont été sauvegardées localement. Cet intervalle est à ce jour uniquement configurable lors de la réception d'un paquet *downlink* depuis le réseau.

4. Les paquets *downlink* LoRaWAN sont transmis à la tâche afin que l'application agisse en conséquence.

En fonction du type d'événements reçus, une action est réalisée. En attendant l'événement, la tâche ne consomme pas de ressources. Une représentation simplifiée de l'algorithme appliqué dans la tâche est visible sur le diagramme Figure 6.11.

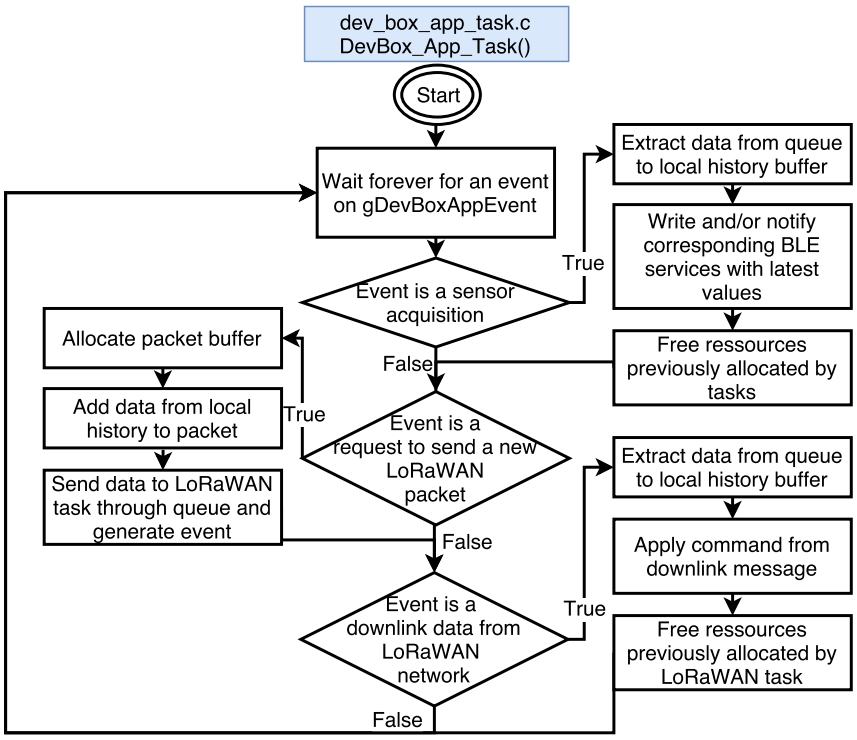


FIGURE 6.11 – Diagramme de fonctionnement de la tâche principale lors de la gestion des événements

6.4.4.2 GPS Neo M8

Le GPS génère périodiquement des trames NMEA qui doivent être lues à l'aide de l'interface SPI du KW41Z. Toute cette acquisition est gérée par la bibliothèque nommée neo-m8 (cf. Section 6.4.2.2.5). À l'heure actuelle, la LED numéro 3 de la carte change d'état pour indiquer que la synchronisation avec le GPS est correcte et qu'au minimum la synchronisation temporelle est possible (*timepulse*). La Figure 6.12 illustre les différentes étapes de l'acquisition et l'envoi des données à l'aide du processus d'échange expliqué précédemment pour la tâche GPS.

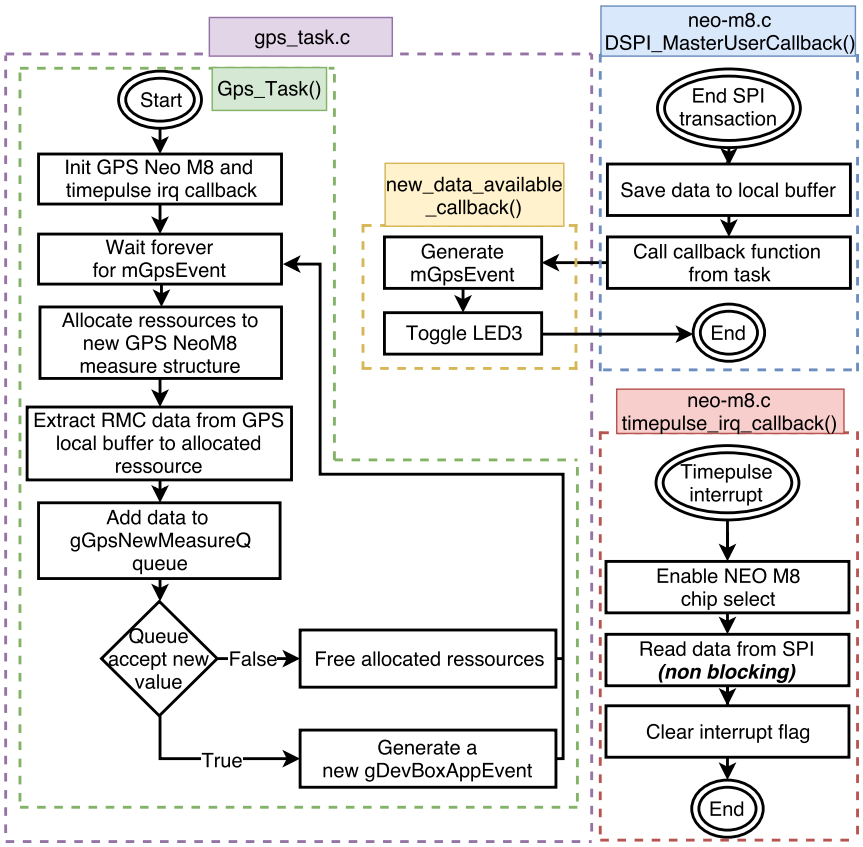


FIGURE 6.12 – Diagramme de fonctionnement de la tâche GPS

6.4.4.3 BNO055

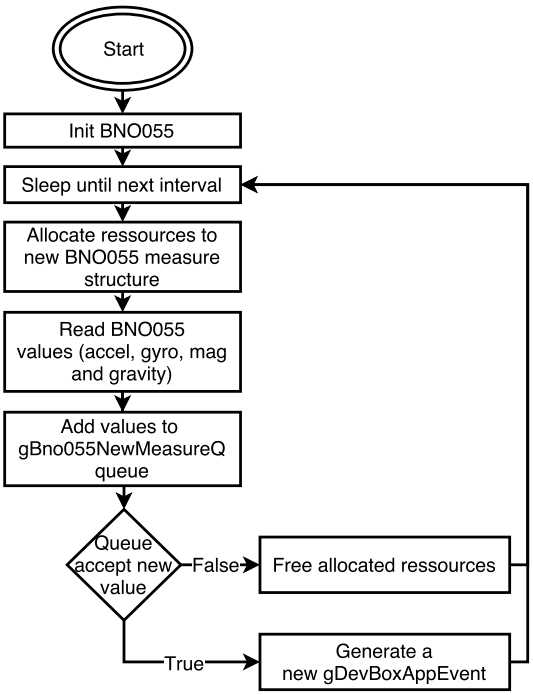


FIGURE 6.13 – Diagramme de fonctionnement de la tâche BNO055

La Figure 6.12 illustre les différentes étapes pour l'acquisition et l'envoi des données pour la tâche du BNO055. L'intervalle entre les mesures est configurable à l'aide d'une fonction appelée lors de l'écriture d'une caractéristique Bluetooth correspondante.

L'implémentation actuelle n'est pas adéquate si l'utilisateur souhaite utiliser une fréquence d'échantillonnage fixe et automatisée par le périphérique. Dans ce cas-là, il faut configurer le BNO055 afin qu'il génère une interruption lorsqu'un *sample* est prêt à être récolté. La gestion de l'interruption a été gérée au niveau de la tâche, il manque juste la configuration du BNO055.

6.4.4.4 BME680

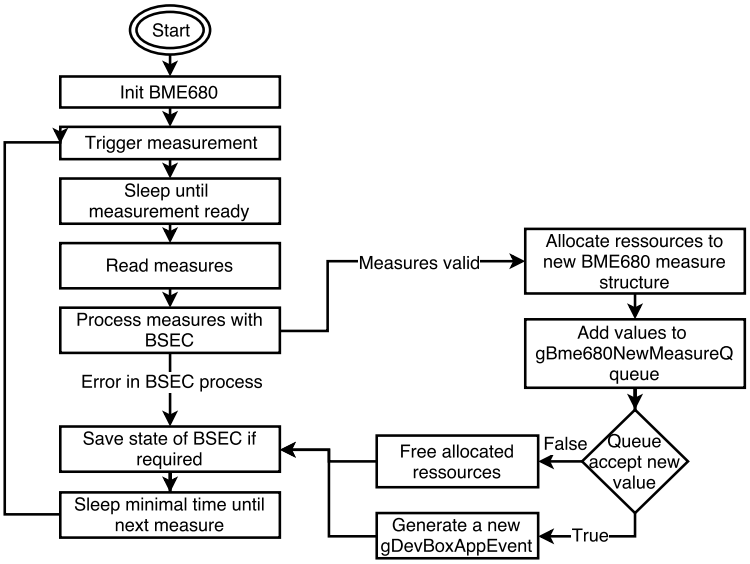


FIGURE 6.14 – Diagramme de fonctionnement de la tâche BME680

La bibliothèque BSEC s'occupe d'une grande partie de l'acquisition et du traitement des données provenant du capteur, comme expliqué en Section 6.4.2.1.5. Il reste donc peu d'éléments à être gérés par la tâche. La Figure 6.14 illustre les différentes étapes pour l'acquisition et l'envoi des données sous forme d'un diagramme.

6.4.4.5 Scanneur Bluetooth Low Energy

Le scanneur Bluetooth n'a pas de tâche propre à lui seul. Il dépend de la tâche *Host* de la *stack* Bluetooth. C'est celle-ci qui appelle une fonction *callback* lorsque de nouveaux périphériques scannés sont disponibles. Il y a donc deux éléments en parallèle. Premièrement, le scanneur qui est directement implémenté par l'hôte et dont le code du *callback* d'un nouveau périphérique BLE est décrit . En deuxième lieu, il y a un *timer* qui récupère le nombre de périphériques qui ont été scannés et qui transfère cette valeur à la tâche principale du système (cf. Figure 6.16). Le lancement du scan s'effectue au démarrage de l'application avec l'appel de la fonction `BleApp_Start`, démontré en Figure 6.16.

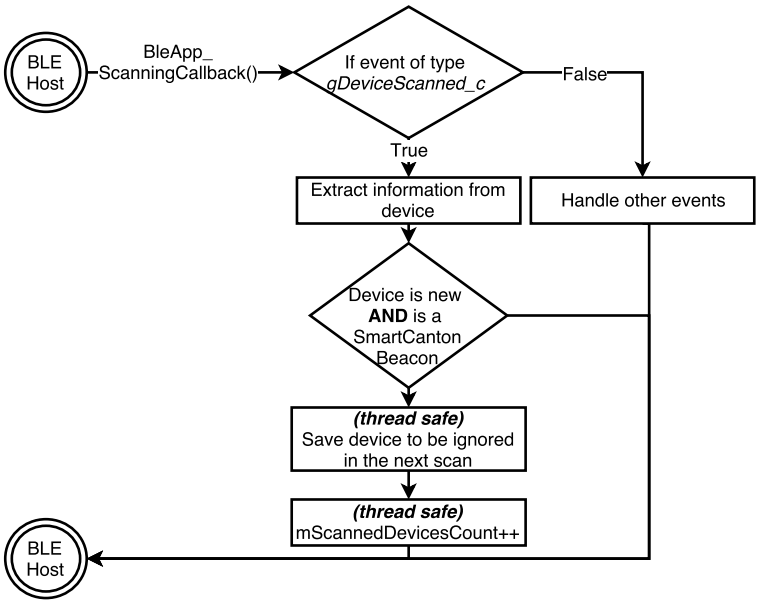


FIGURE 6.15 – Diagramme de fonctionnement de la tâche scanneur Bluetooth lors de la sauvegarde des nouveaux périphériques

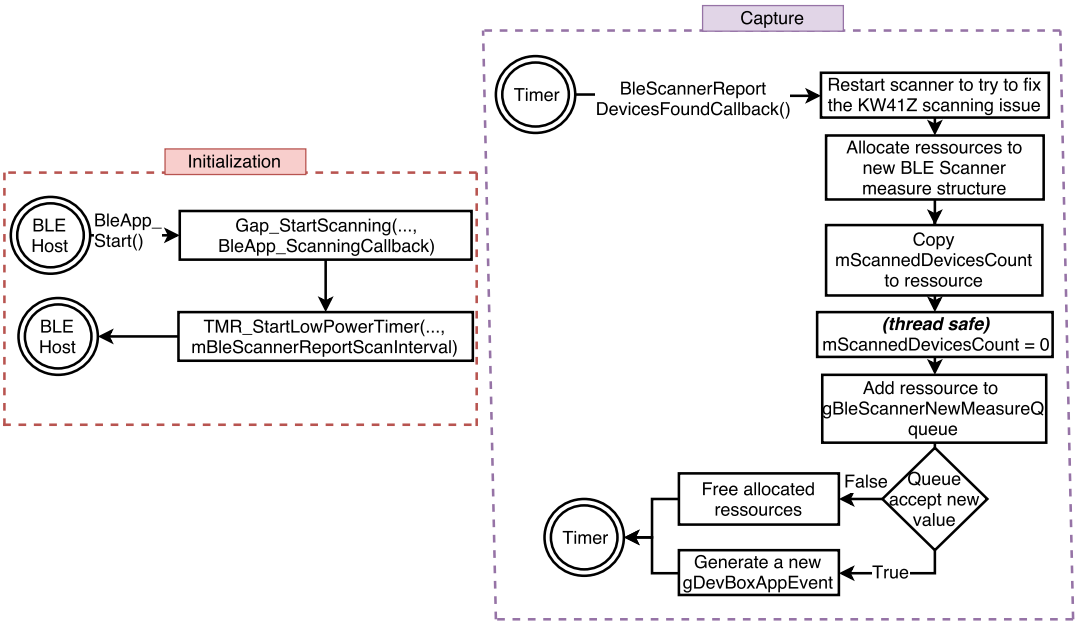


FIGURE 6.16 – Diagramme de fonctionnement de la tâche de scanneur Bluetooth lors de son initialisation et de la capture des données

6.4.4.6 LoRaWAN *Controller*

La tâche LoRaWAN est le nœud central pour communiquer avec le contrôleur LoRaWAN. Son interface est également basée sur des événements présentés sur la Figure 6.10 avec l'utilisateur. C'est elle qui régit l'état lié au contrôleur LoRaWAN avec l'envoi des différentes commandes pour rejoindre et transmettre des données sur le réseau. Tous les messages sont stockés dans une *queue* pour éviter qu'il ne soit perdu si un message est déjà en cours d'envoi, ou que le *duty cycle* maximum a été atteint. Cette tâche notifie également la tâche principale quand le réseau a pu être rejoint correctement afin d'effectuer les actions nécessaires à l'acquisition des capteurs. Le

fonctionnement de l'algorithme de cette tâche est visible sur la Figure 6.17.

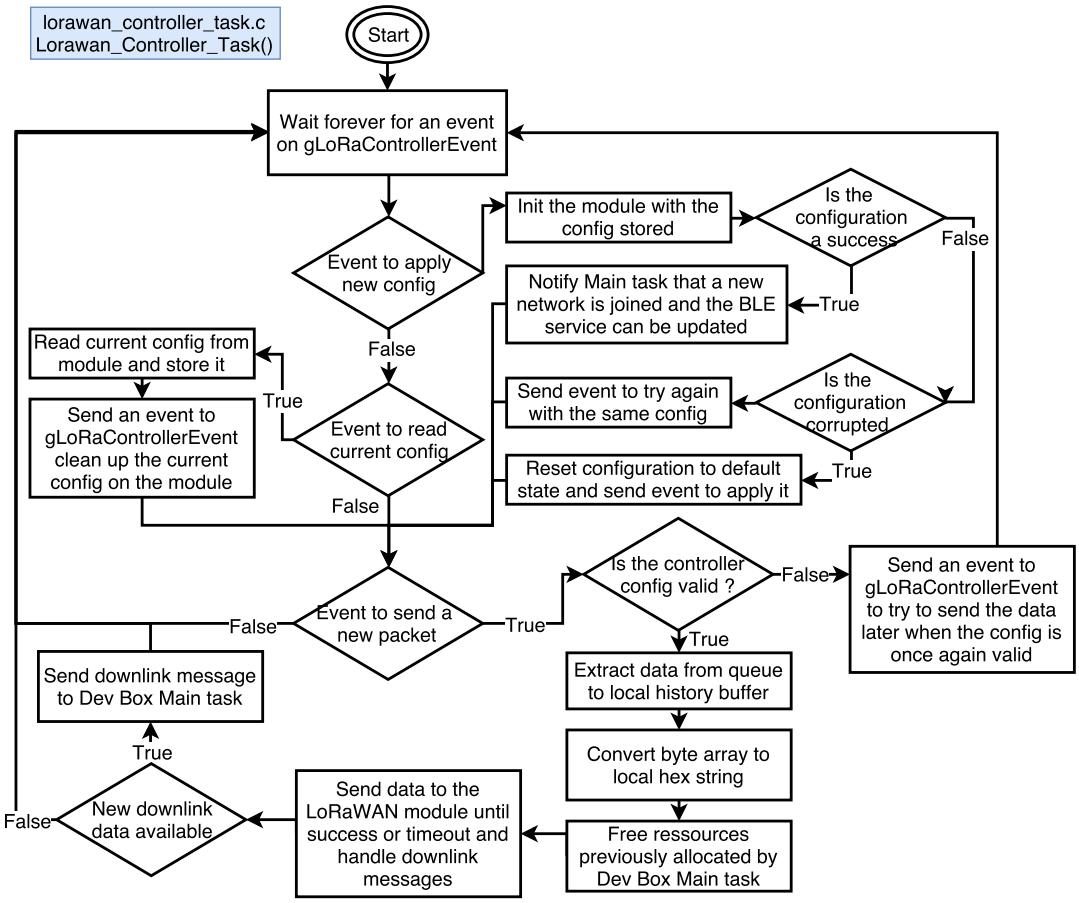


FIGURE 6.17 – Diagramme de fonctionnement de la tâche `LoRaWAN Controller` lors de la gestion des événements

6.4.5 KW41Z *stack* Bluetooth

La *stack* Bluetooth du KW41Z suit le modèle présenté en Section 3.1. Celle-ci n'est pas open source, ce qui peut parfois complique le développement quand une erreur se produit à l'intérieur de celle-ci. L'utilisateur a très peu de contrôle sur l'interface contrôleur. Les tâches qui gèrent le contrôleur et l'hôte ne sont pas directement modifiables par l'utilisateur, contrairement à la couche application qui est entièrement open source et dont plusieurs exemples sont proposés.

Le SDK fournit par NXP est personnalisable par l'utilisateur à l'aide d'un portail en ligne sur le site de NXP, nommé *MCUXpresso SDK Builder*, et visible sur la Figure 6.18. L'utilisateur peut ainsi choisir quel microcontrôleur utiliser et personnaliser les différentes bibliothèques qui sont ajoutées au SDK. Ce SDK est ensuite compilé sur la plateforme en ligne et dans un second temps, un e-mail est ensuite envoyé quand la compilation est achevée. Une archive doit pour la suite être téléchargée et ajoutée à l'IDE MCUXpresso. Divers exemples de projets sont disponibles pour des cartes de développement impliquant le microcontrôleur choisi et ainsi accélérer le développement pour le programmeur. Dans le cadre de ce travail, le projet de base utilisé est celui nommé `frdmkw41z_wireless_examples_bluetooth_heart_`

`rate_sensor_freertos` basé sur la carte de développement FRDMKW41Z présentée en Section 5.3.1.3.

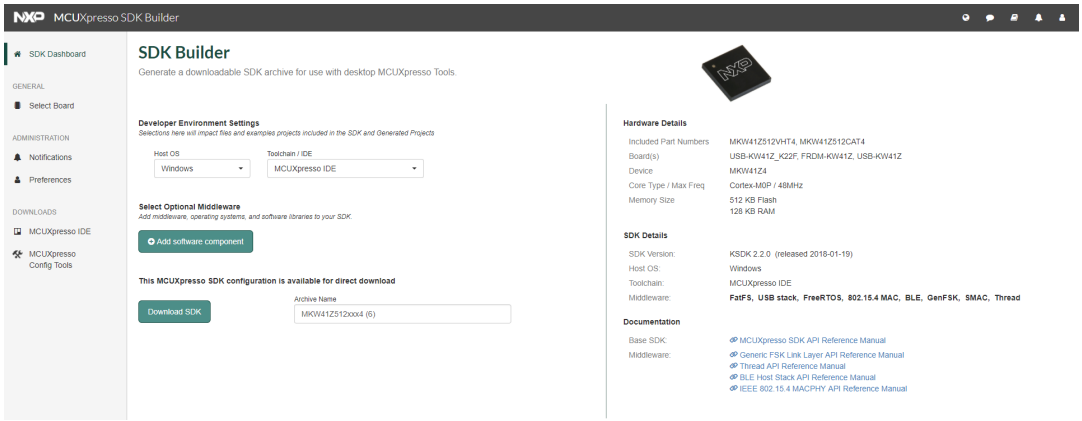


FIGURE 6.18 – MCUXpresso SDK Builder interface

6.4.6 Services Bluetooth

La déclaration des services Bluetooth s'effectue à l'aide de deux fichiers nommés `gatt_uuid128.h` et `gatt_db.h`. Le premier contient uniquement les UUID des différents attributs des services Bluetooth. Ces UUIDs sont créés à l'aide d'une macro nommée `UUID128` ayant comme paramètre une variable qui contiendra le UUID, suivi d'un tableau de bytes de 16 emplacements. Le deuxième fichier est celui qui permet de créer toute la structure des services Bluetooth, avec la spécification des services, caractéristiques et descripteurs nécessaires.

Une partie de ces fichiers peut être générée à l'aide d'un *plug-in* de Freescale pour le logiciel Bluetooth Developper Studio [1] (BDS). Toutefois, ce *plug-in* n'a pas été mis à jour depuis plusieurs versions du SDK. Celui-ci remonte à la version 1.3, alors que la version actuellement supportée est la version 2.2. BDS est développé par le Bluetooth *Special Interest Group* (SIG) et a pour but d'accélérer l'utilisation de profils standards (ex. *Hearth Rate Sensor* HRS)) ou des profils personnalisés. Le logiciel est téléchargeable gratuitement à l'adresse suivante :

`https://www.bluetooth.com/download-developer-studio`

Dans ce projet, tous les services ont été créés à l'aide de cet outil. Si un jour le microcontrôleur utilisé n'est plus le KW41Z, mais un autre, par exemple un NRF52, il est très facile d'exporter les profils personnalisés. Le projet de BDS est disponible dans le dépôt GitHub, dans le répertoire `dev/BluetoothDeveloperStudio`. L'interface de ce projet dans BDS est visible sur la Figure 6.19.

En section Section 3.1.5.3.6, les descripteurs de type CCCD ont été évoqués, de même que la liste des différents descripteurs attribuables aux caractéristiques. Il existe des descripteurs de type *Characteristic User Description Descriptor* (CUDD), qui sont entièrement personnalisables par l'utilisateur. En effet, comme on peut le voir sur le Tableau 6.2, la seule chose qui est imposée est le format, un string UTF8.

1. `https://www.bluetooth.com/develop-with-bluetooth/developer-resources-tools/developer-kits/bluetooth-developer-plugins`

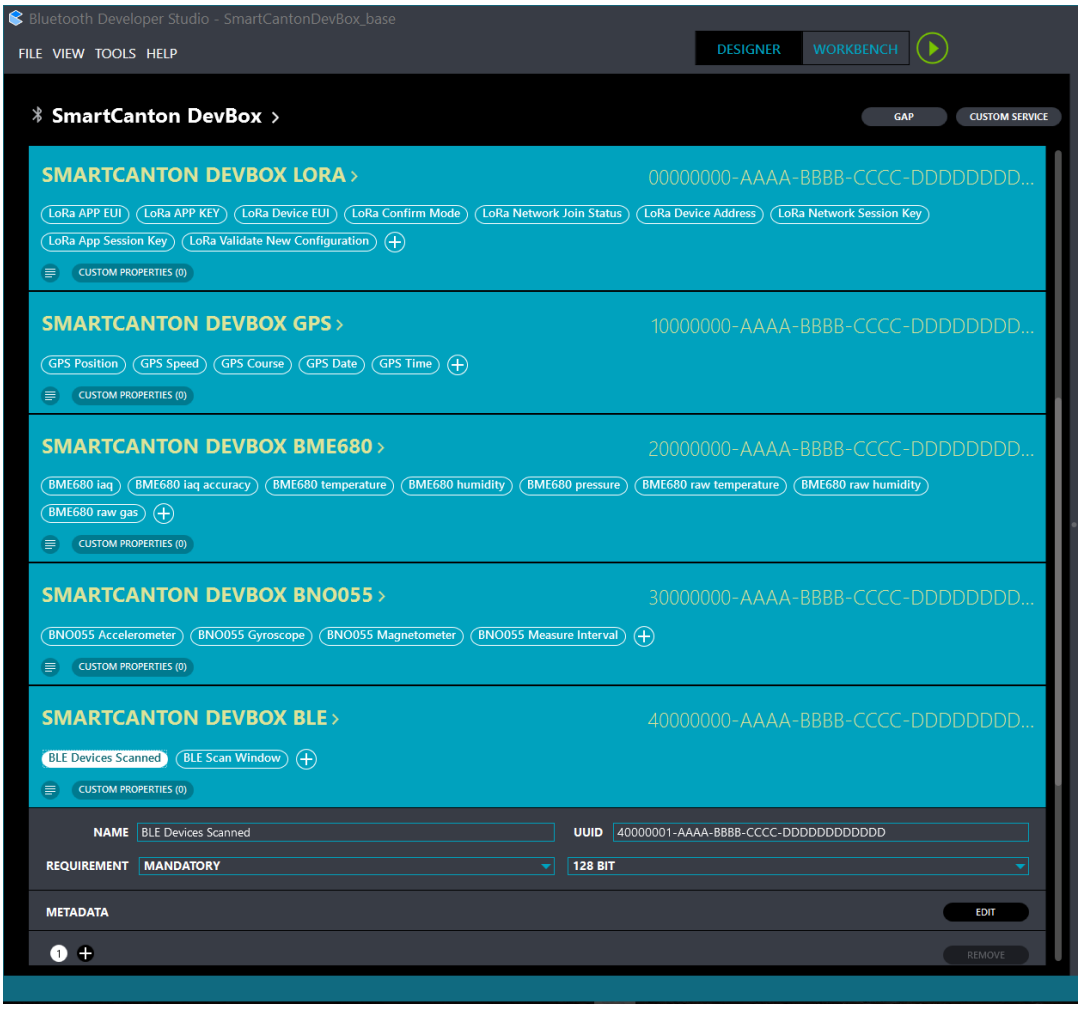


FIGURE 6.19 – Bluetooth Developer Studio avec les services BLE du projet SmartCanton DevBox

Ce descripteur a donc été utilisé dans ce projet, afin de stocker un string décrivant la fonction de chaque caractéristique. Le code hexadécimal de ce type de descripteurs est `0x2901`.

TABLE 6.2 – Paramètres d'un *Characteristic User Description Descriptor*

Bit Field Characteristic User Description Descriptor (CUDD)			
Format	Minimum Value	Maximum Value	Additional Information
utf8s	N/A	N/A	None

6.4.6.1 Création d'un profil personnalisé

Le fichier `gatt_db.h` utilise des macros pour définir les différents attributs d'un service. Voici par exemple la définition du service GPS avec sa caractéristique de lecture de la position, nommée `char_gps_position`, et ses deux descripteurs (CUDD et CCCD) :

gatt_db.c

```
78  PRIMARY_SERVICE_UUID128(service_smartcanton_devbox_gps,
    ↪  uuid_service_smartcanton_devbox_gps)
79          CHARACTERISTIC_UUID128(char_gps_position, uuid_gps_position,
            ↪  (gGattCharPropNotify_c | gGattCharPropRead_c))
80                  VALUE_UUID128_VARLEN(value_gps_position, uuid_gps_position,
                    ↪  (gPermissionFlagReadable_c), 23, 0, "")
81                  CCCD(cccd_gps_position)
82                  DESCRIPTOR(descriptor_gps_position, 0x2901,
                    ↪  (gPermissionFlagReadable_c), 12, "GPS Position")
```

La ligne 78 définit le service avec son *handle*, nommé `service_smartcanton_devbox_gps`, assigné au UUID défini par la variable `uuid_service_smartcanton_devbox_gps`. La ligne 79, quant à elle, définit la caractéristique, son *handle* et son UUID, ainsi que les permissions d'accès à la caractéristique. La caractéristique `char_gps_position` est accessible en tant que notification, ainsi qu'en lecture simple de la dernière valeur envoyée en notification. Puisque les notifications sont autorisées sur cette caractéristique, il est indispensable de rajouter le descripteur CCCD en ligne 81. Finalement, le dernier descripteur est le CUDD, qui ne dispose pas de sa macro personnalisée comme le CCCD et doit être déclaré manuellement.

En utilisant le SDK de NXP, les services Bluetooth sont défini dans le répertoire `<project_name>/bluetooth/profiles/`. Dans le cadre de ce projet, tous les services liés au smartcanton ont été créés de rien en analysant les divers exemples proposés par NXP. La Figure 6.20 expose la hiérarchie des profils Bluetooth. Les services `Battery` et `Device Info` ont été repris et adaptés du projet `Heart Rate Sensor`.

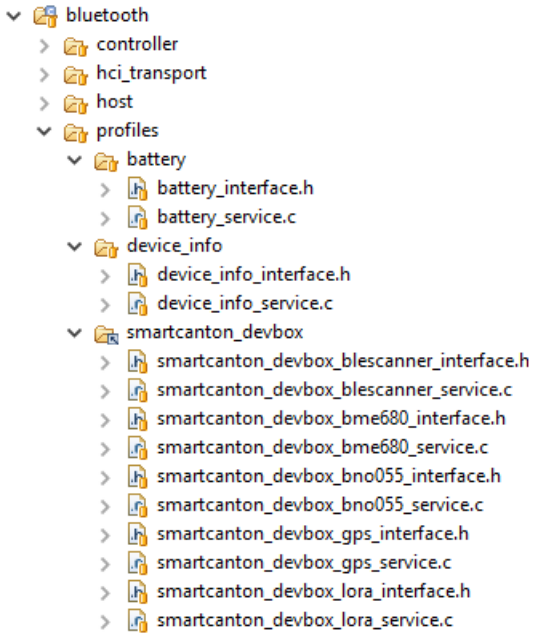


FIGURE 6.20 – Fichiers de déclaration des services Bluetooth

6.4.6.1.1 Fonctions minimales d'un profil

Sur le KW41Z, deux méthodes doivent être implémentées dans chaque profil. La première est la fonction start qui sera appelée lors de l'initialisation de l'application, avec la réception de l'événement `gInitializationComplete_c` dans le *callback* de l'hôte, nommé `BleApp_GenericCallback` :

dev_box_app_task.c

```
536 static void BleApp_Config()
537 {

    ...

559         ScDbLoRa_Start(&scdbLoRaServiceConfig);
```

L'unique but de cette fonction est d'initialiser l'ID du périphérique connecté avec la valeur `gInvalidDeviceId_c`. Les deux autres fonctions se nomment `subscribe` et `unsubscribe`. L'objectif de celles-ci est d'enregistrer la connexion au service d'un nouveau maitre et informer les applications, qui doivent diffuser des données, que quelqu'un est peut-être intéressé à les recevoir. L'ID est simplement stockée en local dans le profil et est libéré lorsque le maitre se déconnecte :

smartcanton_devbox_gps_service.c

```
79  bleResult_t ScDbGPS_Subscribe(deviceId_t deviceId)
80  {
81          /* Subscribe a new client to this service */
82          mScDbGPS_SubscribedClientId = deviceId;
83
84          return gBleSuccess_c;
85  }

    ...

87  bleResult_t ScDbGPS_Unsubscribe()
88  {
89          /* Clear current subscribed client ID */
90          mScDbGPS_SubscribedClientId = gInvalidDeviceId_c;
91          return gBleSuccess_c;
92  }
```

Aucun exemple prodigué par NXP ne supporte le multi maitre. Cependant, la *datasheet* du KW41Z spécifie que deux maitres peuvent être connectés simultanément. On peut voir que cette implémentation n'est pas pensée pour gérer deux connexions. Toutefois, pour la DevBox, cette multiple connexion n'a pas été activée dans les options du SDK, car elle n'a pas forcément de sens pour le projet.

6.4.6.1.2 Écritures et lecture depuis la base de donnée GATT

La base de donnée GATT est régie par une suite d'interfaces très complètes contenues dans le répertoire `<project_name>/bluetooth/interface/`. Lorsque l'utilisateur souhaite sauvegarder des données dans la table GATT, il dispose d'une fonction nommée *GattDb_WriteAttribute*. L'entête de cette fonction est disponible ci-dessous :

gatt_db_app_interface.h

```
78  bleResult_t GattDb_WriteAttribute
79  (
80      uint16_t    handle,
81      uint16_t    valueLength,
82      uint8_t*    aValue
83  );
```

Le premier paramètre est l'*handle* de la caractéristique du service désiré à l'aide d'une fonction nommée `GattDb_FindCharValueHandleInService`. La nouvelle valeur de la caractéristique est ensuite écrite à l'aide des deux paramètres suivants. Si les données écrites sont fausses ou que la caractéristique est invalide, la fonction retourne

une erreur. La lecture est également réalisée à l'aide d'une fonction du même fichier, en spécifiant des paramètres similaires :

gatt_db_app_interface.h

```
99  bleResult_t GattDb_ReadAttribute
100 (
101     uint16_t    handle,
102     uint16_t    maxBytes,
103     uint8_t*    aOutValue,
104     uint16_t*   pOutValueLength
105 );
```

6.4.6.1.3 Notifications

Dans le cadre de ce projet, uniquement les notifications ont été implémentées. Les indications sont plus complexes à mettre en place, car il faut créer manuellement des machines d'états et des *queues* pour l'envoi et ensuite l'attente de la confirmation (cf. Section 3.1.5.3.6 pour la différence entre notifications et indications). Les notifications n'ont jamais été utilisées avec des données critiques où l'ont doit être sûr d'avoir la réception sur le maitre, cet *overhead* était alors inutile dans ce projet.

Une fois les données des caractéristiques enregistrées dans la base de données avec la fonction `GattDb_WriteAttribute`, celles-ci peuvent être notifiées aux clients qui se sont enregistrés en écrivant dans le CCCD. Cette opération de notification est activée à l'aide de la fonction `GattServer_SendNotification` :

gatt_server_interface.h

```
268 bleResult_t GattServer_SendNotification
269 (
270     deviceId_t              deviceId,
271     uint16_t                handle
272 );
```

Le premier paramètre spécifié est l'identifiant du maitre et le deuxième est l'*handle* de la caractéristique.

Lorsque le flux des données est trop important, il est inutile de les sauvegarder. Cela affecte les performances du microcontrôleur puisque celles-ci doivent être à chaque fois copiées dans la base de données. Par exemple, dans le cas de la capture des données provenant d'un accéléromètre, la fréquence d'acquisition peut attendre plusieurs centaines d'Hertz ce qui crée des écritures inutiles dans la base de données. Le framework propose donc la possibilité de notifier l'utilisateur sans sauvegarder les informations. Ceci s'effectue à l'aide de la fonction `GattServer_SendInstantValueNotification`. Voici l'entête de cette fonction :

gatt_server_interface.h

```
302 bleResult_t GattServer_SendInstantValueNotification
303 (
304     deviceId_t              deviceId,
305     uint16_t                handle,
306     uint16_t                valueLength,
307     uint8_t*                aValue
308 );
```

Les paramètres sont similaires à la fonction précédente, mais cette fois, la nouvelle doit être spécifiée lors de l'appel. Cette fonction a été utilisée dans ce projet pour tous les services Bluetooth qui ont des caractéristiques listées comme étant uniquement

accessibles en tant que notifications (ex. les données du BNO055, cf. Section 6.4.6.2.5). Le KW41Z ne supporte qu'au maximum 16 notifications par maitre connecté et ce chiffre affecte directement toutes les notifications, même si le maitre ne s'abonne pas à celles-ci. Cette limitation ne peut pas être modifiée, cela a donc limité le nombre de caractéristiques qui ont pu être assignées comme notifications pour l'ensemble du projet.

6.4.6.2 Services BLE disponibles sur la DevBox

La documentation de tous les services Bluetooth implémentés dans ce projet est disponible en Annexe A sous forme de tableaux décrivant tous les UUIDs, les droits d'accès ainsi que des exemples de données fournies par les divers services. Les quatre sous-sections qui suivent expliquent plus en détail les différents services, de même que leurs rôles dans ce projet. Ces services peuvent ainsi être accédés par n'importe quel client BLE, par exemple, en utilisant l'application NRF Connect sur Android (cf. Figure 6.21).

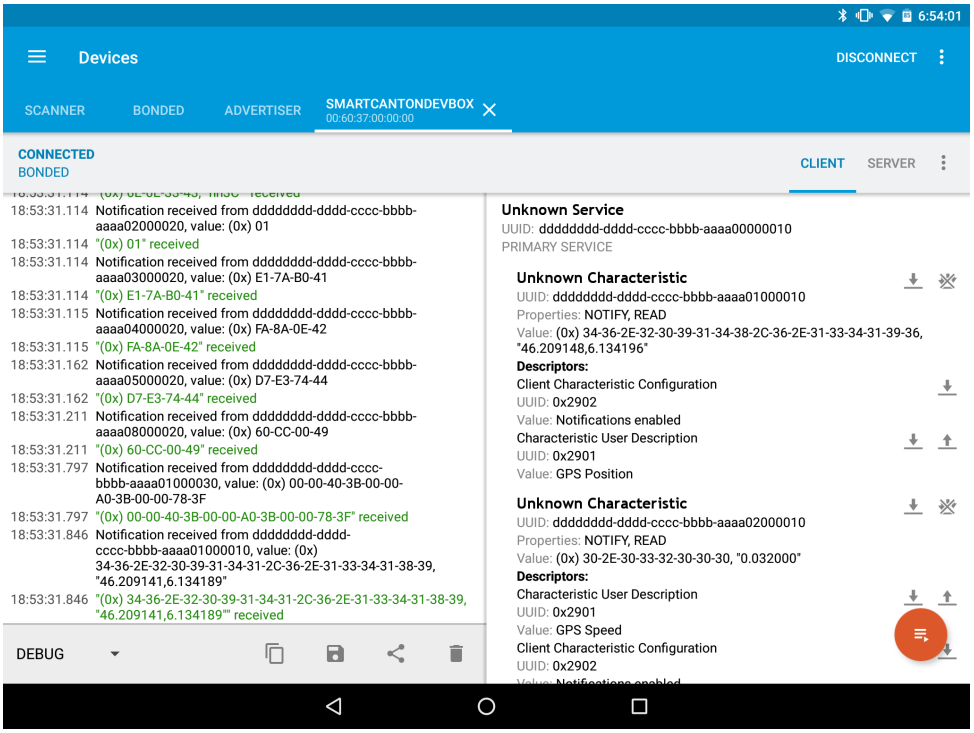


FIGURE 6.21 – Lecture des caractéristiques du service de GPS à l'aide de l'application *NRF Connect*

6.4.6.2.1 *Generic Access*, *Generic Attribute* et *Battery*

Ces trois services utilisent les profils Bluetooth standards et n'ont donc pas été modifiés pour être utilisés sur ce projet. `Generic Access` permet uniquement d'indiquer quelques informations sur le périphérique connecté, comme les différentes versions du *firmware*, *hardware* ou le constructeur. Il peut ainsi être utilisé par le maitre pour garantir que le périphérique connecté est bien une DevBox et non un autre périphérique. Le contenu de ces services est visible en annexe sur le Tableau A.1, Tableau A.2 et Tableau A.8.

Le service de batterie est celui qui est fourni par l'exemple *Hearth Rate Sensor* de NXP. Pour notifier l'état de la batterie, c'est l'application qui actualise la valeur de celle-ci toutes les 30 secondes à l'aide d'un *timer*. Cette mise à jour s'effectue dans la tâche principale du projet (DevBox Main Task).

6.4.6.2.2 SmartCanton DevBox LoRaWAN

Ce service est l'un des plus importants de la DevBox, car il permet la mise à jour des différents paramètres de la connexion LoRaWAN. Les opérations autorisées sur les différentes caractéristiques sont décrites sur le Tableau A.3 en annexe. Sur ce tableau, les caractéristiques listées avec le tag `[DEBUG]` signifient que celles-ci ne sont accessibles que pour le développement actuel. Elles sont placées en lecture seule si la DevBox doit être utilisée pour un cas d'utilisation définitif. Voici une liste des différentes caractéristiques implémentées :

1. LoRaWAN App EUI ;
2. LoRaWAN AppKey ;
3. LoRaWAN Device EUI ;
4. LoRaWAN mode de confirmation des paquets (confirmés ou non) ;
5. LoRaWAN statu du réseau (rejoint ou non) ;
6. LoRaWAN *Device Address* assignée par le réseau ;
7. LoRaWAN Network Session Key ;
8. LoRaWAN App Session Key ;
9. Validation de la configuration appliquée. Cette dernière option permet de valider une configuration une fois que toutes les caractéristiques ont été écrites. Celle-ci va annuler la configuration actuelle du module LoRaWAN afin d'appliquer la dernière configuration entrée par l'utilisateur.

6.4.6.2.3 SmartCanton DevBox GPS

Le service Bluetooth implémenté pour le GPS permet de recevoir les différentes informations provenant de celui-ci. A l'heure actuelle, uniquement les données des trames de type RMC sont supportées [64].

Toutes les informations sont disponibles en tant que notifications. Comme le débit n'est pas très élevé (une fois par seconde), la dernière valeur de chaque caractéristique est stockée dans la base de données. Ceci permet également de savoir quand la dernière valeur GPS a pu être capturée (que ce soit la position ou le temps). Voici la liste des différentes caractéristiques qui ont été implémentées :

1. Position avec la latitude et longitude en degrés ;
2. Vitesse en m/s ;
3. Trajectoire ;
4. Date en format `hh:mm:ss` ;
5. Heure en format `hh:mm:ss`.

Le service complet est disponible sur le Tableau A.4 en annexe. Pour simplifier le débogage, les données sont toutes en format string UTF8. Le traitement rajouté sur les données est faible, car elles sont en partie récupérées sous ce format directement depuis la trame RMC transmise par le GPS. Cela alourdit l'envoi à travers la communication Bluetooth, mais le débit qui est actuellement d'une trame par seconde reste négligeable.

6.4.6.2.4 SmartCanton DevBox BME680

Le capteur de température, d'humidité, de pression et de qualité de l'air dispose d'un grand nombre de données accessibles. Cependant, puisque le KW41Z ne supporte que 16 caractéristiques en notifications, des concessions ont été faites sur les flux de données disponibles. Voici la liste des caractéristiques implémentées :

1. *Indoor Air Quality* (IAQ) facteur de 0 à 400 ;
2. Qualité de l'IAQ en échelle de 0 à 3 (0 : données inutilisables et une calibration est nécessaire, 3 : capteur parfaitement calibré et données parfaites) ;
3. Température en °C ;
4. Humidité en % ;
5. Pression en hPa ;
6. Température brute en °C (sans le post traitement de la bibliothèque BSEC) ;
7. Humidité brute en % (sans le post traitement de la bibliothèque BSEC) ;
8. Mesure du GAS brut en ohms (sans le post traitement de la bibliothèque BSEC).

Le service complet est disponible sur le Tableau A.5 en annexe.

6.4.6.2.5 SmartCanton DevBox BNO055

Le BNO055 peut fournir beaucoup de données, mais il a été décidé de seulement garder l'accès aux trois capteurs principaux (accéléromètre, gyroscope et magnétomètre). Voici la liste des caractéristiques implémentées :

1. Trois axes de l'accéléromètre en g codé chacun sur 4 bytes pour un total de 12 bytes ;
2. Trois axes du gyroscope en degrés par secondes codé chacun sur 4 bytes pour un total de 12 bytes ;
3. Trois axes du magnétomètre en microtesla codé chacun sur 4 bytes pour un total de 12 bytes ;
4. Le temps entre chaque nouvelle mesure. L'utilisateur peut choisir un temps de 300 à 10000 ms pour être à nouveau notifié.

Le service complet est disponible sur le Tableau A.6 en annexe. Actuellement l'envoi des données n'est pas optimisé pour les grands débits, c'est pour cela que la configuration entre chaque mesure est limitée à 300ms. Pour pallier à cette limitation, il faut envoyer plusieurs échantillons dans une même notification afin d'éviter l'*overhead* généré par les paquets BLE.

6.4.6.2.6 SmartCanton DevBox BLE Scanner

Le scanneur Bluetooth est basique, il ne propose actuellement pas beaucoup de personnalisation. Voici la liste des caractéristiques implémentées :

1. Nombre de périphériques scannés lors de la dernière fenêtre de capture ;
2. Le temps entre chaque nouvelle fenêtre de capture. L'utilisateur peut choisir un temps de 10 à 600 secondes pour être à nouveau notifié.

Le service complet est disponible sur le Tableau A.7 en annexe.

6.4.7 Sécurité

La *stack* BLE du KW41Z s'occupe d'une grande partie de la sécurité en interne. Les sous-sections qui suivent expliquent les différents paramètres configurés afin de suivre les recommandations du NIST citées en Section 4.1.3.

La méthode d'authentification retenue a été le *passkey*. Celui-ci est généré aléatoirement pour chaque DevBox et programmé dans celle-ci. En section Section 4.1, il a été vu que si un périphérique souhaite forcer l'utilisation d'un *passkey*, il doit indiquer qu'il dispose d'un écran pour afficher ce dernier, même si cette information est fausse. Cela aura pour conséquence de forcer l'initiateur de la connexion à demander un *passkey*. Cette demande de *passkey* ne s'effectue que si l'initiateur dispose d'un écran et d'un clavier, ce qui est le cas pour un smartphone. Si ce n'est pas le cas, la connexion est refusée, car le mode 1 niveau 4 a été activé et exigeant une connexion authentifiée et sécurisée. Le NIST (cf. Section 4.1.3) conseille uniquement de ne pas utiliser de *passkey* simple. Dans le cas présent, le *passkey* est différent pour chaque périphérique. Le *passkey* est ensuite uniquement accessible sur un serveur sécurisé. On peut voir cela comme un autre type d'échange d'information de type Out Of the Band (OOB). La solution la plus sure serait d'échanger directement vraies LTK Bluetooth à l'aide du serveur et de les programmer dans le dispositif souhaitant se connecter sur la DevBox. Toutefois, dans ce projet un smartphone a été utilisé, et ce type d'injection de clés n'est pas possible dans Android sans être en mode développeur ou root [66].

Si l'attaquant arrive à *brute force* le *passkey*, celui-ci peut alors se connecter sur le périphérique. Toutefois, celui-ci ne peut pas lire les clés, car celles-ci sont en lecture seules. Il peut tout de même modifier certains paramètres et également changer le périphérique de réseau LoRaWAN. Néanmoins, dans un cas comme celui-là, une alerte via un message *uplink* LoRa peut être générée pour indiquer une nouvelle connexion Bluetooth.

6.4.7.1 Adresses MAC Bluetooth

Les adresses MAC doivent être uniques pour tous les périphériques Bluetooth. Celle-ci doit être configurée manuellement à l'aide de la macro `BD_ADDR` :

app_preinclude.h

```
155 #define BD_ADDR            0x00,0x00,0x00,0x37,0x60,0x00
```

NXP ne fournit pas d'adresses MAC pour les périphériques. Celles-ci doivent être achetées auprès des autorités d'assignation régies par l'IEEE [1]. Les recommandations du NIST (cf. Section 4.1.3) demandent de garder une trace de toutes ces adresses MAC. Donc par cet achat, cela permet de connaitre la plage d'adresses MAC assignée aux différents périphériques vendus ou développés.

6.4.7.2 Sécurité des services Bluetooth

La sécurité des services Bluetooth a été placée en suivant les recommandations du NIST exposées en Section 4.1.3. Il s'agit de mettre la sécurité en Mode 1 et Level 3. La stack Bluetooth du KW41Z est configurée avec les des différents paramètres qui suivent :

1. https://standards.ieee.org/develop/regauth/grpmac/index.html

app_config.c

```
166 /* Device Security Requirements */
167 static const gapSecurityRequirements_t masterSecurity = {
168                 gSecurityMode_1_Level_4_c,
169         FALSE,
170         gDefaultEncryptionKeySize_d
171 };
172
173 /* Device Security Requirements */
174 static const gapServiceSecurityRequirements_t serviceSecurity[0] = {
175 };
176
177 gapDeviceSecurityRequirements_t deviceSecurityRequirements = {
178     .pMasterSecurityRequirements    = (void*)&masterSecurity,
179     .cNumServices                   = 0,
180     .aServiceSecurityRequirements   = (void*)serviceSecurity
181 };
```

La *stack* Bluetooth permet également de mettre en place des niveaux de sécurité personnalisés pour trois services à l'aide de l'option `aServiceSecurityRequirements`. Si aucun service n'est spécifié dans cette personnalisation, c'est la configuration `pMasterSecurityRequirements` qui est appliquée à tous les services. Cette option de sécurité personnalisée n'a pas été utilisé dans ce projet, mais on peut imaginer laisser un service sans sécurité, afin d'informer les utilisateurs d'informations non critiques concernant le périphérique.

6.4.7.3 *Passkey* Bluetooth Low Energy

Le *passkey* du Bluetooth doit être généré lors de la programmation des cartes DevBox. La définition de celui-ci s'effectue à l'aide de la ligne suivante :

app_preinclude.h

```
35 /*! Passkey asked when bonding with the device */
36 #define gPasskeyValue_c                111100
```

Le *passkey* doit être de minimum 6 digits et doit être unique pour chaque DevBox. Il n'y a actuellement pas d'autre possibilité de le changer à l'exception de cette ligne de code. Malheureusement, il n'est pas possible de suivre la recommandation du NIST au sujet du *passkey* aléatoire à chaque connexion (cf. Section 4.1.3).

6.4.7.4 Stockage des clés LoRaWAN

Dans l'idéal, les clés LoRaWAN doivent être stockées dans le module LoRaWAN. Cependant, l'implémentation n'a pas pu être faite dans le temps imparti à ce travail. À l'heure actuelle, les clés sont stockées dans le microcontrôleur KW41Z en flash à l'aide de la structure `lorawanControllerConfiguration_t` présentée en Section 6.4.2.2.4.

6.5 Application Android SmartCanton Manager

L'application Android, nommée SmartCanton Manager, a pour but de créer un pont entre deux technologies, comme illustré en Figure 6.22 mais également de créer un canal d'échange sécurité pour le transfert de l'AppKey LoRaWAN.

L'application offre également la possibilité de tester les différents capteurs présents sur une DevBox en se connectant aux différents services Bluetooth présentés en Section 6.4.6.2. La récupération des données s'effectue à l'aide de lectures BLE ou via des flux de notifications BLE.

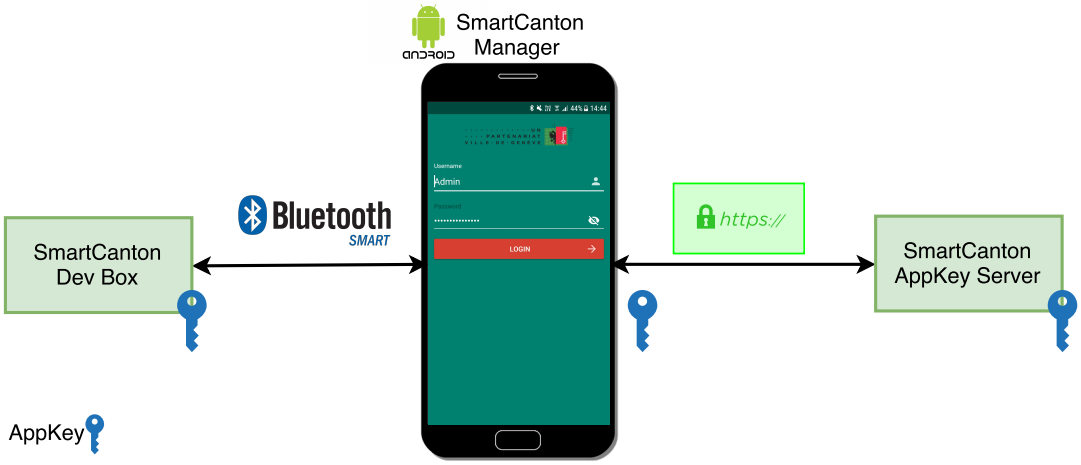


FIGURE 6.22 – Rôle de l'application mobile

6.5.1 Frameworks utilisés

Afin d'accélérer la programmation sur Android, deux frameworks majeurs ont été utilisés. Ceux-ci sont listés dans les sous-sections qui suivent.

6.5.1.1 SweetBlue

Android est censé s'abstraire du matériel afin d'offrir une interface générique pour chaque plateforme utilisée, surtout sur les smartphones. Néanmoins, le comportement du Bluetooth diffère encore entre les différents périphériques Android utilisés. Par exemple, Samsung bride la fréquence des scans Bluetooth possible sur ses smartphones lorsque ceux-ci sont verrouillés. Cette décision a été prise pour réduire la consommation énergétique des applications qui s'exécutent en arrière-plan sur un smartphone. Ceci n'est pas le cas sur la plupart des autres constructeurs. La *stack* Bluetooth sur Android est également très *bas niveau*. Par exemple, si on souhaite lire plusieurs caractéristiques BLE à la suite, l'utilisateur doit lui-même créer une file de lecture avec la gestion de toutes les erreurs, car deux lectures ne peuvent pas être effectuées en parallèle. Voici une liste complète des divers problèmes qui sont aujourd'hui encore non résolus sur les diverses versions d'Android :

https://github.com/iDevicesInc/SweetBlue/wiki/Android-BLE-Issues

Pour pallier à cela, une bibliothèque nommée SweetBlue a été développée par l'entreprise iDevicesInc [1]. Celle-ci a pour but de simplifier l'intégration du Bluetooth

1. https://idevicesinc.com/sweetblue/

dans une application et d'essayer de corriger certaines instabilités. La bibliothèque est gratuite dans son intégralité pour un usage privé même si l'application finale n'est pas vendue à des utilisateurs.

6.5.1.2 Retrofit

Sur Android, la gestion des appels asynchrones peut vite devenir compliquée et répétitive. Retrofit est un client *type-safe* pour les API REST sur Java, avec un accent prononcé pour Android. Cette bibliothèque est développée par l'entreprise Square [1], spécialisée dans les logiciels *open sources*. Cette bibliothèque fournit un framework complet et simple à utiliser pour interagir avec les APIs et recevoir des requêtes avec le client OkHttp [2].

La gestion des erreurs HTTP sont directement traitées par le framework à l'aide de *callbacks* avec les méthodes `success` et `failure`.

6.5.2 Manuel d'utilisation de l'application

L'Annexe C contient un manuel d'utilisation de l'application Android. Avec toutes les étapes qui sont nécessaires pour se connecter au serveur, lister les périphériques Bluetooth à proximité, ainsi que la connexion avec ces derniers.

6.5.3 Activités

Pour ne pas surcharger la présentation de cette section, toutes les captures d'écran de l'application sont présentées dans le manuel d'utilisateur situé à l'Annexe C.

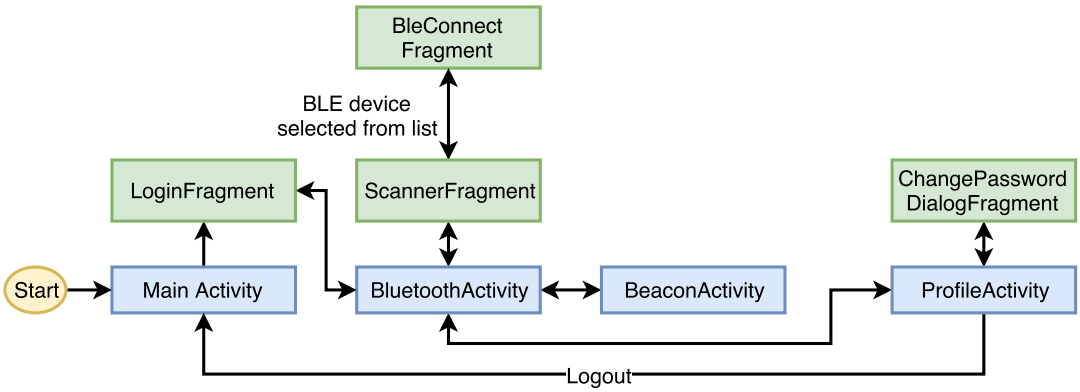


FIGURE 6.23 – Diagramme d'interconnexions entre les différentes activités Android

L'architecture générale de l'application avec la présentation des diverses activités et fragments est visible sur la Figure 6.23. On constate que l'application n'est composée que de quatre activités. Voici leurs différents rôles :

— `MainActivity` : affiche le fragment `LoginFragment` qui offre la possibilité à l'utilisateur de s'authentifier auprès du serveur, afin de recevoir un *token* JWT (cf. Figure C.1). Une fois le login accepté, l'activité `BluetoothActivity` est appelée.

— `BluetoothActivity` : gère les diverses communications Bluetooth entre les différents périphériques. Cette activité peut afficher deux fragments :

1. http://square.github.io/retrofit/
2. http://square.github.io/okhttp/

1. `ScannerFragment` : fragment instancié par défaut au démarrage de l'activité et a pour but de lister tous les périphériques Bluetooth Low Enerny à proximité (cf. Figure C.2). Les dispositifs sont affichés dans une liste et peuvent être sélectionnés par l'utilisateur.
2. `BleConnectFragment` : fragment affiché lorsque l'utilisateur sélectionne un périphérique. Lors du chargement du fragment, une requête vers le serveur est automatiquement envoyée afin de savoir si le dispositif est connu en fonction de l'adresse MAC de ce dernier. Si le périphérique est connu, l'utilisateur peut se connecter au périphérique via Bluetooth et ainsi accéder aux divers périphériques (cf. Figure C.4). Il peut également mettre à jour le périphérique avec les dernières configurations LoRaWAN contenues sur le serveur.

— `BeaconActivity` : ne contient pas de fragment, car elle n'a qu'un seul et unique rôle, qui est de générer un certain nombre de *beacons* afin de simuler des périphériques Bluetooth Low Energy à proximité comme vu en Section 6.2.

— `ProfileActivity` : Cette activité permet d'afficher les informations sur l'utilisateur connecté, ainsi que sur le *token* JWT récupéré lors de la connexion. Ici l'utilisateur peut également se déconnecter de sa session (cf. Figure C.9).

6.5.4 Sécurité

Pour ce qu'il s'agit de la sécurité, tout est directement géré par Android même. Dans le cas des requêtes HTTPS, il suffit de spécifier que l'URL du serveur est en HTTPS pour que toute la communication soit automatiquement chiffrée à l'aide du certificat fournit par le serveur.

Dans le cas du Bluetooth, c'est le périphérique qui décide du mode de sécurité qu'il accepte (cf. Section 4.1). Android s'occupe donc d'effectuer la connexion avec requêtes de celui-ci. La demande de *passkey* est demandée directement par l'OS et ne peut pas être directement validée par l'application. Pour informer l'utilisateur sur la présence du *passkey*, une popup est affichée lors de la connexion dévoilant ainsi le PIN que l'utilisateur doit rentrer sur le périphérique (cf. Figure C.5).

6.6 Serveur de management de SmartCanton DevBox

Pour stocker les diverses informations sur les périphériques DevBox, et gérer les utilisateurs, un serveur avec une API REST [1], ainsi qu'une base de données SQLite [2] ont été mis en place. Un digramme illustrant la connexion est visible à l'aide de la Figure 6.24). En utilisant une API REST, le développeur final peut utiliser la platefrome avec laquelle il est le plus à l'aise. Dans le cadre de ce travail, c'est une application Android qui a été utilisée lors de la communication. Contrairement à la mise à disposition d'une base de données directe, l'API laisse une plus grande flexibilité lorsque des modifications sur les tables doivent être faites au fur et à mesure de l'avancement d'un projet, et ceci sans affecter le code précédemment implémenté par les développeurs.

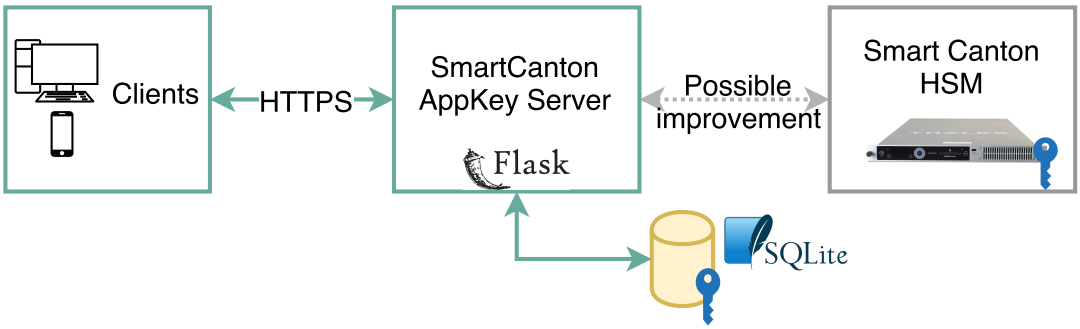


FIGURE 6.24 – Diagramme de communication entre les utilisateurs et le serveur de gestion des AppKey

À l'heure actuelle, le serveur doit être provisionné manuellement avec les différentes AppKey dans sa base de données. Dans une vision plus finale du projet, il serait intéressant que le serveur ait accès au HSM lors de l'enregistrement du périphérique (lien *possible improvement* sur la Figure 6.24). Lorsque'un périphérique souhaite s'enregistrer sur le réseau, il demande la génération de sa clé AppKey via l'API REST du serveur, qui elle communique avec le HSM pour créer la clé. Ensuite, cette clé n'est plus stockée dans le serveur, mais directement transmisse au périphérique, afin que celui-ci puisse rejoindre le réseau. Ainsi, la seule copie de la clé est sur le périphérique et dans le HSM.

6.6.1 Frameworks utilisés

Python est un langage de programmation très riche en bibliothèques. Dans le cadre de l'implémentation de ce serveur, deux bibliothèques ont principalement été utilisées pour accélérer le processus de développement.

6.6.1.1 Flask

Flask [3] est un framework de développement web en Python. Son optique est d'être léger afin de laisser le maximum de souplesse liée à la programmation en Python [67]. Plusieurs *templates* sont également disponibles, de même que différentes extensions. Par exemple, pour les accès à la base de données, un sous module de Flask nommé Flask-SQLAlchemy [4] a été utilisé pour faciliter les accès en écriture et en lecture à la

1. https://en.wikipedia.org/wiki/Representational_state_transfer
2. https://en.wikipedia.org/wiki/SQLite
3. http://flask.pocoo.org/
4. http://flask-sqlalchemy.pocoo.org/2.3/

base de données SQLite.

Une route d'accès à l'API peut être créée en quelques lignes en utilisant Flask. Voici par exemple comment afficher le message Hello World ! à l'utilisateur se connectant à l'URL `http://127.0.0.1:5000/` (le port par défaut est 5000) :

```
from flask import Flask
app = Flask(__name__)

@app.route("/")
def hello():
    return "Hello World!"

if __name__ == "__main__":
    app.run()
```

6.6.1.2 Flask-JWT-Extended

Le principe des JWT a été présenté en Section 4.3.2. Plusieurs frameworks pour la génération et la vérification de ces *tokens* ont été testés, il y eut tout d'abord `PyJWT`[1] et `python-jwt`[2]. Le principal inconvénient de ces deux implémentations réside dans le fait qu'elles ne sont pas parfaitement adaptées pour fonctionner en tandem avec Flask. La validité du *token* doit être traitée à l'intérieur du *callback* défini par le décorateur de la route Flask (`@app.route`). Certaines autres bibliothèques ont donc été développées afin que le décorateur de flask appelle premièrement une fonction de validation du token. Les deux bibliothèques essayées ont été `FlaskJWT`[3] et `Flask-JWT-Extended`[4]. Après plusieurs tests d'implémentation, il a été retenu que la bibliothèque `Flask-JWT-Extended` est beaucoup plus simple d'utilisation et contient plus de fonctionnalités. Elle est également beaucoup plus active en termes de développement. `FlaskJWT` ne dispose que de 99 *commits* qui remontent à 2015, alors que `Flask-JWT-Extended` reçoit plusieurs *commits* chaque mois. La documentation les étapes d'installation sont disponibles à l'adresse suivante :

`https://github.com/vimalloc/flask-jwt-extended`

6.6.2 Base de données

La structure de la base de données est visible sur la Figure 6.25 avec la présence de deux tables nommées `user` et `smartcanton_devbox_device`. Le couple `ble_mac_addr` et `device_eui` est unique dans la table du device. Chaque périphérique est lié à un propriétaire à l'aide d'un ID unique autoincrémenté à l'ajout d'un utilisateur de la table `user`. Le mot de passe est stocké à l'aide d'un *hash* généré avec l'algorithme sha256. Le champ `public_id` est l'identifiant UUID qui doit être utilisé lors des requêtes impliquant un utilisateur à l'API. Celui-ci est unique et permet de ce fait d'anonymiser le nom de l'utilisateur. Un administrateur, spécifié par le champ `admin`, a des privilèges spéciaux qui sont décrits en Section 6.6.3. Le *passkey* Bluetooth n'est pas obligatoire, car certaines DevBox peuvent être installées sans ce mécanisme de sécurité, cependant cela n'est pas conseillé.

1. `https://github.com/jpadilla/pyjwt`
2. `https://github.com/davedoesdev/python-jwt`
3. `https://github.com/mattupstate/flask-jwt`
4. `https://github.com/vimalloc/flask-jwt-extended`

La base de données du serveur est stockée dans un fichier SQLite. Lors de la présentation de Flash en Section 6.6.1.1, celui-ci a été couplé avec le module SQLAlchemy qui facilite la création des tables, ainsi que l'accès à celles-ci depuis les différentes routes REST implémentées. Si l'utilisateur souhaite utiliser une autre base de données, SQLAlchemy peut reconstruire les tables simplement en lui spécifiant une nouvelle base de données.

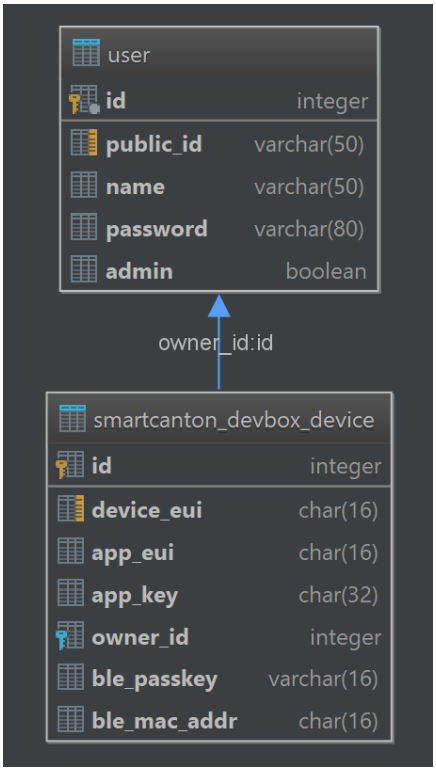


FIGURE 6.25 – Contenu de la base de données

6.6.3 Routes REST

La documentation complète de l'API REST ainsi que des exemples sont disponibles en Annexe B de ce document. Quatre types de requêtes disponibles sur les API REST ont été utilisées (`POST`, `GET`, `PUT` et `DELETE`). Une gestion des droits d'utilisateurs a également été respectée à l'aide du champ `admin` de la base de données. Voici un résumé des fonctionnalités qui ont été implémentées sur le serveur :

— [`POST`] Authentification de la paire nom d'utilisateur et mot de passe, puis si authentification acceptée, la génération d'un *token* JWT en réponse ;

— Gestion des utilisateurs :

1. [`GET`] Lister tous les utilisateurs ;
2. [`GET`] Récupérer des informations détaillées sur un utilisateur précis ;
3. [`PUT`] Mise à jour des informations d'utilisateurs existants ;
4. [`POST`] Création de nouveaux utilisateurs ;
5. [`DELETE`] Suppression d'utilisateurs existants ;

— Gestion des SmartCanton DevBox :

1. [`GET`] Lister tous les périphériques DevBox ;
2. [`GET`] Récupérer des informations détaillées sur un périphérique précis ;

3. [`PUT`] Mise à jour des informations d'un périphérique existant ;
4. [`POST`] Création de nouvelles DevBox ;
5. [`DELETE`] Suppression des DevBox existantes.

Pour la gestion des utilisateurs, si le compte connecté n'est pas administrateur, alors il peut uniquement accéder, modifier et supprimer les informations le concernant. Par exemple, il ne peut lister que ses périphériques et non ceux de tout le réseau. Certaines fonctionnalités sont limitées seulement aux utilisateurs de types administrateurs, en voici la liste :

— Modification des mots de passe des utilisateurs (ne peut pas consulter, uniquement modifier le *hash*) ainsi que les diverses informations de n'importe quel utilisateur ;
— Ajout de nouveaux utilisateurs ;
— Promulguer un utilisateur (nouveau ou existant) au rang d'administrateur ;
— Supprimer des utilisateurs.

Pour spécifier un périphérique DevBox, c'est le champ `ble_mac_addr` qui est utilisé, car c'est le seul élément que le smartphone peut lire sans avoir besoin de se connecter à la DevBox. La documentation en Annexe B explique plus en détail les permissions pour chaque commande.

6.6.4 Certificats SSL

Puisqu'une application Android doit être développée dans ce projet pour accéder à des URLs en HTTPS, une condition doit être respectée. Il est impératif d'utiliser un serveur qui utilise des certificats qui sont signés par une autorité de certifications. Si ce n'est pas le cas, Android refuse d'accéder à une quelconque ressource provenant de ces URLs. Il est possible d'ajouter son propre certificat dans Android, mais c'est une tâche compliquée, qui n'est pas très agréable pour l'utilisateur.

Pour la génération des certificats SSL, on peut directement contacter des sociétés comme Verisign ou Comodo, moyennant une cotisation annuelle, ils peuvent prodiguer un certificat pour un nom de domaine désiré. Cependant, depuis le 12 avril 2016, l'autorité de certification nommée *Let's Encrypt* offre la possibilité de générer des certificats gratuitement [68]. *Let's Encrypt* est un service fourni par l'*Internet Security Research Group* (ISRG), financé par de grandes entreprises et associations.

L'utilisateur désirant un certificat SSL doit prouver qu'il a accès au port 80 ou 443 du serveur derrière un nom de domaine. Pour cela, un logiciel doit être exécuté sur le serveur pour d'utiliser un de ces deux ports. Le plus connu est CertBot [1]. Dans le cas de ce projet, le serveur Web est directement implémenté par Flask. Il faut donc générer les clés et ensuite les passés en paramètres dans le code. Mais si l'utilisateur utilise Apache ou Nginx comme serveur web, il existe des *plug-ins* pour ceux-ci qui s'occupent de renouveler automatiquement le certificat. Celui-ci peut donc être généré avec la commande suivante :

```
$ sudo certbot certonly --standalone --preferred-challenges http -d lsn.eig.ch
```

1. https://certbot.eff.org/

Tous les 3 mois, cette commande doit être réexcutée afin que le service n'invalide pas le sous domaine. Les fichiers généré peuvent être listés à l'aide de la commande suivante :

```
$ certbot certificates
Saving debug log to /var/log/letsencrypt/letsencrypt.log
-------------------------------------------------------------------------------
Found the following certs:
  Certificate Name: lsn.eig.ch
    Domains: lsn.eig.ch
    Expiry Date: 2018-03-20 12:11:40+00:00 (VALID: 89 days)
    Certificate Path: /etc/letsencrypt/live/lsn.eig.ch/fullchain.pem
    Private Key Path: /etc/letsencrypt/live/lsn.eig.ch/privkey.pem
-------------------------------------------------------------------------------
```

Ceux-ci doivent être passés en paramètres à Flask afin de mettre en place des connexions HTTPS :

```
context = ssl.SSLContext(ssl.PROTOCOL_TLSv1_2)
chain_path = 'ssl_certificates/fullchain.pem'
chain_path = os.path.join(os.path.dirname(__file__), chain_path)
privkey_path = 'ssl_certificates/privkey.pem'
privkey_path = os.path.join(os.path.dirname(__file__), privkey_path)
context.load_cert_chain(chain_path, privkey_path)
```

Chapitre 7

Démonstrateur

Une fois la carte électronique développée, il était nécessaire de déterminer un *use case* pouvant être utilisé comme démonstrateur complet du travail réalisé. En somme, la création d'un capteur générique qui puisse fournir tous les types de données générées. Le but était d'utiliser l'intégralité de l'infrastructure mise à disposition, en partant du *périphérique* (SmartCanton DevBox dans ce cas) jusqu'à la visualisation des données à l'aide d'une interface, en passant par l'approvisionnement de l'AppKey LoRaWAN.

7.1 Architecture finale

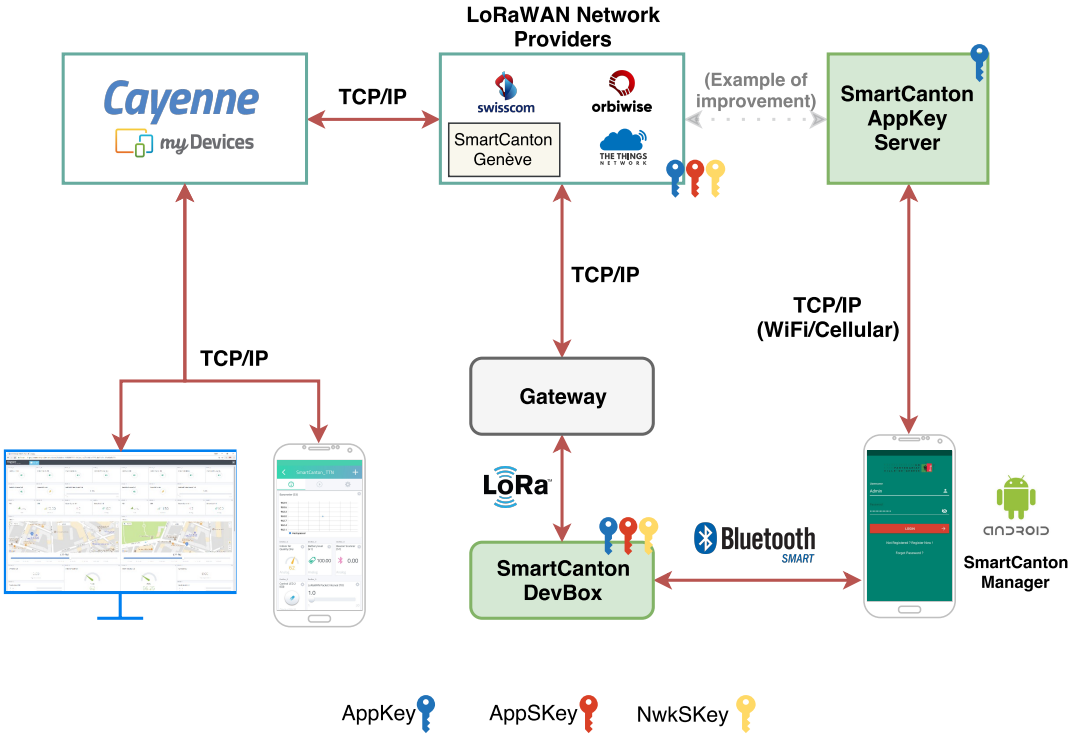


FIGURE 7.1 – Diagramme du projet avec l'application de visualisation des données

La plateforme Cayenne de l'entreprise MyDevice a été raccordée au flux de données proposé par les différents opérateurs LoRaWAN. Ce démonstrateur permet d'utiliser tous les périphériques disponibles sur la carte électronique développée et de les visualiser sur des terminaux. Les données sont toutes accessibles via *Bluetooth Low Energy* ou **LoRaWAN**. Le débit des données par LoRaWAN a été limité à maximum

un paquet par seconde, et ceci en désactivant le contrôle du *duty cycle* de la *stack* LoRaWAN.

7.2 Cayenne de MyDevices

Cayenne est une plateforme qui offre la possibilité d'afficher des *widgets* contenant diverses données prodiguées par l'utilisateur. L'utilisateur peut choisir plusieurs formats d'affichage pour les données (ex. valeur instantanée, graphique, jauge, etc.). Des actuateurs peuvent également être intégrés afin de générer des données ou des événements en direction du périphérique.

Le support des périphériques LoRa est encore en bêta sur la plateforme, mais beaucoup d'opérateurs sont déjà supportés pour la récupération des données. Parmi ces opérateurs LoRaWAN, on identifie la présence de Swisscom, Orbiwise et The Things Network, tous actifs dans la région genevoise. The Things Network dispose d'un très bon tutoriel [1] pour l'intégration avec Cayenne. C'est d'ailleurs par ce biais que la plateforme a été découverte.

7.2.1 Format des données

Dans la tâche RTOS DevBox Main, il était nécessaire de respecter le format de données imposé par Cayenne, de sorte que celles-ci soient interprétées par la plateforme. La description complète des commandes, de même que des exemples de trames sont disponibles dans la documentation de Cayenne :

https://mydevices.com/cayenne/docs/lora/

Bien que le format des données ne soit nullement optimisé pour une application prévue en production, il est nécessaire pour que Cayenne puisse interpréter correctement ces dernières. Une bibliothèque C++ a été adaptée en C afin que la conversion des données soit simplifiée. La bibliothèque initialement utilisée est disponible à l'adresse suivante :

https://github.com/sabas1080/CayenneLPP

7.2.2 Fonctionnalités implémentées

Toutes les données des capteurs sont envoyées à l'aide de paquets *uplinks*. Voici la liste de celles-ci :

1. RSSI ;
2. SNR ;
3. latitude et longitude GPS ;
4. BNO055 (accéléromètre et gyroscope) ;
5. BME680 (température, humidité, pression et IAQ) ;
6. *beacons* Bluetooth Low Energy scannés ;
7. niveau de batterie.

Des actuateurs sont disponibles sur les *dashboards* pour générer des messages *downlinks* LoRaWAN. L'état de ces actuateurs est également envoyé dans les paquets *uplink*, afin que l'affichage soit toujours à jour. En voici l'énumération :

1. https://www.thethingsnetwork.org/docs/devices/arduino/api/cayennelpp.html

1. autoriser l'envoi des données du GPS ;
2. autoriser l'envoi des données du BNO055 (accéléromètre + gyroscope) ;
3. autoriser l'envoi des données du BME680 (température, humidité, pression et IAQ) ;
4. autoriser l'envoi des données du niveau de batterie ;
5. autoriser l'envoi des données du scanneur de *beacons* BLE ;
6. intervalle entre chaque paquet LoRaWAN configurable en secondes ;
7. contrôle de l'état de la LED verte connectée au KW41Z.

Dans l'état actuel du démonstrateur, ces configurations ne sont pas sauvegardées dans la mémoire non volatile du KW41Z. Lors du *reset* de celui-ci, une configuration préprogrammée est appliquée. Par défaut, toutes les autorisations d'envoi des mesures sont activées et l'intervalle entre chaque paquet est de 4min. La LED verte est quant à elle éteinte.

7.2.3 Dashboard

Le dashboard est visualisable en ligne dans un navigateur Web. Le dashboard est personnalisable par le développeur et peut ensuite être partagé avec un lien. Lors de ce partage, des droits d'accès peuvent être spécifiés, tels que l'activation des actuateurs. Un exemple de dashboard affichant les données d'une DevBox est présenté sur la Figure 7.2. Cayenne propose également la possibilité de visualiser les diverses informations des capteurs à l'aide d'une application Android [1], visible sur la Figure 7.3. Celle-ci est toute fois un peu instable selon les menus sélectionnés.

7.3 Échangeur d'AppKey

L'application SmartCanton Manager offre la possibilité de mettre à jour l'AppKey et l'App EUI présents sur la DevBox. Les images démontrant le fonctionnement de l'application ainsi que les étapes de mise à jour sont décrites dans l'Annexe C de ce document.

Le serveur est hébergé à hepia à l'aide d'une adresse IP publique fixe. Actuellement, l'AppKey doit être provisionnée à la main dans la base de données SQLite. À l'avenir, la solution de connecter le serveur au HSM ou à une API intermédiaire serait optimale.

7.4 Opérateurs testés

L'application finale a été utilisée principalement sur deux opérateurs : The Things Network (TTN) et Orbiwise. La plus grande partie du développement a été réalisée sur TTN, plusieurs *gateways* LoRaWAN étant disponibles dans les locaux d'hepia. Cayenne supporte les deux opérateurs pour recevoir des données sur les *dashboards*. Il est ainsi aisé d'alterner les réseaux à l'aide de l'application SmartCanton Manager et de visualiser le changement du flux de données entre les deux *dashboards* (TTN et Orbiwise). Toutefois, les *downlinks* sur Orbiwise ne semblent pas être correctement supportés par la plateforme Cayenne pour l'instant.

1. https://play.google.com/store/apps/details?id=com.mydevices.cayenne

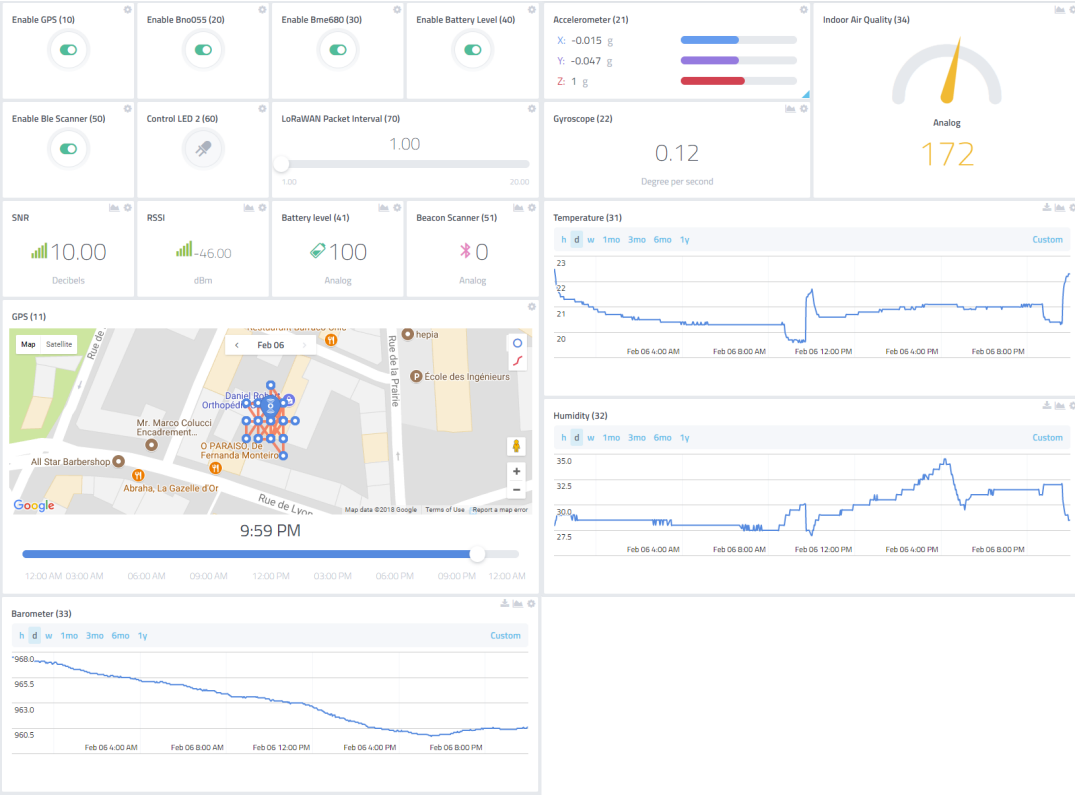


FIGURE 7.2 – *Dashboard* du capteur DevBox à l'aide de Cayenne sur un navigateur Web

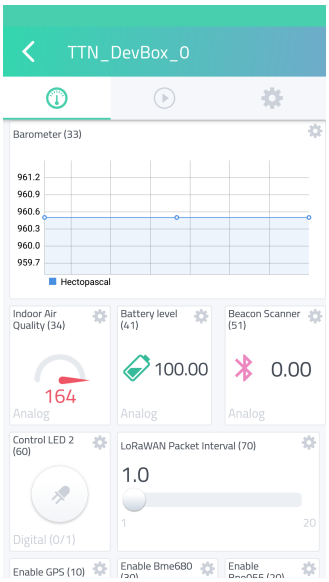


FIGURE 7.3 – *Dashboard* du capteur DevBox à l'aide de Cayenne sur l'application Android

7.5 Consommation électrique

La consommation électrique du périphérique n'a malheureusement pas pu être optimisée pour le démonstrateur final. C'est un choix qui s'est imposé lors du développement pour accélérer la réalisation d'un prototype fonctionnel. Lors de la réalisation du PCB, quelques erreurs ont naturellement été commises, ce qui a provoqué des

problèmes de consommation (ex. la connexion de l'antenne LoRa, cf. Section 5.6).

Le circuit présente actuellement une consommation moyenne de 100 mA. La source de 50 % de cette consommation est connue et émane du GPS. En effet, avec son antenne active, le GPS consomme environ 50 mA (cf. Section 5.3.4.5). Le reste de la consommation provient des divers périphériques, tels que les LEDs qui utilisent chacune 4 à 6 mA. Le processeur n'a pas été mis en mode *low power*. Ce mode est actionnable à l'aide d'une *macro* dans le SDK. Pour des raisons de temps, ce mode n'a toutefois pas pu être testé en vue d'examiner sa stabilité.

Chapitre 8

Gestion du projet

8.1 Planning

Au démarrage de ce projet, suite à l'établissement du cahier des charges, un planning GANTT a été réalisé. Celui-ci a permis de lister les différentes tâches à réaliser et de fixer des *milestones* pour la réalisation. Le planning GANTT relatif au temps estimé est visible en Annexe D.1 ; celui relatif au temps réalisé est quant à lui exposé à l'Annexe D.2.

Dans l'ensemble, les prévisions ont été respectées. Quelques changements sont néanmoins survenus sur les différentes tâches à réaliser. Nous nous sommes en effet rendu compte qu'il n'était pas forcément nécessaire de faire 4 petites applications distinctes, puisqu'à l'heure actuelle, seules deux cartes électroniques ont été montées. Nous avons ainsi décidé de faire une seule et unique application qui intègre toutes les fonctionnalités désirées. Cette application est toutefois modulable et les différents capteurs peuvent être désactivés en supprimant les différentes tâches au démarrage (cf. Section 6.4.3). La mise en place de l'API entre les deux microcontrôleurs a été simplifiée par la reprise du projet de commandes AT créé par STMicroelectronics.

Il découle du planning final en Annexe D.2 que les éléments qui ont requis le plus de temps sont surtout liés au développement logiciel sur le KW41Z. C'était une première expérience avec ce microcontrôleur, de même qu'avec une *stack* Bluetooth sur RTOS, complexifiant ainsi l'implémentation. Certaines tâches se sont révélées plus complexes que prévu, par exemple, la réalisation de l'application Android. À l'origine, il a été prévu 10 jours pour le développement de l'application et les tests avec le serveur ; au final, il en a fallu le double, en raison, notamment de l'intégration fastidieuse du Bluetooth sur Android et l'aspect visuel de l'application qui prend du temps si l'on souhaite obtenir un résultat agréable et pratique.

Tous ces légers retards ont été rattrapés lors de la mise en place du démonstrateur. Au départ, il avait été imaginé de créer une application web afin d'afficher les différents capteurs, mais un gain de temps a été effectué en utilisant directement une solution comme Cayenne. Finalement, le planning a été respecté, la plus grande partie du projet a été finalisé le 12 janvier 2018, nous laissant ainsi du temps pour préparer la documentation du code et corriger les derniers petits problèmes de stabilité.

8.2 Séparation des dépendances

Afin d'éviter une dépendance au matériel électronique, une partie de la programmation a été démarrée sur les cartes de développement des deux processeurs (cf. Chapitre 5 pour les différentes cartes de développement disponibles). Cette démarche

a accéléré la prise en main des différents IDE imposés par les fabricants et les tests des différentes implémentations logicielles possibles (ex. les différents RTOS disponibles).

8.3 Gestionnaire de version

Pour être le plus rigoureux possible, un système de gestion des versions a été utilisé. C'est sur GitHub que le projet a été hébergé :

https://github.com/Rasnar/SmartCanton

Ce dépôt GitHub a aujourd'hui plus de 300 commits. La plupart de ces commits recensent les divers changements effectués. Des balises dans les titres ont été utilisées pour différencier les types de modifications et leur impact (ex. [Android], [KW41Z], etc).

Pour faciliter la tâche, un seul dépôt git a été créé pour le projet intégral. Celui-ci sera splitté en différents sous-projets une fois ce mémoire fini. Le lien GitHub susmentionné restera toujours le même, avec une vue d'ensemble du projet.

8.4 Documentation du code

L'intégralité du code source écrit dans ce projet a été commenté en utilisant la syntaxe Doxygen :

https://www.stack.nl/~dimitri/doxygen/manual/docblocks.html

Cette syntaxe a été appliquée à trois langages différents ; C, Java/Android et Python. Il est ainsi possible de créer la documentation à partir des commentaires créés, mais le temps a manqué pour que cette opération puisse être réalisée dans le cadre de ce projet.

Chapitre 9

Discussions

9.1 Problèmes rencontrés

Cette section offre une discussion autour des difficultés qui ont été rencontrées au cours de ce projet. La plupart ont pu être résolues, mais certaines d'entre elles restent encore aujourd'hui sans correctif, en raison d'un manque de temps ou du fait que l'erreur n'émane pas de l'utilisateur, mais plutôt de l'implémentation réalisée par une autre entité (bibliothèque fournie, limites d'un framework choisi, etc.).

9.1.1 Réalisation matérielle de la DevBox

Lors de la conception de la DevBox, certaines erreurs ont été commises mais elles ont toutes pu être corrigées sur la carte de démonstration. La Section 5.6 explique toutes les erreurs commises, ainsi que les améliorations qui pourraient être apportées pour une nouvelle version de la carte.

Il existe un phénomène actuellement inexplicable se produisant avec les données de la température provenant du BME680 et exposé sur la Figure 9.1. Toutes les 12 heures environ, le capteur transmet un *glitch* qui a pour effet de fausser la mesure de température d'un delta de 1.5 °C. Cet effet est également visible sur la mesure de l'humidité mais avec un impact moindre (± 4 %). La valeur de la température affichée sur la Figure 9.1 est calculée à l'aide de la bibliothèque BSEC. Des tests avec les données brutes du BME680 ont toutefois aussi prouver que ce n'est pas la bibliothèque qui altère les données. Malheureusement, le temps a manqué pour explorer ce phénomène plus en détail.

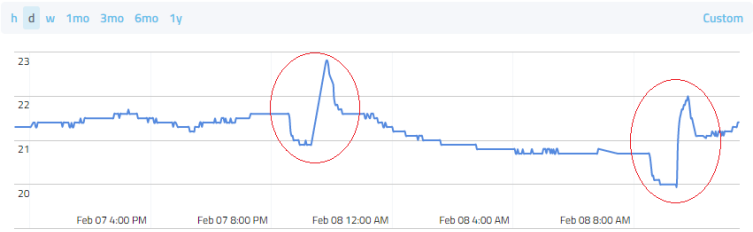


FIGURE 9.1 – Mesure de la température faussée sur le BME680

9.1.2 Développement logiciel de la DevBox

La programmation du KW41Z a été l'étape qui a demandé le plus d'investissement en terme de temps. La programmation d'un RTOS est complexe et surtout le débogage peut vite devenir difficile en fonction de l'erreur commise.

La liste qui suit n'est pas dans un ordre chronologique, mais elle contient les plus principaux problèmes rencontrés :

1. Le débogueur fournit par la FDRM-KW41Z carte d'évaluation n'a pas pu être utilisé pour programmer la DevBox à cause d'un manque de documentation concernant les *jumpers* pour utiliser la carte de développement. Heureusement, un JTAG Link de Segger était compatible avec le KW41Z et a donc été utilisé tout au long du projet pour la programmation et le débogage.
2. Les principaux problèmes liés à la programmation du RTOS proviennent du manque de documentation disponible. En effet, depuis le rachat de Freescale par NXP, certains documents ne sont pas disponibles sous la rubrique documentation du microcontrôleur KW41Z. Il s'agit principalement des documentations se référant à l'OS Abstraction layer de Freescale. La documentation du SDK n'est pas encore intégralement complète sur le site de NXP. Freescale propose cette documentation au format PDF, mais NXP a décidé de la rendre disponible sous forme de page web (`http://mcuxpresso.nxp.com/api_doc/dev/224/`), mais plusieurs chapitres ne sont pas disponibles, contrairement au PDF nommé `Kinetis SDK v.2.0 API Reference Manual`[1]. Ce dernier a donc été utilisé comme principale source d'informations.
3. Une nouvelle version du SDK est disponible, intitulé KSDK 2.3 ou MCUXpresso SDK selon les plateformes. Suite à cela, NXP a mis à jour son site web, mais malheureusement certains outils ne sont plus fonctionnels. Par exemple, l'outil nommé `PIN TOOLS`, ayant pour but de faciliter la configuration des pins du microcontrôleur selon de leurs fonctionnalités assignées, n'est maintenant plus accessible pour le KW41Z. Une nouvelle version de l'IDE MCUXpresso est également disponible pour le KW41Z, mais les exemples de l'ancien SDK ne sont pas compatibles avec la nouvelle version. Deux messages[2] ont été postés sur le forum d'aide de NXP, dans le but d'obtenir des informations sur le sujet. Ils ont répondu qu'ils étaient conscients des problèmes, mais qu'à l'heure actuelle, le changement de SDK et d'IDE n'est pas envisageable pour le KW41Z.
4. Il y a un problème avec le scanner Bluetooth sur le KW41Z : lorsque l'utilisateur souhaite scanner en permanence les périphériques l'entourant, le scan arrête de transmettre des informations après un laps de temps indéterminé. Cette erreur semble être présente sur le KW41Z depuis septembre 2017, à en croire les divers messages[3] qui peuvent être lus sur le forum NXP. Pour corriger ce problème, le support de NXP conseille le redémarrage du contrôleur Bluetooth, sans garantir que cela puisse corriger le problème. Il faut ainsi attendre une mise à jour du SDK afin de pouvoir avoir un correctif fonctionnel.

En somme, la documentation éparse de ce microcontrôleur, plus particulièrement au niveau logiciel fait qu'il est souvent plus facile de consulter le code source (si disponible) que de chercher la documentation complète. Le rachat de Freescale par NXP est l'une des causes due à la fusion de certains éléments logiciels entre les deux entités. Cela semble toutefois s'améliorer au fur et à mesure de la publication de nouvelles versions. Le code est très bien structuré et aisément compréhensible, on s'habitue donc assez vite à cette approche.

1. `https://www.nxp.com/docs/en/reference-manual/KSDK20APIRM.pdf`
2. `https://community.nxp.com/thread/466353` `https://community.nxp.com/thread/464380`
3. `https://community.nxp.com/thread/453829`

Le support en ligne de NXP a été utilisé à plusieurs reprises. C'est un support très réactif et expérimenté, chaque message posté ayant obtenu une réponse en moins de 24 heures.

9.1.3 Serveur

L'implémentation du serveur a été grandement facilitée par l'utilisation du langage Python et de ses bibliothèques très complètes. Les seuls problèmes rencontrés sont dus à l'implémentation de l'authentification basée sur les JWT. Après avoir testé différentes bibliothèques, les limitations des premiers tests ont disparu.

9.1.4 Application Android

La plus grande difficulté de l'application Android provient de l'utilisation du Bluetooth. Même avec l'utilisation de la bibliothèque SweetBlue, il y a tout de même certaines instabilités qui demeurent. Android a pour but d'offrir une couche d'abstraction entre les différents appareils, mais en ce qui concerne le Bluetooth, le comportement diffère encore beaucoup d'un dispositif à un autre. Par exemple, sur une tablette Nexus 9 avec Android 8.0, parfois le pop-up du *passkey* BLE n'est pas affiché. Il est alors nécessaire d'aller dans le menu de notifications afin d'avoir une carte demandant à l'utilisateur de s'appairer. Ce type d'erreur ne se produit pas avec un Samsung Galaxy S7 sous Android 7.0.

9.1.5 Démonstrateur

Pour le démonstrateur, en collaboration avec la DGSI, il a été décidé de tester la connexion de la DevBox sur l'infrastructure SmartCanton. En novembre 2017, un premier test d'intégration a eu lieu entre la DevBox et le réseau LoRaWAN avec la fédération SmartCanton et Orbiwise qui s'est déroulé correctement. La carte a pu rejoindre le réseau et envoyer des données. Orbiwise a gracieusement fourni un compte de test qui a permis l'utilisation du *Join Server* implémenté par le SmartCanton pour rejoindre le réseau. Cependant, le POC a dû être modifié entre temps et cet accès n'était plus valide. Fin décembre 2017, nous avons voulu retenter l'intégration, mais une erreur au niveau de l'intégration avec Orbiwise a rendu le système non fonctionnel. En raison des fêtes de fin d'année, cette erreur ne pouvait pas être résolues avant les premières semaines de janvier. Nous avons donc décidé de se passer du SmartCanton et de n'utiliser ni la plateforme avec FIWARE pour la partie applicative, ni utiliser le réseau avec le Join Server du SmartCanton. Une démonstration avec The Things Network et Cayenne a été mise en place afin d'avoir un démonstrateur visuel et, surtout, de pouvoir tester les messages LoRaWAN en *uplink* et *downlink*.

Pour le démonstrateur final, la plateforme Cayenne a été utilisée pour réaliser un *dashboard*. Néanmoins, la plateforme Cayenne n'est pas encore stable. Voici les quelques erreurs qu'y on été rencontrées et la manière dont celles-ci ont été corrigées :

1. Impossible d'ajouter un périphérique supprimé précédemment : si on supprime un périphérique de son *dashboard*, il est censé réapparaitre lorsqu'une nouvelle trame de données est reçue par Cayenne. Néanmoins, ce périphérique n'est parfois pas recréé. La seule solution est de recommencer à zéro depuis The Things Network, c'est-à-dire supprimer l'application ainsi que les périphériques LoRa.

2. Messages *downlink* non confirmés : selon le guide d'utilisation de Cayenne, les messages *downlink* (de l'application vers le périphérique) génèrent une confirmation visuelle pour l'utilisation. Par exemple, si l'utilisateur appuie sur un bouton pour allumer une LED, celui-ci doit afficher un petit cercle de chargement en attendant la réponse du périphérique. Cependant l'implémentation actuelle de Cayenne ne supporte pas cet état d'attente. La LED apparait tout de suite allumée lorsque l'on appuie sur le bouton, alors qu'aucun message *uplink* n'a été reçu du périphérique. Il n'y a pas de solution à ce problème ; il faut attendre une mise à jour de Cayenne.
3. Capteurs non actualisés sur la plateforme : il arrive parfois que certains capteurs ne soient plus actualisés ou qu'un deuxième capteur, avec le même identifiant et les mêmes données, apparaisse dans le *device*. La première solution est de supprimer le capteur et d'attendre que le *device* envoie quelques nouvelles trames de données (le nombre de trames pour actualiser le *device* est aléatoire, mais il varie entre 1 et 5 trames). Si celui-ci ne réapparait pas, la seule solution est de supprimer entièrement le périphérique.
4. Plateforme hors ligne : de temps en temps les paquets reçus par The Things Network ne sont pas mis à jour par Cayenne. Le problème provient de leur infrastructure qui passe en maintenance. Malheureusement, aucune communication sur ces maintenances n'est effectuée, le seul moyen étant de visiter le forum et de demander l'état du système.

Il faut être conscient que les périphériques LoRa sur Cayenne sont encore en *bêta*, raison pour laquelle que le nombre de problèmes est élevé. Pour un service gratuit se trouvant en Bêta, il s'agit d'un très bon portail pour réaliser des tests.

9.2 Améliorations possibles

Dans l'ensemble, les objectifs initiaux ont pu être réalisés dans ce projet, malgré le fait qu'il reste toutefois des améliorations possibles dans plusieurs domaines. Le rôle de cette section est d'explorer certaines de ces améliorations qui pourraient être implémentées dans une future version du projet.

9.2.1 Sécurité

La sécurité a certes été explorée dans le cadre de ce projet, mais celle-ci peut encore être améliorée. Voici dans les quelques sections qui suivent les principaux axes à explorer.

9.2.1.1 Stockage des clés

À l'heure actuelle, l'AppKey est stockée dans le KW41Z (cf. Figure 9.2). Ce n'est pas idéal pour un projet commercial, car cela implique que l'AppKey soit transférée à chaque initialisation. Celle-ci doit plutôt être stockée à l'intérieur du module LoRaWAN puisqu'il dispose d'un coffre fort nommé STSAFE (cf. Section 5.3.3.1 et Section 4.4.3) prévu à cet effet. Comme évoqué en Section 4.4.3, la documentation de ce module n'est à l'heure actuelle pas publique ; comme pour beaucoup de circuits de cryptographie, il faut accepter des conditions spéciales envers le fabricant afin d'avoir accès aux guides d'utilisation et aux spécifications de ce module. Cela reste néanmoins l'une des meilleures solutions pour garder en lieu sûr les clés sur un système embarqué. La situation actuelle n'est que temporaire et doit être résolue en utilisant un stockage unique à l'intérieur du CMWX1ZZABZ, comme illustré sur la Figure 9.2.

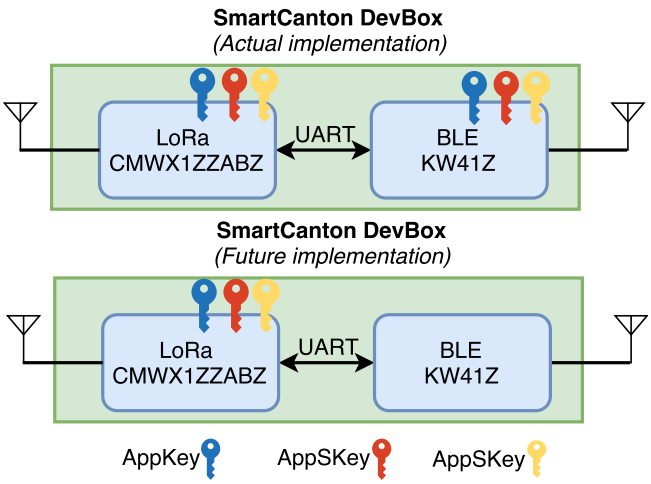


FIGURE 9.2 – Amélioration du stockage des clés sur le KW41Z

9.2.1.2 Confiance des données du périphérique

En IoT, il est essentiel de savoir si le périphérique qui envoie des données est de confiance. En effet, si quelqu'un arrive à usurper l'identité d'un périphérique, par exemple en récupérant les clés de communication, il est important de savoir si les données qui sont reçues sont valides ou non. Selon le type d'utilisation cette information est essentielle, surtout si les données servent à piloter des systèmes critiques, tels que des systèmes de régulation. La confiance des données peut aussi provenir du fait que le capteur a un défaut. Une partie du traitement de cette confiance peut être effectuée en logiciel, par exemple, dans le cas d'une mesure de température, si le capteur a un problème électronique (ex. variation de température trop brusque), une alerte peut être générée afin d'avertir les exploitants du capteur.

9.2.1.3 Bluetooth

La sécurité Bluetooth appliquée n'est pas parfaite, puisque le *passkey* n'est à l'heure actuelle pas généré aléatoirement à chaque appairage, comme proposé par les recommandations NIST (cf. Section 4.1.3). Pour palier à cela, il faut que le périphérique puisse afficher ce pin à l'aide d'une interface. On peut ainsi imaginer la mise en place d'un petit écran qui ne serait alimenté que lorsque l'utilisateur souhaite initialiser un appairage Bluetooth. Ce pin pourrait être généré et affiché à la programmation de la carte et faciliter ainsi le stockage du *passkey*, car à l'heure actuelle il s'agit d'une constante dans le code.

9.2.2 Réalisation matérielle de la DevBox

La DevBox a été réalisée afin de créer une plateforme de développement pour ce projet et les projets à venir dans le cadre du SmartCanton. Après quelques mois de prise en main, plusieurs améliorations ont pu être retenues qui sont présentées ci-dessous de manière non exhaustive.

1. Miniaturiser la carte. La première version a été pensée afin de faciliter les différents tests durant le développement, mais a eu pour conséquence de créer une carte de taille importante (80×68 mm). Tous les points de tests peuvent notamment être enlevés afin d'optimiser au mieux la surface.

2. Tester avec de nouvelles antennes afin de comparer les différentes portées possibles en fonction des gains.
3. Opter pour des composants moins chers si l'optique finale est la production de masse. En effet, le GPS par exemple, est un composant onéreux (environ 25 CHF selon les quantités). Ce composant ne devrait être utilisé que s'il est primordial dans le *use case* final.
4. Créer un boîtier avec des contraintes environnementales afin de pouvoir utiliser la DevBox comme capteur extérieur.
5. Utilisation du NRF52840 comme microcontrôleur Bluetooth. Comme expliqué dans la Section 5.3.1.2, le NRF52840 semblait être le microcontrôleur parfait pour cette carte ; malheureusement il n'était pas disponible quand ce projet a commencé. Si la carte était à refaire et que le temps le permet, il serait intéressant de l'utiliser en remplacement du KW41Z.

9.2.3 Programmation de la DevBox

La programmation de la DevBox est l'élément qui a requis un important investissement en temps. Dans les sous-sections qui suivantes, quelques exemples sont proposés pour apporter de nouvelles fonctionnalités à celles déjà implémentées.

9.2.3.1 RTOS

Actuellement, seul FreeRTOS est compatible avec le KW41Z. Il serait toutefois intéressant de garder un œil sur Zephyr et Mynewt (présentés en Section 2.9.3). Il s'agit là de projets qui semblent tous deux prometteurs pour les domaines IoT que ce soit pour leurs aspects en matière de sécurité, ou leurs *stacks* open source pour le Bluetooth.

9.2.3.2 Mise en place de la basse consommation

Pour l'instant la consommation n'a pas pu être optimisée comme souhaité dû aux problèmes matériels rencontrés d'une part (cf. Section 5.6), et au temps de développement restreint d'autre part. En effet, lorsque l'on souhaite activer tous les modes basse consommation, cela complique l'implémentation logicielle. Par exemple, sur un microcontrôleur, lorsqu'on passe en *sleep* on perd souvent la main sur le débogueur, compliquant ainsi le développement.

9.2.3.3 Mises à jour logicielles

La mise à jours des *firmwares* des périphériques est une problématique très répandue dans les systèmes IoT. Le problème des systèmes LoRa est que le débit en *downlink* (*gateway* vers périphérique) est limité par jour. Il est donc très difficile de mettre à jour via cette méthode à l'heure actuelle. Néanmoins, puisque la DevBox est équipée d'une interface Bluetooth, on peut utiliser cette dernière comme alternative. Cette mise à jour peut s'effectuer à l'aide d'un *bootloader* Bluetooth supportant l'*Over-the-Air programming*[1]. NXP offre des exemples avec cette implémentation, mais le temps a malheureusement manqué pour explorer cette possibilité.

1. https://en.wikipedia.org/wiki/Over-the-air_programming

9.2.4 Serveur

Le serveur est assez basique dans la structure de sa base de données. On peut imaginer rajouter des tables plus complètes, notamment pour l'aspect de gestion du profil de l'utilisateur. L'API REST peut être étendue avec de nouvelles commandes comme par exemple pour partager un dispositif entre plusieurs utilisateurs.

La principale amélioration reste tout de même l'intégration avec le HSM pour la création des clés, comme exposé en Section 6.6. Cela rendrait ainsi le système plus sécurisé, car même si le compte de l'utilisateur venait à être compromis, il ne pourra que rajouter des nouvelles clés via le HSM, sans jamais accéder aux clés déjà créées.

9.2.5 Application Android

L'application Android peut être un véritable HUB pour le SmartCanton sur laquelle on peut imaginer de demander directement la génération d'une APPKEY et la programmation sur le serveur. Le serveur intermédiaire n'aurait pas besoin de stocker les clés si la génération et la programmation s'effectuent sur la même paire de connexions (Bluetooth et HTTP).

L'application Android pourrait également offrir la possibilité de mettre à jour le *firmware* de la carte électronique comme évoqué en Section 9.2.3.

Il possible d'injecter le code pin en utilisant une réflection de méthode Java pour accéder à une méthode privée [1] de la classe `BluetoothDevice` [2]. Toutefois, cette implémentation est plutôt un *workaround*, car l'API Bluetooth standard ne l'accepte pas. L'avantage de cette solution serait de faire en sorte que la connexion et l'appairage au périphérique soient des étapes invisibles pour l'utilisateur.

9.2.6 Démonstrateur

La solution Cayenne est idéale pour du prototypage, mais l'implémentation des paquets est bien trop lourde pour être utilisée pour un projet commercial. Il existe une méthode de compression d'informations afin de limiter les données et ainsi économiser le temps d'émission sur le réseau LoRa.

Si l'on part dans l'optique d'utiliser l'infrastructure FIWARE pour récupérer les données stockées dans l'infrastructure SmartCanton, l'idéal serait d'implémenter un client web qui puisse récupérer les données à travers un IoT Agent de FIWARE. Ces données pourraient ensuite être affichées sur un *dashboard* personnalisé. Le format des données pourrait être grandement réduit afin d'économiser du temps d'émission et ainsi pouvoir envoyer un nombre supérieur de trames LoRaWAN.

1. https://stackoverflow.com/questions/11623696/android-bluetooth-setpin-function
2. https://developer.android.com/reference/android/bluetooth/BluetoothDevice.html

9.3 Conclusions

La carte électronique a pu être achevée et s'est avérée parfaitement opérationnelle, tous les capteurs intégrés ayant pu être testés et validés. Deux exemplaires de cette carte ont été montés et utilisés pour la réalisation du présent travail de Master.

Le code développé dans ce projet s'est voulu modulaire pour faciliter la réutilisation de la carte notamment pour divers projets pédagogiques au sein d'hepia. Les professeurs pourront ainsi proposer des travaux aux étudiants sur la thématique du LoRa ou du Bluetooth. La documentation fournie dans ce rapport, de même que celle sur le dépôt Git facilitera l'utilisation de la carte.

L'interface Bluetooth Low Energy est utilisable aussi bien pour l'approvisionnement des paramètres LoRaWAN, que pour la récupération des données des divers capteurs de la carte DevBox. L'interface Bluetooth présente un avantage pour la récupération du flux de données en local. Par exemple, lorsqu'un opérateur doit consulter l'état du capteur, il peut récupérer les données en direct sans nécessiter l'attente d'une émission d'un paquet LoRaWAN. La méthode utilisée pour l'approvisionnement des clés pourrait être imposée sur des périphériques à risque, sur lesquels on souhaite pouvoir modifier ou consulter les données une fois le périphérique installé sur le terrain.

Il est toutefois navrant que le projet n'ait pu être testé au dernier moment sur la plateforme SmartCanton pour l'intégration finale. Cela étant, les différents modules développés ont pu être testés sur les plateformes TTN et Orbiwise avec succès. Par conséquent, FIWARE n'a été utilisé ni pour la récupération de données ni pour la visualisation de celles-ci dans le cadre du démonstrateur. Un dashboard — créé au moyen de la plateforme Cayenne de MyDevices — a été mis en place en guise de démonstrateur ; toutes les données des capteurs sont visibles sur ce dernier, de même que les actuateurs permettant d'altérer l'état des périphériques ou les paramètres de la carte électronique.

D'un point de vue personnel, je considère ce projet comme étant idéal pour concrétiser et démontrer les compétences acquises au fil de mes années de formation. En effet, le projet a deux extrêmes : d'une part, la création d'une carte avec de l'électronique, et d'autre part, la programmation haut niveau avec de l'Android et du Python. A cela s'ajoute la programmation d'un microcontrôleur avec un RTOS. Le présent travail permet de concilier ces divers éléments en vue de la réalisation d'un projet exhaustif et instructif. L'IoT est un domaine en pleine expansion et ce fut un réel plaisir d'analyser et d'utiliser les diverses technologies qui sont à l'heure actuelle en en phase de développement. La sécurité des systèmes d'information est un domaine passionnant. Ce n'était pas le but initial de ce projet, mais le résultat obtenu est plaisant et m'a permis d'acquérir de nouvelles compétences tout en explorant des problématiques auxquelles je n'avais pas encore été confronté.

Conseiller de travail de Master
Prof. Fabien, Vannel
Ingénierie des Technologies de l'Information,
Institut inIT,
hepia

. . . . . .
Signature
Genève, le 09 février 2018

Diplômant
M. David, Da Silva Andrade
Technologies de l'Information et de la Communication,
Systèmes Embarqués et Mobiles,
HES-SO//Master

. . . . . .
Signature
Genève, le 09 février 2018

Bibliographie

[1] Wikipedia. *LPWAN - Wikipedia*. https://en.wikipedia.org/wiki/LPWAN. (Accédé le 09/10/2017).

[2] *lora-alliance*. https://www.lora-alliance.org/. (Accédé le 10/08/2017).

[3] Wikipedia. *Smart city - Wikipedia*. https://en.wikipedia.org/wiki/Smart_city. (Accédé le 01/21/2018).

[4] Teena Maddox. *66% of US cities are investing in smart city technology - TechRepublic*. https://www.techrepublic.com/article/66-of-us-cities-are-investing-in-smart-city-technology/. (Accédé le 01/21/2018). Nov. 2017.

[5] Markets et Markets. *Smart Cities Market worth 1,201.69 Billion USD by 2022*. https://www.marketsandmarkets.com/PressReleases/smart-cities.asp. (Accédé le 22/01/2018).

[6] Dejan Nikolic. *Genève brille par son absence au classement des villes intelligentes - Le Temps*. https://www.letemps.ch/economie/2017/05/30/geneve-brille-absence-classement-villes-intelligentes. (Accédé le 19/10/2017). Mai 2017.

[7] Smart City Suisse. *Smart City Suisse*. http://www.smartcity-suisse.ch/fr/smart-city/. (Accédé le 28/01/2018).

[8] Dejan Nikolic. *Le parking intelligent débarque à Genève - Le Temps*. https://www.letemps.ch/economie/2017/06/19/parking-intelligent-debarque-geneve. (Accédé le 05/02/2018). Juin 2017.

[9] Wikipedia. *LPWAN - Wikipedia*. https://en.wikipedia.org/wiki/LPWAN. (Accédé le 22/01/2018).

[10] Waviot. *LPWAN is Low-power Wide-area Network | WAVIoT LPWAN*. http://waviot.com/technology/what-is-lpwan. (Accédé le 22/01/2018).

[11] Ferran Adelantado. *Understanding the Limits of LoRaWAN*. https://arxiv.org/pdf/1607.08011.pdf. (Accédé le 22/01/2018). Jan. 2017.

[12] M2COMM. *WAN - Ultra-Narrow Band (UNB) - M2COMM*. https://www.m2comm.co/front-page/technology/wan-ultra-narrow-band-unb/. (Accédé le 22/01/2018).

[13] Semtech. *AN1200_22_Semtech_LoRa_Basics_v2_STD*. https://www.semtech.com/uploads/documents/an1200.22.pdf. (Accédé le 05/02/2018). Mai 2015.

[14] Wikipedia. *Code rate - Wikipedia*. https://en.wikipedia.org/wiki/Code_rate. (Accédé le 23/01/2018).

[15] Exploratory Engineering. *Exploratory Engineering : Data Rate and Spreading Factor*. https://docs.exploratory.engineering/lora/dr_sf/. (Accédé le 24/01/2018).

[16] Aloÿs Augustin. *A Study of LoRa : Long Range & Low Power Networks for the Internet of Things*. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5038744/. (Accédé le 23/01/2018). Sept. 2016.

[17] Libelium. *Waspmote LoRa 868MHz 915MHz SX1272 Networking Guide*. http://www.libelium.com/downloads/documentation/waspmote_lora_868mhz_915mhz_sx1272_networking_guide.pdf. (Accédé le 05/02/2018).

[18] ETSI. *EN 300 220-1 - V2.4.1 - Electromagnetic compatibility and Radio spectrum Matters (ERM); Short Range Devices (SRD); Radio equipment to be used in the 25 MHz to 1 000 MHz frequency range with power levels ranging up to 500 mW; Part 1 : Technical characteristics and test methods*. http://www.etsi.org/deliver/etsi_en/300200_300299/30022001/02.04.01_40/en_30022001v020401o.pdf. (Accédé le 05/02/2018). Jan. 2012.

[19] Wikipedia. *ISM band - Wikipedia*. https://en.wikipedia.org/wiki/ISM_band. (Accédé le 19/10/2017).

[20] Radio Electronics. *Dipole Length | Antenna Length Calculation & Formula*. http://www.radio-electronics.com/info/antennas/dipole/length-calculation-formula.php. (Accédé le 19/10/2017).

[21] Martin Woolley. *Exploring Bluetooth 5 - Going the Distance | Bluetooth Technology Website*. https://blog.bluetooth.com/exploring-bluetooth-5-going-the-distance. (Accédé le 07/01/2018). Fév. 2017.

[22] Wikipedia. *Bluetooth - Wikipedia*. https://en.wikipedia.org/wiki/Bluetooth. (Accédé le 17/01/2018).

[23] Wikipedia. *IEEE 802.15.4 - Wikipedia*. https://en.wikipedia.org/wiki/IEEE_802.15.4. (Accédé le 10/03/2017).

[24] Wikipedia. *ZigBee — Wikipédia*. https://fr.wikipedia.org/wiki/ZigBee. (Accédé le 01/21/2018).

[25] Silicon Labs. *What is the difference between an end device, a router, and a coordinator ?* https://www.silabs.com/community/wireless/zigbee-and-thread/knowledge-base.entry.html/2012/07/02/what_is_the_differen-IYze. (Accédé le 01/21/2018).

[26] Wikipedia. *6LoWPAN - Wikipedia*. https://en.wikipedia.org/wiki/6LoWPAN. (Accédé le 01/21/2018).

[27] Wikipedia. *Thread (network protocol) - Wikipedia*. https://en.wikipedia.org/wiki/Thread_(network_protocol). (Accédé le 01/21/2018).

[28] *Seven keys to understanding Thread protocol - Electronic Products*. https://www.electronicproducts.com/Digital_ICs/Communications_Interface/Seven_keys_to_understanding_Thread_protocol.aspx. (Accédé le 01/21/2018). Juin 2016.

[29] Texas Instruments. *General RTOS Concepts*. http://dev.ti.com/tirex/content/simplelink_academy_cc2640r2sdk_1_12_00_00/modules/rtos_concepts/rtos_concepts.html. (Accédé le 05/02/2018).

[30] Embedded.com. *Bare metal embedded software development with & without an RTOS | Embedded*. https://www.embedded.com/design/prototyping-and-development/4237636/Bare-metal-embedded-software-development-with---without-an-RTOS. (Accédé le 06/01/2018). Mar. 2012.

[31] Wikipedia. *FreeRTOS - Wikipedia*. https://en.wikipedia.org/wiki/FreeRTOS. (Accédé le 06/01/2018).

[32] EMBEDDED.COM. *2017 Embedded Market Survey | Embedded*. https://www.embedded.com/electronics-blogs/embedded-market-surveys/4458724/2017-Embedded-Market-Survey. (Accédé le 06/01/2018). Août 2017.

[33] WIKIPEDIA. *Zephyr (operating system) - Wikipedia*. https://en.wikipedia.org/wiki/Zephyr_(operating_system). (Accédé le 07/01/2018).

[34] LITEPOINT. *Bluetooth Low Energy Whitepaper*. https://litepoint.com/wp-content/uploads/2016/10/Bluetooth-Low-Energy-Whitepaper-102416.pdf. (Accédé le 05/02/2018). Oct. 2016.

[35] NXP. *Lesson 3 : Introduction to Bluetooth Low Energy|NXP*. https://www.nxp.com/video/lesson-3-introduction-to-bluetooth-low-energy:LESSON-3-KW41Z-WIRELESS-LAB. (Accédé le 17/01/2018). Juil. 2017.

[36] Martin WOOLLEY. *Bluetooth Technology Protecting Your Privacy | Bluetooth Technology Website*. https://blog.bluetooth.com/bluetooth-technology-protecting-your-privacy. (Accédé le 16/01/2018). Avr. 2015.

[37] MBED. *Understanding the different types of BLE Beacons | Mbed*. https://os.mbed.com/blog/entry/BLE-Beacons-URIBeacon-AltBeacons-iBeacon/. (Accédé le 16/01/2018). Mar. 2015.

[38] *Bluetooth Classic Vs Bluetooth Low Energy (BLE) : Whats different*. https://www.plugintoiot.com/bluetoothclassic-vs-bluetoothlowenergy/. (Accédé le 05/02/2018). Juin 2016.

[39] R. SCHRADER et al. « Advertising power consumption of bluetooth low energy systems ». In : *2016 3rd International Symposium on Wireless Systems within the Conferences on Intelligent Data Acquisition and Advanced Computing Systems (IDAACS-SWS)*. Juil. 2016, p. 62–68. DOI : 10.1109/IDAACS-SWS.2016.7805787.

[40] The Things NETWORK. *Home - The Things Network Wiki*. https://www.thethingsnetwork.org/wiki/LoRaWAN/Home. (Accédé le 23/01/2018).

[41] PROXIMUSENCO. *LoRaWAN device classes – Developers Support*. https://enablingsupport.zendesk.com/hc/en-us/articles/206069491-LoRaWAN-device-classes. (Accédé le 24/01/2018).

[42] The Things NETWORK. *Week 2 LoRaWAN Architecture Network Server*. https://lorawanacademy.azureedge.net/Week+2/Week+2+-+LoRaWAN+Architecture+Network+Server.mp4. (Accédé le 26/01/2018). Déc. 2017.

[43] The Things NETWORK. *Limitations : data rate, packet size, 30 seconds uplink and 10 messages downlink per day Fair Access Policy - End Devices (Nodes) - The Things Network*. https://www.thethingsnetwork.org/forum/t/limitations-data-rate-packet-size-30-seconds-uplink-and-10-messages-downlink-per-day-fair-access-policy/1300. (Accédé le 24/01/2018). Oct. 2017.

[44] Mahendra TAILOR. *What's New with Bluetooth 5.0 ? - Laird Technologies Wireless Connectivity Blog*. http://www.summitdata.com/blog/whats-new-with-bluetooth-5-0/. (Accédé le 07/01/2018). Fév. 2017.

[45] MICHROCHIP. *Bluetooth® Low Energy Security Modes and Procedures - Developer Help*. http://microchipdeveloper.com/wireless:ble-gap-security#toc6. (Accédé le 07/02/2018).

[46] *BLE pairing vs. bonding | Pirate Comm.* https://piratecomm.wordpress.com/2014/01/19/ble-pairing-vs-bonding/. (Accédé le 18/01/2018). Jan. 2014.

[47] Kai Ren. *Bluetooth Pairing Part 1 Pairing Feature Exchange | Bluetooth Technology Website.* http://blog.bluetooth.com/bluetooth-pairing-part-1-pairing-feature-exchange. (Accédé le 18/01/2018). Mar. 2016.

[48] Matthew Bon. *A Basic Introduction to BLE Security - Wireless - eewiki.* https://eewiki.net/display/Wireless/A+Basic+Introduction+to+BLE+Security. (Accédé le 07/02/2018). Oct. 2016.

[49] *Difference between Secure Simple Pairing and Secure Connections in Bluetooth ? - Information Security Stack Exchange.* https://security.stackexchange.com/questions/116027/difference-between-secure-simple-pairing-and-secure-connections-in-bluetooth/140679. (Accédé le 08/02/2018).

[50] Kai Ren. *Bluetooth Pairing Part 4 : LE Secure Connetions - Numeric Comparison | Bluetooth Technology Website.* https://blog.bluetooth.com/bluetooth-pairing-part-4. (Accédé le 07/02/2018). Jan. 2017.

[51] NIST. *Guide to Bluetooth Security.* http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-121r2.pdf. (Accédé le 19/01/2018). Mai 2017.

[52] The Things Network. *Week 8 LoRaWAN Sessions.* https://lorawanacademy.azureedge.net/Week8/Week+8+LoraWAN+Sessions.mp4. (Accédé le 27/01/2018). Déc. 2017.

[53] Rovbert Miller. *LoRa Security : Building a secure LoRa solution.* https://labs.mwrinfosecurity.com/publications/lo/. (Accédé le 19/01/2018). Mar. 2016.

[54] Johan Stokking. *What is new in LoRaWAN 1.1 ? | Webinar by Johan Stokking.* https://www.youtube.com/watch?v=ewsXKc3bk1U. (Accédé le 05/02/2018). Nov. 2017.

[55] Wikipedia. *HyperText Transfer Protocol Secure — Wikipédia.* https://fr.wikipedia.org/wiki/HyperText_Transfer_Protocol_Secure. (Accédé le 01/02/2018).

[56] Wikipedia. *JSON Web Token - Wikipedia.* https://en.wikipedia.org/wiki/JSON_Web_Token. (Accédé le 01/02/2018).

[57] Wikipedia. *Bluetooth - Wikipedia.* https://en.wikipedia.org/wiki/Bluetooth. (Accédé le 10/03/2017).

[58] *CC2650 : Bluetooth 5 stack - Bluetooth® low energy Forum - Bluetooth® low energy - TI E2E Community.* https://e2e.ti.com/support/wireless_connectivity/bluetooth_low_energy/f/538/t/579431. (Accédé le 09/01/2018). Mar. 2017.

[59] Murata. *Murata launches ultra compact low cost stand-alone LoRa module | Murata Manufacturing Co., Ltd.* https://www.murata.com/about/newsroom/news/product/wireless/2016/0719. (Accédé le 10/01/2018). Juil. 2016.

[60] GPS gov. *GPS.gov : GPS Accuracy.* https://www.gps.gov/systems/gps/performance/accuracy/. (Accédé le 08/02/2018).

[61] Wikipedia. *GPS signals - Wikipedia.* https://en.wikipedia.org/wiki/GPS_signals. (Accédé le 08/02/2018).

[62] STMIcroelectronics. *STM32 LoRa software expansion for STM32Cube*. http://www.st.com/content/ccc/resource/technical/document/user_manual/group0/31/96/2f/3b/df/c1/40/2e/DM00300436/files/DM00300436.pdf/jcr:content/translations/en.DM00300436.pdf. (Accédé le 01/29/2018).

[63] Wikipedia. *NMEA 0183 - Wikipedia*. https://en.wikipedia.org/wiki/NMEA_0183. (Accédé le 31/01/2018).

[64] Dale DePriest. *NMEA data*. http://www.gpsinformation.org/dale/nmea.htm#RMC. (Accédé le 16/01/2018).

[65] Bosch Sensortec. *Bosch Sensortec Environmental Cluster (BSEC) Software*. https://www.bosch-sensortec.com/bst/products/all_products/bsec. (Accédé le 31/01/2018).

[66] Frontline. *Getting the Android Link Key for Classic Bluetooth Decryption*. http://www.fte.com/docs/whitepapers/whitepapergetandroidlinkkey.pdf. (Accédé le 07/02/2018). Sept. 30.

[67] Wikipedia. *Flask (framework) — Wikipédia*. https://fr.wikipedia.org/wiki/Flask_(framework). (Accédé le 01/02/2018).

[68] Wikipedia. *Let's Encrypt - Wikipedia*. https://en.wikipedia.org/wiki/Let%27s_Encrypt. (Accédé le 13/01/2018).

Annexe A

SmartCanton DevBox : Bluetooth Services

A.1 Generic Access

Table A.1 – Description du Generic Access service

Generic Access Service			
Service UUID	**00001800-0000-1000-8000-00805f9b34fb**		
Characteristic Name	**Characteristic UUID**	**Rights [R,W,N]**	**Example Value**
Device Name	00002a00-0000-1000-8000-00805f9b34fb	R,W	"SmartCanton DevBox"
Appearance	00002a01-0000-1000-8000-00805f9b34fb	R	0x0
Peripheral Preferred Connection Parameters	00002a04-0000-1000-8000-00805f9b34fb	R	0x0

A.2 Generic Attribute

Table A.2 – Description du Generic Attribute service

Generic Attribute Service			
Service UUID	**00001801-0000-1000-8000-00805f9b34fb**		
Characteristic Name	**Characteristic UUID**	**Functions [R,W,N]**	**Example Value**
Service Changed	00002a05-0000-1000-8000-00805f9b34fb	R,N	0x0
Service Changed CCCD	00002902-0000-1000-8000-00805f9b34fb	R,W	<notifications status>

A.3 SmartCanton DevBox LoRaWAN

Table A.3 – Description du SmartCanton DevBox LoRaWAN service

SmartCanton DevBox LoRaWAN Service			
Service UUID	**ddddddd-dddd-cccc-bbbb-aaaa00000000**		
Characteristic Name	**Characteristic UUID**	**Rights [R,W,N]**	**Example Value**
AppEUI	ddddddd-dddd-cccc-bbbb-aaaa01000000	R,W	0x70B3D57ED 0008E8C
AppEUI CUDD	00002901-0000-1000-8000-00805f9b34fb	R	"LoRaWAN Application EUI"
AppKey	ddddddd-dddd-cccc-bbbb-aaaa02000000	W	0x509C1B97989D40C1 62D84681C91E67CD
AppKey CUDD	00002901-0000-1000-8000-00805f9b34fb	R	"LoRaWAN Application Key"
Device EUI	ddddddd-dddd-cccc-bbbb-aaaa03000000	R	0x323831395C367815
Device EUI CUDD	00002901-0000-1000-8000-00805f9b34fb	R	"LoRaWAN Application Key"
Confirmation Mode	ddddddd-dddd-cccc-bbbb-aaaa04000000	R,W	0x01
Confirmation Mode CUDD	00002901-0000-1000-8000-00805f9b34fb	R	"LoRaWAN Confirmation Mode"
Confirmation Mode	ddddddd-dddd-cccc-bbbb-aaaa05000000	R,W	0x01
Confirmation Mode CUDD	00002901-0000-1000-8000-00805f9b34fb	R	"LoRaWAN Network Join Status"
Device Addr	ddddddd-dddd-cccc-bbbb-aaaa06000000	R	0x32383000
Device Addr CUDD	00002901-0000-1000-8000-00805f9b34fb	R	"LoRaWAN Device Address"
Network Session Key	ddddddd-dddd-cccc-bbbb-aaaa07000000	R [Debug]	0x27B3A1BB9CA9AC 0E67880B4632CAEB4A
Network Session Key CUDD	00002901-0000-1000-8000-00805f9b34fb	R	"LoRaWAN Network Session Key"
App Session Key	ddddddd-dddd-cccc-bbbb-aaaa08000000	R [Debug]	0x59F8407F851FB75 57640B8345E50E729
App Session Key CUDD	00002901-0000-1000-8000-00805f9b34fb	R	"LoRaWAN App Session Key"
Validate New Config	ddddddd-dddd-cccc-bbbb-aaaa09000000	W	0x1
Validate New Config CUDD	00002901-0000-1000-8000-00805f9b34fb	R	"LoRaWAN Validate New Configuration"

A.4 SmartCanton DevBox GPS

Table A.4 – Description du SmartCanton DevBox GPS service

SmartCanton DevBox GPS Service			
Service UUID	**ddddddd-dddd-cccc-bbbb-aaaa00000010**		
Characteristic Name	**Characteristic UUID**	**Rights [R,W,N]**	**Example Value**
GPS Position	ddddddd-dddd-cccc-bbbb-aaaa01000010	R,W,N	"46.209126, 6.134028"
GPS Position CUDD	00002901-0000-1000-8000-00805f9b34fb	R	"GPS Position"
GPS Position CCCD	00002902-0000-1000-8000-00805f9b34fb	R,W	<notifications status>
GPS Speed	ddddddd-dddd-cccc-bbbb-aaaa02000010	R,W,N	"0.095"
GPS Speed CUDD	00002901-0000-1000-8000-00805f9b34fb	R	"GPS Speed"
GPS Speed CCCD	00002902-0000-1000-8000-00805f9b34fb	R,W	<notifications status>
GPS Course	ddddddd-dddd-cccc-bbbb-aaaa03000010	R,W,N	"0.1"
GPS Course CUDD	00002901-0000-1000-8000-00805f9b34fb	R	"GPS Course"
GPS Course CCCD	00002902-0000-1000-8000-00805f9b34fb	R,W	<notifications status>
GPS Date	ddddddd-dddd-cccc-bbbb-aaaa04000010	R,W,N	"16/01/18"
GPS Date CUDD	00002901-0000-1000-8000-00805f9b34fb	R	"GPS Date"
GPS Date CCCD	00002902-0000-1000-8000-00805f9b34fb	R,W	<notifications status>
GPS Time	ddddddd-dddd-cccc-bbbb-aaaa05000010	R,W,N	"13:35:02"
GPS Time CUDD	00002901-0000-1000-8000-00805f9b34fb	R	"GPS Time"
GPS Time CCCD	00002902-0000-1000-8000-00805f9b34fb	R,W	<notifications status>

Le service Bluetooth implémenté pour le GPS permet de recevoir les différentes informations provenant de celui-ci. Pour l'heure actuelle, uniquement les trames de type RMC sont supportée [64].

A.5 SmartCanton DevBox BME680

Table A.5 – Description du SmartCanton DevBox BME680 service

SmartCanton DevBox BME680 Service			
Service UUID	**ddddddd-dddd-cccc-bbbb-aaaa00000020**		
Characteristic Name	**Characteristic UUID**	**Rights [R,W,N]**	**Example Value**
IAQ	ddddddd-dddd-cccc-bbbb-aaaa01000020	N	0xACDAFE41
IAQ CUDD	00002901-0000-1000-8000-00805f9b34fb	R	"BME680 IAQ"
IAQ CCCD	00002902-0000-1000-8000-00805f9b34fb	R,W	<notifications status>
IAQ Accuracy	ddddddd-dddd-cccc-bbbb-aaaa02000020	N	0x01
IAQ Accuracy CUDD	00002901-0000-1000-8000-00805f9b34fb	R	"BME680 IAQ Accuracy"
IAQ Accuracy CCCD	00002902-0000-1000-8000-00805f9b34fb	R,W	<notifications status>
Temperature	ddddddd-dddd-cccc-bbbb-aaaa03000020	N	0x1F85B741
Temperature CUDD	00002901-0000-1000-8000-00805f9b34fb	R	"BME680 Temperature"
Temperature CCCD	00002902-0000-1000-8000-00805f9b34fb	R,W	<notifications status>
Humidity	ddddddd-dddd-cccc-bbbb-aaaa04000020	N	0x6b021121
Humidity CUDD	00002901-0000-1000-8000-00805f9b34fb	R	"BME680 Humidity"
Humidity CCCD	00002902-0000-1000-8000-00805f9b34fb	R,W	<notifications status>
Pressure	ddddddd-dddd-cccc-bbbb-aaaa05000020	R,N	0x710D7044
Pressure CUDD	00002901-0000-1000-8000-00805f9b34fb	R	"BME680 Pressure"
Pressure CCCD	00002902-0000-1000-8000-00805f9b34fb	R,W	<notifications status>
Raw Gas	ddddddd-dddd-cccc-bbbb-aaaa08000020	N	0x206AEA48
Raw Gas CUDD	00002901-0000-1000-8000-00805f9b34fb	R	"BME680 Raw Gas"
Raw Gas CCCD	00002902-0000-1000-8000-00805f9b34fb	R,W	<notifications status>
Raw Temperature	ddddddd-dddd-cccc-bbbb-aaaa06000020	R	0x52B8DC41
Raw Temperature CUDD	00002901-0000-1000-8000-00805f9b34fb	R	"BME680 Raw Temperature"
Raw Humidity	ddddddd-dddd-cccc-bbbb-aaaa07000020	R	0xBA49E441
Raw Humidity CUDD	00002901-0000-1000-8000-00805f9b34fb	R	"BME680 Raw Humidity

A.6 SmartCanton DevBox BNO055

Table A.6 – Description du SmartCanton DevBox BNO055 service

SmartCanton DevBox BNO055 Service			
Service UUID	**dddddddd-dddd-cccc-bbbb-aaaa00000030**		
Characteristic Name	**Characteristic UUID**	**Rights [R,W,N]**	**Example Value**
Accel xyz	dddddddd-dddd-cccc-bbbb-aaaa01000030	N	0x0000E0BB0000 44BD00807D3F
Accel xyz CUDD	00002901-0000-1000-8000-00805f9b34fb	R	"BNO055 Accelerometer xyz"
Accel xyz CCCD	00002902-0000-1000-8000-00805f9b34fb	R,W	<notifications status>
Gyroscope xyz	dddddddd-dddd-cccc-bbbb-aaaa02000030	N	0x0000E0BB0000 44BD00807D3F
Gyroscope xyz CUDD	00002901-0000-1000-8000-00805f9b34fb	R	"BNO055 Gyroscope xyz"
Gyroscope xyz CCCD	00002902-0000-1000-8000-00805f9b34fb	R,W	<notifications status>
Magnetometer xyz	dddddddd-dddd-cccc-bbbb-aaaa03000030	N	0x0000E0BB0000 44BD00807D3F
Magnetometer xyz CUDD	00002901-0000-1000-8000-00805f9b34fb	R	"BNO055 Magnetometer xyz"
Magnetometer xyz CCCD	00002902-0000-1000-8000-00805f9b34fb	R,W	<notifications status>
Measure interval	dddddddd-dddd-cccc-bbbb-aaaa04000030	R,W	0xB80B0000
Measure interval CUDD	00002901-0000-1000-8000-00805f9b34fb	R	"BNO055 Measure Interval"

A.7 SmartCanton DevBox BLE Scanner

Table A.7 – Description du SmartCanton DevBox BLE Scanner service

SmartCanton DevBox BLE Scanner Service			
Service UUID	**ddddddd-dddd-cccc-bbbb-aaaa00000040**		
Characteristic Name	**Characteristic UUID**	**Rights [R,W,N]**	**Example Value**
Devices Scanned	ddddddd-dddd-cccc-bbbb-aaaa01000040	R,N	0x0002
Devices Scanned CUDD	00002901-0000-1000-8000-00805f9b34fb	R	"BLE Devices Scanned"
Devices Scanned CCCD	00002902-0000-1000-8000-00805f9b34fb	R,W	<notifications status>
Measure interval	ddddddd-dddd-cccc-bbbb-aaaa02000040	R,W	0x1E00
Measure interval CUDD	00002901-0000-1000-8000-00805f9b34fb	R	"BLE Window Scan"

A.8 Batterie

Table A.8 – Description du Battery Service

Battery Service			
Service UUID	**0000180f-0000-1000-8000-00805f9b34fb**		
Characteristic Name	**Characteristic UUID**	**Functions [R,W,N]**	**Example Value**
Battery Level	0000180f-0000-1000-8000-00805f9b34fb	R,N	0x64
Battery Level CCCD	00002902-0000-1000-8000-00805f9b34fb	R,W	<notifications status>

Annexe B

AppKey Server : REST API Documentation

This document contain all the routes available on the AppKeyServer. The Figure B.1 illustrate the database model used by the server.

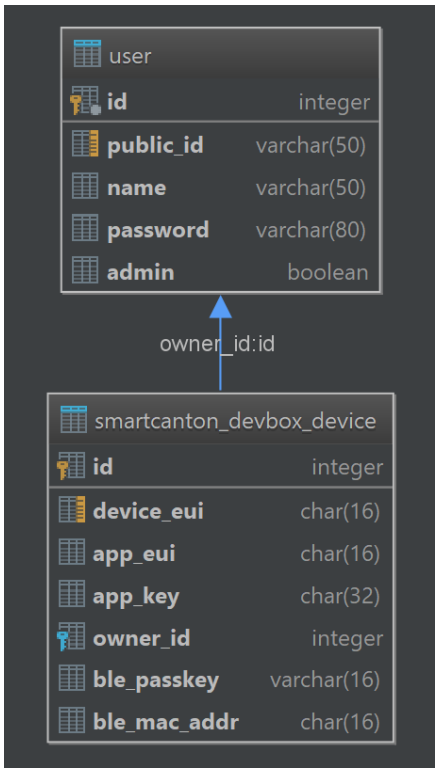


FIGURE B.1 – Database model

This documentation is also provided on the root path of the server :

```
$ curl -X GET https://127.0.0.1:5000/
```

B.1 Authentification

Command rights : Anyone can request a JWT token. The only requirement it that the user/password combinaison on the POST body is correct.

Curl command for a JWT authentification :

```
$ curl -X POST \
  http://127.0.0.1:5000/auth \
  -H 'cache-control: no-cache' \
  -H 'content-type: application/json' \
  -d '{
    "username": "<YOUR_USERNAME",
    "password": "<YOUR_PASSWORD>"
}'
```

Example of a correct response (HTTP code 200) :


```
{
    "access_token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJleHAiOjE1MTg0ND
    g2MDcsImlhdCI6MTUxNTg1NjYwNywibmJmIjoxNTE1ODU2NjA3LCJqdGkiOiIyYmE5NDE2ZC
    1lOTVmLTQ3ZDktYmM0Ny0xOTY4MmY5NzlhMTMiLCJwdWJsaWNfaWQiOiJkZjQ1ZThlMi02ZT
    EwLTRlYmEtOGJjZC0yMjJjYWNlMjk3NGUiLCJmcmVzaCI6ZmFsc2UsInR5cGUiOiJhY2Nlc3
    MifQ.2XzkxKE-qDDtooZGfW2Gp5tYXISvnUH1QHibyGSAkAI"
}
```


Token decrypted :


```
{
  "typ": "JWT",
  "alg": "HS256"
}
{
  "exp": 1518448607,
  "iat": 1515856607,
  "nbf": 1515856607,
  "jti": "2ba9416d-e95f-47d9-bc47-19682f979a13",
  "public_id": "df45e8e2-6e10-4eba-8bcd-222cace2974e",
  "fresh": false,
  "type": "access"
}
```

B.2 User routes

Those routes are only used to manage the user information. It does not interfer with the existing SmartCanton DevBox devices.

B.2.1 Get all users

Command rights : **Admin only**

Curl command :

```
$ curl -X GET \
  https://127.0.0.1:5000/user/df45e8e2-6e10-4eba-8bcd-222cace2974e \
  -H 'authorization: JWT eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJl
  eHAiOjE1MTg0NDg2MDcsImlhdCI6MTUxNTg1NjYwNywibmJmIjoxNTE1ODU2NjA3
  LCJqdGkiOiIyYmE5NDE2ZC1lOTVmLTQ3ZDktYmM0Ny0xOTY4MmY5NzlhMTMiLCJw
  dWJsaWNfaWQiOiJkZjQ1ZThlMi02ZTEwLTRlYmEtOGJjZC0yMjJjYWNlMjk3NGUi
  LCJmcmVzaCI6ZmFsc2UsInR5cGUiOiJhY2Nlc3MifQ.2XzkxKE-qDDtooZGfW2Gp
  5tYXISvnUH1QHibyGSAkAI' \
  -H 'cache-control: no-cache'
```

Example of a correct response (HTTP code 200) :

```
{
    "users": [
        {
            "admin": true,
            "name": "Admin",
            "public_id": "df45e8e2-6e10-4eba-8bcd-222cace2974e"
        },
        {
            "admin": false,
            "name": "david",
            "public_id": "93cb11d2-bbdd-4d2a-8e66-e970e6d6bd25"
        },
        {
            "admin": true,
            "name": "david2",
            "public_id": "0b6a165d-5126-46da-bd0d-9d35bdda4bfd"
        }
    ]
}
```

B.2.2 Get a specific user information from a public id

Command rights : Non admin users can only see their own profiles. Admin users can access all profiles.

Curl command :

```
$ curl -X GET \
  https://127.0.0.1:5000/user/df45e8e2-6e10-4eba-8bcd-222cace2974e \
  -H 'authorization: JWT eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJl
  eHAiOjE1MTg0NDg2MDcsImlhdCI6MTUxNTg1NjYwNywibmJmIjoxNTE1ODU2NjA3
  LCJqdGkiOiIyYmE5NDE2ZC1lOTVmLTQ3ZDktYmM0Ny0xOTY4MmY5NzlhMTMiLCJw
  dWJsaWNfaWQiOiJkZjQ1ZThlMi02ZTEwLTRlYmEtOGJjZC0yMjJjYWNlMjk3NGUi
  LCJmcmVzaCI6ZmFsc2UsInR5cGUiOiJhY2Nlc3MifQ.2XzkxKE-qDDtooZGfW2Gp
  5tYXISvnUH1QHibyGSAkAI' \
  -H 'cache-control: no-cache'
```

Example of a correct response (HTTP code 200) :

```
{
    "admin": true,
    "public_id": "df45e8e2-6e10-4eba-8bcd-222cace2974e",
    "username": "Admin"
}
```

B.2.3 Update information about a specific user

Command rights : Non admin users can only change their own profile and can't give themselves the Admin rights. They need to know their current password to be able to change it. Admin users can change all profiles and passwords, even if they don't know the old password.

Curl command :

```
$ curl -X PUT \
  https://127.0.0.1:5000/user/df45e8e2-6e10-4eba-8bcd-222cace2974e \
  -H 'authorization: JWT eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJl
  eHAiOjE1MTg0NDg2MDcsImlhdCI6MTUxNTg1NjYwNywibmJmIjoxNTE1ODU2NjA3
  LCJqdGkiOiIyYmE5NDE2ZC1lOTVmLTQ3ZDktYmM0Ny0xOTY4MmY5NzlhMTMiLCJw
  dWJsaWNfaWQiOiJkZjQ1ZThlMi02ZTEwLTRlYmEtOGJjZC0yMjJjYWNlMjk3NGUi
  LCJmcmVzaCI6ZmFsc2UsInR5cGUiOiJhY2Nlc3MifQ.2XzkxKE-qDDtooZGfW2Gp
  5tYXISvnUH1QHibyGSAkAI' \
  -H 'cache-control: no-cache' \
  -H 'content-type: application/json' \
  -d '{
        "password" : "test",
        "new_password" : "SmartCanton2017",
        "admin" : true
}'
```

Example of a correct response (HTTP code 200) :

```
{
    "message": "User information updated!"
}
```

B.2.4 Create a new user

Command rights : **Admin only**

Curl command :

```
$ curl -X POST \
  http://127.0.0.1:5000/user \
  -H 'authorization: JWT eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJl
  eHAiOjE1MTg0NDg2MDcsImlhdCI6MTUxNTg1NjYwNywibmJmIjoxNTE1ODU2NjA3
  LCJqdGkiOiIyYmE5NDE2ZC1lOTVmLTQ3ZDktYmM0Ny0xOTY4MmY5NzlhMTMiLCJw
  dWJsaWNfaWQiOiJkZjQ1ZThlMi02ZTEwLTRlYmEtOGJjZC0yMjJjYWNlMjk3NGUi
  LCJmcmVzaCI6ZmFsc2UsInR5cGUiOiJhY2Nlc3MifQ.2XzkxKE-qDDtooZGfW2Gp
  5tYXISvnUH1QHibyGSAkAI' \
  -H 'cache-control: no-cache' \
  -H 'content-type: application/json' \
  -d '{"username": "david2", "password": "SmartCanton"}'
```

Example of a correct response (HTTP code 200) :

```
{
    "message": "New user created!"
}
```

B.2.5 Delete an existing user

Command rights : **Admin only**

Curl command :

```
$ curl -X DELETE \
  https://127.0.0.1:5000/user/88320280-42c6-4ad1-b747-cc4844580fa7 \
  -H 'authorization: JWT eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJl
  eHAiOjE1MTg0NDg2MDcsImlhdCI6MTUxNTg1NjYwNywibmJmIjoxNTE1ODU2NjA3
  LCJqdGkiOiIyYmE5NDE2ZC1lOTVmLTQ3ZDktYmM0NyOxOTY4MmY5NzlhMTMiLCJw
  dWJsaWNfaWQiOiJkZjQ1ZThlMiO2ZTEwLTRlYmEtOGJjZCOyMjJjYWNlMjk3NGUi
  LCJmcmVzaCI6ZmFsc2UsInR5cGUiOiJhY2Nlc3MifQ.2XzkxKE-qDDtooZGfW2Gp
  5tYXISvnUH1QHibyGSAkAI' \
  -H 'cache-control: no-cache'
```

Example of a correct response (HTTP code 200) :

```
{
    "message": "The user has been deleted!"
}
```

B.3 SmartCanton DevBox routes

Those routes are only used to manage the DevBox devices. You will need to know the user informations to update or understand the *owner_public_id* field.

B.3.1 Get all devices

Command rights : Non admin users can only see their own devices (those where their public id is used as the owner id). Admin users can see all devices.

Curl command :

```
$ curl -X GET \
  https://127.0.0.1:5000/device \
  -H 'authorization: JWT eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJl
  eHAiOjE1MTg0NDg2MDcsImlhdCI6MTUxNTg1NjYwNywibmJmIjoxNTE1ODU2NjA3
  LCJqdGkiOiIyYmE5NDE2ZC1lOTVmLTQ3ZDktYmM0NyOxOTY4MmY5NzlhMTMiLCJw
  dWJsaWNfaWQiOiJkZjQ1ZThlMiO2ZTEwLTRlYmEtOGJjZCOyMjJjYWNlMjk3NGUi
  LCJmcmVzaCI6ZmFsc2UsInR5cGUiOiJhY2Nlc3MifQ.2XzkxKE-qDDtooZGfW2Gp
  5tYXISvnUH1QHibyGSAkAI' \
  -H 'cache-control: no-cache'
```

Example of a correct response (HTTP code 200) :

```
{
    "devices": [
        {
            "ble_mac_addr": "006037000000",
            "dev_eui": "323831395C367815"
        },
        {
            "ble_mac_addr": "006037000001",
            "dev_eui": "323831395C367715"
        }
    ]
}
```

B.3.2 Get information about a specific device

Command rights : Non admin users can only see their own devices (those where their public id is used as the owner id). Admin users can see all devices.

Curl command :

```
$ curl -X GET \
  https://127.0.0.1:5000/device/006037000001 \
  -H 'authorization: JWT eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJl
  eHAiOjE1MTg0NDg2MDcsImlhdCI6MTUxNTg1NjYwNywibmJmIjoxNTE1ODU2NjA3
  LCJqdGkiOiIyYmE5NDE2ZC1lOTVmLTQ3ZDktYmM0NyOxOTY4MmY5NzlhMTMiLCJw
  dWJsaWNfaWQiOiJkZjQ1ZThlMiO2ZTEwLTRlYmEtOGJjZCOyMjJjYWNlMjk3NGUi
  LCJmcmVzaCI6ZmFsc2UsInR5cGUiOiJhY2Nlc3MifQ.2XzkxKE-qDDtooZGfW2Gp
  5tYXISvnUH1QHibyGSAkAI' \
  -H 'cache-control: no-cache'
```

Example of a correct response (HTTP code 200) :

```
{
    "app_eui": "<APP_EUI>",
    "app_key": "<APP_KEY>",
    "ble_mac_addr": "006037000001",
    "ble_passkey": "111100",
    "dev_eui": "323831395C367715",
    "owner_public_id": "df45e8e2-6e10-4eba-8bcd-222cace2974e"
}
```

B.3.3 Create a new device

Command rights : Devices are automatically assigned to the authentified user for non admin users. An admin can change the `owner_public_id` field to transfer the ownership of a device.

Curl command :

```
$ curl -X POST \
  http://127.0.0.1:5000/device \
  -H 'authorization: JWT eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJl
  eHAiOjE1MTg0NDg2MDcsImlhdCI6MTUxNTg1NjYwNywibmJmIjoxNTE1ODU2NjA3
  LCJqdGkiOiIyYmE5NDE2ZC1lOTVmLTQ3ZDktYmM0Ny0xOTY4MmY5NzlhMTMiLCJw
  dWJsaWNfaWQiOiJkZjQ1ZThlMi02ZTEwLTRlYmEtOGJjZC0yMjJjYWNlMjk3NGUi
  LCJmcmVzaCI6ZmFsc2UsInR5cGUiOiJhY2Nlc3MifQ.2XzkxKE-qDDtooZGfW2Gp
  5tYXISvnUH1QHibyGSAkAI' \
  -H 'cache-control: no-cache' \
  -H 'content-type: application/json' \
  -d '{
    "app_eui": "<APP_EUI>",
    "app_key": "<APP_KEY>",
    "dev_eui": "323831395C367801",
    "ble_mac_addr": "006037000005",
    "ble_passkey": "111111",
    "owner_public_id": "df45e8e2-6e10-4eba-8bcd-222cace2974e"
}'
```

Example of a correct response (HTTP code 200) :

```
{
    "message": "New device created!"
}
```

B.3.4 Update an existing device

Command rights : Non admin users can only edit their own devices (those where their public id is used as the owner id). Admin users can edit all devices.

Curl command to edit a specidic field (ble passkey in this case) :

```
$ curl -X PUT \
  http://127.0.0.1:5000/device \
  -H 'authorization: JWT eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJl
  eHAiOjE1MTg0NDg2MDcsImlhdCI6MTUxNTg1NjYwNywibmJmIjoxNTE1ODU2NjA3
  LCJqdGkiOiIyYmE5NDE2ZC1lOTVmLTQ3ZDktYmM0Ny0xOTY4MmY5NzlhMTMiLCJw
  dWJsaWNfaWQiOiJkZjQ1ZThlMi02ZTEwLTRlYmEtOGJjZC0yMjJjYWNlMjk3NGUi
  LCJmcmVzaCI6ZmFsc2UsInR5cGUiOiJhY2Nlc3MifQ.2XzkxKE-qDDtooZGfW2Gp
  5tYXISvnUH1QHibyGSAkAI' \
  -H 'cache-control: no-cache' \
  -H 'content-type: application/json' \
  -d '{
    "ble_passkey": "111122",
}'
```

Example of a correct response (HTTP code 200) :

```
{
    "message": "Device updated successfully!"
}
```

B.3.5 Delete an existing device

Command rights : Non admin users can only edit their own devices (those where their public id is used as the owner id). Admin users can edit all devices.

Curl command :

```
$ curl -X DELETE \
  http://127.0.0.1:5000/device/006037000005 \
  -H 'authorization: JWT eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJl
  eHAiOjE1MTg0NDg2MDcsImlhdCI6MTUxNTg1NjYwNywibmJmIjoxNTE1ODU2NjA3
  LCJqdGkiOiIyYmE5NDE2ZC1lOTVmLTQ3ZDktYmM0NyOxOTY4MmY5NzlhMTMiLCJw
  dWJsaWNfaWQiOiJkZjQ1ZThlMiO2ZTEwLTRlYmEtOGJjZCOyMjJjYWNlMjk3NGUi
  LCJmcmVzaCI6ZmFsc2UsInR5cGUiOiJhY2Nlc3MifQ.2XzkxKE-qDDtooZGfW2Gp
  5tYXISvnUH1QHibyGSAkAI' \
  -H 'cache-control: no-cache'
```

Example of a correct response (HTTP code 200) :

```
{
    "message": "SmartCanton Devbox item deleted!"
}
```

Annexe C

Android SmartCanton Manager : User's Guide

C.1 Login Activity

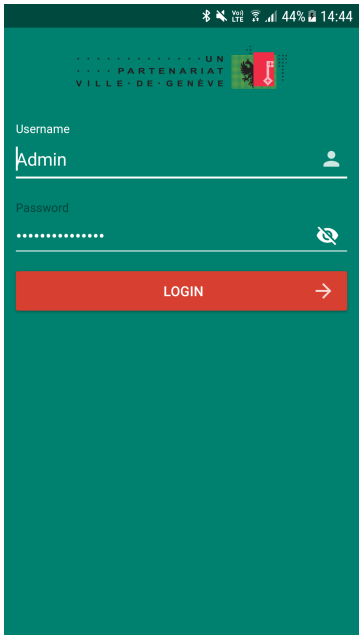


FIGURE C.1 – SmartCanton Manager login activity

When the application is launched, the user is required to log in using his credentials from the AppKey Server, as shown by the Figure C.1.

C.2 Bluetooth Activity : Scanner BLE Fragment

Once the user logged, all devices broadcasting ble packets nearby will be displayed in a list, as shown by the Figure C.2. The list can take a bit to be updated, just wait a moment. If the list still not display itself, just activate the *airplane mode* on the phone and restart the SmartCanton Manager app.

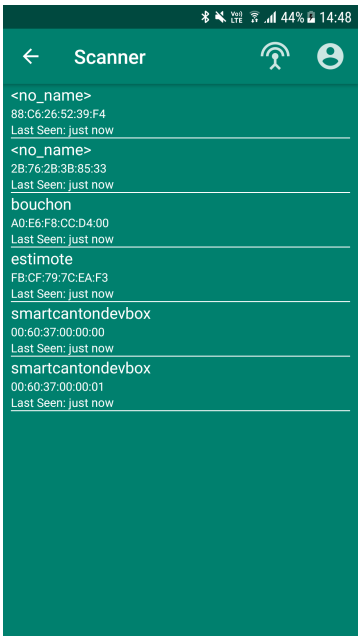


FIGURE C.2 – SmartCanton Manager scanner fragment

C.3 Bluetooth Activity : Device Management Fragment

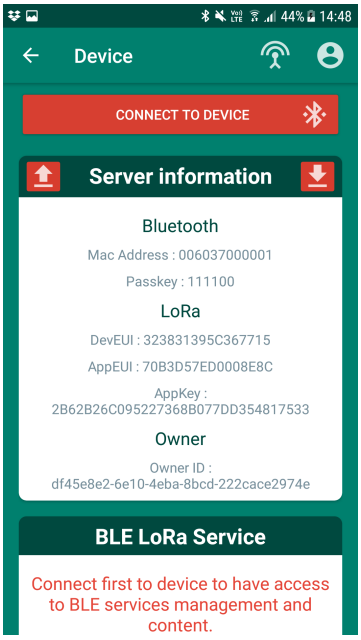


FIGURE C.3 – SmartCanton Manager device found on the server

When a Bluetooth device is selected on the BLE Scanner list, the fragment illustrated by the Figure C.3 is shown. If the device is found on the server, his information are displayed inside the `Server information` card. Those data can be updated by pressing the arrow on the right of the card.

To connect to the device using Bluetooth Low Energy, the red button named `CONNECT TO DEVICE` on the top of the fragment needs to be pressed. The Figure C.5 shows the sequence of the connection. First a dialog inform the user that a BLE Passkey is present in the device and that the user need to remember it. Then the

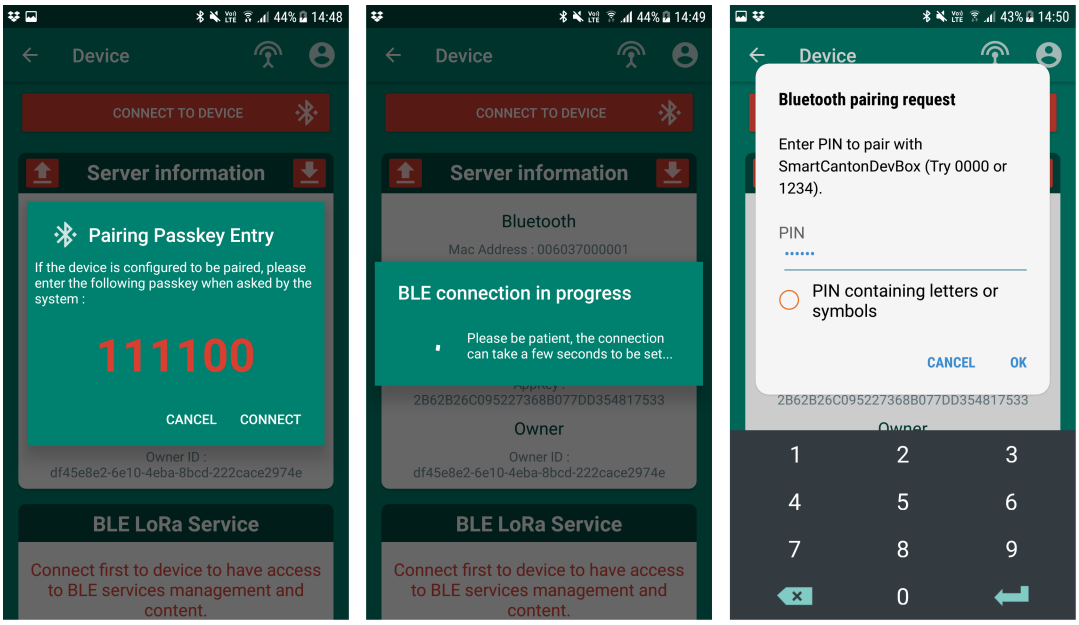


FIGURE C.4 – SmartCanton Manager connection progress

application will try to connect to the device. Finally, Android will ask the user to pair and bond with the device using the BLE Passkey previously displayed.

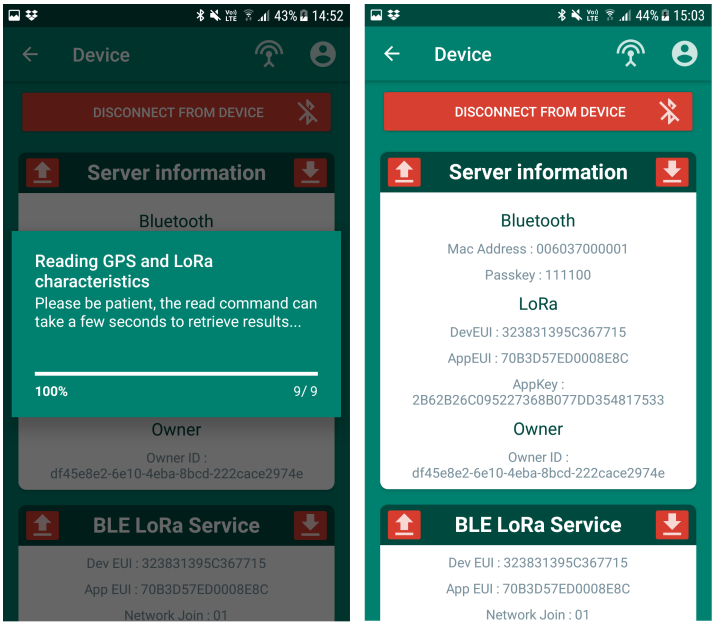


FIGURE C.5 – SmartCanton Manager device connected and characteristics read

Once the device is connected, the characteristics from the LoRa Service and the GPS of the DevBox are read and displayed in their respective cards, as shown on the Figure C.5. The user can access to the multiples Bluetooth services available on the device. The user can disconnect from the device by leaving the current fragment (back button) or by pressing the `DISCONNECT FROM DEVICE` button on the top of the page.

When the red arrow on the left of the `BLE LoRaWAN Service` is pressed, the device will be configured with the data read from the Server and diplayed in the

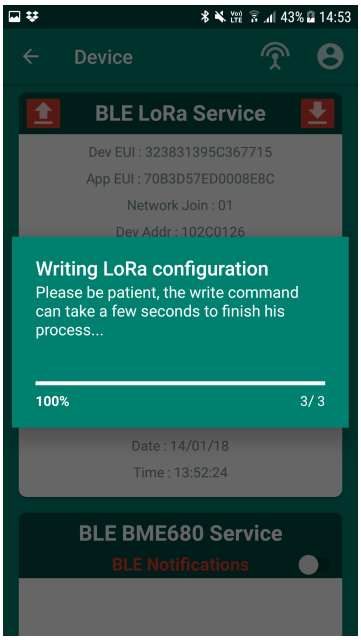


FIGURE C.6 – SmartCanton Manager update LoRaWAN config from server configuration

`Server information` card. While the LoRaWAN service is been updated, a progress dialog is shown like in the Figure C.6.

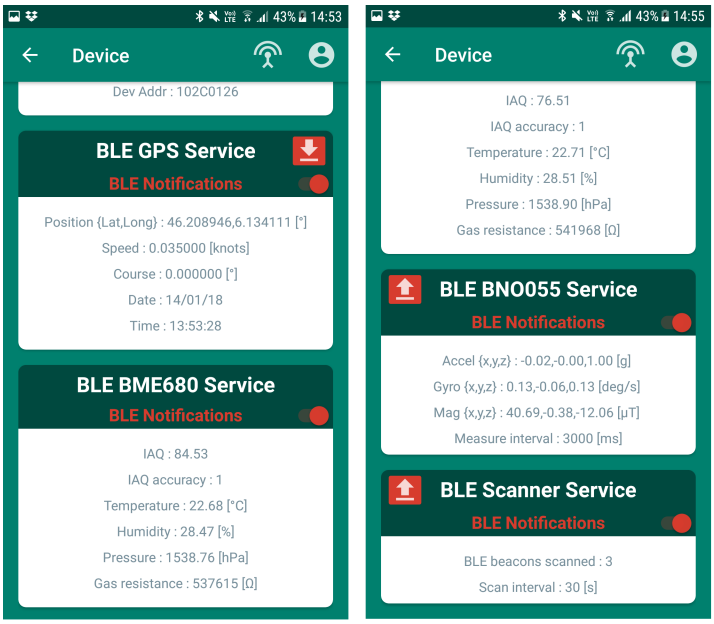


FIGURE C.7 – SmartCanton Manager enable sensors notifications

The user can toggle the notifications button for each sensor, as shown by the Figure C.7. Everytime a new data is read from the sensors, the application will display it.

The user can configure the interval beetween each notifications from the BNO055 or the BLE Scanner, as shown by the Figure C.8.

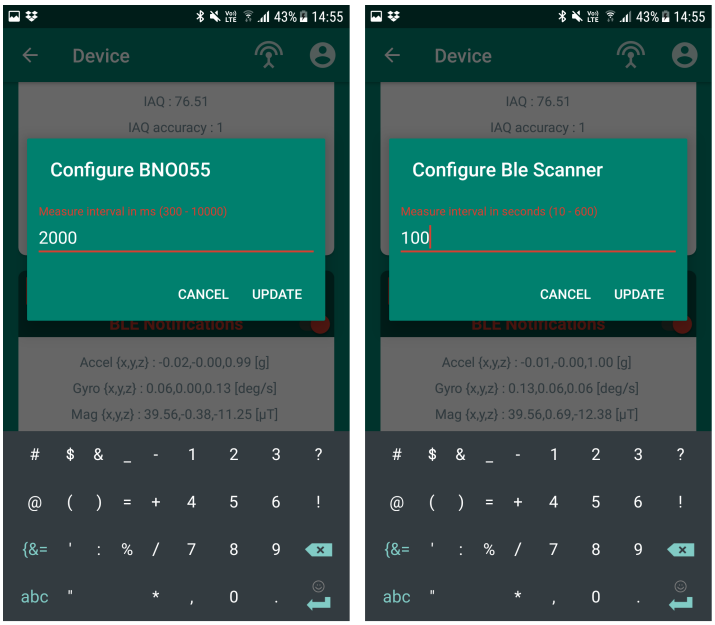


FIGURE C.8 – SmartCanton Manager edit bno055 or ble scanner interval

C.4 Profile Activity

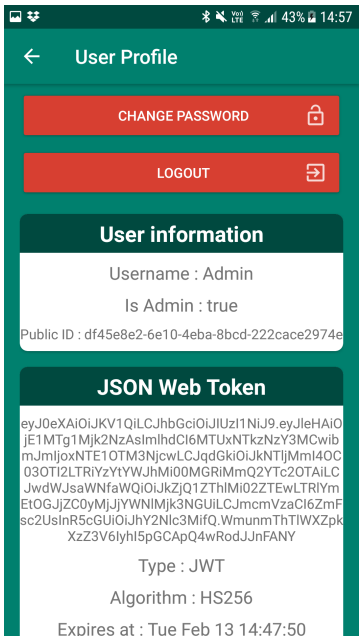


FIGURE C.9 – SmartCanton Manager profile activity

The user can access his profile by clicking on the user icon on the Action Bar. The profile activity, as it can be seen on the Figure C.9, contain all information stored on the server for the current logged user. The user can logout from the current session by pressing the `LOGOUT` button.

C.5 Beacon Generator Activity

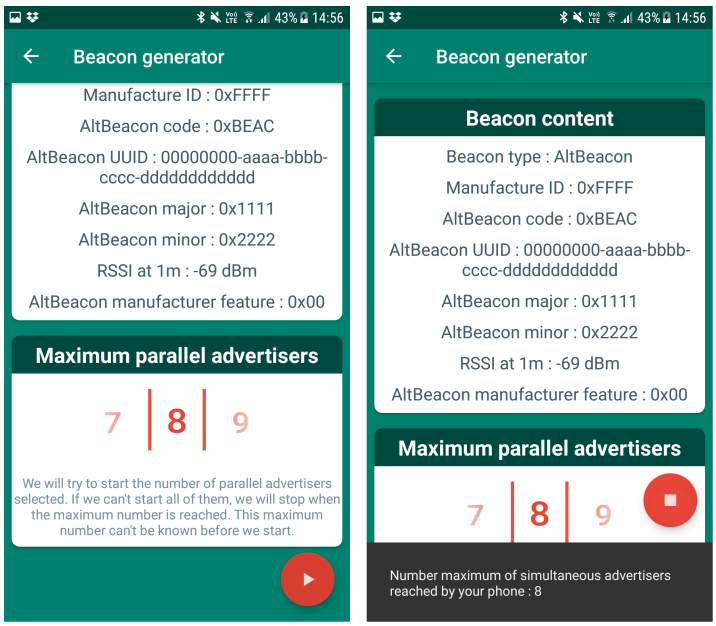


FIGURE C.10 – SmartCanton Manager beacon generator activity

A beacon generator has been integrated inside the Android application. His purpose is to simulate nearby devices that can be scanned by a DevBox. The Figure C.10 show the activity layout and how the app react when the *play* button is pressed.

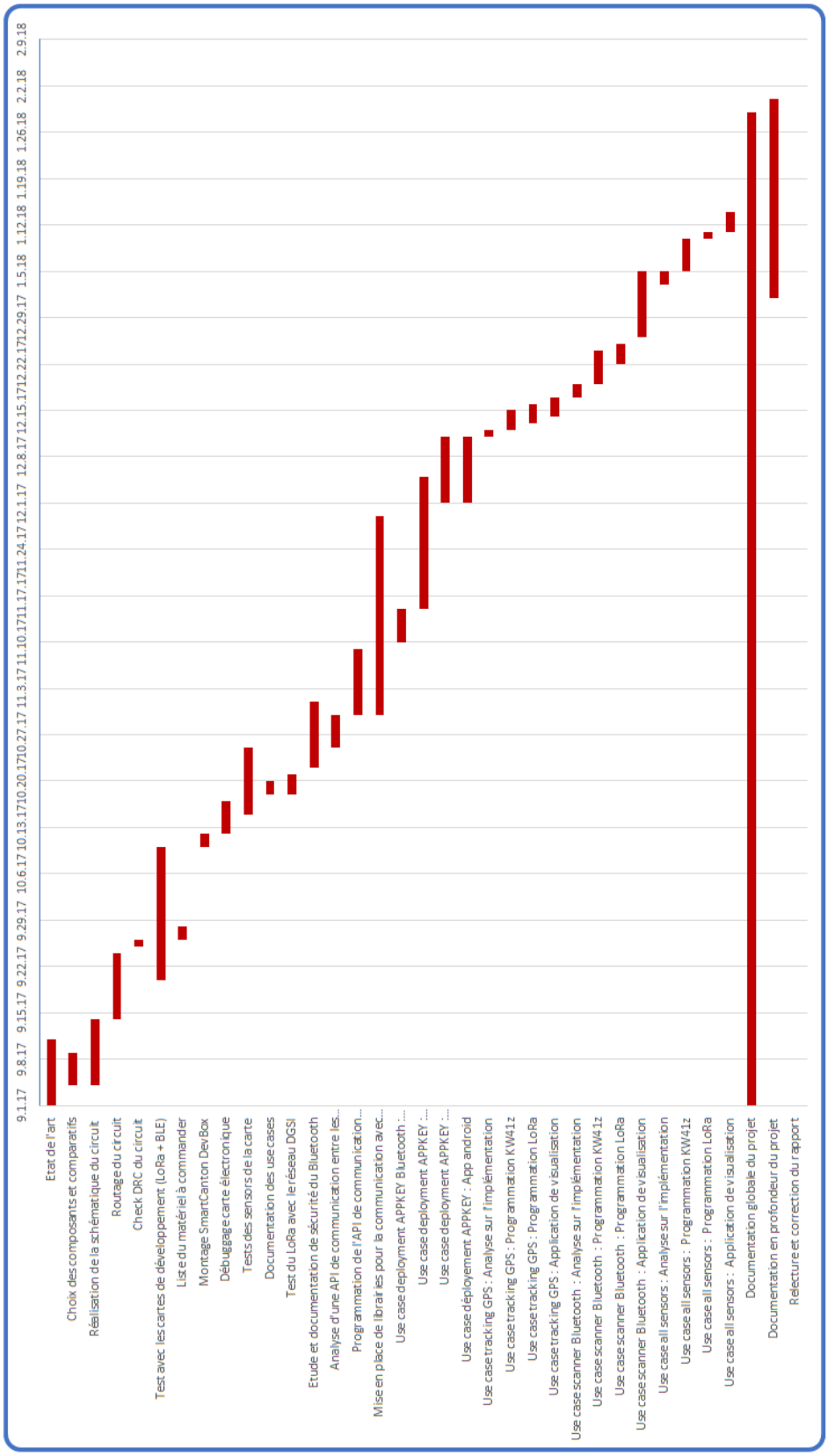
The user can select the number of devices that will be simulated by the application. However, each device has a number of maximum advertiser that can run at the same time. If the user select a number of devices greater than the maximum number available, the maximum number will be displayed to the user, as shown by the Figure C.10.

Annexe D

Planification

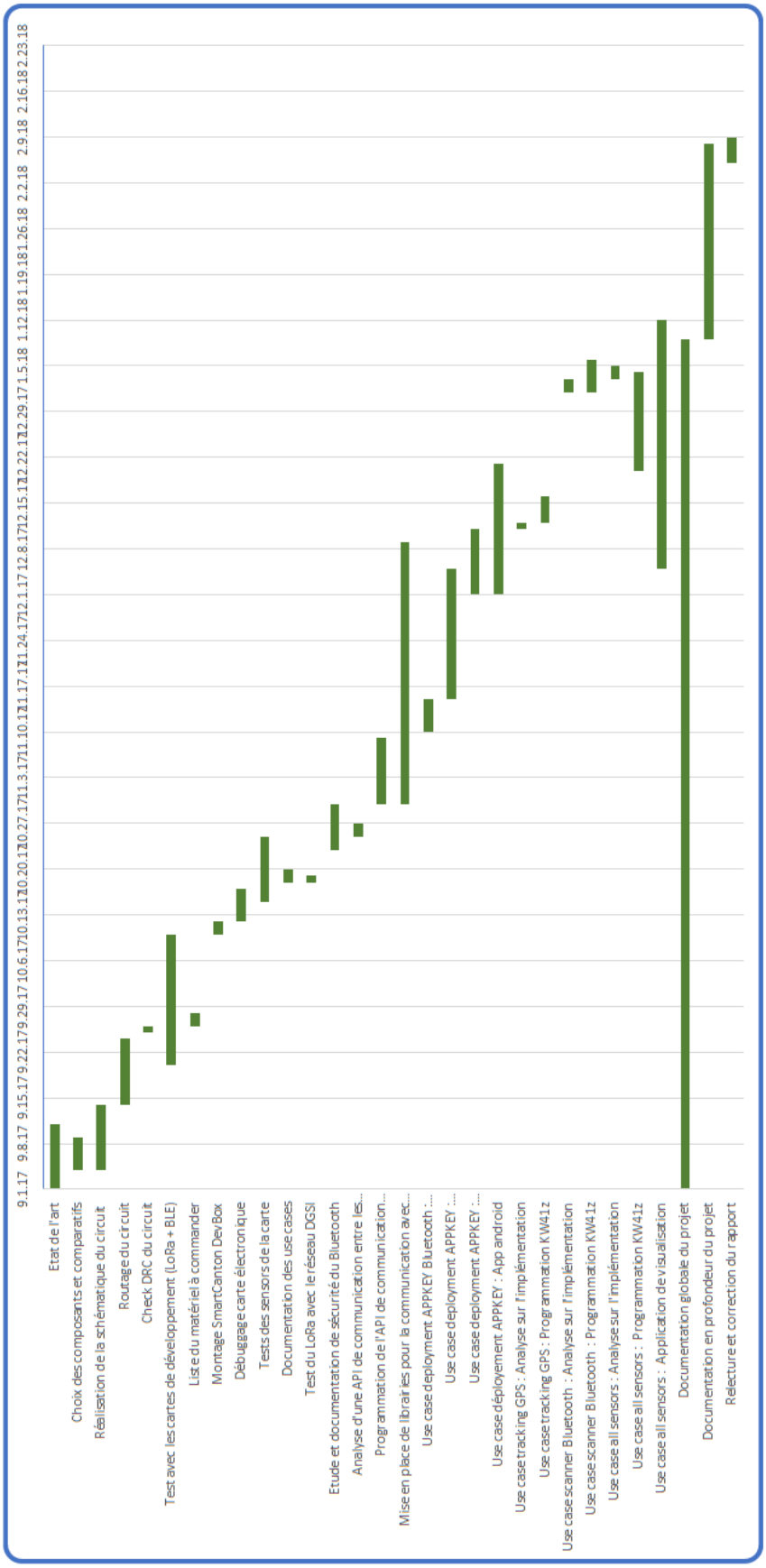
D.1 Temps planifié pour chaque tâche

Task Name	Start Date	End Date	Duration (Days)	Days Complete	Days Remaining	Percent Complete
Etat de l'art	01.09.2017	11.09.2017	10	0.00	10.00	0%
Choix des composants et comparatifs	04.09.2017	09.09.2017	5	0.00	5.00	0%
Réalisation de la schématique du circuit	04.09.2017	14.09.2017	10	0.00	10.00	0%
Routage du circuit	14.09.2017	24.09.2017	10	0.00	10.00	0%
Check DRC du circuit	25.09.2017	26.09.2017	1	0.00	1.00	0%
Test avec les cartes de développement (LoRa + BLE)	20.09.2017	10.10.2017	20	0.00	20.00	0%
Liste du matériel à commander	26.09.2017	28.09.2017	2	0.00	2.00	0%
Montage SmartCanton DevBox	10.10.2017	12.10.2017	2	0.00	2.00	0%
Débuggage carte électronique	12.10.2017	17.10.2017	5	0.00	5.00	0%
Tests des sensors de la carte	15.10.2017	25.10.2017	10	0.00	10.00	0%
Documentation des use cases	18.10.2017	20.10.2017	2	0.00	2.00	0%
Test du LoRa avec le réseau DGSI	18.10.2017	21.10.2017	3	0.00	3.00	0%
Etude et documentation de sécurité du Bluetooth	22.10.2017	01.11.2017	10	0.00	10.00	0%
Analyse d'une API de communication entre les deux microcontrôleur (LoRa <-> BLE)	25.10.2017	30.10.2017	5	0.00	5.00	0%
Programmation de l'API de communication entre microcontrôleurs	30.10.2017	09.11.2017	10	0.00	10.00	0%
Mise en place de librairies pour la communication avec les différents périphériques	30.10.2017	29.11.2017	30	0.00	30.00	0%
Use case deployment APPKEY Bluetooth : Analyse sur l'implémentation	10.11.2017	15.11.2017	5	0.00	5.00	0%
Use case deployment APPKEY : Programmation KW41z + LoRa	15.11.2017	05.12.2017	20	0.00	20.00	0%
Use case deployment APPKEY : Programmation serveur REST de partage de clés	01.12.2017	11.12.2017	10	0.00	10.00	0%
Use case déployement APPKEY : App android	01.12.2017	11.12.2017	10	0.00	10.00	0%
Use case tracking GPS : Analyse sur l'implémentation	11.12.2017	12.12.2017	1	0.00	1.00	0%
Use case tracking GPS : Programmation KW41z	12.12.2017	15.12.2017	3	0.00	3.00	0%
Use case tracking GPS : Programmation LoRa	13.12.2017	16.12.2017	3	0.00	3.00	0%
Use case tracking GPS : Application de visualisation	14.12.2017	17.12.2017	3	0.00	3.00	0%
Use case scanner Bluetooth : Analyse sur l'implémentati	17.12.2017	19.12.2017	2	0.00	2.00	0%
Use case scanner Bluetooth : Programmation KW41z	19.12.2017	24.12.2017	5	0.00	5.00	0%
Use case scanner Bluetooth : Programmation LoRa	22.12.2017	25.12.2017	3	0.00	3.00	0%
Use case scanner Bluetooth : Application de visualisatio	26.12.2017	05.01.2018	10	0.00	10.00	0%
Use case all sensors : Analyse sur l'implémentation	03.01.2018	05.01.2018	2	0.00	2.00	0%
Use case all sensors : Programmation KW41z	05.01.2018	10.01.2018	5	0.00	5.00	0%
Use case all sensors : Programmation LoRa	10.01.2018	11.01.2018	1	0.00	1.00	0%
Use case all sensors : Application de visualisation	11.01.2018	14.01.2018	3	0.00	3.00	0%
Documentation globale du projet	01.09.2017	29.01.2018	150	7.50	142.50	5%
Documentation en profondeur du projet	01.01.2018	31.01.2018	30	0.00	30.00	0%
Relecture et correction du rapport	30.01.2017	09.02.2017	10	0.00	10.00	0%



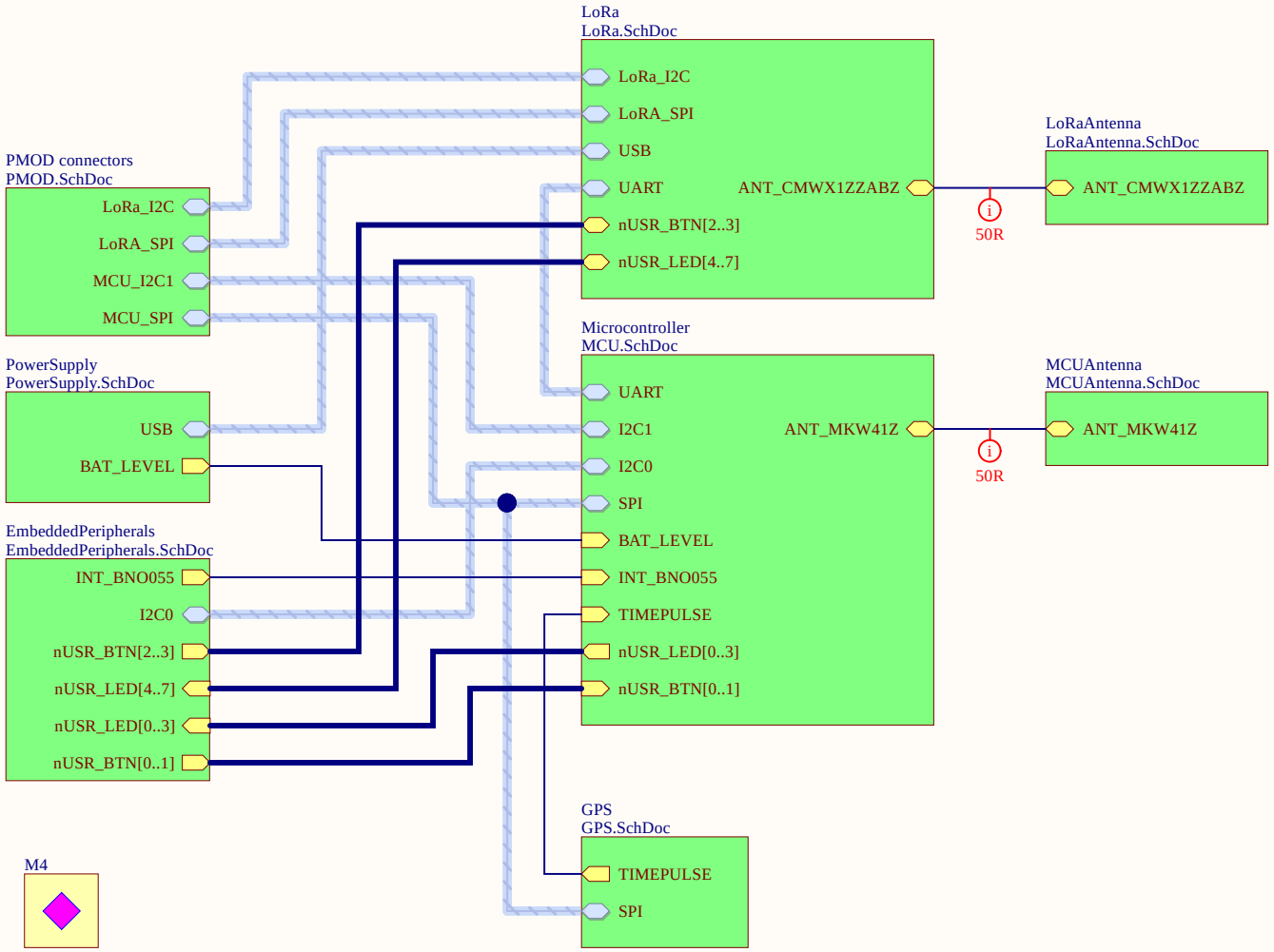
D.2 Temps écoulé pour chaque tâche

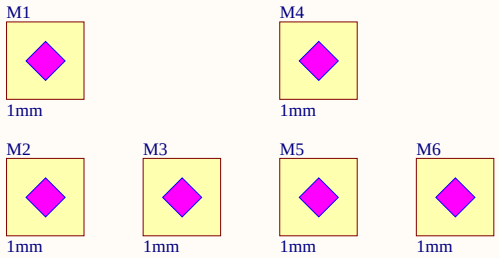
Task Name	Start Date	End Date	Duration (Days)	Days Complete	Days Remaining	Percent Complete
Etat de l'art	01.09.2017	11.09.2017	10	10.00	0.00	100%
Choix des composants et comparatifs	04.09.2017	09.09.2017	5	5.00	0.00	100%
Réalisation de la schématique du circuit	04.09.2017	14.09.2017	10	10.00	0.00	100%
Routage du circuit	14.09.2017	24.09.2017	10	10.00	0.00	100%
Check DRC du circuit	25.09.2017	26.09.2017	1	1.00	0.00	100%
Test avec les cartes de développement (LoRa + BLE)	20.09.2017	10.10.2017	20	20.00	0.00	100%
Liste du matériel à commander	26.09.2017	28.09.2017	2	2.00	0.00	100%
Montage SmartCanton DevBox	10.10.2017	12.10.2017	2	2.00	0.00	100%
Débuggage carte électronique	12.10.2017	17.10.2017	5	5.00	0.00	100%
Tests des sensors de la carte	15.10.2017	25.10.2017	10	10.00	0.00	100%
Documentation des use cases	18.10.2017	20.10.2017	2	2.00	0.00	100%
Test du LoRa avec le réseau DGSI	18.10.2017	19.10.2017	1	1.00	0.00	100%
Etude et documentation de sécurité du Bluetooth	23.10.2017	30.10.2017	7	7.00	0.00	100%
Analyse d'une API de communication entre les deux microcontrôleur (LoRa <-> BLE)	25.10.2017	27.10.2017	2	2.00	0.00	100%
Programmation de l'API de communication entre microcontrôleurs	30.10.2017	09.11.2017	10	10.00	0.00	100%
Mise en place de librairies pour la communication avec les différents périphériques	30.10.2017	09.12.2017	40	40.00	0.00	100%
Use case deployment APPKEY Bluetooth : Analyse sur l'implémentation	10.11.2017	15.11.2017	5	5.00	0.00	100%
Use case deployment APPKEY : Programmation KW41z + LoRa	15.11.2017	05.12.2017	20	20.00	0.00	100%
Use case deployment APPKEY : Programmation serveur REST de partage de clés	01.12.2017	11.12.2017	10	10.00	0.00	100%
Use case déployement APPKEY : App android	01.12.2017	21.12.2017	20	20.00	0.00	100%
Use case tracking GPS : Analyse sur l'implémentation	11.12.2017	12.12.2017	1	1.00	0.00	100%
Use case tracking GPS : Programmation KW41z	12.12.2017	16.12.2017	4	4.00	0.00	100%
Use case scanner Bluetooth : Analyse sur l'implémentation	01.01.2018	03.01.2018	2	2.00	0.00	100%
Use case scanner Bluetooth : Programmation KW41z	01.01.2018	06.01.2018	5	5.00	0.00	100%
Use case all sensors : Analyse sur l'implémentation	03.01.2018	05.01.2018	2	2.00	0.00	100%
Use case all sensors : Programmation KW41z	20.12.2017	04.01.2018	15	15.00	0.00	100%
Use case all sensors : Application de visualisation	05.12.2017	12.01.2018	38	38.00	0.00	100%
Documentation globale du projet	01.09.2017	09.01.2018	130	130.00	0.00	100%
Documentation en profondeur du projet	09.01.2018	08.02.2018	30	30.00	0.00	100%
Relecture et correction du rapport	05.02.2018	09.02.2018	4	4.00	0.00	100%


9.1.17
9.8.17
9.15.17
9.22.17
9.29.17
10.6.17
10.13.17
10.20.17
10.27.17
11.3.17
11.10.17
11.17.17
11.24.17
12.1.17
12.8.17
12.15.17
12.22.17
12.29.17
1.5.18
1.12.18
1.19.18
1.26.18
2.2.18
2.9.18
2.16.18
2.23.18
Etat de l'art
Choix des composants et comparatifs
Réalisation de la schématique du circuit
Routage du circuit
Check DRC du circuit
Test avec les cartes de développement (LoRa + BLE)
Liste du matériel à commander
Montage SmartCanton DevBox
Débuggage carte électronique
Tests des sensors de la carte
Documentation des use cases
Test du LoRa avec le réseau DGSI
Etude et documentation de sécurité du Bluetooth
Analyse d'une API de communication entre les...
Programmation de l'API de communication...
Mise en place de librairies pour la communication avec...
Use case deployment APPKEY Bluetooth :...
Use case deployment APPKEY :...
Use case deployment APPKEY :...
Use case déployement APPKEY : App android
Use case tracking GPS : Analyse sur l'implémentation
Use case tracking GPS : Programmation KW41z
Use case scanner Bluetooth : Analyse sur l'implémentation
Use case scanner Bluetooth : Programmation KW41z
Use case all sensors : Analyse sur l'implémentation
Use case all sensors : Programmation KW41z
Use case all sensors : Application de visualisation
Documentation globale du projet
Documentation en profondeur du projet
Relecture et correction du rapport

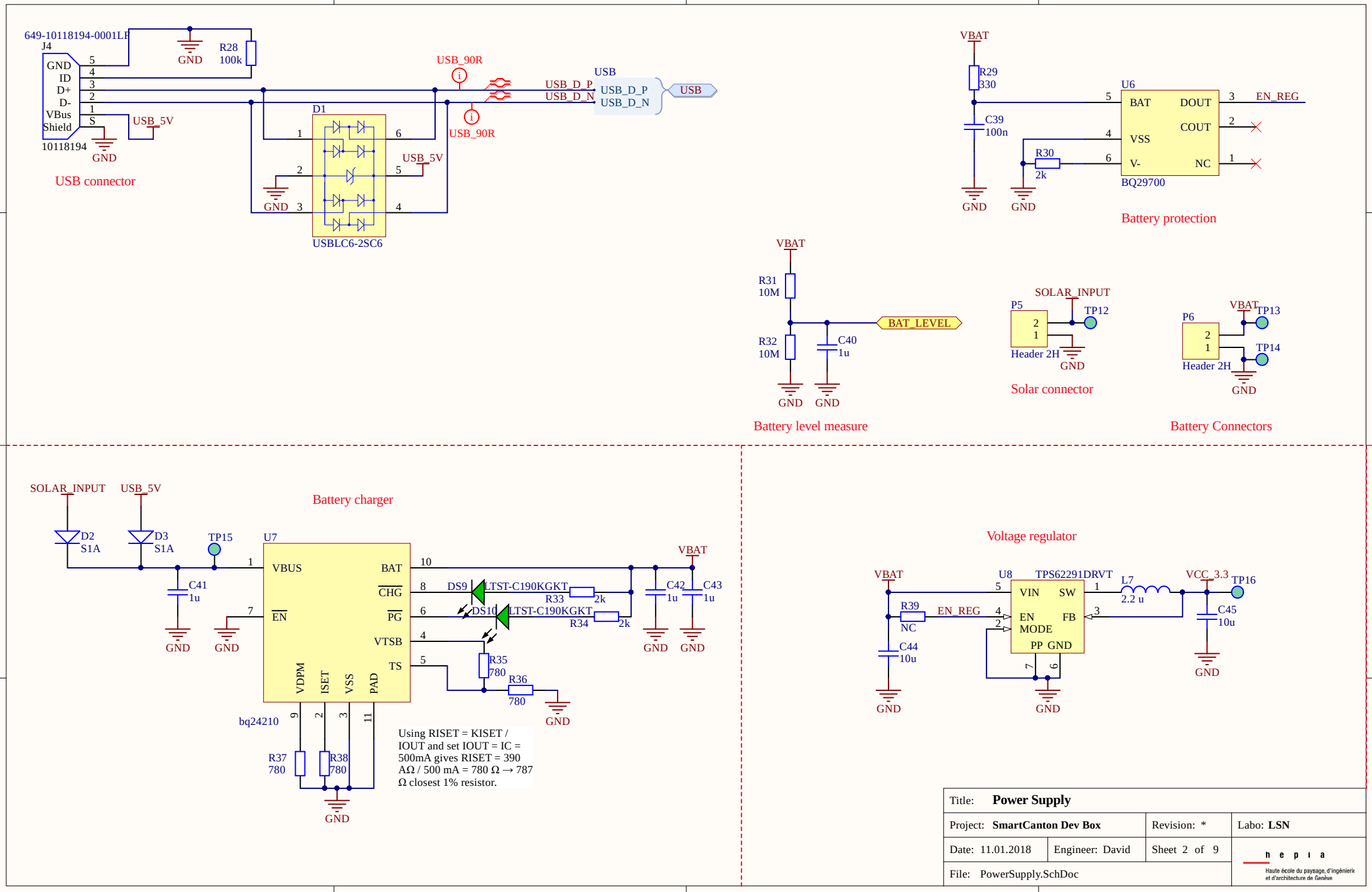
Annexe E

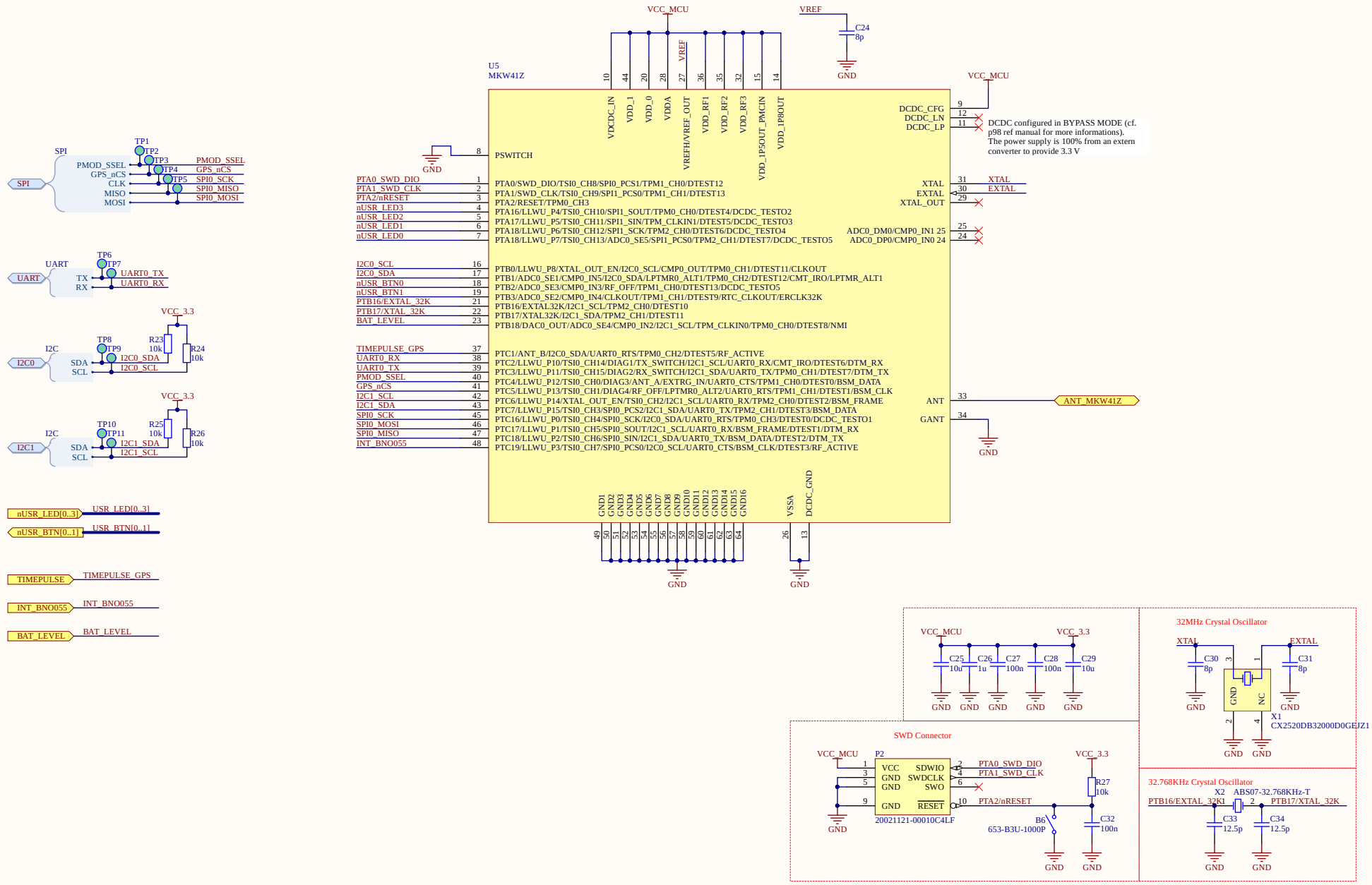
SmartCanton DevBox : Electronic Designs





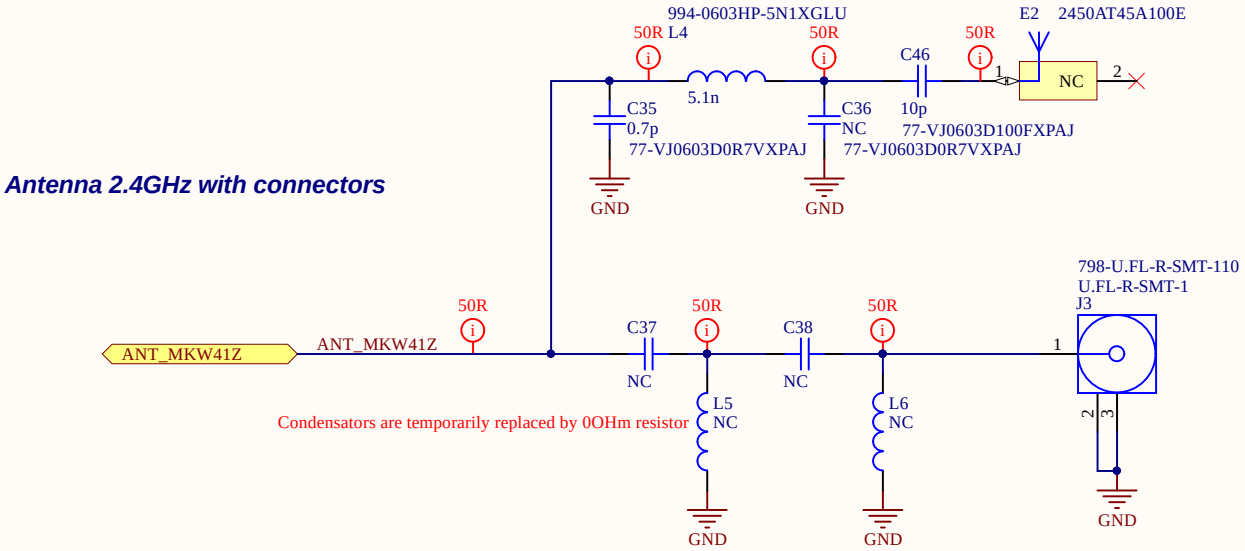
Title: **Top View**			
Project: **SmartCanton Dev Box**		Revision: *	Labo: **LSN**
Date: 11.01.2018	Engineer: David	Sheet 1 of 9	hepia Haute école du paysage, d'ingénierie et d'architecture de Genève
File: SmartCantonDevBox_TopPage.SchDoc			


649-10118194-0001LF
J4
GND
ID
D+
D-
VBus
Shield
10118194
R28
100k
USB_5V
USB_90R
USB_D_P
USB_D_N
USB
USB connector
D1
USBLC6-2SC6
VBAT
R29
330
C39
100n
U6
BAT
DOUT
COUT
VSS
V-
NC
EN_REG
R30
2k
BQ29700
Battery protection
R31
10M
R32
10M
C40
1u
BAT_LEVEL
Battery level measure
SOLAR_INPUT
P5
TP12
Header 2H
Solar connector
P6
TP13
TP14
Battery Connectors
Battery charger
D2
S1A
D3
S1A
TP15
U7
VBUS
BAT
CHG
EN
PG
VTSB
TS
VDPM
ISET
VSS
PAD
bq24210
C41
1u
C42
1u
C43
1u
DS9
DS10
LTST-C190KGKT
R33
2k
R34
2k
R35
780
R36
780
R37
780
R38
780
Using RISET = KISET / IOUT and set IOUT = IC = 500mA gives RISET = 390 AΩ / 500 mA = 780 Ω → 787 Ω closest 1% resistor.
Voltage regulator
U8
TPS62291DRVT
VIN
SW
EN
FB
MODE
PP
R39
NC
C44
10u
L7
2.2 u
VCC_3.3
TP16
C45
10u
Title: Power Supply
Project: SmartCanton Dev Box
Revision: *
Labo: LSN
Date: 11.01.2018
Engineer: David
Sheet 2 of 9
File: PowerSupply.SchDoc
hepia
Haute école du paysage, d'ingénierie et d'architecture de Genève

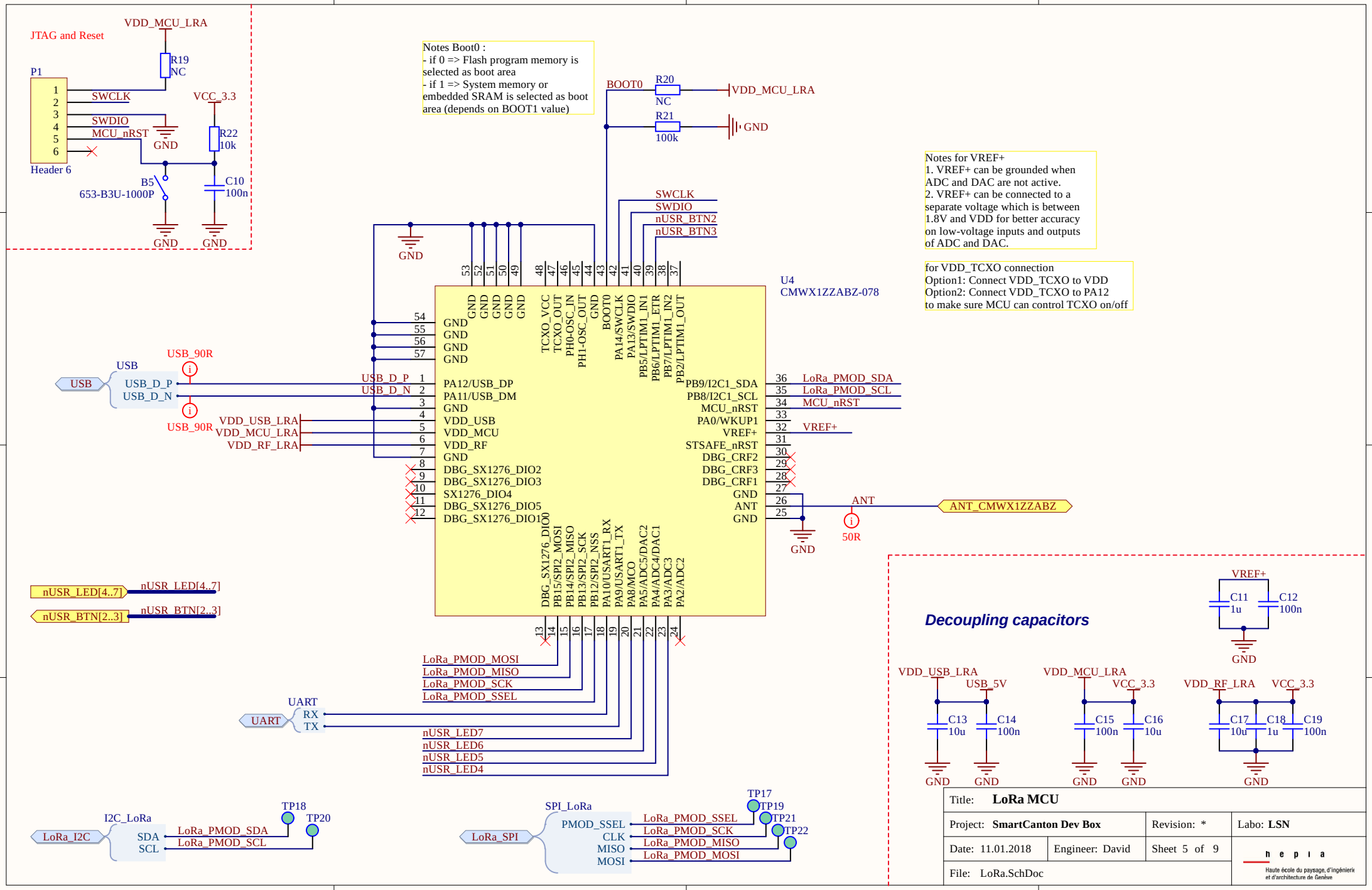


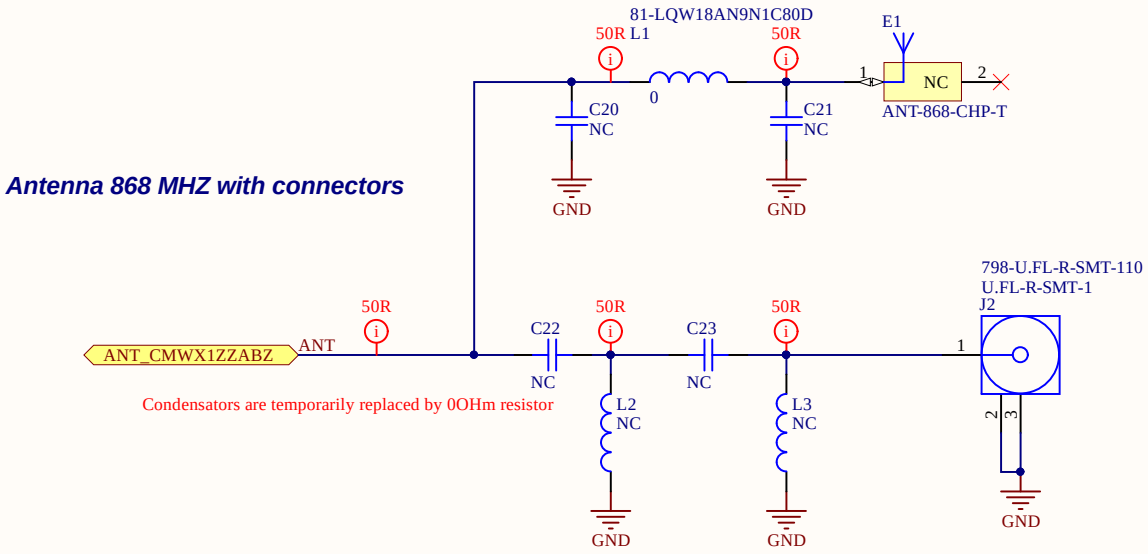
Title: **Bluetooth MCU**			
Project: **SmartCanton Dev Box**		Revision: *	Labo: **LSN**
Date: 11.01.2018	Engineer: David	Sheet 3 of 9	
File: MCU.SchDoc			

h e p i a
Haute école du paysage, d'ingénierie et d'architecture de Genève

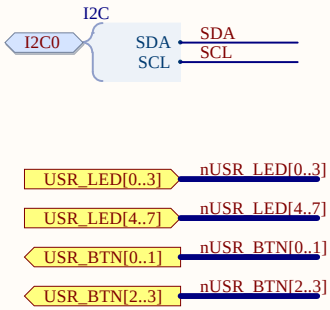


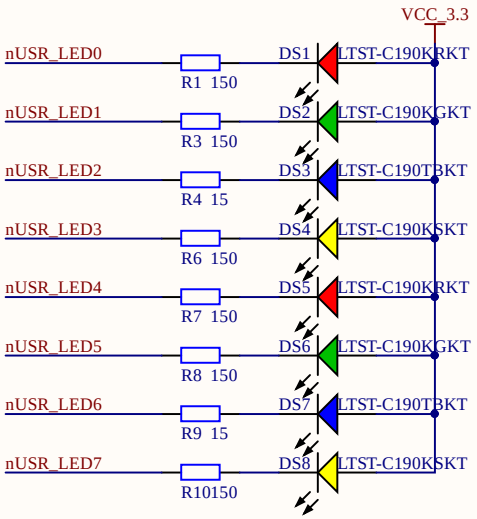
Title: **Bluetooth MCU Antenna**			
Project: **SmartCanton Dev Box**		Revision: *	Labo: **LSN**
Date: 11.01.2018	Engineer: David	Sheet 4 of 9	hepia Haute école du paysage, d'ingénierie et d'architecture de Genève
File: MCUAntenna.SchDoc			


JTAG and Reset
VDD_MCU_LRA
R19
NC
P1
SWCLK
SWDIO
MCU_nRST
VCC_3.3
R22
10k
GND
Header 6
B5
653-B3U-1000P
C10
100n
Notes Boot0 :
- if 0 => Flash program memory is selected as boot area
- if 1 => System memory or embedded SRAM is selected as boot area (depends on BOOT1 value)
BOOT0
R20
NC
R21
100k
Notes for VREF+
1. VREF+ can be grounded when ADC and DAC are not active.
2. VREF+ can be connected to a separate voltage which is between 1.8V and VDD for better accuracy on low-voltage inputs and outputs of ADC and DAC.
for VDD_TCXO connection
Option1: Connect VDD_TCXO to VDD
Option2: Connect VDD_TCXO to PA12
to make sure MCU can control TCXO on/off
U4
CMWX1ZZABZ-078
USB
USB_D_P
USB_D_N
USB_90R
VDD_USB_LRA
VDD_MCU_LRA
VDD_RF_LRA
PA12/USB_DP
PA11/USB_DM
VDD_USB
VDD_MCU
VDD_RF
DBG_SX1276_DIO2
DBG_SX1276_DIO3
SX1276_DIO4
DBG_SX1276_DIO5
DBG_SX1276_DIO1
TCXO_VCC
TCXO_OUT
PH0-OSC_IN
PH1-OSC_OUT
BOOT0
PA14/SWCLK
PA13/SWDIO
PB5/LPTIM1_IN1
PB6/LPTIM1_ETR
PB7/LPTIM1_IN2
PB2/LPTIM1_OUT
PB9/I2C1_SDA
PB8/I2C1_SCL
MCU_nRST
PA0/WKUP1
VREF+
STSAFE_nRST
DBG_CRF2
DBG_CRF3
DBG_CRF1
ANT
LoRa_PMOD_SDA
LoRa_PMOD_SCL
nUSR_BTN2
nUSR_BTN3
ANT_CMWX1ZZABZ
50R
nUSR_LED[4..7]
nUSR_BTN[2..3]
DBG_SX1276_DIO0
PB15/SPI2_MOSI
PB14/SPI2_MISO
PB13/SPI2_SCK
PB12/SPI2_NSS
PA10/USART1_RX
PA9/USART1_TX
PA8/MCO
PA5/ADC5/DAC2
PA4/ADC4/DAC1
PA3/ADC3
PA2/ADC2
LoRa_PMOD_MOSI
LoRa_PMOD_MISO
LoRa_PMOD_SCK
LoRa_PMOD_SSEL
UART
RX
TX
nUSR_LED7
nUSR_LED6
nUSR_LED5
nUSR_LED4
Decoupling capacitors
VREF+
C11
1u
C12
100n
VDD_USB_LRA
USB_5V
C13
10u
C14
100n
VDD_MCU_LRA
VCC_3.3
C15
100n
C16
10u
VDD_RF_LRA
C17
10u
C18
1u
C19
100n
I2C_LoRa
LoRa_I2C
SDA
SCL
TP18
TP20
SPI_LoRa
LoRa_SPI
PMOD_SSEL
CLK
MISO
MOSI
TP17
TP19
TP21
TP22
Title: LoRa MCU
Project: SmartCanton Dev Box
Revision: *
Labo: LSN
Date: 11.01.2018
Engineer: David
Sheet 5 of 9
File: LoRa.SchDoc
hepia
Haute école du paysage, d'ingénierie et d'architecture de Genève

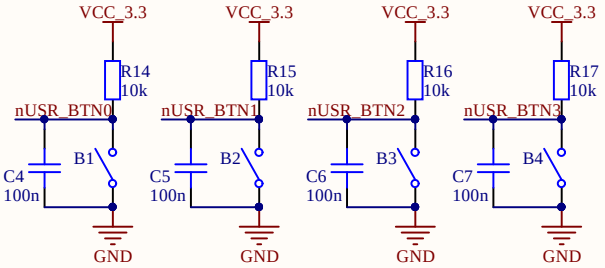


Title: **LoRa Antenna**			
Project: **SmartCanton Dev Box**		Revision: *	Labo: **LSN**
Date: 11.01.2018	Engineer: David	Sheet 6 of 9	hepia Haute école du paysage, d'ingénierie et d'architecture de Genève
File: LoRaAntenna.SchDoc			



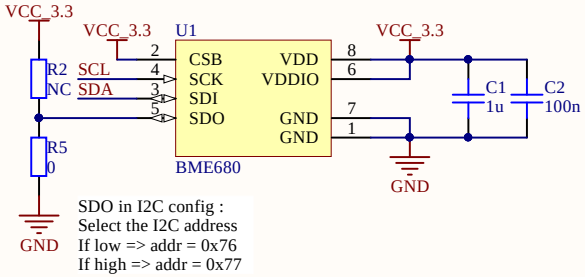


Leds controlled by the user
Half from the LoRa MCU, the other half from the bluetooth MCU



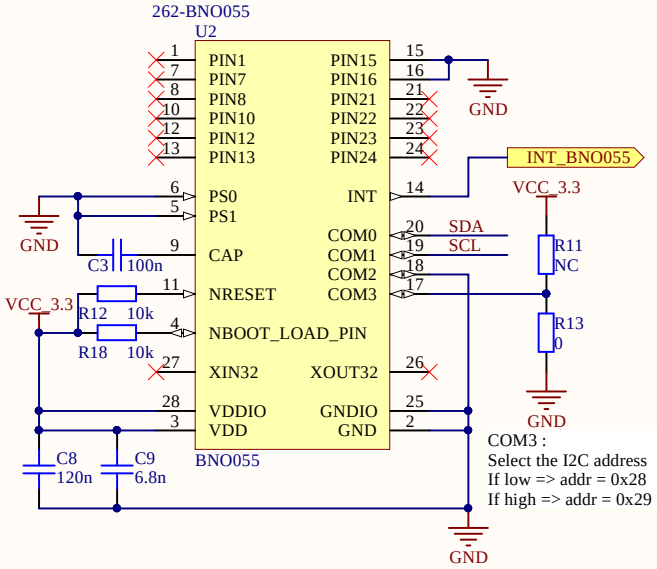
Buttons read by the user
Half from the LoRa MCU, the other half from the bluetooth MCU

Combined humidity, pressure, temp and gas sensor



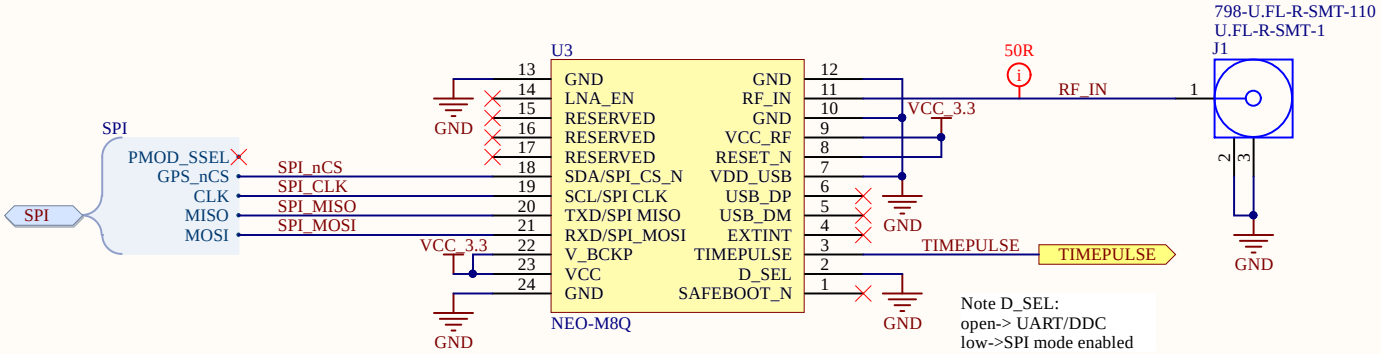
SDO in I2C config :
Select the I2C address
If low => addr = 0x76
If high => addr = 0x77

Inertial platform 9Axis

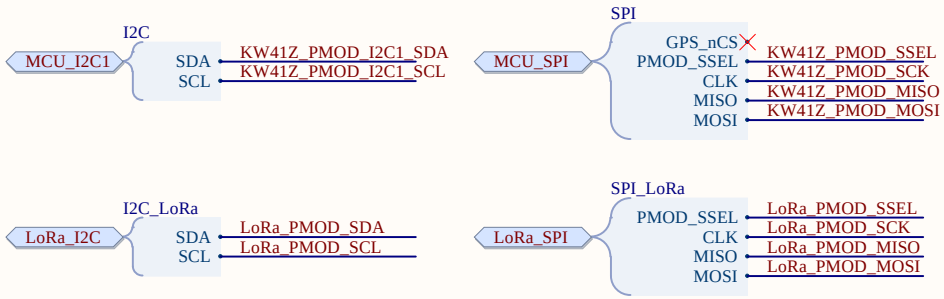


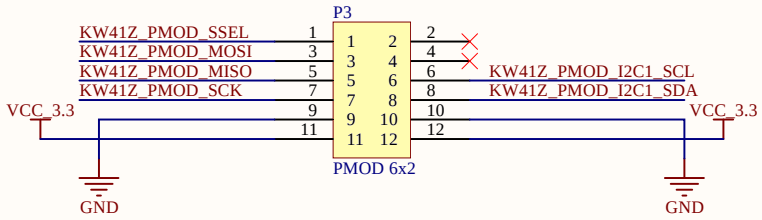
Title: **Embedded Sensors**			
Project: **SmartCanton Dev Box**		Revision: *	Labo: **LSN**
Date: 11.01.2018	Engineer: David	Sheet 7 of 9	
File: EmbeddedPeripherals.SchDoc			

h e p i a
Haute école du paysage, d'ingénierie et d'architecture de Genève

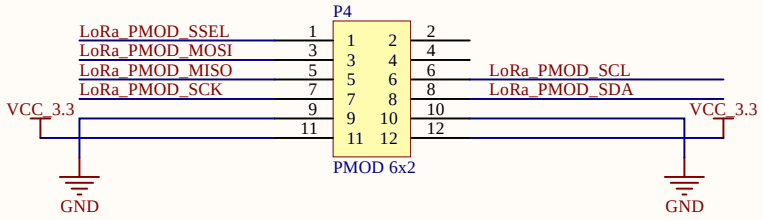


Title: **GPS**			
Project: **SmartCanton Dev Box**		Revision: *	Labo: **LSN**
Date: 11.01.2018	Engineer: David	Sheet 8 of 9	hepia Haute école du paysage, d'ingénierie et d'architecture de Genève
File: GPS.SchDoc			



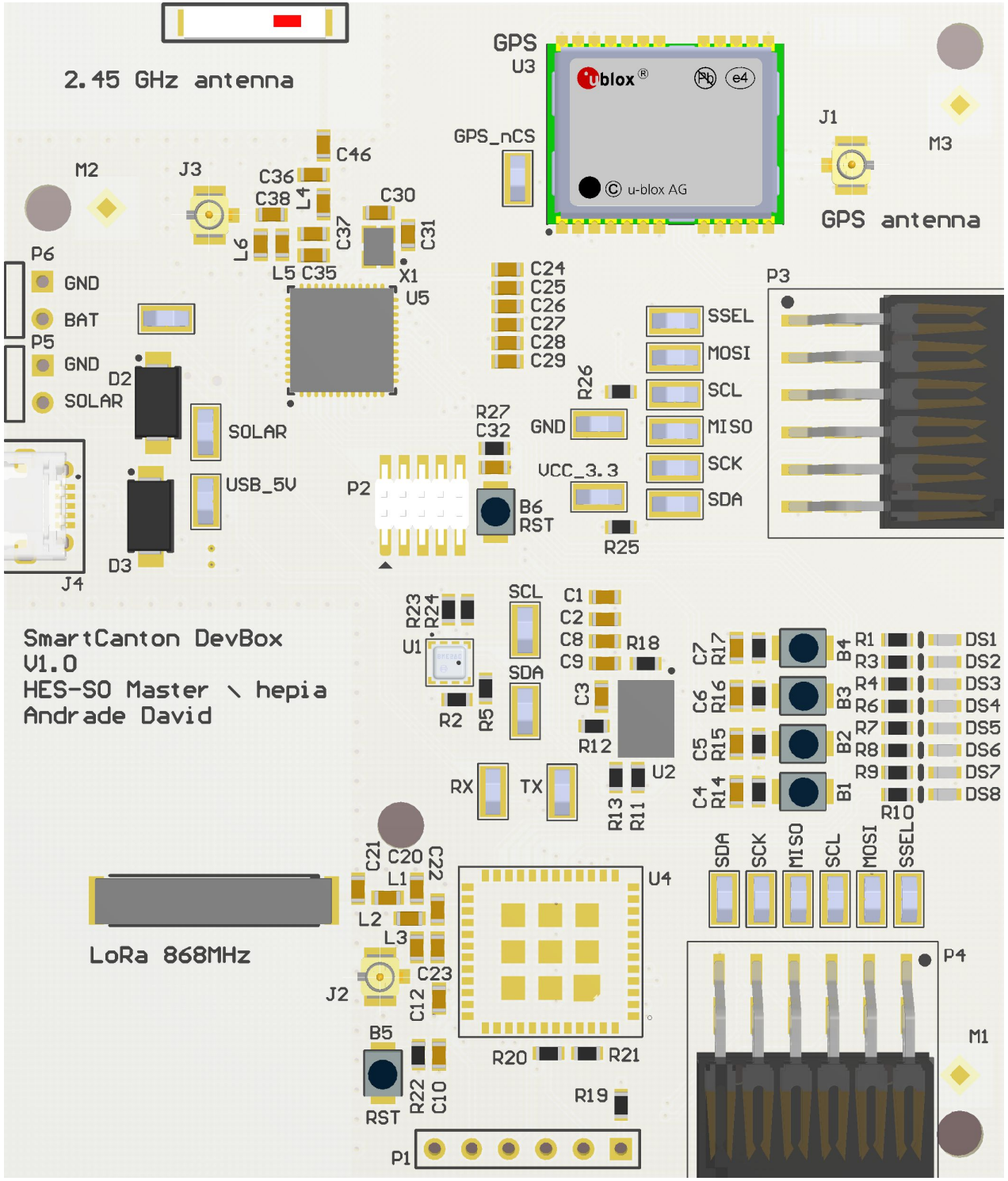


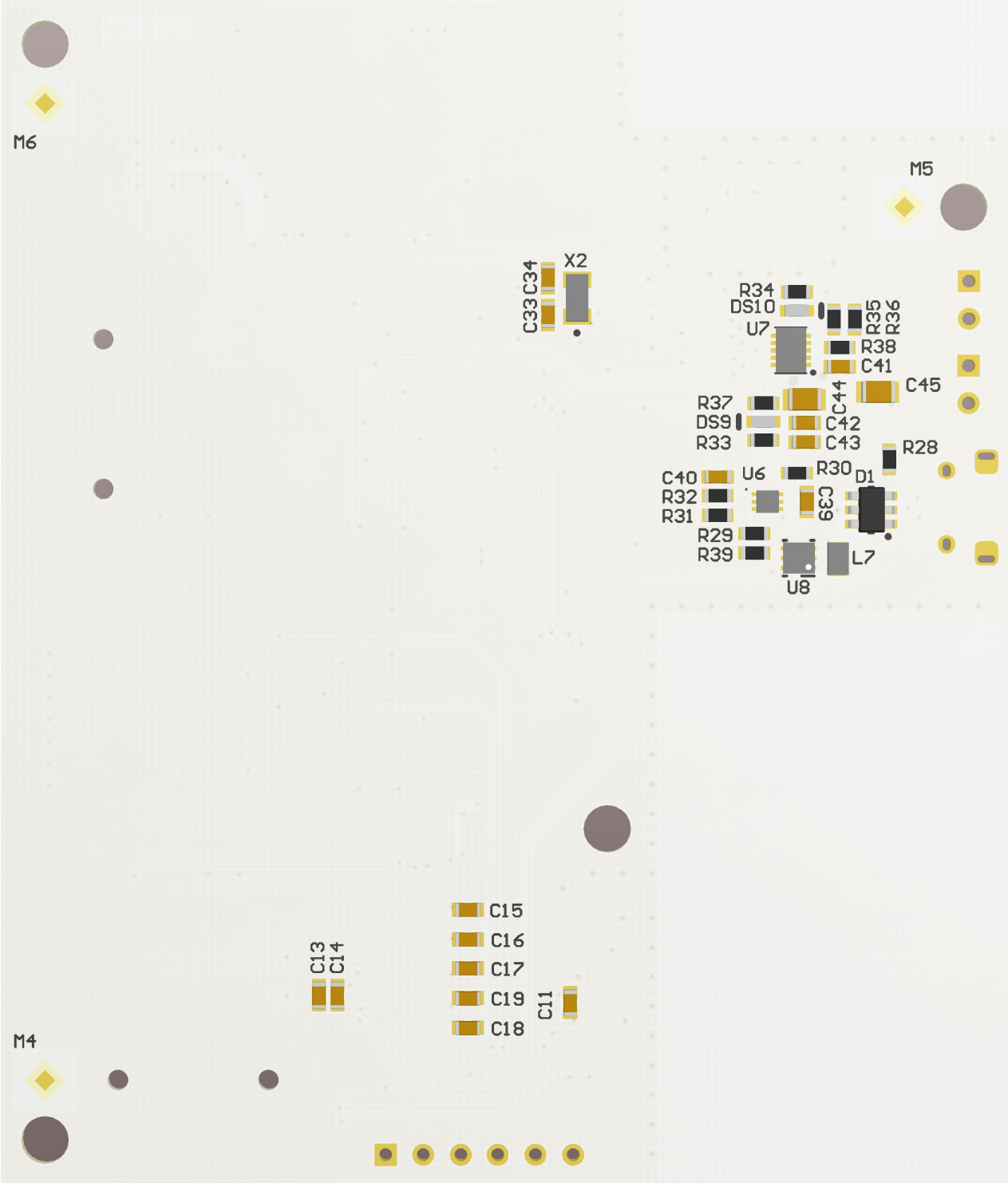
Connector Pmod I2C and SPI connected to the KW41Z



Connector Pmod I2C and SPI connected to the LoRa module

Title: *			
Project: **Project name**		Revision: *	Labo: **LSN**
Date: 11.01.2018	Engineer: *	Sheet 9 of 9	hepia Haute école du paysage, d'ingénierie et d'architecture de Genève
File: PMOD.SchDoc			

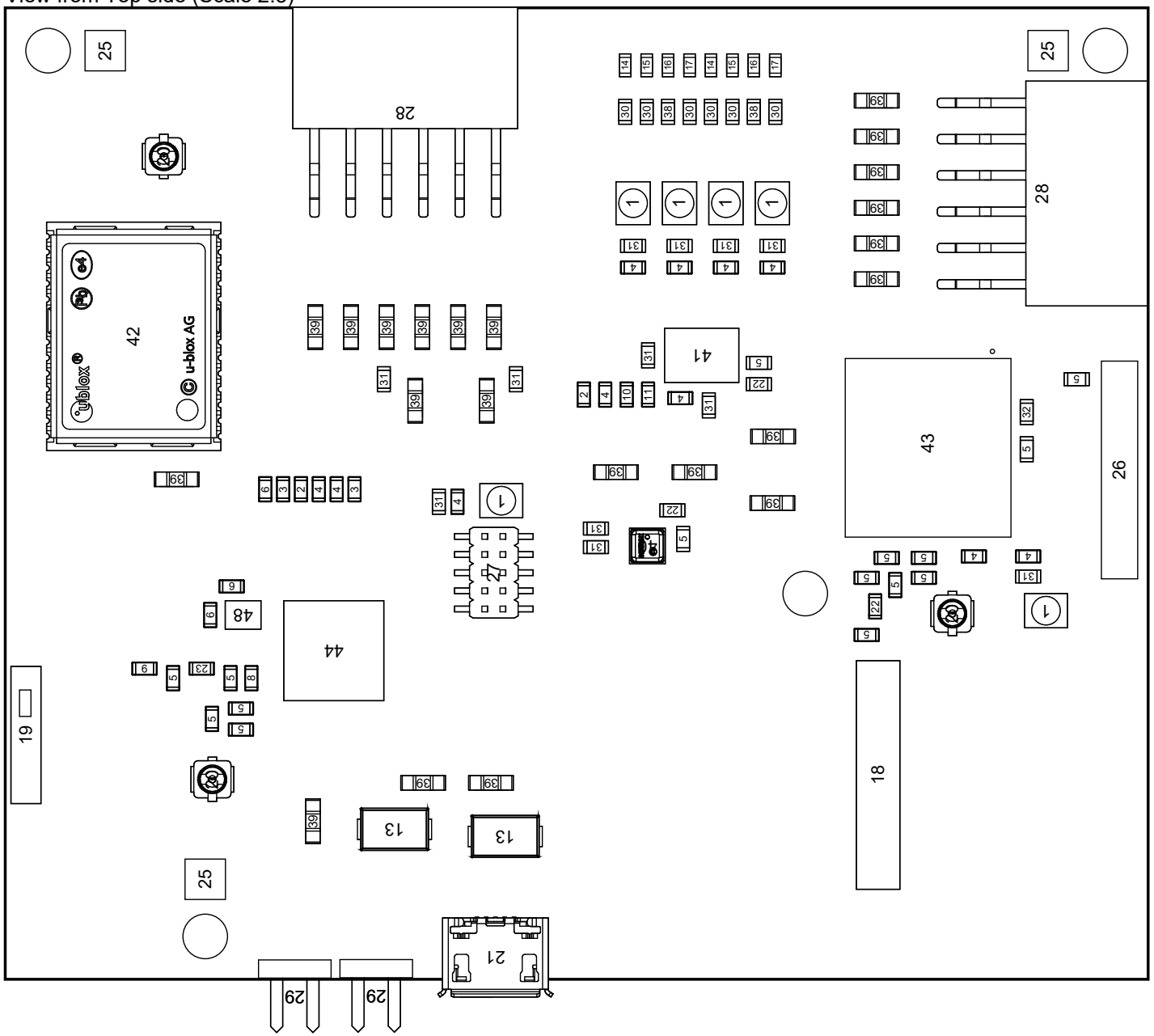
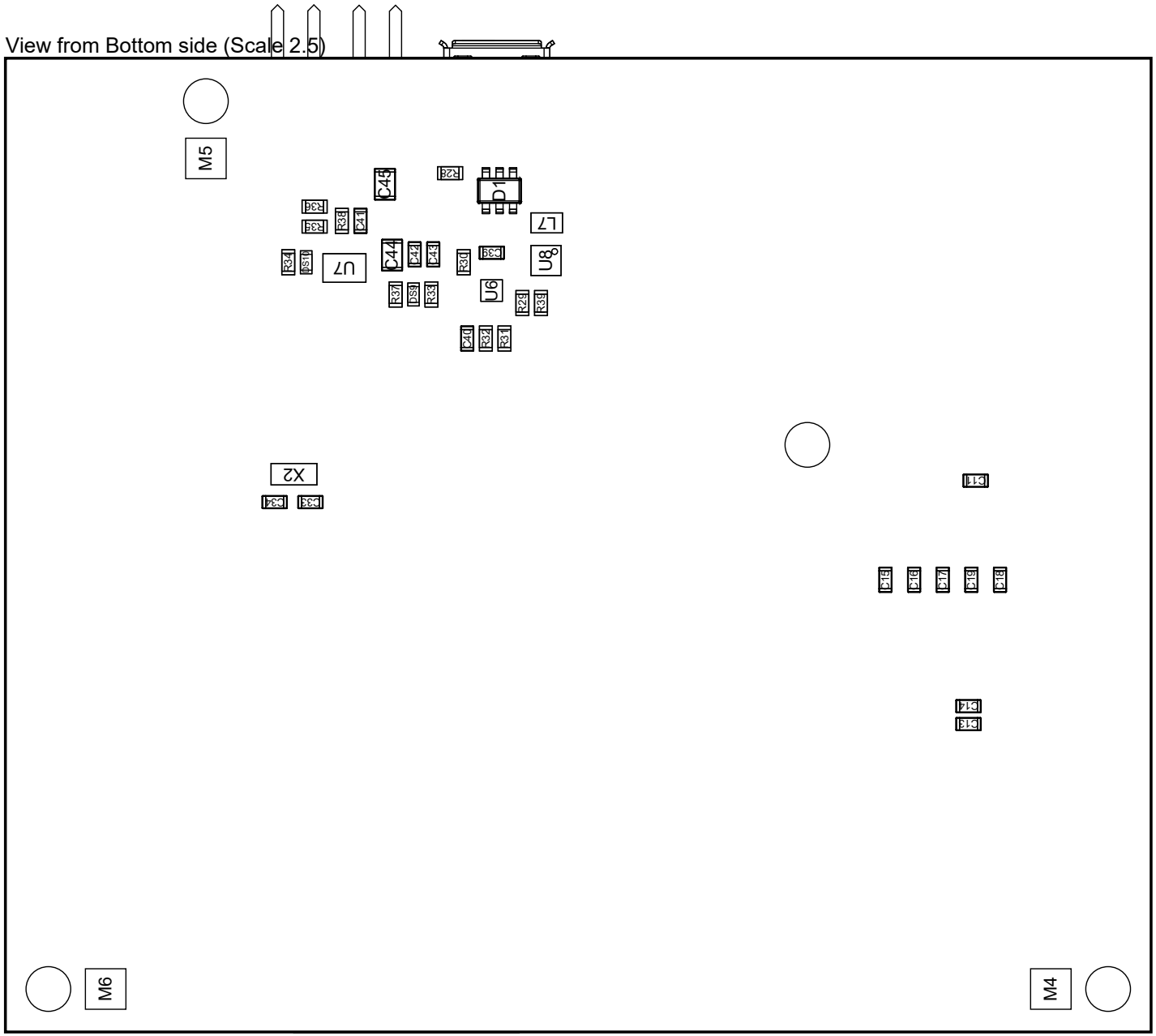

2.45 GHz antenna
GPS
U3
GPS_nCS
J1
M3
GPS antenna
M2
J3
C46
C36
C38
L4
C30
C37
C31
L6
L5
C35
X1
U5
P6
GND
BAT
P5
GND
SOLAR
D2
SOLAR
USB_5V
D3
J4
C24
C25
C26
C27
C28
C29
P3
SSEL
MOSI
SCL
MISO
SCK
SDA
R26
GND
R27
C32
VCC_3.3
P2
B6
RST
R25
SmartCanton DevBox
V1.0
HES-SO Master \ hepia
Andrade David
R23
R24
U1
R2
R5
SCL
SDA
C1
C2
C8
C9
R18
C3
R12
U2
RX
TX
R13
R11
C7
R17
B4
R1
R3
R4
R6
R7
R8
R9
R10
C6
R16
B3
C5
R15
B2
C4
R14
B1
DS1
DS2
DS3
DS4
DS5
DS6
DS7
DS8
SDA
SCK
MISO
SCL
MOSI
SSEL
LoRa 868MHz
C21
C20
C22
L1
L2
L3
U4
J2
C23
C12
B5
RST
R22
C10
R20
R21
R19
P1
P4
M1


M6
M5
X2
C33
C34
R34
DS10
U7
R35
R36
R38
C41
C45
C44
R37
DS9
C42
R33
C43
R28
C40
U6
R30
D1
R32
C39
R31
R29
R39
L7
U8
C15
C16
C13
C14
C17
C19
C11
C18
M4

Bill Of Materials

Item	Designator	Comment	Quantity	Footprint
1	B1-B6	653-B3U-1000P	6	B3U-1000P
2	C1, C11, C18, C26, C40-C43	1u	8	CAPC1608X08
3	C13, C16, C17, C25, C29, C44, C45	10u	7	CAPC1608X08N
4	C2-C7, C10, C12, C14, C15, C19, C27, C28, C32, C39	100n	15	CAPC1608X08
5	C20-C23, C36-C38, L2, L3, L5, L6, R2, R11, R19, R20, R39	NC	16	CAPC1608X08
6	C24, C30, C31	8p	3	CAPC1608X08
7	C33, C34	12.5p	2	CAPC1608X08
8	C35	0.7p	1	CAPC1608X08
9	C46	10p	1	CAPC1608X08
10	C8	120n	1	CAPC1608X08
11	C9	6.8n	1	CAPC1608X08
12	D1	USBLC6-2SC6	1	STM-SOT23-6L_V
13	D2, D3	S1A	2	DO-214AC
14	DS1, DS5	LTST-C190KRKT	2	LTST-C191KRKT_V
15	DS2, DS6, DS9, DS10	LTST-C190KGKT	4	LITE-ON-LTST-C190GKT-2_V
16	DS3, DS7	LTST-C190TBKT	2	LTST-C190TBKT-2_V
17	DS4, DS8	LTST-C190KSKT	2	LTST-C190KSKT_V
18	E1	ANT-868-CHP-T	1	ANT-868-916-CHP
19	E2	2450AT45A100E	1	Antenne 2.4ghz Ceramique
20	J1-J3	U.FL-R-SMT-1	3	U.FL-R-SMT-1
21	J4	10118194	1	10118194-0001LF
22	L1, R5, R13	0	3	INDC1608X08
23	L4	5.1n	1	INDC1608X08
24	L7	2.2 u	1	TDK-IND0805-125_V
25	M1-M6	1mm	6	Losange-1mm
26	P1	Header 6	1	HDR1X6
27	P2	20021121-00010C4LF	1	PCBComponent
28	P3, P4	PMOD 6x2	2	PMOD SMD Header 6x2
29	P5, P6	Header 2H	2	HDR1X2H
30	R1, R3, R6-R8, R10	150	6	RESC1608X06
31	R12, R14-R18, R22-R27	10k	12	RESC1608X06
32	R21	100k	1	RESC1608X06
33	R28	100K	1	RESC1608X06
34	R29	330	1	RESC1608X06
35	R31, R32	10M	2	RESC1608X06
36	R30, R33, R34	2k	3	RESC1608X06
37	R35-R38	780	4	RESC1608X06
38	R4, R9	15	2	RESC1608X06
39	TP1-TP22	5015	22	5015
40	U1	BME680	1	LGA-8-30x30x09
41	U2	BNO055	1	BOSC-LGA-28_N
42	U3	NEO-M8Q	1	NEO
43	U4	CMWX1ZZABZ-078	1	CMWX1ZZABZ-078
44	U5	MKW41Z	1	QFN50P700X700X80_HS-49M
45	U6	BQ29700	1	SON50P150X75-6M
46	U7	bq24210	1	WSON50P200X300X80-11N
47	U8	TPS62291DRVT	1	DRV0006A_V
48	X1	CX2520DB32000D0GEJZ1	1	AVX-CX2520DB32000D0GEJZ1-4_V
49	X2	ABS07-32.768KHz-T	1	ABRA-ABS07-2_V

View from Top side (Scale 2.5)



View from Bottom side (Scale 2.5)