Hes·so VALAIS WALLIS
Haute Ecole d'Ingénierie
Hochschule für Ingenieurwissenschaften

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

# Filière Systèmes industriels

## Orientation Infotronics

# Travail de bachelor
# Diplôme 2017

*Pierre Mendicino*

## *Low-power networks relays*

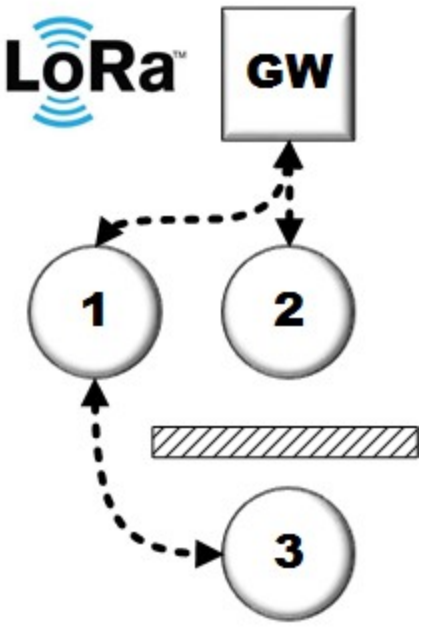
*Professeur*
Pierre-André Mudry

*Expert*
César Papilloud

*Date de la remise du rapport*
18.08.2017

# Low-power networks relays

Diplômant/e Pierre Mendicino




Travail de diplôme
| édition 2017 |

Filière
*SYND*

Domaine d'application
*Infotronics*

Professeur responsable
*Pierre-André Mudry*
*pierre-andre.mudry@hevs.ch*

## Objectif du projet

Analyser et comparer les principales technologies de réseaux longue portée et basse consommation d'énergie existantes. Développer un protocole basé sur la technologie de modulation LoRa permettant le forwarding de paquets.

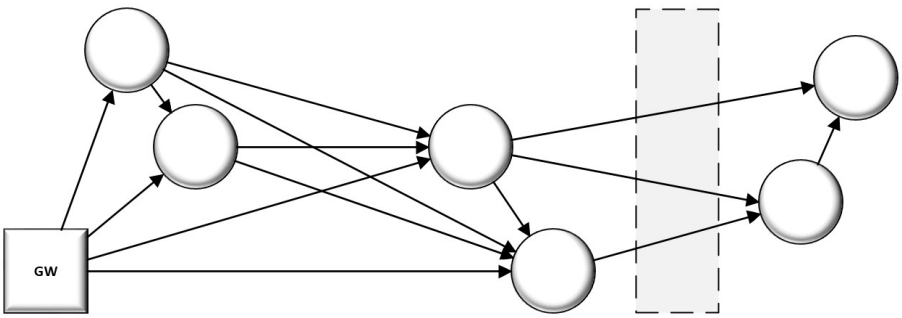
## Méthodes | Expériences | Résultats

L'augmentation massive du nombre d'objets connectés dans le monde induit une forte augmentation des communication machine vers machine. Différents acteurs commerciaux se partagent le marché en ce qui concerne l'établissement des réseaux capables de gérer ce flux d'informations.

Ce travail propose, dans un premier temps, de comparer les différentes approches actuellement utilisées dans le domaine de la longue portée.

La technologie LoRa, en passe de devenir un standard dans ce domaine, a été plus profondément étudiée et des tests de portée dans différents milieux de transmission ont été réalisés. Un protocole de communication basé sur cette technologie a ensuite été proposé et implémenté. Ce dernier est responsable de l'établissement d'un réseau avec une topologie particulière, puisque les nœuds qui le composent ont la possibilité de réexpédier des paquets. Il pourrait être utilisé dans des situations où la portée des antennes LoRa serait trop faible ou l'augmentation du nombre d'antennes de couverture, trop onéreuse.

A la fin de ce travail, une application démontrant que le protocole est fonctionnel a été mise sur pied.



# 1 Table des matières

# 2 Acronymes

Les acronymes listés dans le Tableau 1 peuvent être rencontrés plus loin dans ce travail. Bien que suffisante pour la compréhension, cette liste n'est pas exhaustive.

| Acronyme | Anglais | Français |
|---|---|---|
| **ADC** | Analog-to-digital converter | Convertisseur analogique vers numérique |
| **BW** | Bandwidth | Largeur de bande |
| **CSS** | Chirp spread spectrum | Etalement de spectre |
| **EXTI** | External interrupt/event controller | Contrôleur d'interruptions/événements externes |
| **FIFO** | First-in-first-out | Premier dedans, premier dehors |
| **GSM** | Global system for mobile communications | Système mondial pour les communications mobiles |
| **GPIO** | General-purpose I/Os | Entrées/sorties à usage général |
| **GPS** | Global positioning system | Système de positionnement mondial |
| **GW** | Gateway | Passerelle |
| **HCI** | Host controller interface | Interface de commande hôte |
| **IOT** | Internet of things | Internet des objets |
| **LBT** | Listen before transmit | Ecoute avant de transmettre |
| **LPWAN** | Low-power wide area network | Réseau étendu à faible puissance |
| **LTE** | Long term evolution | Evolution à long terme |
| **LTE-M** | Long term evolution machine-type communications | Evolution à long terme des communications entre machines |
| **MAC** | Media access control | Contrôle d'accès au support |
| **M2M** | Machine-to-machine | Machine à machine |
| **NB-IOT** | Narrow band internet of things | Internet des objets à bande étroite |
| **PER** | Packet error rate | Taux d'erreurs des paquets |
| **PHY** | Physical layer | Couche physique |
| **RF** | Radio frequency | Radio fréquence |
| **RSSI** | Received signal strength indication | Indication de la puissance du signal reçu |
| **SAP** | Service access point | Point d'accès au service |
| **SF** | Spreading factor | Facteur d'étalement |
| **SLIP** | Serial line internet protocol | Protocole internet pour les lignes séries |
| **SNR** | Signal to noise ratio | Rapport signal sur bruit |
| **TIM** | Timer | Minuteur |
| **UART** | Universal synchronous asynchronous receiver transmitter | Transmetteur/récepteur synchrone/asynchrone universel |
| **3GPP** | 3rd generation partnership project | Project partenarial de 3ème génération |

**Tableau 1 : Liste des acronymes utilisés**

# 3 Introduction

Le domaine des communications M2M subit actuellement un nouvel essor en raison de l'émergence de l'IOT. Les objets connectés de demain auront besoin de toujours plus d'autonomie et seront de plus en plus nombreux. Pour gérer ce nouveau flux d'informations, des nouveaux réseaux de communication sans fil doivent être déployés.

Ce travail de diplôme traite spécifiquement des technologies de communications sans fil et bas-débits, dans le domaine des communications M2M longue portée et à basse consommation d'énergie (LPWAN). La télémétrie est un bon exemple d'applications pour ce type de réseau : un capteur sur batterie mesurant des valeurs simples n'a besoin que de transmettre quelques Bytes à la fois, mais exige des contraintes fortes au niveau de sa consommation d'énergie.


**Figure 1 : Domaines d'application des LPWAN**[1]

Ce type de réseau utilise généralement une topologie en étoile à l'instar des réseaux TCP/IP. C'est-à-dire qu'il est composé d'un certain nombre d'appareils, tous jumelés avec une même passerelle centrale. Dans les LPWAN, cette dernière fait ensuite le lien avec un réseau externe, dénué des contraintes de consommation. Différents acteurs commerciaux donnent leur envol à ces réseaux basse consommation :

- Les **opérateurs téléphoniques** traditionnels veulent devenir des acteurs incontournables pour l'IOT. Ils disposent déjà de plages de fréquences propriétaires et d'antennes relais installées un peu partout pour leurs réseaux téléphoniques. Dans le domaine des communications M2M, ils utilisent actuellement des solutions comme la 2G et projettent maintenant de diversifier leur offre avec des solutions moins énergivores. Leur business-model repose essentiellement sur la vente d'abonnements et de cartes SIM.
- Des **sociétés tierces** lancent leurs propres standards, concurrentiels aux solutions des opérateurs, basés sur d'autres technologies et vendent des packages incluant un abonnement à leurs services cloud. Elles ne possèdent pas de bandes de fréquences propriétaires et doivent donc passer par les plages de fréquences dites « libres » pour leurs transmissions. On peut notamment citer Sigfox, Ingenu mais également certains opérateurs téléphoniques, comme Orange et Bouygue en France ou Swisscom en Suisse, qui proposent des solutions basées sur la technologie LoRa.

[1] Crédit : Semtech, *http://www.semtech.com/wireless-rf/internet-of-things/img/home-LoRa-IoT.png*

# 4 Objectifs

Les objectifs principaux de ce travail sont l'analyse des différentes technologies LPWAN existantes et le développement d'un protocole de communication, basé sur une d'entre elles, permettant une extension de la portée des antennes. Pour ce faire, il faut considérer les différents objectifs présentés dans ce chapitre.

## 4.1 Analyse des technologies existantes

Différentes technologies existent et sont déjà sur le marché en ce qui concerne les réseaux LPWAN. Les analyser semble être une phase essentielle pour comprendre les principes qui régissent leur fonctionnement. Un comparatif sera également fait du point de vue de la faisabilité, de la vitesse d'exécution et du prix de mise en œuvre pour ce projet spécifique.

L'objectif de cette phase est de choisir une technologie et des appareils compatibles, pour lancer des tests et pouvoir planifier la suite du travail.

## 4.2 Développement du protocole

Le protocole à implémenter doit permettre le forwarding (réexpédition) de paquets dans une technologie LPWAN, modifiant ainsi la topologie habituelle de ces réseaux. En effet, la topologie en étoile est habituellement privilégiée pour réduire au maximum le nombre de communications.

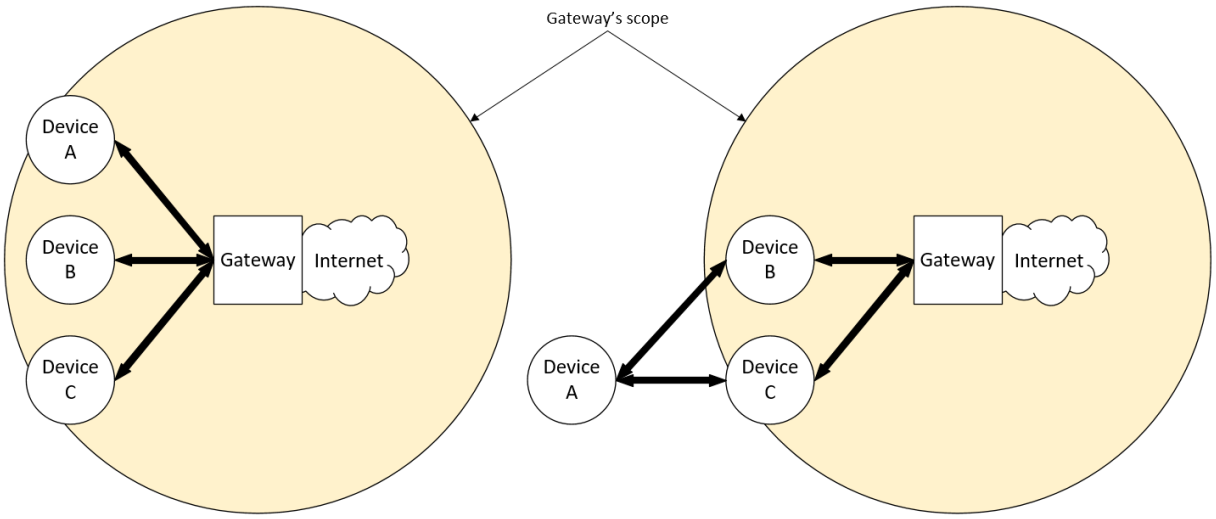



**Figure 2 : Principe de base de l'extension de la portée du réseau LPWAN**

On peut voir sur la Figure 2 : à gauche, le principe de la topologie en étoile et à droite, le principe du protocole à développer, permettant une augmentation de la portée de la passerelle. Chaque nœud du réseau devient une porte d'accès possible pour atteindre cette dernière.

## 4.3 Mise en œuvre avec une application test

La deuxième partie du travail consiste à mettre en œuvre une application concrète qui utilise le protocole développé. Cette phase a pour but de prouver le bon fonctionnement du protocole et permettra d'en tester différents aspects.

## 4.4 Documentation

Une fois le protocole implémenté et testé, la documentation le concernant doit être mise sur pied afin de permettre une intégration facile de celui-ci dans de futures applications. Un utilisateur ne doit pas avoir besoin de connaître l'implémentation détaillée du protocole pour l'utiliser. Il lui suffit de prendre connaissance de la documentation et de la mettre en application.

# 5 Comparaison des technologies LPWAN

## 5.1 Les différentes approches

### 5.1.1 LTE-M / NB-IOT

Les technologie LTE-M et NB-IOT sont les réponses des acteurs traditionnels du GSM face aux nouveaux venus dans le domaine de l'IOT. Leur objectif est de créer des réseaux assurant une couverture comparable à celle de la 4G tout en s'adaptant aux contraintes des objets connectés (basse consommation et faible quantité de données). L'avantage qu'ils ont par rapport aux autres acteurs du secteur est qu'ils disposent déjà des antennes et des plages de fréquences nécessaires à la mise en œuvre de leurs systèmes. Ces technologies sont encore en cours de normalisation et seront appliquées dans les plages de fréquences propriétaires actuellement occupées par leurs bandes GSM ou LTE, comme illustré sur la Figure 3. Sur celle-ci, on a représenté la fréquence sur l'axe horizontal et on observe les possibilités de déploiement de la technologie NB-IOT à l'intérieur des plages de fréquences propriétaires actuellement utilisées pour les réseaux LTE ou GSM.

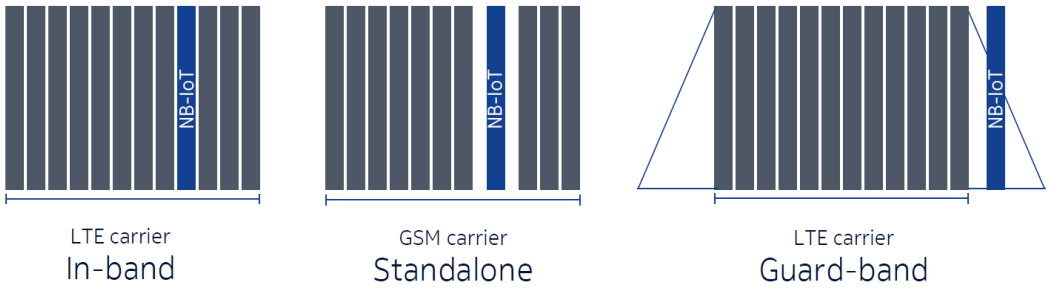



**Figure 3 : Possibilité de déploiement NB-IOT[2]**

En ce qui concerne la technologie NB-IOT, l'entente 3GPP, qui développe les standards pour la télécommunication, a déjà normalisé la technologie en 2016[3]. Cependant elle reste toujours en cours de développement et elle devrait être finalisé en 2018[4].

Pour ce qui est du standard LTE-M, la déclaration de soutien de certains de ces opérateurs au déploiement des réseaux LTE-M liste les acteurs impliqués dans son développement [1]. On peut notamment y voir que cette technologie est actuellement en phase de test à grande échelle aux Etats-Unis et en phase d'expérimentation à plus petite échelle en Europe.

### 5.1.2 LoRa

LoRa est une technique de modulation propriétaire, dérivée des modulations CSS (Semtech Corporation, 2015) et utilisant des largeurs de bandes fixes. Elle appartient à l'entreprise Semtech, la seule à fabriquer les composants RF compatibles. La portée théorique d'un appareil basé LoRa serait de plus de 15km en milieu rural et entre 2 et 5km en milieu urbain, selon eux[5]. Le fait que seule une entreprise fabrique ces composants peut sembler être un point négatif, mais une multitude de fabricants utilisent ces composants et les incluent dans leurs propres

[2] Crédit : Nokia, tiré de « LTE evolution for IoT connectivity »
[3] Source : *http://www.3gpp.org*, communiqué « Standardization of NB-IOT completed », 22 juin 2016
[4] Source : *http://www.informatiquenews.fr*, article « LTE M, un pavé dans la mare des jardins de Sigfox et LoRA », par Alain Baritault, 05 mars 2017
[5] Source: *http://www.semtech.com*, communiqué « Semtech Adds New Devices To Its LoRa™ Ultra Long Range Transceiver Platform », 24 octobre 2013

modules à bas prix (une vingtaine de francs[6]). Grâce à eux, on dispose d'un large panel de modules avec lesquels on peut très rapidement concevoir des systèmes simples sans se soucier du fonctionnement détaillé de la modulation. Quelques exemples sont illustrés ci-dessous, à la Figure 4.

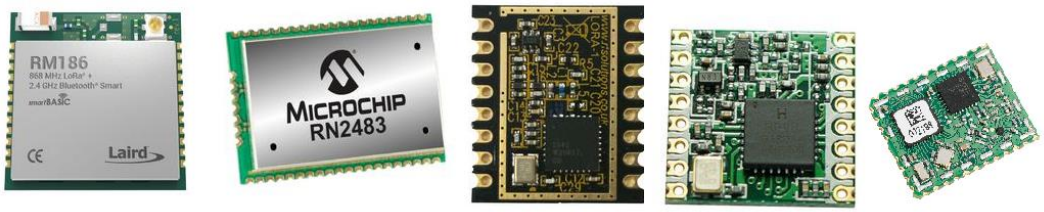



**Figure 4 : Modules LoRa, en vrac**

Chaque fabricant propose une compatibilité – ou non – avec une couche MAC comme celles développées plus loin. Le principe de fonctionnement du système est dans presque tous les cas le même : ces modules possèdent une interface sérielle (UART, SPI, ...) et sont prévus pour être pilotés par un hôte externe (typiquement un microcontrôleur).

### 5.1.2.1 *LoRaWAN*

Basé sur la couche physique LoRa, LoRaWan est la couche MAC développée par l'alliance LoRa. Cette dernière compte de nombreuses entreprises du domaine des télécommunications en son sein, notamment Bouygues, Orange, Swisscom ou encore Semtech. On peut donc dire que cette couche est en bonne passe pour devenir un standard d'interopérabilité au niveau de la modulation LoRa.

Elle permet d'avoir un réseau maillé d'antennes LoRa disséminées dans une large zone. En flux montant, les messages sont transmis à toutes les antennes à proximité. Ensuite, les données sont réexpédiées à des serveurs web. La complexité du réseau est gérée à l'intérieur de ces serveurs, simplifiant la complexité des relais et des nœuds. En flux descendant, seule l'antenne la plus proche du nœud lui transmet les données.

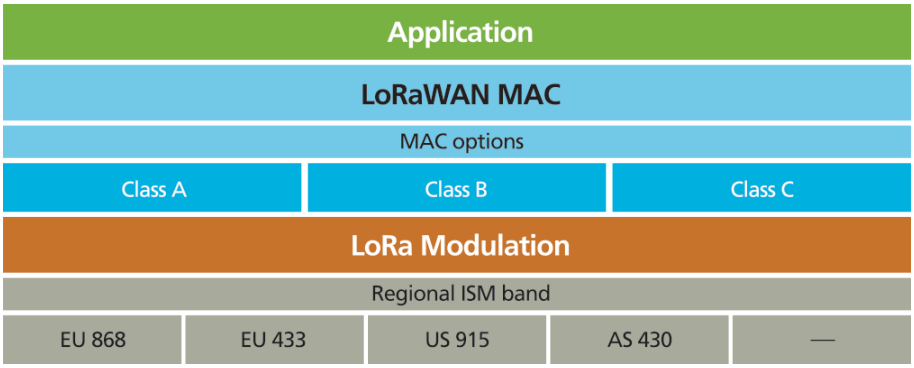



**Figure 5 : Couches et classes de LoRaWAN[7]**

La spécification définit trois classes d'appareils distinctes :

- **Classe A** : Classe minimale, implémentée par tous les appareils LoRaWAN. Chaque transmission en flux montant est suivie de deux courtes ouvertures en flux descendant. Tout le reste du temps, l'appareil peut dormir. C'est la classe la moins énergivore définie. Elle convient bien aux appareils sur batterie ou sur piles.

[6] Catalogue Farnell : *http://ch.farnell.com/fr-CH/Search?storeId=10178&catalogId=15001&st=module+lora*
[7] Crédit : Semtech, *http://semtech.com*

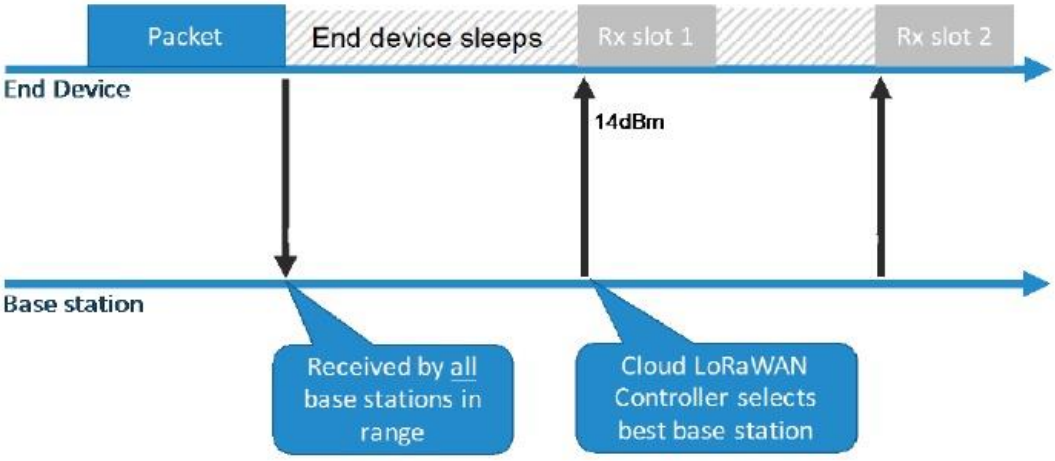



**Figure 6 : Configuration d'une classe A[8]**

- **Classe B** : Les appareils planifient des créneaux supplémentaires pour les flux descendants. En effet, les passerelles émettent périodiquement des beacons et les appareils ouvrent alors une fenêtre de réception synchronisée en plus à ce moment-là. Ce qui a pour conséquence une augmentation de la consommation étant donné que la liaison radio est plus sollicitée.

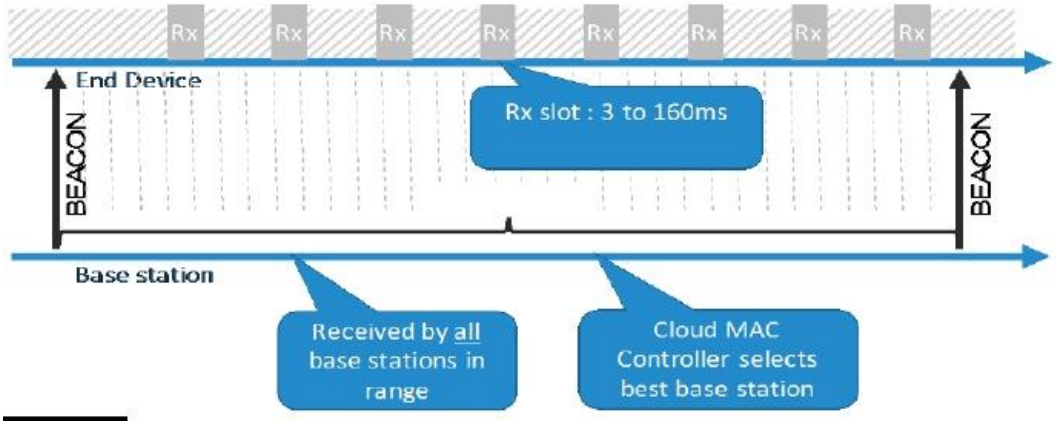



**Figure 7 : Configuration d'une classe B[9]**

- **Classe C** : Les appareils connectés sont tout le temps en écoute, sauf quand ils émettent. Ceci implique une augmentation significative de la consommation de batterie et donc cette classe n'est pas du tout adaptée à des appareils sur batterie.

[8] Crédit : http://fr.scribd.com « Etude Lpwan », Hajji
[9] Crédit : http://fr.scribd.com « Etude Lpwan », Hajji

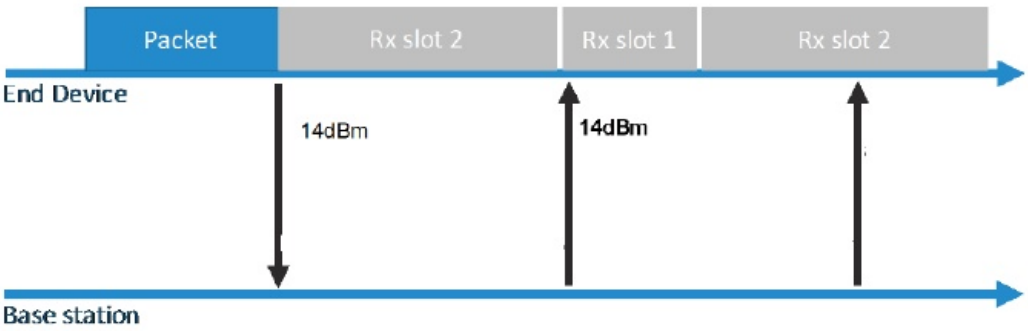



**Figure 8 : Configuration d'une classe C[10]**

La classe dans laquelle se trouve actuellement un appareil connecté au réseau dépend de la couche applicative. Cependant, pour se connecter à une passerelle, les appareils sont obligés d'être en classe A.

#### 5.1.2.2 *Symphony Link*

Egalement basé sur la couche physique LoRa, Symphony Link est la couche MAC développée par la société LinkLabs. Cette technologie offre des éléments tels que la fiabilité (tous les messages sont automatiquement confirmés au niveau de la couche MAC) ou la possibilité d'avoir des répéteurs. Cependant un Gateway coûte aux alentours de CHF 1'000. –[11]. La technologie est encore en phase de test en Europe et n'est donc pas utilisable en l'état.

Elle offre principalement les fonctionnalités suivantes :

- Utilisation de répéteurs (Figure 9)
- Débit de données adaptatif (plus la connexion est bonne, plus le débit sera élevé)
- 100% des messages confirmés
- Possibilité de mise à jour des appareils par les airs
- Possibilité d'envoyer des messages multicast

[10] Crédit : *http://fr.scribd.com* « Etude Lpwan », Hajji
[11] Source : *http://www.arrow.com/en/products/ll-bst-8-868-lrw-w-o-eu/link-labs* , consulté en aout 2017

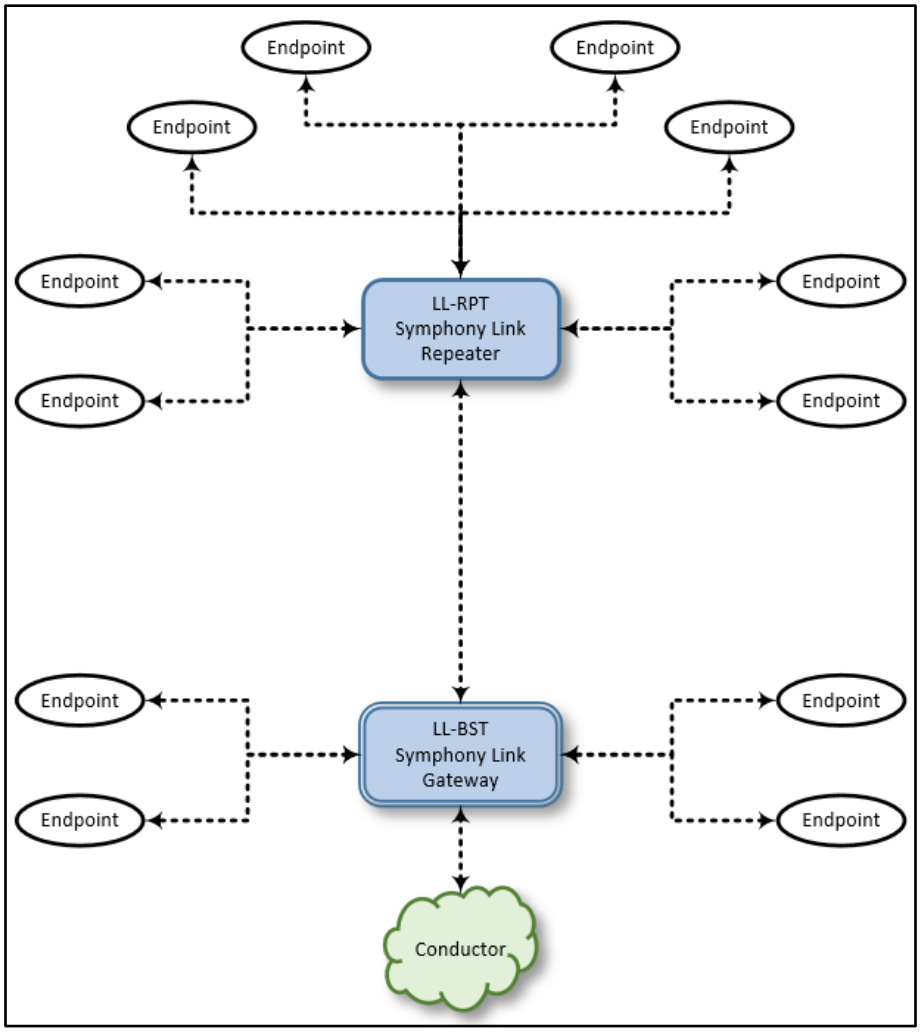



**Figure 9 : Intégration d'un répéteur dans un système Symphony Link[12]**

### 5.1.3 Sigfox

Sigfox est le nom d'un opérateur téléphonique international et également le nom de la modulation utilisée par celui-ci. C'est une compagnie française qui développe son réseau en Europe mais qui n'est pas encore bien installée aux Etats-Unis. Le système utilise la bande non-propriétaire 868Mhz en Europe. Il repose sur une modulation BPSK UNB (Binary Phase-Shift Keying Ultra Narrow-Band). La société annonce une portée de 30km pour son système en milieu rural, avec un débit < 300bits/s, avec des messages de 12 Bytes maximum. Le protocole Sigfox est bidirectionnel sous condition : un appareil Sigfox peut recevoir 4 messages par jour à des instants définis et envoyer une dizaine de messages. La technologie est efficace pour le flux montant mais très peu pour le flux descendant.

Il faut payer une souscription comme avec n'importe quel opérateur mais la couverture est encore très limitée en Suisse (Figure 10). On peut noter que Sigfox est compatible avec les puces SX1272 et SX1276 développées par Semtech et permettant de communiquer dans les bandes de fréquences 868MHz.

[12] Source : Link-labs, *http://link-labs.com*

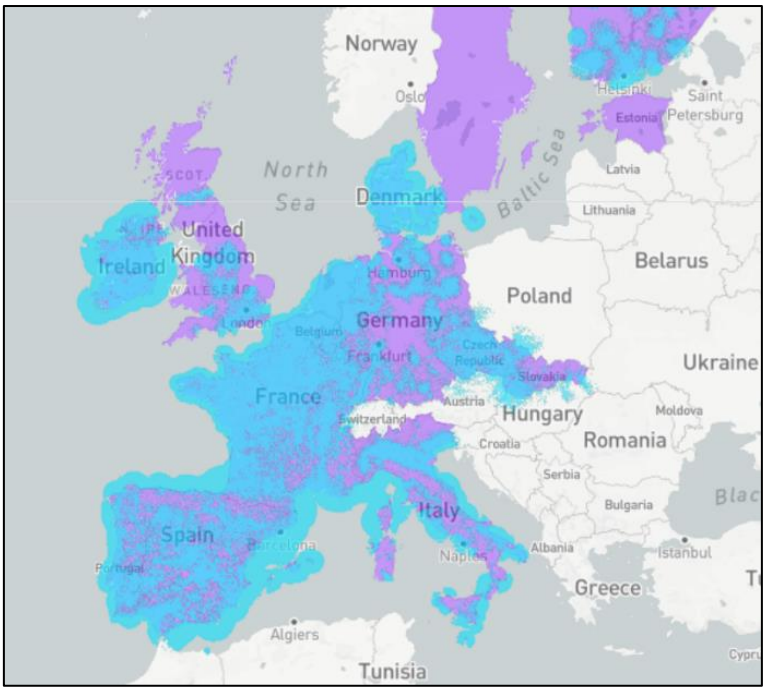



**Figure 10 : Couverture de Sigfox en Europe centrale au 1er juin 2017[13]**

Des fabricants se sont mis à concevoir du matériel compatible ces dernières années mais la technologie peine à prendre un réel envol car c'est un concurrent direct aux opérateurs téléphoniques actuels, au niveau de la communication M2M.

### 5.1.4 Ingenu

Ingenu est une société qui a développé une modulation propriétaire RPMA (Random Phase Multiple Access), utilisant la bande libre 2.4GHz, la même que celle utilisée pour le Wifi. L'avantage de cette bande de fréquence est qu'elle est libre dans le monde entier et donc les appareils compatibles peuvent être vendus indépendamment de la zone géographique. La société se targue de permettre une couverture bien supérieure que ses principaux concurrents, à savoir LoRa et Sigfox[14] mais pour être complet, il faut prendre en compte les différences régionales sur les législations en termes de puissance d'émission. Tout comme Sigfox, la société veut développer son réseau de communication M2M mondial et développe des solutions de stockage cloud. Bien qu'un réseau soit en cours de développement dans des grandes villes américaines elle ne s'est pas encore étendue en Europe. Choisir cette technologie dans un projet revient à s'obliger de travailler avec Ingenu, car ils en sont propriétaires et sont les seuls à la commercialiser. Ils ne communiquent pas directement sur le prix de mise en œuvre d'un réseau avec leur solution, mais un article du site IoTnow[15] parle de près de 15'000$.

## 5.2 Technologie utilisée

Etant donné la disponibilité des solutions LTE-M, NB-IOT et Symphony Link ainsi que les contraintes que nécessiteraient l'option Sigfox ou Ingenu en termes d'abonnements et de dépendance à un seul acteur, la solution retenue est de développer une couche MAC propriétaire, basé sur la couche physique LoRa. Ce choix est idéal pour un projet comme celui-ci

[13] Crédit : Sigfox, *http://sigfox.com*
[14] Source : Ingenu, *http://www.ingenu.com, « INGENU-RPMA-to-LPWA-Comparison.pdf »*
[15] Source : *http://www.iot-now.com*, article « RPMA technology coming to Europe in 2017 », par Sheetal Kumbhar, 10 janvier 2017

car les modules basés LoRa sont généralement simples d'utilisation et ne nécessitent pas une connaissance accrue de cette modulation.

L'HES-SO Valais/Wallis possède déjà un kit de développement, suite à des projets antérieurs concernant cette technologie. Ils seront donc utilisés pour mener à bien des tests de portée.


**Figure 11 : Kit de développement SK-iM880A**[16]

Ce kit de développement est produit par la société IMST GmbH. Il comporte une plaque de base, « WiMOD Demo-Board » sur laquelle on peut insérer une carte d'extension composée d'un module LoRa de type iM880 et d'une antenne 868MHz.

[16] Crédit : IMST GmbH, *http://imst.com*

# 6 Test de portée avec la modulation LoRa

## 6.1 Principe

Ces tests sont destinés à se faire une idée des performances de la modulation LoRa dans différents milieux de transmission et à comparer les résultats obtenus avec les résultats théoriques affichés par le fabricant. Pour cela, deux modules LoRa équipés d'une antenne seront placés à différentes distances l'un de l'autre et différents paramètres seront ajustés, comme illustré à la Figure 12.

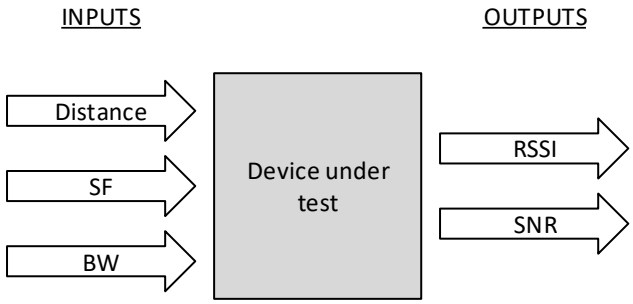



**Figure 12 : Principe des tests de portée avec la modulation LoRa**

L'une des antenne, connectée à un ordinateur, sera fixe et fera office de collecteur pour les données transmises. L'autre sera montée sur batterie et se chargera de la transmission, pilotée par un ordinateur de poche de type Raspberry.

**Le paramètre SF** (Spreading factor) représente l'étalement de spectre de la modulation. Il correspond au nombre de gazouillement nécessaire à la transmission d'un symbole. Plus il est haut, plus le temps de transmission d'un seul symbole va augmenter, ce qui impactera le débit de la transmission. Toutefois, il faut bien comprendre qu'un grand facteur d'étalement va également permettre au récepteur de distinguer plus facilement le signal du bruit en raison de l'augmentation de ce temps de transmission. Ce phénomène induit une baisse de la sensibilité aux perturbations et donc une augmentation de la portée. Ce paramètre peut prendre une valeur entre 7 et 12.

**Le paramètre BW** (Bandwidth) représente la largeur de bande utilisable pour la modulation. Plus celui-ci est haut, plus la transmission aura un débit de symboles important. Cependant, la portée en sera impactée dans le sens inverse. Ce paramètre peut être : 125kHz, 250kHz ou 500kHz.

En résumé, la configuration radio offrant la portée maximale est atteinte lorsque le SF est au plus haut et la BW au plus bas. Dans celle-ci, le temps de transmission d'un symbole étant également au plus long, la consommation de batterie sera la plus conséquente.

### 6.1.1 Débit

Le débit de la transmission dépend entièrement des paramètres SF, BW et du codage d'erreur (CR). En effet, la distance n'a aucun impact sur cette donnée. Les débits peuvent donc être calculé grâce aux formules fournies par Semtech (Semtech Corporation, 2015).

$$\text{Débit}\left[\frac{\text{bits}}{s}\right] = \text{SF} * \frac{\text{CR}}{\frac{2^{SF}}{BW}} = SF * \frac{\frac{4}{4+CR}}{\frac{2^{SF}}{BW}} = \frac{4 * SF * BW}{(4+CR) * 2^{SF}}$$

Ce qui nous donne, pour un SF de 7 à 12, trois BW différentes et une correction d'erreur maximale (CR = 4), le résultat présenté à la Figure 13.

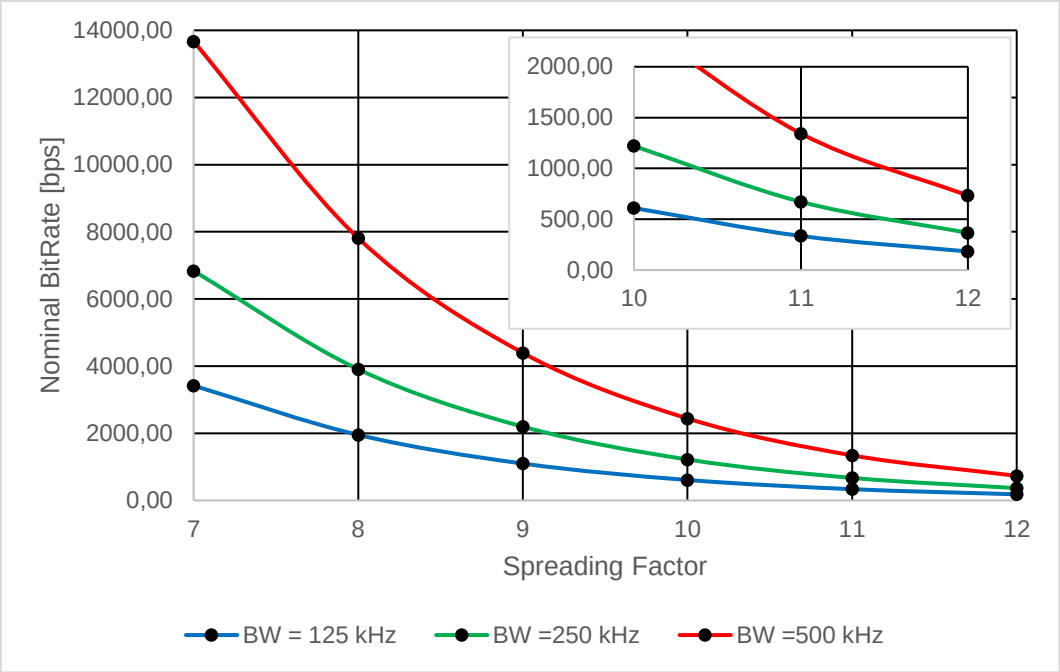



**Figure 13 : Influence des paramètres BW et SF sur le débit**

# 6.2 Préparations

## 6.2.1 Matériel utilisé

La liste du matériel utilisé est présentée dans le Tableau 2. La colonne « Composants internes » indique les composants électroniques utilisés sur la carte de développement et les logiciels utilisés sur le PC portable.

| Matériel | Composants internes | Pièce(s) |
|---|---|---|
| **Raspberry Pi 3 Model B** | | 1 |
| **Carte de développement WiMOD Demo-Board** | | 2 |
| | Module iM880 | |
| | Antenne 868MHz | |
| | Bouton poussoir « B3 » | |
| | Commutateur dipôle « Dip1 » | |
| | LED de contrôle « D2 » | |
| **PC portable** | | 1 |
| | WiMOD LR Studio v.1.18.4 | |
| | FTDI VCP driver v.2.12.26 | |
| | Windows 10 v.10.0.12393 | |
| **Câble USB type B** | | 1 |
| **Câble micro-USB type B** | | 1 |
| **Batterie portable mophie 5V, 1A** | | 1 |

**Tableau 2 : Matériel utilisé pour les tests de portée LoRa**

## 6.2.2Raspberry Pi

Un petit programme de test a été développé pour le Raspberry Pi, son fonctionnement est décrit à la Figure 14 et le code de son implémentation est présenté à l'annexe [2].

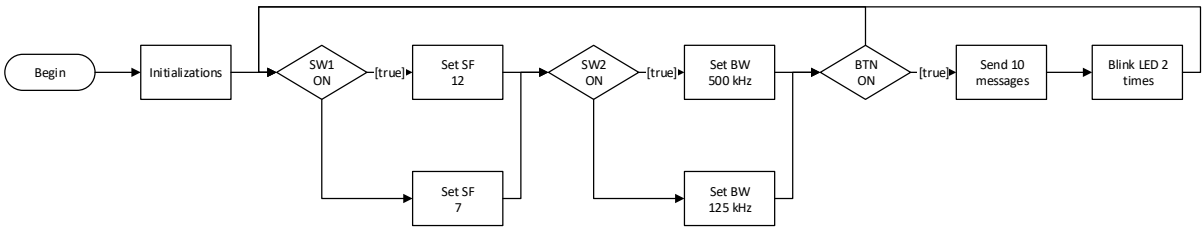



**Figure 14 : Algorithme de test du Raspberry Pi**

Cette implémentation utilise la librairie WiringPi (*http://wiringpi.com*) afin d'accéder de façon simple aux pins du Raspberry. Pour utiliser cette librairie, il faut lancer le programme de test en tant qu'utilisateur `root`. Pour cela, il suffit d'ajouter la commande `sudo` au lancement : `sudo ./LoRaGateway`.

Pour pouvoir démarrer le programme au démarrage du Raspberry, une astuce supplémentaire est utilisée : il faut éditer le fichier rc.local. Pour cela, il faut entrer la commande `sudo nano /etc/rc.local` et rajouter la ligne `./home/[chemin vers l'executable]`, comme présenté à la Figure 15.

```
#!/bin/sh -e
#
# rc.local
#
# This script is executed at the end of each multiuser runlevel.
# Make sure that the script will « exit 0 » on success or any other
# value on error.
#
# In order to enable or disable this script just change the execution
# bits.
#
# By default this script does nothing

# Print the IP address
_IP=$(hostname -I) || true
if [ « $_IP » ]; then
      printf « My IP address is %s\n »  « $_IP »
fi

# Launch this program at the end of the user runlevel
./home/pi/Desktop/LoRaGW/LoRaGateway &

exit 0
```

**Figure 15 : Contenu du fichier /etc/rc.local après édition**

### 6.2.3 IM880

Les deux modules LoRa sont calibrés sur une fréquence porteuse fixe, à savoir 868.5 MHz. En fonction du test, le facteur de dispersion commutera entre SF = 7 ou SF = 12 et la largeur de bande sur BW = 125kHz ou BW = 500kHz. Le codage d'erreur sera fixé sur EC = 4/5 (valeur par défaut) et la puissance d'émission sur 17dBm. Il faut savoir que les puces RF embarquées dans ces modules sont capable de recevoir des données avec un RSSI de -137dBm.

Le module connecté au PC se configure grâce au logiciel WiMOD LR Studio (Figure 16) alors que le module connecté au Raspberry se configure via l'interface UART, à l'initialisation du

système. On configure les deux modules pour qu'ils utilisent le mode « trame étendues ». Ce paramètre oblige les modules à envoyer des Bytes supplémentaires à chaque transmission, contenant des informations sur la qualité de la transmission (RSSI et SNR).

| Radio | COM6 |
|---|---|
| Type | iM880A-L (128k) |
| Device ID | 0x00000DB6 (3510) |
| Firmware | WiMOD_LR_Base |
| Version | V1.12 (13.01.2017) |
| Build Count | 61 |
| Operation Mode | Application Mode |
| Radio Mode | Standard |
| Device L-Address | 10:1234 |
| Frequency | 868500000 Hz |
| Modulation | LoRa |
| Spreading | SF7 |
| Bandwidth | 500000 Hz |
| Error Coding | 4/5 |
| Power Level | 17 |

**Figure 16 : Configuration type d'un module LoRa pour les tests de portée**

### 6.2.4 Branchements

La liste des branchements effectués entre le Raspberry et la carte WiMOD Demo-Board est présentée dans le Tableau 3. Il faut également alimenter le Raspberry Pi à l'aide de la batterie portable et connecter le PC à la deuxième carte de développement. La définition des pins du Raspberry Pi est présentée en annexe [3] et celle de la carte de développement peut être obtenue en ligne (IMST GmbH, 2013).

| Description | Raspberry Pi | WiMOD Demo-Board |
|---|---|---|
| **VCC** | 02 | X8.1 |
| **GND** | 06 | X8.2 |
| **Raspberry Pi « Tx »** | 08 | X7.20 |
| **Raspberry Pi « Rx »** | 10 | X7.14 |
| **LED « D2 »** | 12 | X6.3 |
| **Bouton « B3 »** | 16 | X5.1 |
| **Switch « Dip1.1 »** | 18 | X5.3 |
| **Switch « Dip1.2 »** | 22 | X5.15 |

**Tableau 3 : Connexions Raspberry Pi <=> WiMOD Demo-Board**

**Figure 17 : Batterie portable + Raspberry Pi + WiMOD Demo-Board, test en intérieur**

## 6.3 Test en intérieur

### 6.3.1 Situation

Ce test a été mené dans un bâtiment d'habitation standard afin de mesurer l'impact de la pénétration d'une dalle sur la portée de la modulation en intérieur. L'immeuble en question est doté de 3 étages d'appartements, un rez-de-chaussée et deux sous-sols comme on peut le voir sur la Figure 18. On y distingue étalement le dispositif récepteur (PC+iM880A) installé au 3ème étage, symbolisé par un « R » et les cinq points de mesures, un par étage, symbolisés par un numéro.

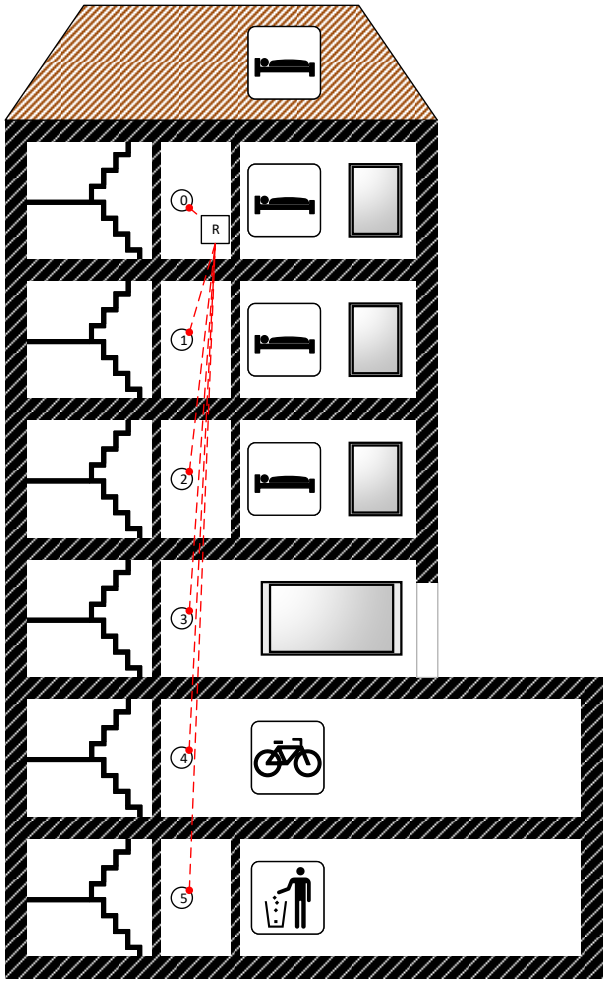



**Figure 18 : Coupe de principe du bâtiment test**

Bien sûr tous les bâtiments ne sont pas construits de la même façon et la transmission s'est toujours faite de la manière la plus verticale possible. Les résultats obtenus sont donc à prendre avec des pincettes et ne peuvent en aucun cas servir de référence pour spécifier le système de manière générique. Ils nous donnent cependant un bon aperçu des performances de la modulation.

### 6.3.2 Résultats

Les résultats obtenus à la réception sont pour le moins surprenants. En effet, comme présenté à la Figure 19, on voit que le signal modulé en LoRa a pu être mesuré à travers les cinq dalles de l'immeuble. On constate que la largeur de bande a toutefois une influence significative dans le rapport signal-sur-bruit. On peut noter également au passage que sur dix paquets envoyés, le PER est de 0% et que le seuil de -137dBm n'a pas été atteint.

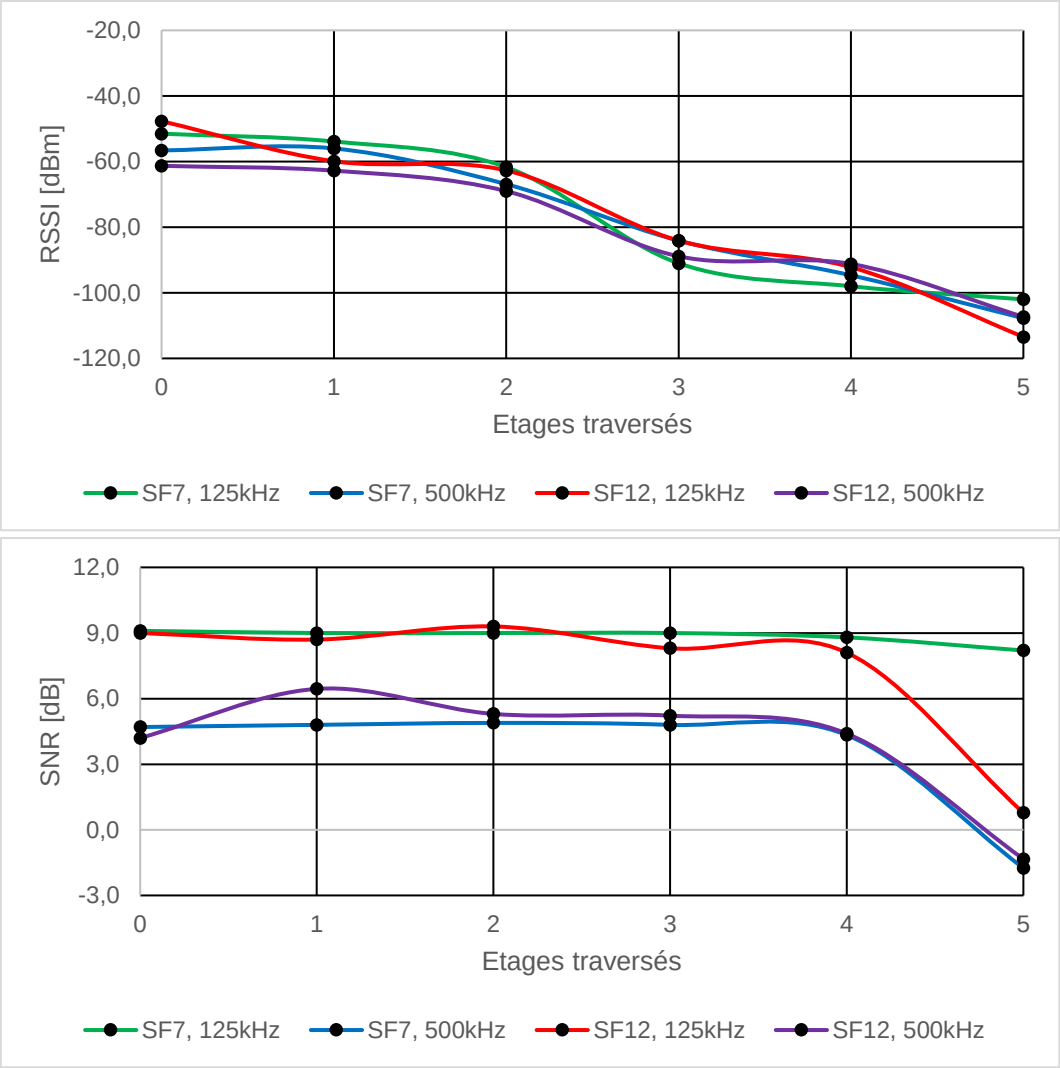



**Figure 19 : Résultats du test en intérieur (RSSI moyen et SNR moyen)**

En analysant ce graphique, on ne peut pas dire qu'une série se distingue de manière évidente des autres, cependant on peut noter l'influence significative de la largeur de bande sur le SNR qui, pour rappel est directement lisible sur les trames reçues grâce au format de trames étendu.

## 6.4 Test en extérieur

### 6.4.1 Situation

Le récepteur est installé en extérieur, dans un endroit bénéficiant d'une vue dégagée sur une dizaine de kilomètres. Les coupes géologiques présentées dans ce chapitre ont été réalisées grâce au site HeyWhatsThat Path Profiler (*http://www.heywhatsthat.com/profiler.html*), un profileur de chemin qui produit des coupes de terrain. On notera que le récepteur est toujours représenté sur la gauche et l'émetteur sur la droite de ces coupes. La ligne rouge qui relie les deux points représente le chemin parcouru par le signal en ligne droite et les lignes jaunes, qui l'entourent représente la zone dans laquelle des obstacles influeraient sur le signal. Cette donnée est automatiquement générée par le site, quand on spécifie la fréquence porteuse du signal. Ici, ce paramètre a été fixé à 868.5MHz.

Le principe de ce test reste le même que celui effectué en intérieur. Seulement cette fois l'émetteur est déplacé entre zéro et huit kilomètres par sauts de deux kilomètres. À chaque point de mesure, on relève les données grâce au PC qui joue le rôle de récepteur.

### 6.4.1.1 *Récepteur*

Le lieu choisit pour poser le récepteur se situe en Suisse, dans le canton du Valais, à côté du village de Corin-sur-Sierre. Ses coordonnées GPS sont : 46.28804° N 7.506214° E et il se situe à 733m d'altitude. La spécificité du lieu réside dans sa proximité avec la ville de Sierre. En effet, on peut s'éloigner autant que l'on veut du point de départ, il faut de toute façon que le signal passe au-dessus de cette petite ville pour atteindre le récepteur. Cette situation particulière nous donne un bon aperçu de ce que peut donner une transmission en extérieur, dans un milieu quasi-urbain.


**Figure 20 : Localisation du récepteur et de la zone de test sur la carte du valais**

### 6.4.1.2 *Emetteur à 2km*

Le point se situe aux coordonnées 46.29344° N 7.531233° E. Il est en plein centre-ville de Sierre et est entouré de deux immeubles. On s'attend toutefois à recevoir les signaux sans trop de difficulté car on peut apercevoir facilement l'endroit où est situé le récepteur à l'œil nu. Les seuls obstacles que peut potentiellement rencontrer le signal sont à proximité de l'émetteur et devraient donc peu influer sur la distance maximale du signal.

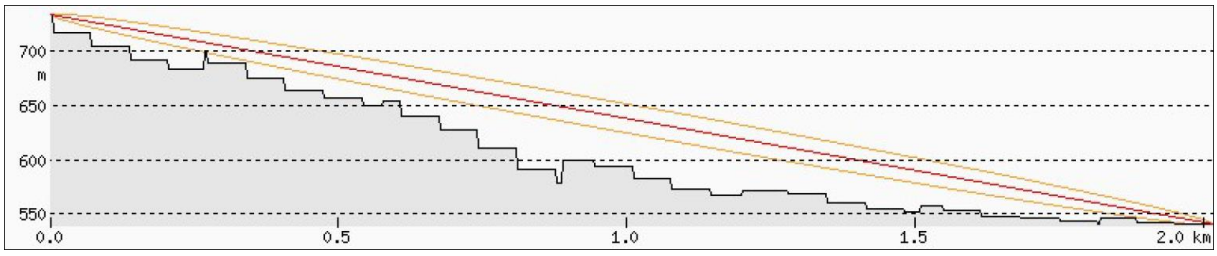



**Figure 21 : Situation de l'émetteur à 2km**

### 6.4.1.3 *Emetteur à 4km*

Le point se situe aux coordonnées 46.29871° N 7.555825° E. Il est entouré de champs et se trouve à une altitude de 539 mètres. On peut donc s'attendre à ce que le signal passe bien au début, d'autant plus que l'on voit encore bien le lieu où se situe le récepteur à l'aide de simples jumelles. Toutefois, étant donné l'altitude plutôt basse, le signal va devoir passer vraiment très proche de la ville voisine, ce qui rend les perturbations urbaines plus présentes.

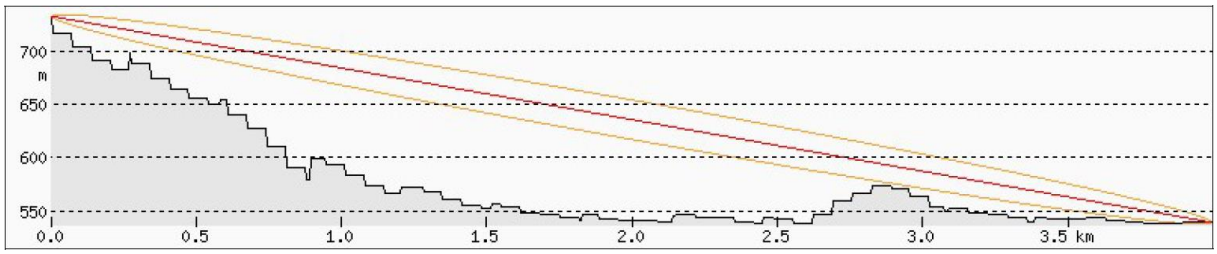



**Figure 22 : Situation de l'émetteur à 4km**

### 6.4.1.4 *Emetteur à 6km*

Le point est situé aux coordonnées 46.30556° N 7.578439° E. Il est un peu surélevé par rapport aux points précédents (565 mètres d'altitude) et donc le signal va devoir passer beaucoup moins proche de la ville. La vue y est dégagée et on aperçoit le point d'arrivée du signal sans aucune difficulté, grâce aux jumelles.

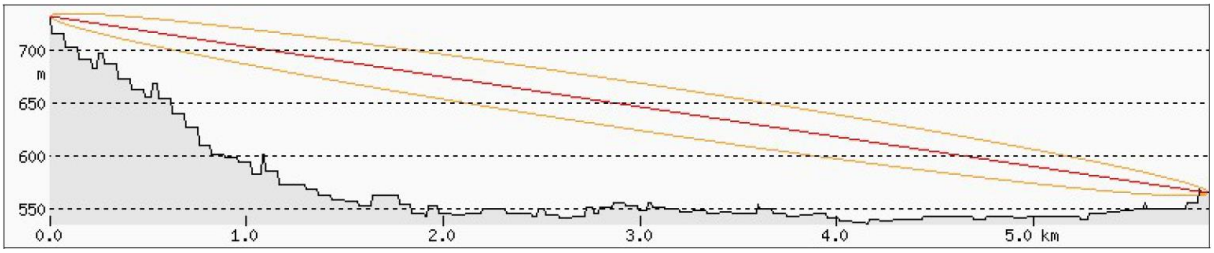



**Figure 23 : Situation de l'émetteur à 6 km**

### 6.4.1.5 *Emetteur à 8km*

Ce point est situé aux coordonnées 46.30402° N 7.607344° E, à 588 mètres d'altitude. On arrive à distinguer l'emplacement du récepteur mais il paraît maintenant bien loin. Cependant aucun obstacle physique ne barre la route du signal.

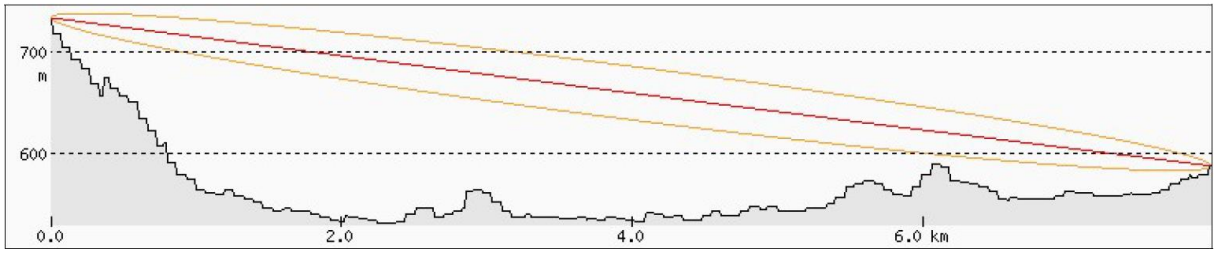



**Figure 24 : Situation de l'émetteur à 8km**

## 6.4.2 Résultats

La Figure 25 résume les résultats obtenus lors de ce test. Il faut noter qu'à deux kilomètres du point de départ, la série « SF7, 125kHz » n'a donné aucun résultat alors qu'elle en a donné à quatre kilomètres. Le point présenté sur le graphique a été interpolé pour les besoin de l'analyse. La distance n'est donc pas le seul paramètre qui influe sur la transmission, l'explication la plus évidente pour expliquer ce résultat et que les perturbations engendrées par la ville sur le signal étaient trop fortes et que la transmission avec un facteur d'étalement de sept n'est peut-être pas la plus fiable en milieu urbain.

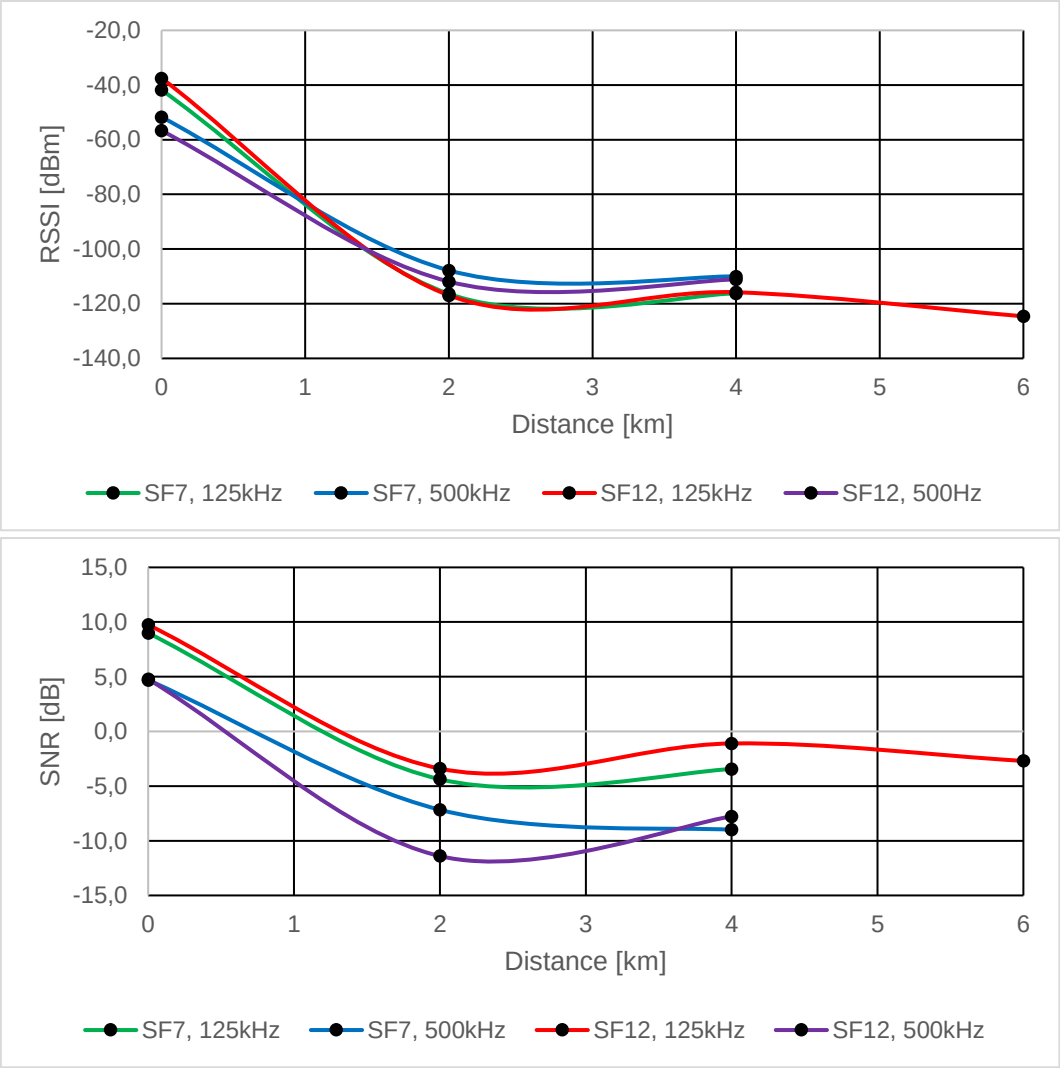



**Figure 25 : Résultats du test en extérieur (RSSI moyen et SNR moyen)**

Aucun signal n'a pu être capté au-delà de six kilomètres, malgré de multiples tentatives. La seule série qui a réussi à passer le test des six kilomètres est la série « SF12, 125kHz ». On pouvait s'attendre à ce qu'elle atteigne une portée plus longue que les autres, comme expliqué en préambule de ces tests. Ce qui pose problème dans ce genre de considérations, c'est le milieu dans lequel le test est effectué.

## 6.5 Synthèse

En conclusion, on a pu vérifier l'influence des paramètres SF et BW sur la portée de la modulation. En effet, on a vu que la portée était maximisée avec le paramètre SF au maximum et le paramètre BW au minimum. Au vu de ces résultats, on peut raisonnablement espérer recevoir un signal de qualité en milieu urbain, si l'on utilise un facteur d'étalement de 12 et une largeur de bande de 125kHz à deux ou trois kilomètres de distance. En intérieur, la traversée de cinq étages en ligne droite n'a pas réussi à empêcher le signal de passer. Ce qui représente une portée en intérieur pour le moins impressionnante.

A titre de comparaison, Semtech nous promet une portée de 30miles (plus de 48km) en milieu rural. Cependant, ils ne fournissent pas la définition de ce qu'ils entendent par « milieu rural ». On ne peut donc pas dire que leurs promesses ne sont pas tenues mais on voit bien là qu'elles sont un peu exagérées, probablement dans un but commercial.

# 7 Développement de la solution

La solution développée durant ce projet est basée sur l'utilisation de cartes de développement embarquant des modules RF LoRa de la série iM880, fabriquées par IMST GmbH et de cartes de développement basées sur des microcontrôleurs STM32 de chez STMicroelectronics. Le microcontrôleur va piloter le module RF via une liaison sérielle pour l'émission et la réception des trames LoRa.

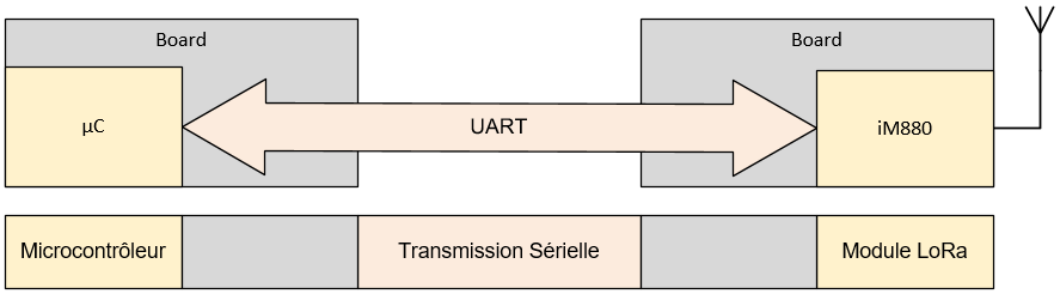



**Figure 26 : Schéma de principe d'un nœud**

L'addition de ces deux cartes de développement constitue un « nœud ». D'un autre côté, le Raspberry Pi utilisé pour les tests de portée (chapitre 6.2.1) accompagné d'un module RF forment la passerelle. Il faut avoir deux de ces nœuds et une passerelle pour commencer à parler de « forwarding » de paquets.

## 7.1 Matériel utilisé

La liste complète du matériel utilisé dans le développement de cette solution est présentée dans le Tableau 4. La colonne « Composants internes » fait référence aux composants électroniques utilisés sur les cartes de développement et aux logiciels utilisés sur le PC portable. Les prochains chapitres explicitent plus en détails certaines entrées de cette liste.

| Matériel | Composants internes | Pièce(s) |
|---|---|---|
| **Raspberry Pi 3 Model B** | | 1 |
| **Carte de développement WiMOD Demo-Board** | | 4 |
| | Module LoRa iM880 | |
| | Antenne 686 MHz | |
| | Bouton poussoir « B3 » | |
| | Commutateur dipôle « Dip1 » | |
| | LED de contrôle « D2 » | |
| | Potentiomètre « Pot1 » | |
| **STM32VLDISCOVERY** | | 3 |
| | Bouton poussoir « B3 » | |
| | LED utilisateur « LD4 » | |
| **PC portable** | | 1 |
| | Windows 10 v.10.0.12393 | |
| | Atollic TrueSTUDIO v.7.0.1 | |
| **Câble micro-USB type B** | | 3 |

**Tableau 4 : Matériel utilisé dans la solution**

### 7.1.1 Raspberry Pi

Pour les informations concernant le Raspberry Pi 3 Model B, se référer au chapitre 6.2.2. Le Makefile utilisé pour compiler le projet est disponible en annexe [6].

### 7.1.2 STM32VLDISCOVERY

Une grande partie des informations fournies ici sont tirées du manuel d'utilisation de la carte STM32VLDISCOVERY (STMMicroelectronics, 2011). Les principales caractéristiques de cette dernière sont :

- Microcontrôleur STM32F1000RBT6B avec 128kB de mémoire Flash et 8kB de RAM
- Debugger ST-Link embarqué
- Alimentation par connecteur USB ou batterie 3.3-5V
- Deux LED utilisateur
- Un bouton utilisateur et un bouton reset

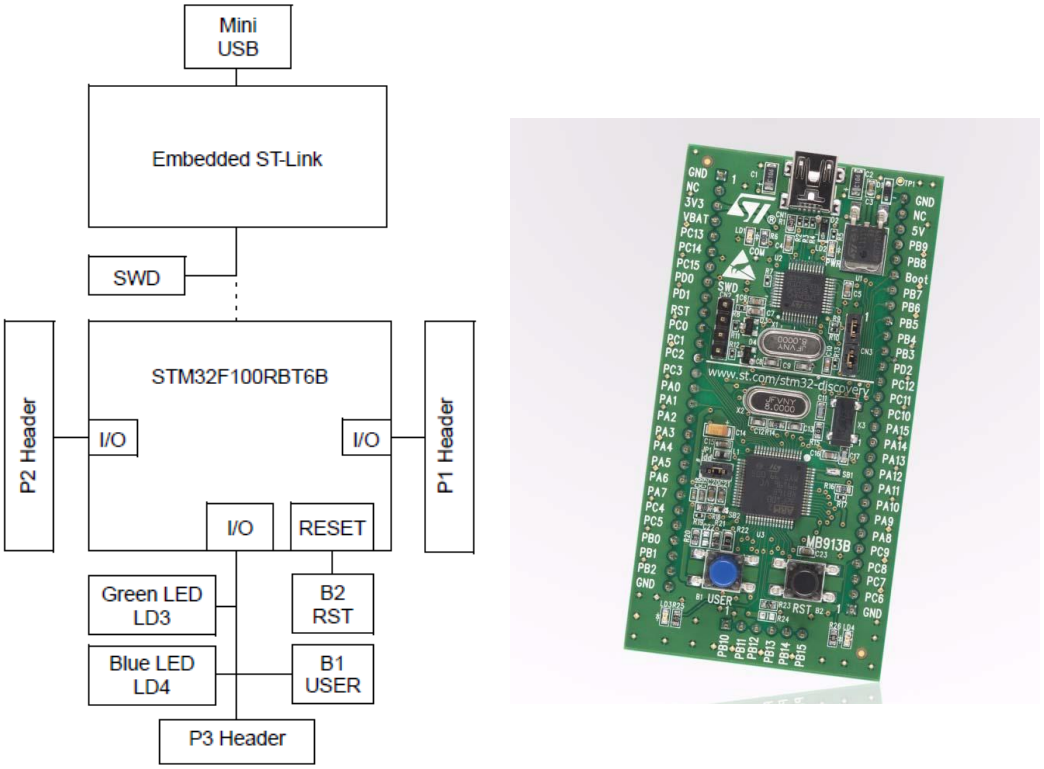



**Figure 27 : Architecture matérielle et photo de la carte de développement STM32VLDISCOVERY[17]**

Ce type de carte est idéal pour débuter un projet rapidement, en raison de son faible coût (~CHF 10. –)[18], de sa rapidité de mise en service et de sa simplicité, grâce aux librairies mises à disposition par le fabricant.

#### 7.1.2.1 *STM32F1000RBT6B*

Ce microcontrôleur est basé sur un ARM® Cortex®-M3 32-bits. Il possède une multitude de périphériques, comme on peut le voir sur la Figure 28. On peut citer ceux qui seront utiles pour ce projet :

- Des convertisseurs analogiques vers numériques (ADCx)
- Un contrôleur d'interruptions/évènements externes (EXTI)
- Des périphériques d'entrée/sortie à usage général (GPIOx)
- Des transmetteurs/récepteurs synchrones/asynchrones universels (UARTx)

[17] Crédit : STMicroelectronics (STMMicroelectronics, 2011)
[18] Catalogue Farnell : *http://ch.farnell.com/fr-CH/stmicroelectronics/stm32vldiscovery/f100-st-link-kit-discovery/dp/1824325*, consulté le 27 juillet 2017

- Des minuteurs à usage général (TIM2 à TIM5)

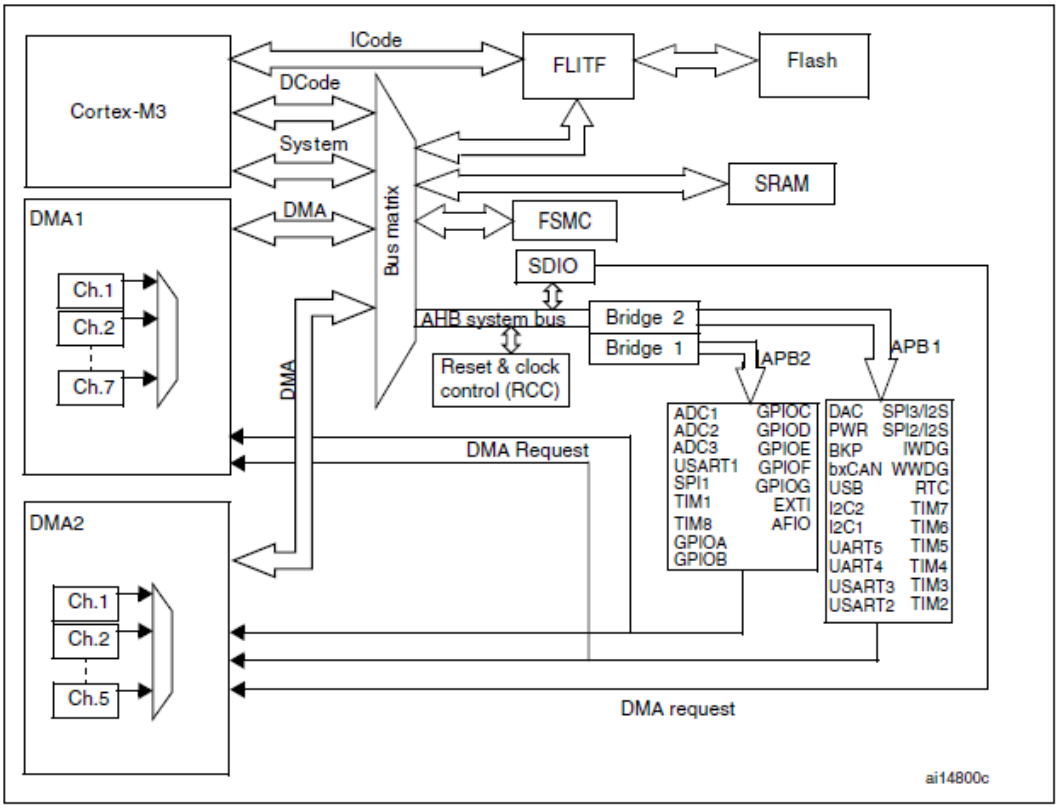



**Figure 28 : Architecture des microcontrôleurs de la famille STM32F10x**[19]

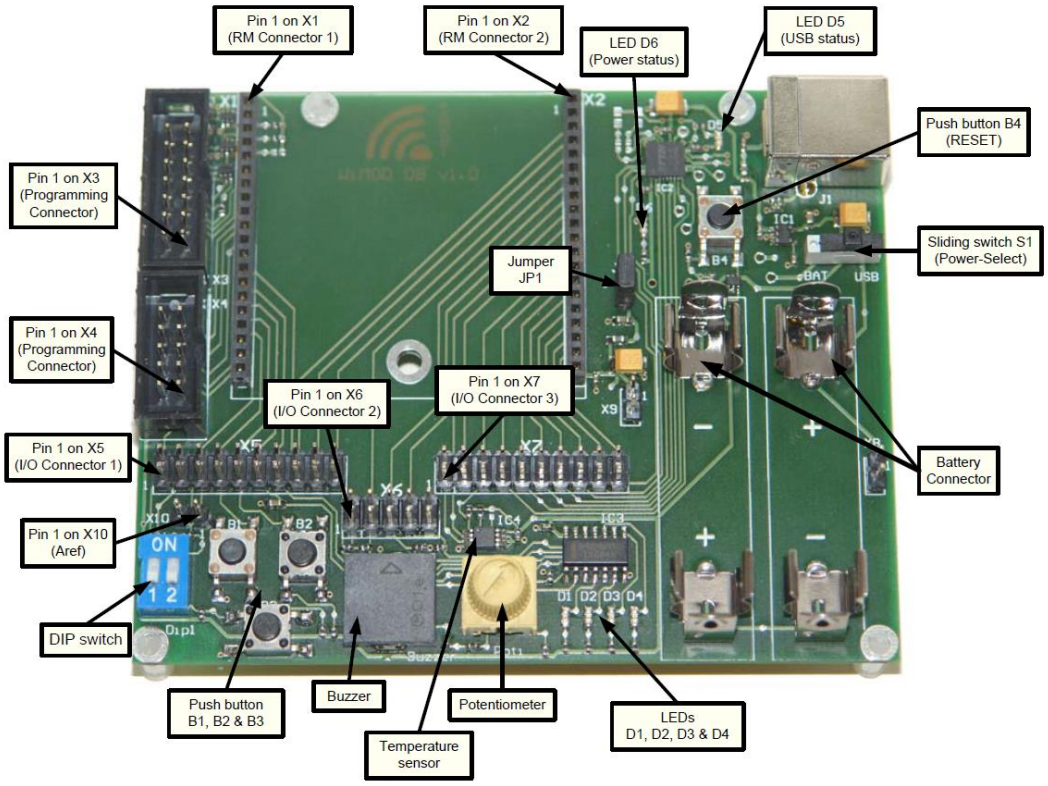
### 7.1.2.2 *Fichiers de configuration*

Les fichiers de configuration liés à la carte de développement et au microcontrôleur peuvent se télécharger gratuitement sur le site de STMicroelectronics (*http://www.st.com/en/evaluation-tools/stm32vldiscovery.html*). Cependant, l'environnement de développement utilisé pour la réalisation de ce projet (cf. 7.1.4) les inclus déjà sans avoir besoin de les télécharger.

## 7.1.3 WiMOD Demo-Board

Ces cartes ont été choisies en raison de leur compatibilité avec les modules RF de la série iM880, très simples d'utilisation. En effet, ces composants se pilotent via une liaison sérielle UART, surmontée d'une interface HCI prédéfinie par le fabricant (IMST GmbH, 2011). Ils s'insèrent dans les connecteurs « X1 » et « X2 » de la carte de démonstration, représentée en Figure 29.

[19] Crédit : STMicroelectronics, tiré de « RM0008 Reference manual », 2015




[20]

**Figure 29 : WiMOD Demo-Board, sans module RF**

### 7.1.4 Atollic TrueSTUDIO for ARM

Ce logiciel, développé par Atollic AB, est construit sur le logiciel Eclipse Neon v.1.a. Il est utilisé pour ce projet dans sa version de démonstration 7.0.1. Son interface graphique diffère donc peu du logiciel Eclipse, un standard dans le développement sur microcontrôleur. Cependant, il bénéficie de quelques fonctionnalités utiles :

- La carte STM32VLDISCOVERY est prise en charge par le logiciel, qui fournit donc toute la configuration de base, bas niveau. On peut donc brancher la carte pour la première fois, écrire quelques lignes de code et tester directement le résultat.
- Des projets de démonstration sont inclus dans l'installation.
- Couplé avec l'interface ST-Link, il devient très puissant pour la recherche de bugs.

## 7.2 Branchements

- Pour les branchements entre la carte RF et le Raspberry Pi, se référer au chapitre 6.2.4.
- Les pins utilisés pour brancher la carte RF et la carte de développement STM32VLDISCOVERY sont listés dans le Tableau 5. La liste des pins de cette dernière est disponible sous le chapitre « 3. Extension connection » (STMMicroelectronics, 2011).

| Description | STM32VLDISCOVERY | WiMOD Demo-Board |
|---|---|---|
| **VCC** | P2.26 (5V) | X8.1 |
| **GND** | P2.28 (GND) | X8.2 |
| **STM32VLDISCOVERY « Tx »** | P3.1 (PB10) | X7.20 |
| **STM32VLDISCOVERY « Rx »** | P3.2 (PB11) | X7.14 |
| **Potentiomètre « Pot1 »** | P1.16 (PA1) | X6.3 |

**Tableau 5 : Connexions STM32VLDISCOVERY <=> WiMOD Demo-Board**

[20] Crédit : IMST GmbH, tiré de « WiMOD Demo-Board User Guide », 2009

- Le module RF iM880 s'insère naturellement dans les connecteurs « X1 » et « X2 » de la carte WiMOD Demo-Board.
- Une antenne 868 MHz est vissée sur le module RF à l'emplacement prévu.

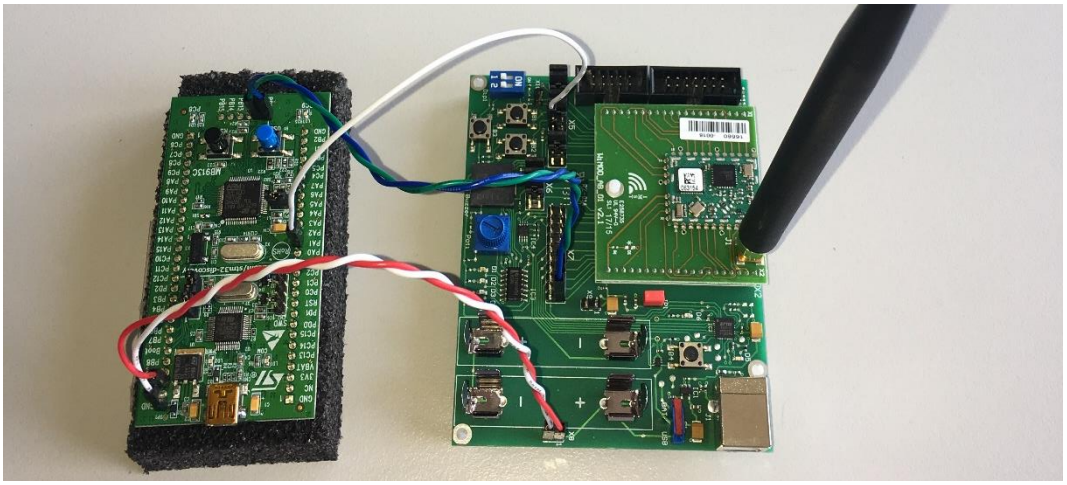


**Figure 30 : STM32VLDISCOVERY + WiMOD Demo-Board, avec module RF**

# 7.3 Développement logiciel

## 7.3.1 Préambule

Ce chapitre traite du développement du logiciel implémenté dans l'hôte des cartes iM880, à savoir : le microcontrôleur STM32 d'un côté et le Raspberry Pi de l'autre. Son architecture a été pensée en différentes couches logicielles qui seront présentées et expliquées par la suite.

Toutes les couches ont été entièrement implémentées lors de ce travail. Cependant, il faut bien se rendre compte que les couches « Application » et « Serial Device » sont fortement liées au système sur lequel est déployé le programme et la couche « HCI » est dépendante des modules RF utilisés. Afin que le rôle de chacune de ces couches soit clair, des diagrammes de séquences [4] s'y rapportant sont annexés à ce travail et des références à ces derniers peuvent apparaître dans les prochains points.

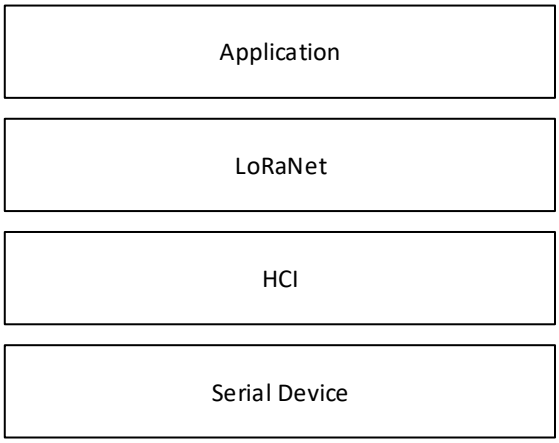



**Figure 31 : Architecture en couche du protocole**

Le microcontrôleur sera dépourvu de système d'exploitation afin de garder une implémentation simple des couches alors que le Raspberry Pi en possède un (Linux raspberrypi 4.4.11-v7+) de base. En effet, le logiciel développé dans les microcontrôleurs est destiné à faire fonctionner des applications simples, ne nécessitant pas forcément de faire des calculs en parallèle ou de faire

tourner plusieurs applications sur la même machine alors que celui développé sur le Raspberry Pi est destiné à faire fonctionner une passerelle, gérant une certaine quantité de connexions.

### 7.3.2 Types de données

Différents types de donnés ont été définis pour faire communiquer ces couches entres-elles. Le but est de garder chaque couche la plus indépendante des autres possible. Les prochains points détaillent ces structures.

#### 7.3.2.1 *Byte buffer*

Ce type de données est très rudimentaire. Il représente un tampon de Bytes à l'aide d'une structure composée d'un pointeur vers le premier Byte et d'un nombre non-signé, codé sur 16 bits, indiquant la taille du tampon.

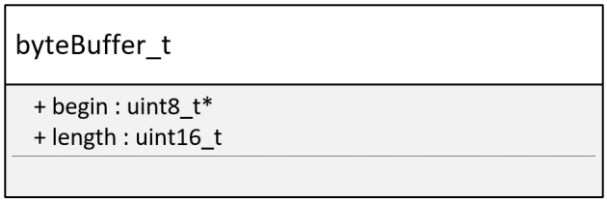



**Figure 32 : Structure byteBuffer_t**

Il est utilisé pour la communication entre les couches, surtout pour transmettre des données brutes. On peut donc le considérer comme un type de base du système. Pour pouvoir l'utiliser, il faut inclure le fichier « *util/byteBuffer.h* ».

#### 7.3.2.2 *Paquet LoRaNet*

Cette structure, déclarée dans le fichier « loranet.h », est utilisée comme interface entre la couche applicative et la couche LoRaNet, s'occupant principalement de la gestion du réseau. Ce type de données décharge l'application de connaître les adresses physiques des autres nœuds du réseau et permet d'envoyer des données en ne spécifiant que quelques informations.

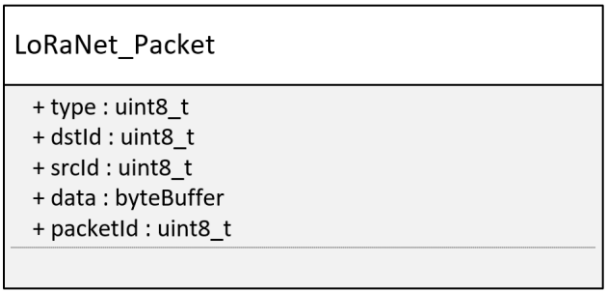



**Figure 33 : Structure LoRaNet_Packet**

On peut décrire les différents attributs qui la composent comme suit :

| Attribut | Description |
|---|---|
| **type** | Représente le type de paquet transporté. Cette valeur peut être :<br>- LORANET_TYPE_UNKNOWN (0x00)<br>- LORANET_TYPE_DATA_UP (0x01)<br>- LORANET_TYPE_DATA_DOWN (0x02)<br>- LORANET_TYPE_PING (0x03)<br>- LORANET_TYPE_PONG (0x04) |
| **dstId** | Représente l'identifiant logique du nœud de destination du paquet. Cette valeur ne peut pas être :<br>- 0x00 (identifiant indéfini)<br>- 0xFF (identifiant de broadcast)<br>- 0xC0 => interdit par l'implémentation du module RF. En effet, c'est un caractère réservé qui définit le début et la fin des trames échangées.<br>Étant donné que c'est un identifiant codé sur huit bits, on a 256 valeurs possibles, moins ces trois cas particuliers => 253 valeurs possibles. |
| **srcId** | Identifiant logique du nœud source du paquet, mêmes restrictions dstId |
| **data** | Données brutes à transmettre, sous la forme d'un tampon de Bytes. Le format des données est définit par l'application. |
| **packetId** | Identifiant du paquet, destiné à l'accompagner à travers les autres nœuds du réseau. Cette valeur est incrémentée à chaque fois que le nœud crée un nouveau paquet. Elle permet une traçabilité des paquets. |

**Tableau 6 : Attributs de la structure LoRaNet_Packet**

Les paquets doivent pouvoir être sérialisés pour descendre les couches jusqu'à la ligne de transmission UART. Pour ce faire, on définit l'ordre de la sérialisation des attributs arbitrairement, comme illustré à la Figure 34 :


**Figure 34 : Sérialisation d'un paquet LoRaNet**

Chaque champ du paquet sérialisé est ensuite facilement atteignable en utilisant un des sauts définis dans le fichier *« loranet.h »* :

- LORANET_OFFSET_DATA (0x04)
- LORANET_OFFSET_DST (0x01)
- LORANET_OFFSET_ID (0x03)
- LORANET_OFFSET_SRC (0x02)
- LORANET_OFFSET_TYPE (0x00)

#### 7.3.2.3 *Message HCI*

Ce type de données permet le passage des données de la couche LoRaNet à la couche HCI, responsable de communiquer de manière protocolaire avec le module RF. J'ai défini cette structure de cette façon car tous les champs qui seront sérialisés à la transmission ou désérialisé à la réception sont présents. Ce qui permet d'unifier les messages transmis à ce niveau.

On voit apparaître ici des attributs faisant référence à un « groupe » d'appareils. C'est une spécificité de la modulation LoRa. En effet, pour pouvoir connecter une multitude d'appareils, chaque appareil fait parti d'un groupe, ce qui permet de multiplier le nombre d'appareils en augmentant le nombre de groupe d'appareils.

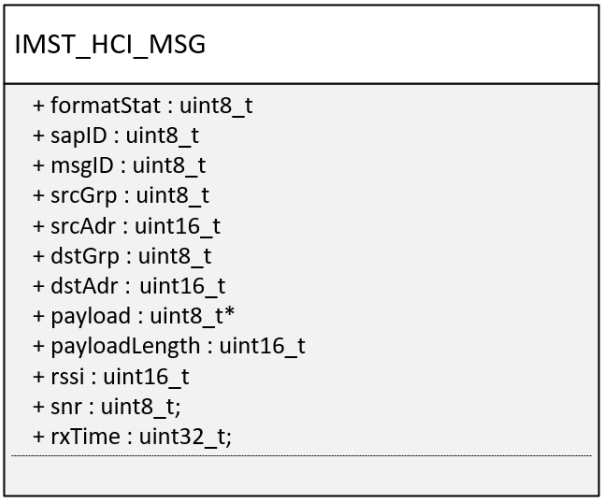



**Figure 35 : Structure IMST_HCI_MSG**

On peut décrire ses attributs de la manière suivante :

| Attribut | Description |
|---|---|
| **formatStat** | Utilisé pour transmettre le format des données à émettre ou le statut des données reçues |
| **sapID** | SAP concerné par le message<br>- IMST_HCI_DEVMGMT_ID (gestion de l'appareil)<br>- IMST_HCI_RLT_ID (test des liens radio)<br>- IMST_HCI_RADIOLINK (lien radio)<br>- IMST_HCI_REM_ID (télécontrôle)<br>- IMST_HCI_HWTEST_ID (test du hardware) |
| **msgID** | Type de message à traiter par le SAP (voir le fichier « *IMST_HCI.h* ») |
| **srcGrp** | Groupe dans lequel se trouve l'appareil source |
| **srcAddr** | Adresse physique de l'appareil source |
| **dstGrp** | Groupe dans lequel se trouve l'appareil de destination |
| **dstAddr** | Adresse physique de l'appareil de destination |
| **payload** | Pointeur vers les données brutes |
| **payloadLength** | Nombre de Bytes qui composent les données brutes (non transmis) |
| **rssi** | RSSI (optionnel, seulement pour la réception) |
| **snr** | SNR (optionnel, seulement pour la réception) |
| **rxTime** | Horodatage du message a été reçu (optionnel, seulement pour la réception). Cette valeur 32bits peut être fournie par l'horloge temps réel. |

**Tableau 7 : Attributs de la structure IMST_HCI_MSG**

### 7.3.2.4 *Configuration Radio HCI*

Cette structure regroupe tous les paramètres configurables des modules RF de type iM880. Des informations détaillées concernant chacun de ces paramètres peuvent être trouvées en ligne (IMST GmbH, 2011). Ledit document en ligne fait le lien entre le protocole HCI et ses formats de trames sérialisées et cette structure sous le chapitre « 3.1.5 Radio Configuration ». J'ai défini cette structure en m'assurant que tous les champs qui doivent apparaître dans les trames sérialisées soient spécifiés. Comme on peut le voir, ils ont tous une longueur bien définie et ils doivent évidemment être transmis dans un ordre spécifique.

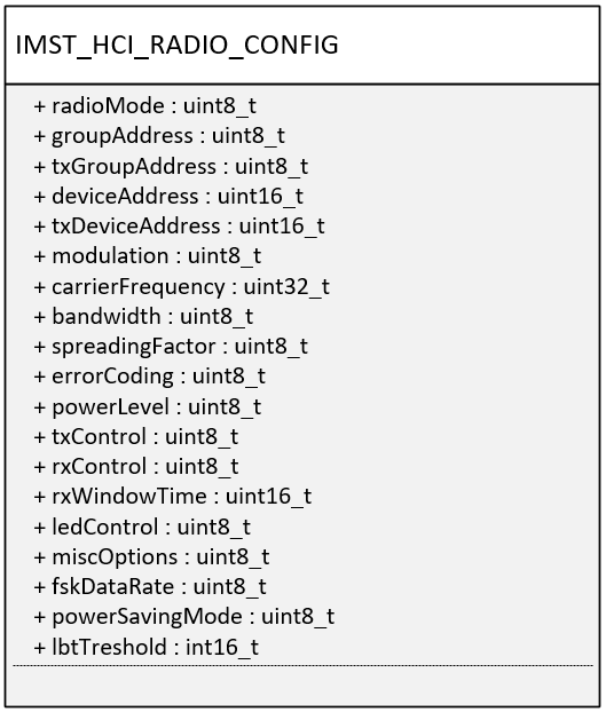



**Figure 36 : Structure IMST_HCI_RADIO_CONFIG**

#### 7.3.2.5 *Nœud LoRaNet*

Cette structure permet de faire le lien entre l'identifiant d'un nœud (nodeId), son adresse physique (address) et son groupe d'appareil (groupId). L'attribut « confirmed » sera expliqué plus tard. Cette structure est interne à la couche LoRaNet.

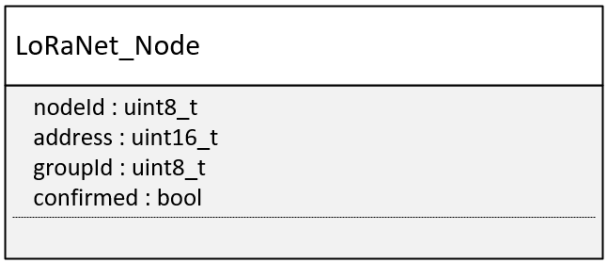



**Figure 37 : Structure LoRaNet_Node**

### 7.3.3 Couche logicielle applicative

L'application utilise directement la couche LoRaNet. Cependant, elle est aussi responsable d'initialiser toutes les couches les plus basses au début du programme, comme illustré sur le diagramme de séquence « Initialisations ». L'ordre d'initialisation des couches a son importance car pour récupérer la configuration radio actuelle du module RF, la couche SerialDevice et la couche HCI doivent être opérationnelles avant la couche LoRaNet.

Elle est également responsable d'appeler périodiquement la méthode « proceed » de la couche LoRaNet et de la couche SerialDevice pour effectuer des actions séquentielles. Sans oublier l'initialisation des outils de temps (voir fichier « *util/time.h* »).

```
int main(void){

	/* INITIALIZATIONS */
	Time_Init();                    //Timers
	SerialDevice_Init();            //Serial device + UART periph.
	HCI_Init();                     //HCI
	LoRaNet_Init(rxCallback, 2);    //LoRaNet + « Get config »

	/* Settings */
	LoRaNet_SetSpreadingFactor(7); //Configure something
	LoRaNet_SendConfig();          //Apply new configuration

	while(1){
		SerialDevice_Proceed();  //Receive waiting frames
		LoRaNet_Proceed() ;      //Send waiting packets

		//Put the application code here
	}
}
```

**Figure 38 : Code de base d'une application LoRaNet**

## 7.3.4 Couche logicielle « LoRaNet »

### 7.3.4.1 *Aperçu*

Cette couche est responsable de gérer le réseau de nœuds. Elle a été conçue pour que les différentes applications ayant besoin d'elle puissent s'adapter facilement. Les méthodes principales utilisables depuis l'application sont :

- « **Init** » : pour initialiser la couche, donner un identifiant logique au nœud et configurer une fonction call-back pour la réception de données au niveau applicatif
- « **PushTxQueue** » : pour demander l'envoi d'une certaine donnée.
- « **Proceed** » : pour garantir une exécution séquentielle des tâches. Cette méthode doit être appelée périodiquement depuis l'application afin de vérifier s'il y a des données à envoyer.

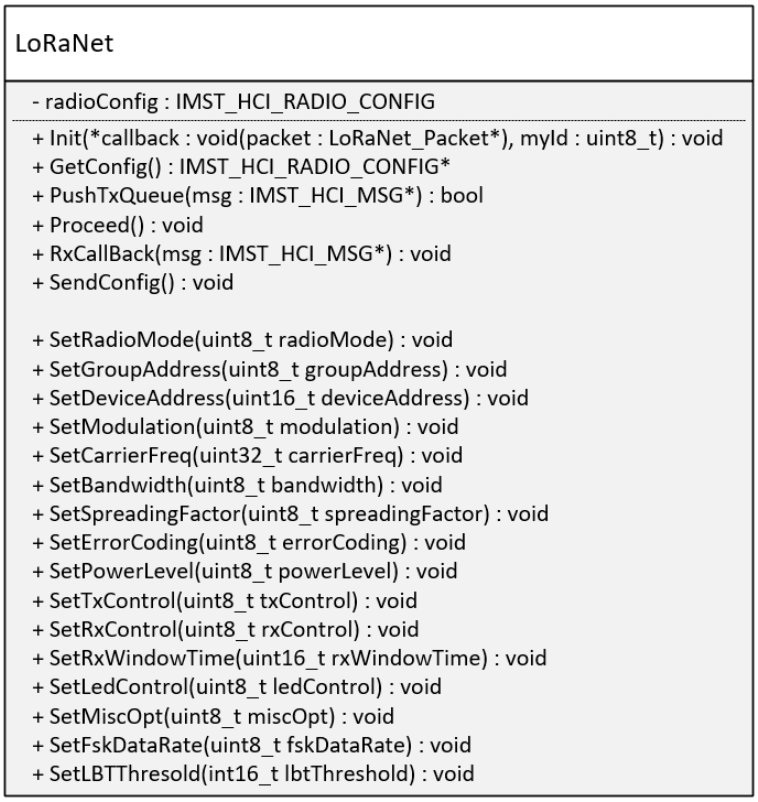



**Figure 39 : Diagramme de classe de la couche « LoRaNet » (méthodes publiques uniquement)**

Elle possède également cinq structures de données qui forment la grosse partie des données dont la couche a besoin pour fonctionner :

- Une liste des nœuds connus dans la direction montante (**uplink**)
- Une liste des nœuds connus dans la direction descendante (**downlink**)
- Un tampon circulaire stockant les n derniers paquets transmis, pour éviter les réexpédition multiples d'un même paquet (**uplink_log**)
- Un tampon de type FIFO, stockant les paquets à envoyer dès l'appel de la méthode « proceed » (**txQueue**)
- La configuration radio locale du module RF, prête à être envoyée dès l'appel de la méthode « sendConfig ». Elle est accessible grâce à la méthode « getConfig » (**radioConfig**)

On voit que la caractéristique du réseau formé par cette couche est que chaque nœud est connecté à d'autres nœuds, mais dans une direction bien particulière (montante ou descendante).

### 7.3.4.2 *Découverte du réseau*

#### 7.3.4.2.1 Principe

La passerelle est considérée comme le maître du réseau, elle peut à tout moment lancer une découverte en envoyant un paquet de type « Ping » avec l'adresse physique 0xFFFF, correspondant à l'adresse de diffusion par défaut (broadcast). Tous les nœuds parvenant à réceptionner ce paquet agissent alors comme suit :

- Enregistrement du nœud source du « Ping » dans la liste uplink
- Envoi d'un paquet de type « Pong » au nœud source du « Ping » pour s'enregistrer dans sa liste downlink

- Envoi d'un paquet de type « Ping » à l'adresse broadcast pour propager la découverte du réseau

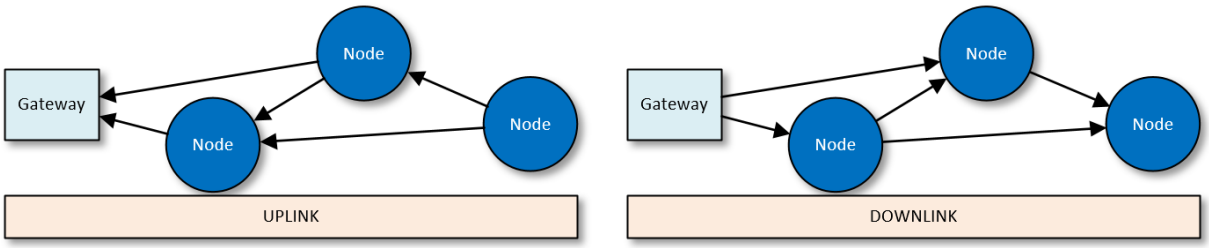



**Figure 40 : Principe des liens entre nœuds**

Le résultat de cette découverte, illustré ci-dessus, nous montre la situation où le nœud le plus éloigné de la passerelle n'a pas réussi à réceptionner le « Ping » de celle-ci mais a en revanche réceptionné les « Ping » des autres nœuds et leur a répondu.

#### 7.3.4.2.2 Problème du paquet perdu

La perte d'un paquet dans un milieu de transmission aussi perturbé que l'air ne peut être totalement exclue. Perdre un paquet contenant des données peut être considéré comme négligeable si cette donnée n'est pas vitale au système global. Toutefois, dans le cas d'un paquet « Ping » ou « Pong », la perte d'un seul paquet peut impacter lourdement le système global car ils sont les outils nécessaires pour assurer les liens entre les nœuds et donc, la stabilité du réseau. Au cours des premiers tests, la couche LoRaNet ne renvoyait sporadiquement pas un paquet « Pong » à la réception d'un paquet « Ping ». Ceci était dû à une erreur d'implémentation mais simulait en définitive la perte d'un paquet dans l'air. Cette erreur a mis en lumière le problème du paquet perdu.

Les nœuds touchés par ce bug ne pouvaient plus s'enregistrer dans la liste descendante des nœuds supérieurs. Ce bug a engendré un phénomène de boucle où deux nœuds pouvaient être liés dans le même sens. Comme l'illustre la Figure 41, où l'on peut voir que le nœud le plus proche de la passerelle considère son lien avec la passerelle comme un lien montant alors qu'elle le considère également comme un lien montant. Le principe du réseau, qui consiste à n'être lié à un autre nœud que dans un sens était donc violé.

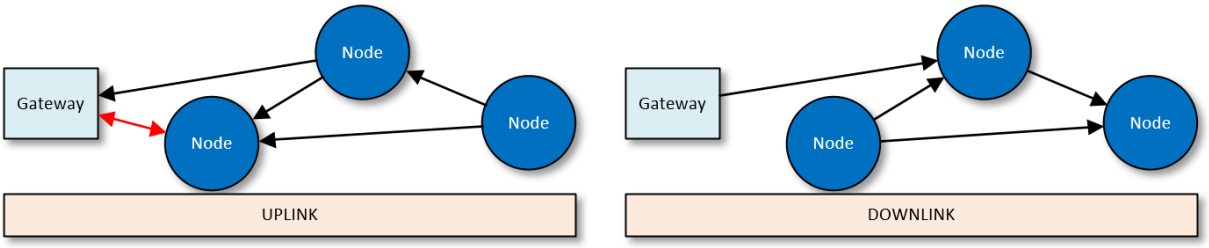



**Figure 41 : Principe des liens entre nœuds avec le bug du paquet perdu**

Pour mieux comprendre ce problème, se référer aux diagrammes de séquences « Découverte réseau » et observer l'annexe [5].

L'attribut « confirmed » nous donne une double sécurité dans le processus de découverte : à la réception d'un « Ping », destiné à tout le monde (broadcasted Ping), les nœuds réagissent maintenant ainsi :

- Enregistrement du nœud source du « Ping » dans la liste uplink
- Envoi d'un paquet de type « Pong » au nœud source du « Ping »

Le nœud supérieur, source du « Ping » enregistre le nœud dans sa liste descendante à la réception de ce « Pong ». Il renvoie alors un paquet de type « Ping », mais cette fois-ci à l'adresse spécifique du nœud en question qui agit alors comme suit :

- Confirmation du nœud source du « Ping » dans la liste montante
- Envoi d'un paquet « Pong » au nœud source du Ping pour être confirmé dans sa liste descendante

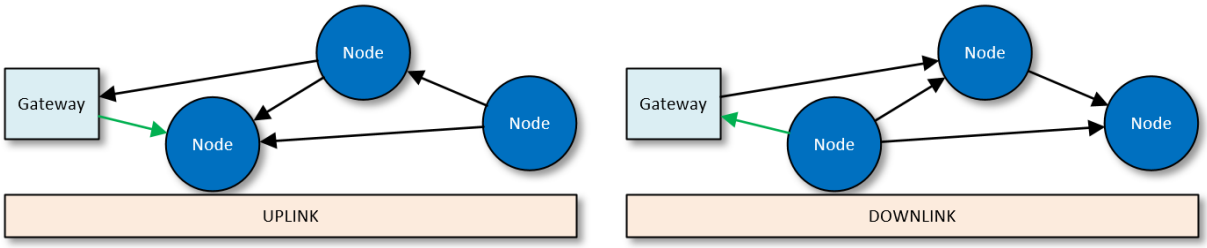



**Figure 42 : Principe des liens entre nœuds avec la résolution du bug du paquet perdu**

Cette nouvelle méthodologie n'évite certes pas les boucles, mais elle règle le problème du lien monodirectionnel qu'elles créaient. Toutefois, étant donné le nombre de messages à transmettre pour créer et confirmer un lien, il faut garder à l'esprit qu'elle engendre une augmentation significative du trafic lors de la découverte du réseau.

### 7.3.4.3 *Expédition de paquets*

Les nœuds peuvent envoyer des paquets montants en direction de la passerelle à tout moment. Il leur suffit de remplir le champ « type » avec la valeur LORANET_TYPE_UP. Chaque nœud utilise la liste d'enregistrement des derniers messages envoyés (uplink_log) pour éviter la redondance sur un même lien. Pour illustrer ce principe, on a repris le principe des liens de la Figure 40 :

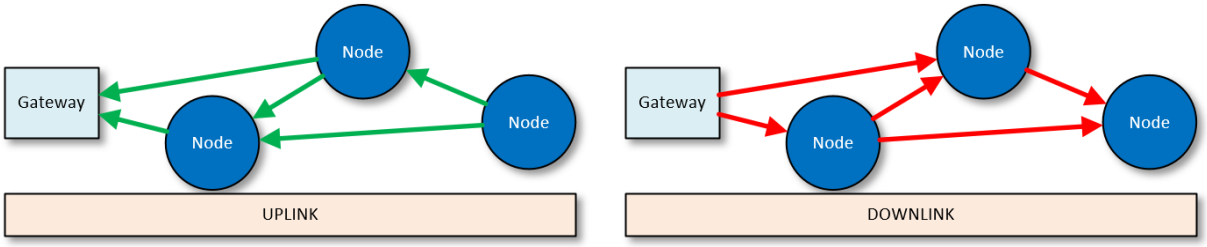



**Figure 43 : Expédition d'un paquet et réponse**

Le premier nœud depuis la passerelle reçoit deux fois le paquet envoyé par le nœud le plus éloigné. Cependant il ne le transmet qu'une fois à la passerelle.

## 7.3.5 Couche logicielle « HCI »

La couche présentée ici gère la sérialisation/désérialisation des trames échangées entre l'hôte et le module RF, conformément au protocole HCI (IMST GmbH, 2011) développé ci-après. Son implémentation est plutôt simple car ce n'est qu'une interface entre le module RF et la couche supérieure.

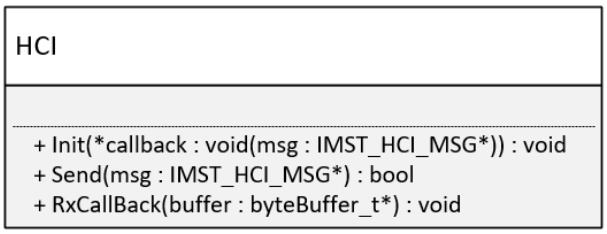



**Figure 44 : Diagramme de classe de la couche « HCI » (méthodes publiques uniquement)**

On peut le voir sur ce diagramme de classe, la méthode « Send » peut être appelée par la couche supérieure pour envoyer des données au module RF à l'aide d'une structure IMST_HCI_MSG passée en paramètre, détaillée au chapitre 7.3.2.3.

### 7.3.5.1 *Protocole HCI*

Ce protocole est basé sur un principe de requête/réponse entre un système hôte et un module de communication externe. Pour exemple l'émission d'un message radio : l'hôte fait une demande de transmission au module, ce dernier tente d'émettre le message et répond ensuite en confirmant l'envoi ou non.

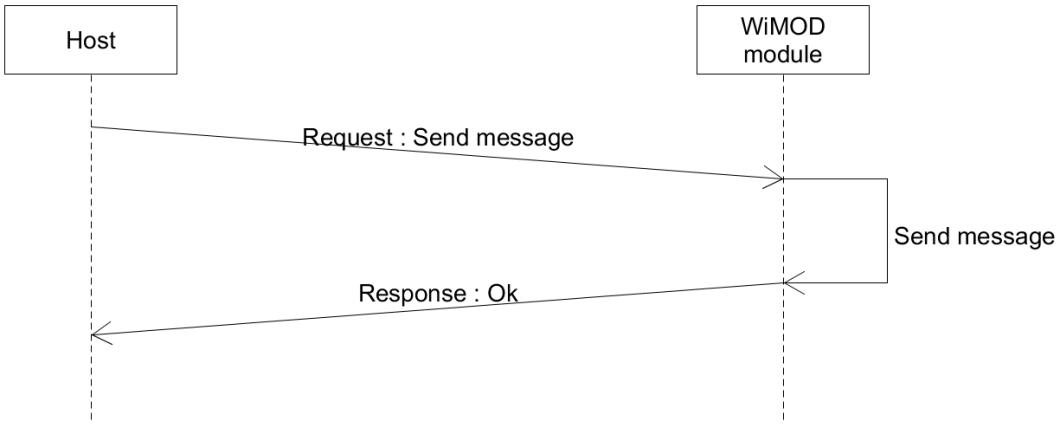



**Figure 45 : Principe du dialogue hôte / module RF pour l'envoi d'un message RF**

Pour être envoyées au module RF, les données doivent respecter la spécification du protocole et doivent être mises en forme selon celle-ci. La Figure 46 montre un message HCI encapsulé dans une enveloppe SLIP, le tout prêt à être transmis via la ligne sérielle. Cette encapsulation permet à la couche inférieure, à l'autre bout de la ligne, de découper le flux de Bytes en trames et inclut également un contrôle d'erreur (ici, FCS : Frame Check Sequence)

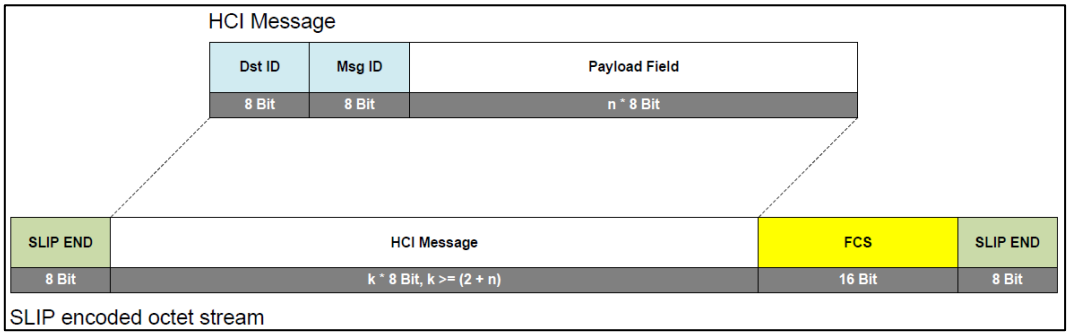



**Figure 46 : Encapsulation d'une trame HCI dans un flux de Bytes**[21]

[21] Crédit : IMST GmbH, tiré de « WiMOD LR Base Host Controller Interface », 2015

#### 7.3.5.1.1 Exemple

Pour bien comprendre comment cette couche fonctionne, la Figure 47 montre la composition d'une commande « Get Radio Configuration », selon la spécification du protocole. La réponse à cette commande est la configuration radio actuelle du module, formattée d'une manière bien définie dans une trame HCI.

3.1.5.1 Get Radio Configuration

This message can be used to read the configuration parameters.

Command Message

| Field | Content | Description |
|---|---|---|
| Endpoint ID | DEVMGMT_ID | Endpoint Identifier |
| Msg ID | DEVMGMT_MSG_GET_RADIO_CONFIG_REQ | Get Radio Config Request |
| Length | 0 | no payload |

**Figure 47 : Extrait de (IMST GmbH, 2011), format des messages "Get Radio Configuration"**

Pour envoyer cette commande au module, il faut donc créer une structure IMST_HCI_MSG, comme vu précédemment, avec les paramètres suivants :

- sapID : DEVMGMT_ID (0x01)
- msgID : DEVMGMT_MSG_GET_RADIO_CONFIG_REQ (0x13)

La couche HCI s'occupe alors de sérialiser la trame et attend la réponse. Une fois celle-ci reçue, elle est transmise à la couche supérieure grâce à la fonction de call-back passée en paramètre à l'initialisation.

### 7.3.6 Couche logicielle « SerialDevice »

Cette couche logicielle est la plus basse du système. Elle peut être considérée comme la couche PHY car elle est responsable de la transmission des trames vers/depuis le périphérique UART. Ses trois principales tâches sont :

- **L'initialisation de la communication UART** : après l'appel à la fonction d'initialisation de la couche, le périphérique UART ainsi que les pins s'y rapportant doivent être opérationnels. Une fonction de call-back est passée en paramètre pour permettre le passage des trames reçues à une des couches supérieures.
- **L'envoi de trames** : une des couches supérieures peut demander l'émission d'une trame. Il faut la décomposer en Bytes et les envoyer un par un au périphérique UART qui s'occupe de la liaison physique avec le module RF.
- **La réception de trames** : le périphérique UART reçoit des Bytes du module RF. La couche développée ici les réceptionne les uns après les autres et les reconditionne sous forme de trames. Elle est ensuite responsable de passer les trames reçues à la couche supérieure, via la fonction de call-back passée à l'initialisation.

La cible sur laquelle sera déployée cette couche impacte beaucoup son implémentation car l'accès au périphérique UART est différent en fonction des systèmes. Cependant, pour garantir une portabilité, le diagramme de classe qui la caractérise reste toujours valide, indépendamment du système. Ce dernier est présenté en Figure 48.

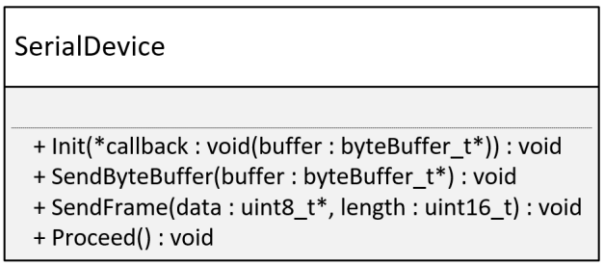



**Figure 48 : Diagramme de classe de la couche « SerialDevice » (méthodes publiques uniquement)**

#### 7.3.6.1.1 Version STM32F100

Sur microcontrôleur STM32, la couche SerialDevice utilise le mécanisme des interruptions pour recevoir les trames Byte par Byte. Cette fonctionnalité permet d'avoir un comportement pseudo-multitâche étant donné qu'au moment où elle intervient, l'interruption prend le contrôle du processeur, s'exécute entièrement et rend ensuite le processeur au programme principal. La première version réalisée fonctionnait sur le principe développé à la Figure 49.

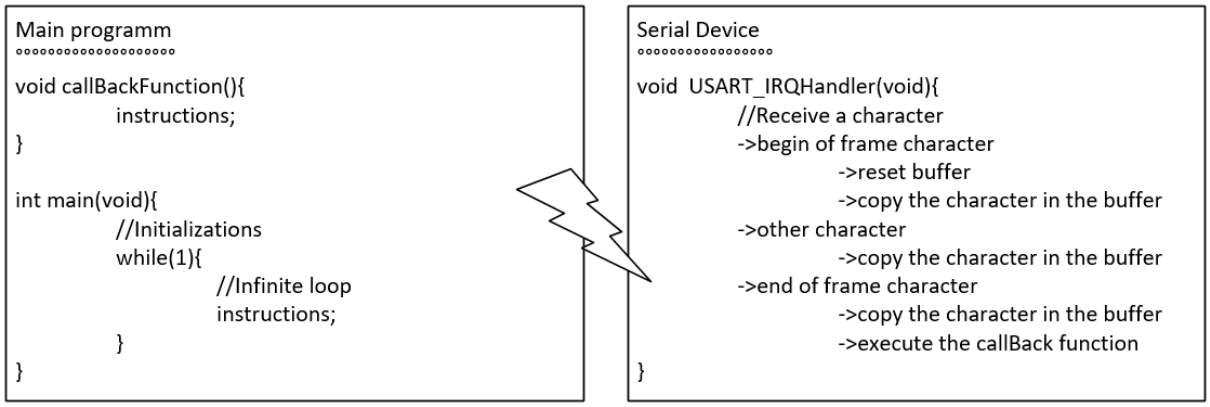



**Figure 49 : SerialDevice : Pseudo-code de la fonction d'interruption (version 1)**

La fonction « UART_IRQHandler » était appelée par la routine d'interruption à chaque réception de caractère par le périphérique UART. Elle n'avait ensuite plus qu'à déterminer si le caractère en question était un caractère spécial de début ou de fin de trame et à agir en conséquence :

- Caractère de début de trame :
  - Réinitialisation du tampon de réception
  - Copie du caractère au début du tampon
- Caractère de fin de trame :
  - Copie du caractère à la suite, dans le tampon
  - Création dynamique d'une structure contenant la trame
  - Appel de la fonction traitant les trames reçues (couche supérieure
- Autre caractère :
  - Copie du caractère à la suite, dans le tampon

Un problème est survenu lors des tests avec cette implémentation : l'appel de la fonction de traitement des trames (« callBack ») peut prendre beaucoup de temps et on peut perdre les Bytes qui arriveraient potentiellement pendant l'exécution de ladite fonction. Une amélioration a donc été apporté pour éliminer ce problème : l'ajout d'une queue de type FIFO, stockant les trames réceptionnées pendant l'interruption. Le programme principal peut ensuite vérifier périodiquement si une trame a été reçue et exécuter lui-même la fonction de traitement des trames, comme illustré à la Figure 50.

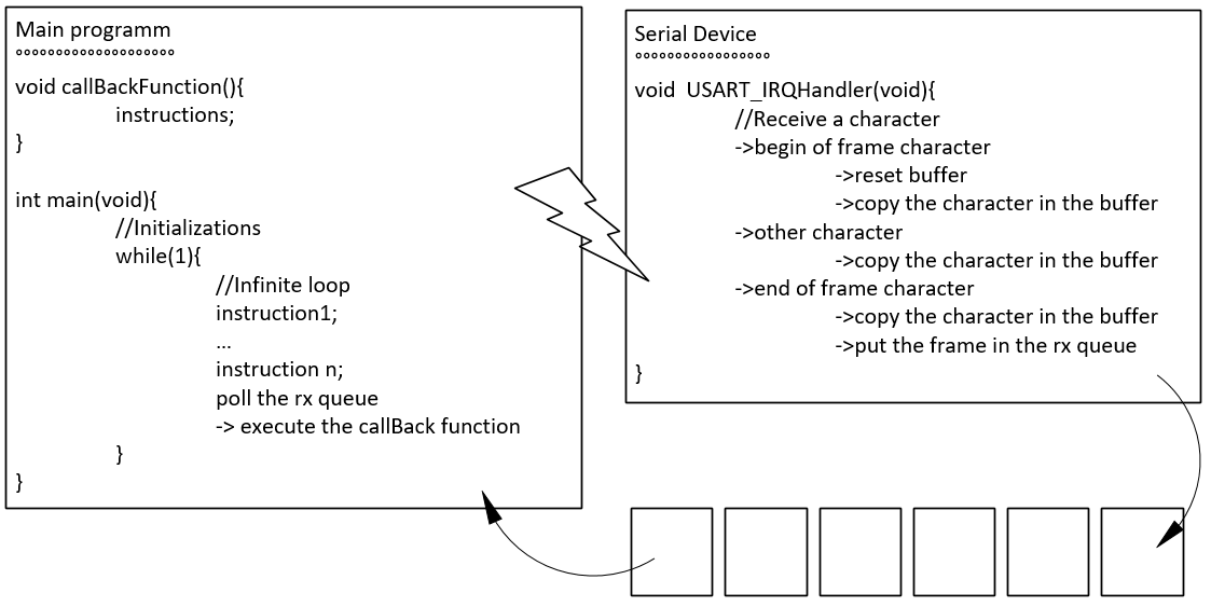



**Figure 50 : SerialDevice : Pseudo-code de ls fonction d'interruption (version 2)**

#### 7.3.6.1.2 Version Raspberry Pi

Cette version de la couche « SerialDevice » utilise la librairie WiringPi (*http://wiringpi.com*) pour gérer les entrées/sorties et permettre une programmation multi-tâche, en utilisant différents processus. Son fonctionnement est présenté en Figure 51.

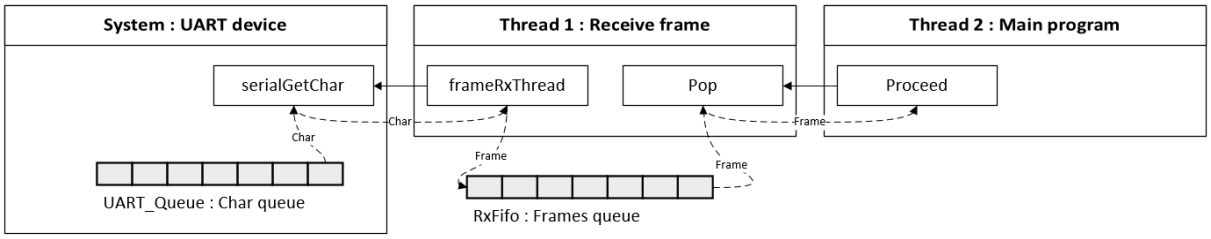



**Figure 51 : SerialDevice : Threads de réception du Raspberry Pi**

On voit que le processus appelé ici « Receive frame » est responsable de récupérer les caractères les uns après les autres depuis le système d'exploitation, en appelant la méthode « serialGetChar ». Il construit ensuite les trames grâce à l'enveloppe SLIP et les pousse dans une queue de type FIFO. Le programme principal, s'exécutant dans un autre processus, n'a plus qu'à venir récupérer les trames dans la queue.

## 7.4 Augmentation du nombre de nœuds

L'augmentation du nombre de nœuds pose un réel problème de complexité du réseau. En effet, si tous les nœuds doivent réexpédier toutes les données de leurs nœuds voisins, le trafic augmente de manière incontrôlée. Le graphique de la Figure 52 montre cette augmentation théorique, en mettant en relation le nombre de nœuds à portée de la passerelle et le nombre de messages nécessaires pour transmettre une seule donnée.

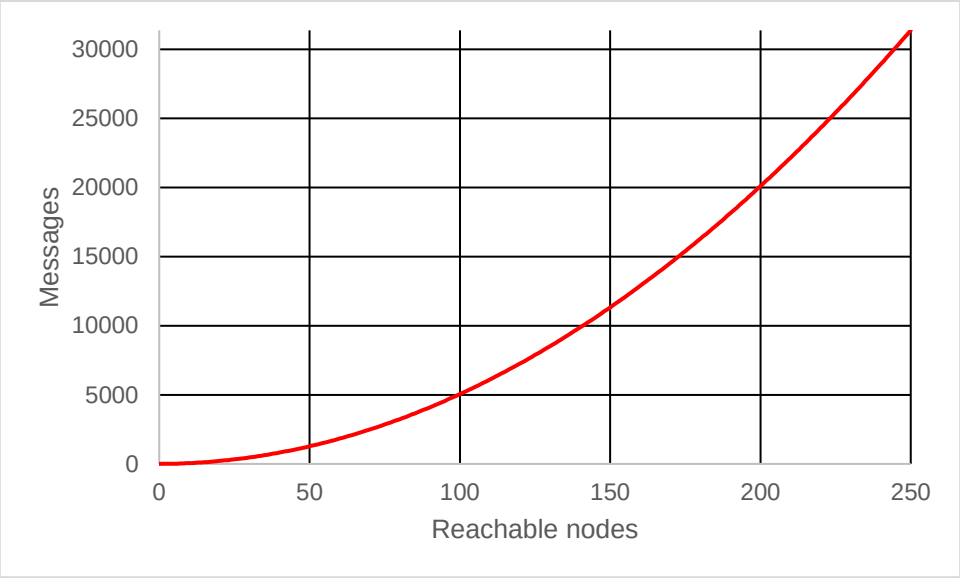



**Figure 52 : Nombre de messages échangés pour un nombre de nœuds à portée**

On voit que le nombre de messages échangés devient alors vite inacceptable. Des algorithmes d'optimisation doivent donc être utilisés. La solution la plus évidente serait que chaque nœud arrive à déterminer le chemin le plus efficace pour atteindre un certain nœud de destination. Ce problème n'est pas traité dans ce travail de diplôme. Cependant, une piste pour arriver à déterminer le meilleur chemin réside dans la possibilité de connaître le RSSI attaché à chaque message reçu, en utilisant le format de trame étendue proposé par les modules iM880. Il faudrait que les nœuds mettent en relation cette donnée et éliminent les liens où le signal passe moins bien, comme illustré ci-dessous. Ceci demanderait une refonte complète de la gestion du réseau.

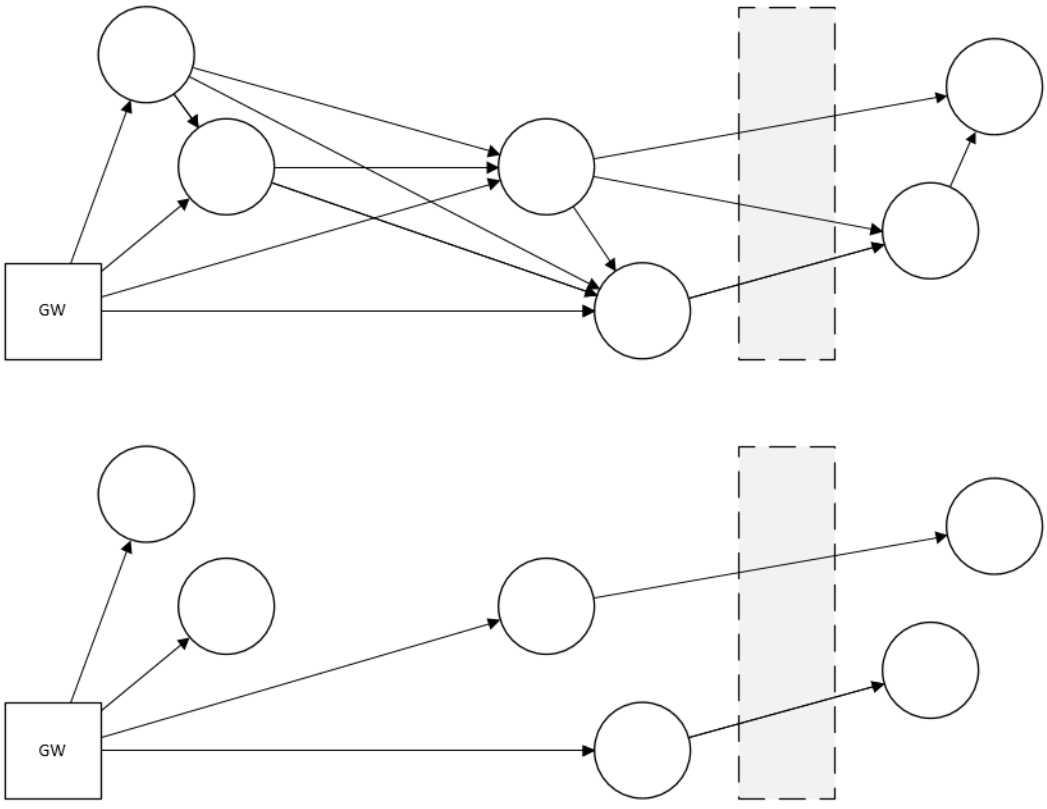



**Figure 53 : Optimisation des liens en flux descendant**

# 8 Tests du protocole

Quantité de tests intermédiaires ont été effectués en cours de développement. Leur description ainsi que leur implémentation sont disponibles dans le fichier « *test.c* ». Les tests développés ici ne concernent que la phase finale du projet, dans laquelle les fonctionnalités ont été vérifiées en mode boîte noire (black-box tests). Pour les réaliser, on a utilisé trois nœuds, la passerelle et un écran, connecté à cette dernière.

*Dans ce chapitre, on entend par :

- La passerelle : le couplage du Raspberry Pi et du module iM880, branché à un écran
- Le nœud : le couplage de la carte STM32VLDISCOVERY et du module iM880.
- Full Debug : mode permettant de visualiser les trames envoyées/reçues par la passerelle sur un écran (en décommentant « __DEBUG_SERIAL__ » dans le fichier « _conf.h »).

## 8.1 Etablissement du réseau à un nœud

### 8.1.1 Situation

Seul un nœud, programmé avec l'identifiant 4, et la passerelle sont alimentés. Ils sont placés à une distance de deux mètres l'un de l'autre et aucun obstacle n'obstrue la transmission.

### 8.1.2 Déroulement

- Lorsque la passerelle est mise sous tension, elle s'initialise et configure son module RF pour avoir l'adresse physique 0x1234.
- On appuie sur le bouton de la carte WiMOD Demo-Board de la passerelle. Elle envoie alors un paquet de type « Ping », à l'adresse de broadcast 0xFFFF.
- Le nœud répond par un paquet de type « Pong », destiné à l'adresse physique de la passerelle.
- Celle-ci enregistre le nœud dans sa liste descendante et renvoie un paquet de type « Ping », à l'adresse physique du nœud.
- Le nœud confirme le lien en envoyant un paquet de type « Pong ».

### 8.1.3 Résultats

Tout se déroule comme prévu, la capture d'écran de la passerelle nous montre les messages échangés à son niveau. On peut y visualiser le processus d'initialisation de celle-ci, puis le processus d'enregistrement/confirmation concernant le nœud 4.

```
pi@raspberrypi:~/Desktop/LoRaGW $ sudo ./LoRaGateway
/////////////// BEGIN ///////////////
[UART opened]
UART configured....
[SERIALDEVICE INIT done]
[HCI INIT done]
Sended :        c0 01 13 85 34 c0
Received :      c0 01 14 00 00 10 10 34 12 bb bb 00 00 20 d9 00 07 01 11 02 01 b8 0b 07 08 02 00 a6 ff b1 82 c0
[LORANET INIT done]
Sended :        c0 01 11 00 00 10 10 34 12 bb bb 00 00 20 d9 00 07 01 11 02 01 b8 0b 07 08 02 00 a6 ff c8 91 c0
[RADIO CONF done]
Received :      c0 01 12 00 31 3a c0
Sended :        c0 01 13 85 34 c0
Received :      c0 01 14 00 00 10 10 34 12 bb bb 00 00 20 d9 00 07 01 11 02 01 b8 0b 07 08 02 00 a6 ff b1 82 c0

Sended :        c0 03 01 ff ff ff 03 ff 01 01 e5 8b c0
Received :      c0 03 02 00 18 1a c0
Received :      c0 03 04 00 10 34 12 10 67 45 04 01 04 01 61 e6 c0
register node 4 in downlink
Sended :        c0 03 01 ff ff ff 03 04 01 02 ec 1c c0
Received :      c0 03 02 00 18 1a c0
Received :      c0 03 04 00 10 34 12 10 67 45 04 01 04 02 fa d4 c0
confirm node 4
Received :      c0 03 04 00 ff ff ff 10 67 45 03 ff 04 03 de d5 c0
```

## 8.2 Etablissement du réseau à deux nœuds

### 8.2.1 Situation

Deux nœuds (Id = 3 et Id = 4) et la passerelle sont alimentés et placés dans une configuration semblable au test précédent.

### 8.2.2 Déroulement

Le déroulement est le même qu'au test précédent, à la différence qu'il y a deux nœuds à portée de la passerelle, ce qui peut engendrer des perturbations. La Figure 54 nous montre le résultat attendu après la phase d'établissement du réseau.

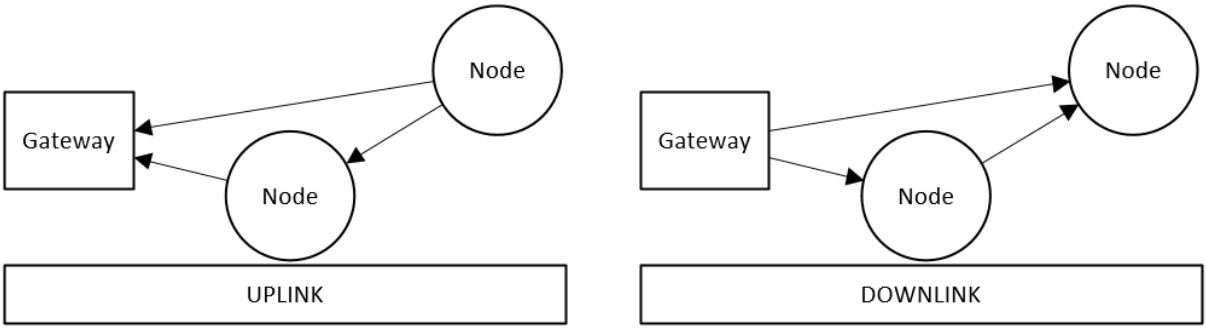



**Figure 54 : Etablissement du réseau à deux nœuds (résultat escompté)**

### 8.2.3 Résultats

L'augmentation du nombre de nœuds pose sporadiquement un problème. Les modules ne semblent pas pratiquer correctement le LBT, malgré de nombreuses tentatives d'ajustement du paramètre du seuil du LBT.

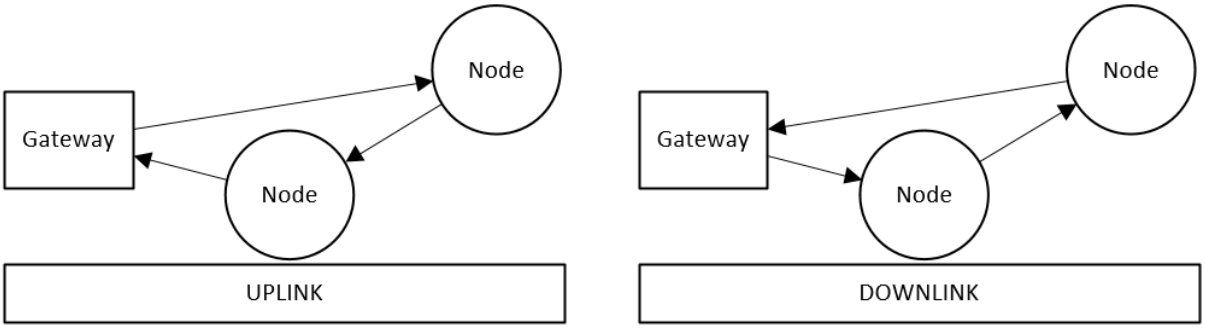



**Figure 55 : Etablissement du réseau à deux nœuds (résultat obtenu)**

On voit bien ici que le résultat obtenu au niveau des liens entre nœuds ne correspond pas aux attentes que l'on aurait pu avoir. En effet, la passerelle a enregistré le nœud le plus éloigné dans sa liste montante. Cependant, étant donné que le protocole garantit des liens bidirectionnels, n'importe quel nœud peut encore être atteint dans les deux sens. Par exemple, pour le cas du nœud le plus éloigné, si la passerelle veut envoyer des données dans le sens descendant, elles transitent simplement par le nœud central.

## 8.3 Etablissement du réseau à trois nœuds

### 8.3.1 Situation

Les trois nœuds (Id = 2, Id = 3 et Id = 4) et la passerelle sont alimentés et à deux mètres les uns des autres. Cependant, leur application a été construite de manière à ce qu'ils simulent la situation illustrée à la Figure 56, grâce à un filtre logiciel :

- Le nœud 4 ne reçoit que les données provenant du nœud 3
- Le nœud 3 n'applique aucun filtre
- Le nœud 2 ne reçoit que les données provenant du nœud 3 et de la passerelle.
- La passerelle ne reçoit que les données des nœuds 2 et 3

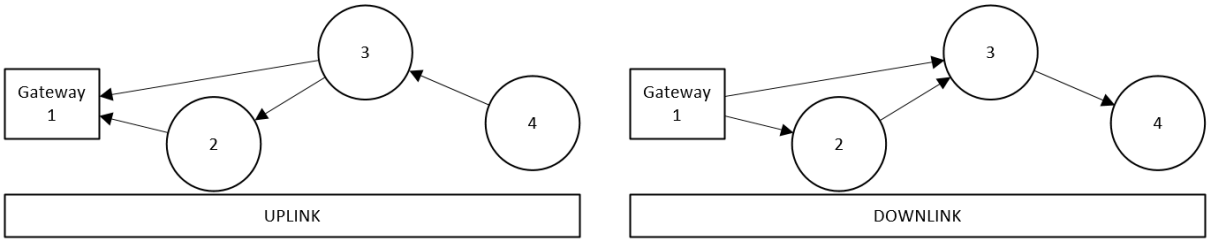



**Figure 56 : Configuration préprogrammée pour la transmission de donnée via deux autres nœuds**

### 8.3.2 Déroulement

- La passerelle est programmée pour envoyer un paquet de type « Ping » chaque 15 secondes
- Dès qu'un nœud possède un lien confirmé dans le sens montant, il envoie la mesure de son potentiomètre chaque deux secondes.
- On visualise la réception des paquets et les connexions sur l'écran de la passerelle

### 8.3.3 Résultats

Le nœud 3 se connecte et confirme le lien avec la passerelle directement après l'envoi du premier paquet « Ping ». Il envoie ensuite un « Ping » en broadcast et le nœud 4 s'y connecte et confirme le lien.

Ces deux nœuds envoient alors des données chaque deux secondes. La passerelle enregistre le nœud 4 à l'adresse physique du nœud 3 dès la réception d'une donnée provenant de celui-ci et relayée par le nœud 3.

Au prochain « Ping » envoyé par la passerelle, le nœud 2 parvient enfin à se connecter et à confirmer le lien et des données peuvent alors transiter par lui. Etant donné ce lien, il peut envoyer un « Ping » à l'adresse de broadcast. Le nœud 3 parvient à s'y connecter et confirme le lien.

Lorsque les premières données provenant du nœud 3 ou 4 transitent par le nœud 2, la passerelle enregistre ces derniers à cette adresse physique. La passerelle connaît donc maintenant les nœuds selon le Tableau 8

| Nœud | Adresse physique | Adresse virtuelle, vue de la passerelle |
|---|---|---|
| **2** | 0x 2345 | 0x 2345 |
| **3** | 0x 3456 | 0x 2345 |
| **3** | 0x 3456 | 0x 3456 |
| **4** | 0x 4567 | 0x 2345 |
| **4** | 0x 4567 | 0x 3456 |

**Tableau 8 : Etablissement du réseau avec trois nœuds : vue de la passerelle**

Le fait que la passerelle connaisse tous les nœuds implique qu'elle peut leur transmettre des données descendantes. Par exemple, pour atteindre le nœud 4, elle sait qu'elle doit passer par l'adresse physique des nœuds 2 ou 3. Cette situation correspond cette fois-ci bien à nos attentes.

## 8.4 Test du protocole en situation réelle

### 8.4.1 Situation

Pour ce test, les nœuds sont programmés de la même façon qu'au test précédent. Le but est de voir si on peut obtenir les mêmes résultats avec une échelle plus grande. Les nœuds sont donc déployés de la manière suivante :

- La passerelle est placée à la fenêtre du troisième étage du bâtiment principal du site de Sion de l'HES-SO Valais/Wallis, salle A304.
- Le nœud 2 est placé au rez-de-chaussée de ce même bâtiment. Suite à des tests effectués au préalable, on sait qu'il est encore à portée de la passerelle.
- Le nœud 3, qui ne pratique aucun filtre logiciel, est branché sur batterie à la vue de la passerelle, à quelques mètres du bâtiment, au rez-de-chaussée. On s'attend donc à ce qu'il parvienne à se connecter facilement.
- Le nœud 4 est placé dans le bâtiment adjacent, dans une salle d'où le nœud 3 est à portée.

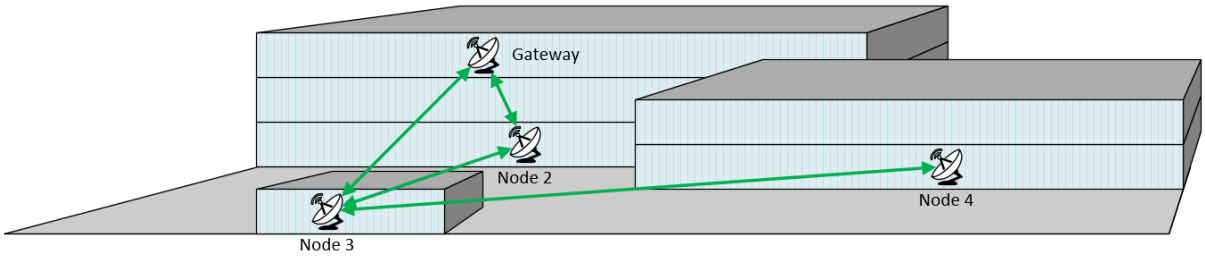



**Figure 57 : Test en situation réelle : plan de situation**

### 8.4.2 Résultats

Dès que la passerelle envoie le premier message de type « Ping », le nœud 2 se connecte sans difficulté et commence à transmettre ses données. Le nœud 3 parvient ensuite à se connecter et les données du nœud 4 nous arrive quelques instants plus tard, ce qui signifie que la connexion entre le nœud 3 et le nœud 4 est faite.

Il faut cependant noter qu'il a fallu s'y reprendre à deux fois pour obtenir de pareils résultats. En effet, au premier essai, le nœud 3 n'était pas à portée de la passerelle. Le problème s'est résolu en éloignant d'un mètre la passerelle de la fenêtre.

## 8.5 Synthèse des résultats

Ces tests nous ont montré que le protocole était fonctionnel. Cependant, on voit bien qu'il est encore instable dans le sens où un paquet perdu lors de la phase de découverte du réseau peut empêcher les nœuds de se connecter les uns aux autres de la bonne manière. Les données se transmettent toutefois d'un nœud à l'autre comme on s'y attendait et le dernier test a prouvé que le protocole était adapté à une situation réelle, où l'on a besoin de transmettre des informations d'un bâtiment à un autre, par exemple.

# 9 Conclusion

## 9.1 Avenir des LPWAN

Les différentes technologies de communication dans le domaine des communications à longue portée M2M étudiées se font actuellement concurrence et on ne peut pas véritablement dire qu'une d'elles a pris le dessus sur les autres à l'heure actuelle. Cependant, ce travail m'a permis de mettre en lumière le fait que les opérateurs téléphoniques déjà en place ont une puissance de frappe financière et logistique tellement importante qu'ils seront à coup sûr des acteurs incontournables dans le déploiement de ces technologies.

Pour ce qui est des solutions qui permettent une modification de la topologie et donc une réexpédition des paquets, le déploiement imminent de la solution « Symphony Link » de la société LinkLabs en Europe sera une avancé dans ce sens. Aucune date n'est encore réellement communiquée mais il est certain que ce déploiement sera effectif dans une poignée d'années. Leur expérience avec cette solution en Amérique centrale leur procure une avance considérable sur leurs potentiels concurrents à ce niveau.

## 9.2 Solution proposée

Ce travail, basé sur la technologie LoRa, a mis en lumière le fait que celle-ci est adaptée pour les applications en intérieur ou dans un rayon d'un ou deux kilomètres dans une ville avec un environnement extérieur perturbé. En effet, malgré les chiffres avancés par les fabricants, on a vu que la transmission à plus longue portée peut difficilement être assurée dès que le signal rencontre des perturbations importantes sur son chemin.

Le protocole développé lors de ce travail remplit les objectifs fixés au départ. En effet, il crée un réseau maillé à plusieurs nœuds, permettant à chacun de réexpédier des paquets vers ses voisins. Cependant, les tests effectués ont démontré des faiblesses au niveau de la fiabilité de la transmission. Cela est essentiellement dû aux modules utilisés pour la transmission. En effet, ils sont conçus au départ pour ne transmettre que sur une fréquence porteuse fixe. On pourrait imaginer un système semblable avec des modules LoRa permettant des transmissions sur plusieurs fréquences simultanément, ce qui démultiplierait les possibilités d'améliorations futures et permettrait d'obtenir une fiabilité plus importante.

On peut dire que dans sa version actuelle, le protocole est encore au stade de prototype et ne peut pas être utilisé dans des applications critiques. En revanche, ce travail permet de sentir le potentiel quant à l'utilisation d'une technologie modifiant la topologie des réseaux longue portée. En effet, puisque les nœuds sont capables de retransmettre les données des nœuds qui ne sont pas directement à portée de la passerelle, ces réseaux voient leur portée s'étendre à chaque fois qu'un nouveau nœud y est ajouté. On peut imaginer par exemple implanter ce protocole dans un système d'éclairage pour une grande ville. Chaque quartier possèderait un nœud LoRa qui retransmettrait les commandes d'enclenchement ou de déclenchement aux quartiers voisins, hors de la portée de la station de base.

# 10 Références

IMST GmbH. (2011). WiMOD LR Base Host Controller Interface v.1.8. Kamp-Lintfort, 47475, Germany.

IMST GmbH. (2013). AN010 WiMOD - iM880A v.0.6 - Software development for iM880A. Kamp-Lintfort, 47475, Germany.

Semtech Corporation. (2015, mai 2). AN1200.22 LoRa Modulation Basics. Camarillo, CA 93012, USA.

STMMicroelectronics. (2011). UM0919 User Manual - STM32VLDISCOVERY.

# 11 Annexes

[1] Déclaration « Back Deployment of LTE-M for Internet of Things », AT&T, KDDI, KPN, NTT DOCOMO, Orange, Telefonica, Telstra, TELUS and Verizon

[2] Fichier « main.c » utilisé sur le Raspberry Pi, pour les tests de portée LoRa

[3] Fichier de définition des pins « WiringPi : GPIO Pin Numbering Tables », wiringpi.com

[4] Diagrammes de séquences

[5] Diagramme de flux pour la réception de paquet dans la couche LoRaNet

[6] Makefile RaspberryPi

[7] Code source de la solution

# AT&T, KDDI, KPN, NTT DOCOMO, Orange, Telefonica, Telstra, TELUS and Verizon Back Deployment of LTE-M for Internet of Things

*LTE-M : The evolution of LTE for IoT is expected to bring the very latest technology to IoT devices and applications across the globe*

**Barcelona, 0900hrs CET, 27 February 2017:** AT&T (US and Mexico), KPN (Netherlands), KDDI (Japan), NTT DOCOMO (Japan), Orange (Europe, Middle East and Africa), Telefonica (Europe), Telstra (Australia), TELUS (Canada) and Verizon (US) confirmed support for the global deployment of LTE-M at the Mobile World Congress in Barcelona.

These operators are working to ensure that LTE-M supports roaming and standards-based local service delivery so that both enterprise and customer-oriented IoT objects, such as trackers or wearables, can be designed for worldwide markets.

The supporting operators are engaging in several activities including pilots, IoT Open Labs and launches of starter kits to support and accelerate the ecosystem of modules and objects.

**AT&T**
AT&T switched on North America's first LTE-M enabled commercial site in October 2016 and plans nationwide U.S. deployment of its LTE-M network ahead of schedule in the second quarter of 2017 and in Mexico by the end of the year. The roll out will ultimately support a North American footprint covering 400 million people in the U.S. and Mexico. LTE-M will put AT&T on the path toward 5G with enhanced features such as low-power, longer battery life, smaller modules and better coverage underground and deep inside buildings.
http://soc.att.com/2kPJot5

**KDDI**
KDDI plans to introduce this technology in its fiscal year 2017. KDDI leads the Japanese IoT market, serving nine of the 10 utilities in Japan using its nationwide LTE network with advanced metering infrastructure services. KDDI will evolve its LTE network with this technology. KDDI, aiming to transform into a “Life Design Company” in all business fields, will provide diverse and beneficial products and services boosted by IoT for the different stages of our customers’ lives.
http://www.kddi.com/english/

**KPN**
As the first operator in Europe, KPN successfully trialled the new Internet of Things (IoT) technology LTE-M in November 2016 (also known as LTE Cat-M1). The LTE-M technology is complementary to the recently introduced LoRaWAN network of KPN as well as to existing Machine-to-Machine (M2M) use cases on 4G. KPN is planning a nationwide LTE-M roll-out by the end of 2017.
https://www.kpn.com/

**NTT DOCOMO**
NTT DOCOMO provides innovative, convenient and secure mobile services that enable smarter living for each customer and is a leading developer of a 5G network that it plans to deploy by 2020. DOCOMO has already deployed various IoT services via its cellular network and soon will support eDRX for longer battery life. DOCOMO is committed to further enhancement of IoT services by launching LTE-based IoT technologies such as LTE-M. www.nttdocomo.co.jp/english.

**Orange**
Orange has primarily chosen LTE-M to be progressively deployed on its 4G networks in Europe, starting with Belgium and Spain this year before the rest of the Group's European footprint. Orange is also launching Europe's first LTE-M IoT Open Lab in France in April. Orange is providing IoT Starter Kits to help IoT developers accelerate their discovery of LTE-M and build a functional prototype faster. The Group has also announced two pilots in 2017 – one will be run in the smart metering domain to connect LTE-M-based electric meters to remotely control power consumption and adapt user subscriptions. A second pilot will test LTE-M-enabled wearable devices that can measure an individual's movement, temperature and other health-related information.
www.orange.com

**Telefónica**
Telefónica is fostering all GSMA LPWA technologies in line with its promise to offer the best IoT connectivity. In the space of LTE-M, on 14th of February 2017, Telefónica completed the first live LTE Cat M1 data call in Europe involving major players of the IOT ecosystem. From 2017 2nd quarter onwards, according also to devices availability, Telefónica in Europe will be ready to support the first customer experiences using the new protocol. Telefónica is running a GSMA LTE-M Open Lab in Madrid (https://www.5tonic.org/) and has plans to connect it with its entrepreneurs ecosystem accelerator Telefónica Open Future (https://www.openfuture.org/en) to give access to the LTE-M network technology, devices and support from experts. During MWC Telefónica will showcase in Barcelona the first live LTE-M network demonstrating a variety of use cases (waste management, electricity metering and wearables) which leverage on the advanced capabilities of LTE-M.
https://iot.telefonica.com/

**Telstra**
Cat M1 can enhance LTE coverage for underground and in-building areas that challenge existing coverage. Combined with Telstra's existing leading LTE coverage, customers can deploy a range of near real time applications in logistics, utilities, medicine, transport, mining, agriculture, manufacturing and many more. As Cat M1 devices and solutions become commercially available, Telstra is set to support their operation across its expansive 4G network which covers over 98% of the Australian population.
www.telstra.com

**TELUS**
TELUS is actively involved in developing 5G wireless technology, and is committed to all the foundational pillars of 5G: Enhanced Mobile Broadband (i.e. HetNet, PicoCell, C-RAN, 30 Gbps wireless speeds), Reliability & Latency, and IoT, including Low Power Wide Area (LPWA) Network Technologies. As a result, TELUS is committed to a standards-based deployment of LPWA LTE-M with pilot customers in the second half of 2017. In 2016, TELUS successfully demonstrated a Smart Parking application as part of its LPWA plan.
www.TELUS.com

**Verizon**
Verizon became the first carrier in the world to deploy LTE Cat M1 commercially, launched in December 2016. Verizon will complete nationwide U.S. coverage of LTE Cat M1 by the end of Q1 2017. The carrier works with industry-leading partners, and in 2016, certified the world's first Cat M1 chipset and module.
www.verizon.com

**Addressing LPWA connectivity requirements of key verticals**
LTE-M is part of Mobile IoT solutions based on 3GPP standard release 13 released in June 2016. It introduces a new Category M1 device optimised for the IoT able to address the LPWA connectivity requirements of key verticals.

Thanks to the easy roll-out capability of LTE-M technology (via software upgrade of existing 4G LTE networks), customers will benefit from a coverage in these respective countries at the time of the launch of LTE-M in respective markets.

LTE-M technology will connect, in a secure and scalable way, a wide variety of IoT devices/ objects such as smart utility meters, asset monitoring trackers, vending machines, alarm systems, fleet of vehicles, heavy equipment, mHealth, oil and gas monitoring and control, agriculture and wearables.

Key features expected from LTE-M are:

- Cat M1 Chipset / module optimised for IoT with a target reduction of complexity and cost over existing LTE Cat 4/1 modules
- Longer battery life; up to 10 years for certain enabled IoT devices when using additional features such as Power Saving Mode, eDRX feature
- Extended coverage compared to LTE for IoT devices underground and deep inside buildings
- Support of Mobility and Voice services enabling the use for wearables and safety related object

- ENDS -


**For more information, contact:**

**AT&T:** Jessica Swain, jessica.swain@att.com, +1 415 613 4267
**KDDI:** Yukiko Habu, yu-habu@kddi.com, +81 3 6678 0690
**KPN:** Stijn Wesselink, stijn.wesselink@kpn.com, +3170 44 66 300
**NTT DOCOMO:** Aya Hokamura, press_event@nttdocomo.com, +81 3 5156 1366
**Orange:** Vanessa Clarke, vanessa.clarke@orange.com, +44 7818 848 848
**Telefónica:** Corporate Communications Department, prensatelefonica@telefonica.com +34 91 482 38 00
**Telstra:** Jon Court, jon.court@team.telstra.com, +61408423516
**TELUS:** Richard Gilhooley, richard.gilhooley@telus.com, +1 (778) 868 0235
**Verizon:** Marc Tracey, marc.tracey@verizon.com, +1 908 307 8378


About AT&T

AT&T Inc. (NYSE:T) helps millions around the globe connect with leading entertainment, business, mobile and high speed internet services. We offer the nation's best data network* and the best global coverage of any U.S. wireless provider.** We're one of the world's largest providers of pay TV. We have TV customers in the U.S. and 11 Latin American countries. Nearly 3.5 million companies, from small to large businesses around the globe, turn to AT&T for our highly secure smart solutions. Additional information about AT&T products and services is available at http://about.att.com. Follow our news on Twitter at @ATT, on Facebook at http://www.facebook.com/att and YouTube at http://www.youtube.com/att.  *Claim based on the Nielsen Certified Data Network Score. Score includes data reported by wireless consumers in the Nielsen Mobile Insights survey, network measurements from Nielsen Mobile Performance and Nielsen Drive Test Benchmarks for Q3+Q4 2016 across 121 markets. **Global coverage claim based on offering discounted voice and data roaming; LTE roaming; and voice roaming in more countries than any other U.S. based carrier. International service required. Coverage not available in all areas. Coverage may vary per country and be limited/restricted in some countries.

About KDDI

KDDI, a comprehensive communications company offering fixed-line and mobile communications services, strives to be a leading company for changing times. For individual customers, KDDI offers its mobile communications (mobile phone) and fixed-line communications (broadband Internet/telephone) services under the brand name au, helping to realize Fixed Mobile and Broadcasting Convergence (FMBC). For business clients, KDDI provides comprehensive Information and Communications services, from Fixed Mobile Convergence (FMC) networks to data centers, applications, and security strategies, which help clients strengthen their businesses. For more information please visit http://www.kddi.com/english.

About KPN

KPN is the leading telecommunications and ICT provider in The Netherlands, offering fixed and mobile telephony, fixed and mobile broadband internet and TV to retail consumers. KPN is also market leader in The Netherlands in ICT services, infrastructure and network related ICT solutions to business customers, including other telecommunications operators. KPN also provides wholesale network services to third parties and operates an IP-based infrastructure for international wholesale customers through iBasis.

About NTT DOCOMO

NTT DOCOMO provides innovative, convenient and secure mobile services that enable smarter living for each customer. The company serves over 73 million mobile customers in Japan via advanced wireless networks, including a nationwide LTE network and one of the world's most progressive LTE-Advanced networks. DOCOMO is a leading developer of a 5G network that it plans to deploy by 2020, as well as network function virtualization (NFV), NFC infrastructure and services, emerging IoT solutions, and more. Outside Japan, the company is providing technical and operational expertise to seven mobile operators and other partner companies, and is contributing to the global standardization of all-new mobile technologies. DOCOMO is listed on stock exchanges in Tokyo (9437) and New York (DCM). Please visit https://www.nttdocomo.co.jp/english/.

About Orange

Orange is one of the world's leading telecommunications operators with sales of 40,9 billion euros in 2016 and 155,000 employees worldwide at 31 December 2016, including 96,000 employees in France. Present in 29 countries, the Group has a total customer base of 263 million customers worldwide at 31 December 2016, including 202 million mobile customers and 18 million fixed broadband customers. Orange is also a leading provider of global IT and telecommunication services to multinational companies, under the brand Orange Business Services. In March 2015, the Group presented its new strategic plan "Essentials2020" which places customer experience at the heart of its strategy with the aim of allowing them to benefit fully from the digital universe and the power of its new generation networks. Orange is listed on Euronext Paris (symbol ORA) and on the New York Stock Exchange (symbol ORAN).
For more information on the internet and on your mobile: www.orange.com, www.orange-business.com or to follow us on Twitter: @orangegrouppr.Orange and any other Orange product or service names included in this material are trademarks of Orange or Orange Brand Services Limited.

About Telefónica
Telefónica is one of the largest telecommunications companies in the world by market capitalization and number of customers with a comprehensive offering and quality of connectivity that is delivered over world class fixed, mobile and broadband networks. As a growing company it prides itself on providing a differential experience based both on its corporate values and a public position that defends customer interests. The company has a significant presence in 21 countries and over 349 million accesses around the world. Telefónica has a strong presence in Spain, Europe and Latin America, where the company focuses an important part of its growth strategy. Telefónica IoT is the Internet of Things global business unit at Telefónica, dedicated to developing and implementing IoT solutions in all industry segments. For more information about Telefónica IoT: visit iot.telefonica.com or follow us on twitter @telefonicaIoT or on LinkedIn.

About Telstra
Telstra is a leading telecommunications and information services company. We offer a full range of services and compete in all telecommunications markets in Australia, operating the largest mobile and Wi-Fi networks. Globally, we provide end-to-end solutions including managed network services, global connectivity, cloud, voice, colocation, conferencing and satellite solutions. For more information visit www.telstra.com.

About TELUS
TELUS (TSX: T, NYSE: TU) is Canada's fastest-growing national telecommunications company, with $12.8 billion of annual revenue and 12.7 million subscriber connections, including 8.6 million wireless subscribers, 1.7 million high-speed Internet subscribers, 1.4 million residential network access lines and more than 1.0 million TELUS TV customers. TELUS provides a wide range of communications products and services, including wireless, data, Internet protocol (IP), voice, television, entertainment and video, and is Canada's largest healthcare IT provider. In support of our philosophy to give where we live, TELUS, our team members and retirees have contributed over $482 million to charitable and not-for-profit organizations and volunteered more than 1 million days of service to local communities since 2000. TELUS' 12 Canadian community boards and 5 International boards have led the Company's support of grassroots charities and have contributed more than $60 million in support of 5,595 local charitable projects, enriching the lives of more than 2 million children and youth, annually. TELUS was honoured to be named the most outstanding philanthropic corporation globally for 2010 by the Association of Fundraising Professionals, becoming the first Canadian company to receive this prestigious international recognition. TELUS has been named to the Dow Jones Sustainability Index for the past 16 years, a feat unequalled by any other North American telecommunications company. As detailed in our TELUS Sustainability Report, our commitment to sustainability is inspired by nature to ensure a healthier future for us all. For more information about TELUS, please visit TELUS.com

About Verizon
Verizon Communications Inc. (NYSE, Nasdaq: VZ), headquartered in New York City, has a diverse workforce of 160,900 and generated nearly $126 billion in 2016 revenues. Verizon operates America's most reliable wireless network, with 114.2 million retail connections nationwide. The company also provides communications and entertainment services over mobile broadband and the nation's premier all-fiber network, and delivers integrated business solutions to customers worldwide.

Verizon's Online News Center: News releases, feature stories, executive biographies and media contacts are available at Verizon's online News Center at www.verizon.com/news/. News releases are also available through an RSS feed. To subscribe, visit www.verizon.com/about/rss-feeds/.

```
/**
  * \file        main.c
  * \author      Pierre Mendicino
  * \version 1.2
  * \date        2017.07.10
  * \brief       This file contains the application layer to do the range tests
  *              with the Raspberry PI 3
  */
//*****************************************************************************
//
// Pre-Includes
//
//*****************************************************************************
#include <stdio.h>
#include <stdbool.h>
#include <unistd.h>
#include <fcntl.h>
#include <termios.h>
#include <wiringPi.h>

//*****************************************************************************
//
// Pre-Definitions
//
//*****************************************************************************
//#define __DEBUG__           //Uncomment for console trace
#define __RPI_3__             //Uncomment for other systems
#define __TST_1__             //Uncomment for the test 1

//*****************************************************************************
//
// Includes
//
//*****************************************************************************
#include "util/IMST_HCI.h"
#include "util/crc16.h"
#include "util/byteBuffer.h"
#include "util/time.h"
#include "serial_device.h"
#include "hci.h"
#include "loranet.h"

//*****************************************************************************
//
// Definitions
//
//*****************************************************************************
#define TX_DATA_SIZE 16       //Size of tx_frame_buffer inc.\0
#define BLINK_TIME 150        //Parameter for the led
#define MAX_MSG_NUM 10        //Maximum message number deft.10
////////////////////////////////////////////////////////////////////////////////
// Pin Definition
// ^^^^^^^^^^^^^^
// RPI SK-IM880A DESCRIPT.
// °°° °°°°°°°° °°°°°°°°°
// 02  X8.1      VCC
// 06  X8.2      GND
// 08  X7.20     RPI Tx
// 10  X7.14     RPI Rx
// 12  X6.3      LED D2
// 16  X5.1      BUT B3
// 18  x5.3      DIP 1
// 22  x5.15     DIP 2
//
#define _LED_PIN 18 //GPIO18 -> PIN 12
#define _BUT_PIN 23 //GPIO23 -> PIN 16
#define _DIP_1_PIN   24 //GPIO24 -> PIN 18
#define _DIP_2_PIN   25 //GPIO25 -> PIN 22
////////////////////////////////////////////////////////////////////////////////

//*****************************************************************************
//
// Variables
//
//*****************************************************************************
char _rxBuffer[255];
uint8_t raw_data_to_send[6] = {
        0x13, 0x7F, 0x00,
        0x00, 0xF7, 0x31};

/**
 * \brief    This function is intended to be used when a led is connected to a
 *           certain pin. The thread that execute this function will be blocked
 *           here because of the use of the blocking function "delay(int ms)".
```

```
 * \param    pin :            the pin where the led is attached
 *           occurrences :    how many times the led will blink
 * \retval   void
 */
void blink(const int pin, const int occurrences){
    int tmp = 0;
    while(tmp < occurrences){
        digitalWrite(pin, HIGH);
        delay(BLINK_TIME);
        digitalWrite(pin, LOW);
        delay(BLINK_TIME);
        tmp++;
    }
}

/**
 * \brief    This function will be passed to the lower layer and will be called
 *           when a new message is received.
 * \retval   void
 */
void LORA_RxCallBack(LoRaNet_Packet* packet){
#ifdef __DEBUG__
    printf("Main_Callback_\n");   //Notify that the callback function is called
#endif // __DEBUG__
    //Local variables

    /* Here will come the code of the application.. */

    //Giving back the memory
    free(packet->data.begin);
    free(packet);
}

/**
 * \brief    This function configure quickly the radio module. After calling it,
 *           the function LoRaNet_SendConfig() should be called to apply changes
 * \retval   void
 */
void configureRadioModule(){
    //Configure the command
    LoRaNet_SetRadioMode(IMST_HCI_DEVMGMT_RADIO_MODE_STD);
    LoRaNet_SetGroupAddress(0x10);
    LoRaNet_SetDeviceAddress(0xCCCC);
    LoRaNet_SetModulation(IMST_HCI_DEVMGMT_MODULATION_LORA);
    LoRaNet_SetCarrierFreq(IMST_HCI_DEVMGMT_CARRIER_FREQ_868_5);
    LoRaNet_SetBandwidth(IMST_HCI_DEVMGMT_LORA_BW_125K);
    LoRaNet_SetSpreadingFactor(IMST_HCI_DEVMGMT_LORA_SF_7);
    LoRaNet_SetErrorCoding(IMST_HCI_DEVMGMT_LORA_RC_4_5);
    LoRaNet_SetPowerLevel(17);
    LoRaNet_SetTxControl(
            IMST_HCI_DEVMGMT_TX_CONTROL_LBT//|
            //IMST_HCI_DEVMGMT_TX_CONTROL_NF
            );
    LoRaNet_SetRxControl(IMST_HCI_DEVMGMT_RX_CONTROL_ALWAYS_ON);
    LoRaNet_SetRxWindowTime(3000);//in ms
    LoRaNet_SetLedControl(
            IMST_HCI_DEVMGMT_LED_CONTROL_ALIVE_D4 |
            IMST_HCI_DEVMGMT_LED_CONTROL_RX_D3 |
            IMST_HCI_DEVMGMT_LED_CONTROL_TX_D2
            );
    LoRaNet_SetMiscOpt(
            //IMST_HCI_DEVMGMT_MISC_OPTIONS_NOTHING |
            IMST_HCI_DEVMGMT_MISC_OPTIONS_EXT_RF |
            //IMST_HCI_DEVMGMT_MISC_OPTIONS_RTC |
            //IMST_HCI_DEVMGMT_MISC_OPTIONS_TX_IND |
            IMST_HCI_DEVMGMT_MISC_OPTIONS_POWER_UP_IND//|
            //IMST_HCI_DEVMGMT_MISC_OPTIONS_BUTTON_IND |
            //IMST_HCI_DEVMGMT_MISC_OPTIONS_AES
            );
    LoRaNet_SetFskDataRate(IMST_HCI_DEVMGMT_FSK_DATARATE_250_KBPS);
    LoRaNet_SetLBTThresold(-90);
}

/**
 * \brief    Main function
 * \retval   0 => nothing wrong
 */
int main(void){
#ifdef __DEBUG__
    printf("/////////////// BEGIN ///////////////\n");
#endif // __DEBUG__
    //Local variables
    int msgNum = 1;              //1 to MAX_MSG_NUM (++ at each iteration)
    bool dataToSend = false;     //We need to send some data
```

```
169       bool dataSent = true;             //The data have been sent
170       LoRaNet_Node myNode;          //Our node basic parameters
171       LoRaNet_Node toRegister;       //A node to register
172       myNode.address = 0x1234;      // ->our physical address (the real one)
173       myNode.groupId = 0x10;            // ->our group id (the real one)
174       myNode.nodeId = 0x01;             // ->our id
175       LoRaNet_Packet* packetToSend; //A packet pointer
176       uint8_t loop_int = 0;             //Just an integer usable in some loops
177       bool ledState = false;            //The next state of the led
178
179       //----- INIT -----
180       wiringPiSetupGpio();          //Initialize the WiringPi library
181       pinMode(_LED_PIN, OUTPUT);        //Led pin
182       pinMode(_BUT_PIN, INPUT);     //The button
183       pinMode(_DIP_1_PIN, INPUT);       //SP switch(7 or 12)
184       pinMode(_DIP_2_PIN, INPUT);       //Frequency switch (125 or 500 kHz)
185       digitalWrite(_LED_PIN, LOW);  //Turn off the led
186
187       Time_Init();                  //Initialize the time functions
188       SerialDevice_Init(HCI_RxCallBack);//Initialize the serial device layer
189   #ifdef __DEBUG__
190       printf("\tSERIAL DEVICE INIT done...\n");
191   #endif // __DEBUG__
192       HCI_Init(LoRaNet_RxCallBack); //Initialize the HCI layer
193   #ifdef __DEBUG__
194       printf("\tHCI INIT done...\n");
195   #endif // __DEBUG__
196       LoRaNet_Init(LORA_RxCallBack, &myNode);//Initialize the LoRaNet layer
197   #ifdef __DEBUG__
198       printf("\tLORANET INIT done...\n");
199   #endif // __DEBUG__
200       configureRadioModule();           //Set the radio configuration
201       LoRaNet_SendConfig();             //Apply it
202   #ifdef __DEBUG__
203       printf("\tRADIO CONF done...\n");
204   #endif // __DEBUG__
205
206   #ifdef __TST_1__
207       // ->Register node 4
208       toRegister.address = 0x4567;
209       toRegister.groupId = 0x10;
210       toRegister.nodeId = 0x04;
211       LoRaNet_Register(&toRegister, LORANET_UP);
212   #endif // __TST_1__
213
214       while(true){
215
216           //----- PROCEED -----
217           LoRaNet_Proceed();            //Send a message if there is one
218           SerialDevice_Proceed();       //Receive a message if there is one
219
220           //----- SEND DATAS -----
221           if(dataToSend && !dataSent){
222   #ifdef __TST_1__
223               // ->Prepare the packet
224               packetToSend = malloc(sizeof(LoRaNet_Packet));
225               packetToSend->type = LORANET_TYPE_DATA_UP;
226               packetToSend->dstId = 0x04;
227               packetToSend->srcId = myNode.nodeId;
228               packetToSend->packetId = LoRaNet_getPacketID();
229               packetToSend->data.length = 6;
230               packetToSend->data.begin = malloc(packetToSend->data.length);
231               loop_int = 0;
232               while(loop_int < packetToSend->data.length){
233                   packetToSend->data.begin[loop_int] = raw_data_to_send[loop_int];
234                   if(loop_int == 3){
235                       // ->Tag the message with the msgNum index
236                       packetToSend->data.begin[loop_int] = msgNum;
237                   }
238                   loop_int++;
239               }
240               // ->Push the packet
241               LoRaNet_PushTxQueue(packetToSend);
242               // ->Actualize the variables
243               if(msgNum == MAX_MSG_NUM){
244                   blink(_LED_PIN, 2);
245                   dataSent = true;
246               }
247               msgNum = msgNum % MAX_MSG_NUM + 1;
248   #endif // __TST_1__
249           }
250
251           //----- USE GPIOS -----
252           // ->Button
```

```
253           if(digitalRead(_BUT_PIN)){        //button is released
254               if(dataSent){
255                   dataToSend = false;
256               }
257               dataSent = false;
258           } else {                          //button is pressed
259               blink(_LED_PIN, 1);
260               dataToSend = true;
261           }
262           //
263           // ->SF Switch
264           if(digitalRead(_DIP_1_PIN)){  //switch is off
265               if(LoRaNet_radioConfig.spreadingFactor != 7){
266   #ifdef __DEBUG__
267                   printf("set to SF 7\n");
268   #endif // __DEBUG__
269                   LoRaNet_SetSpreadingFactor(7);
270                   LoRaNet_SendConfig();
271               }
272           } else {                          //switch is on
273               if(LoRaNet_radioConfig.spreadingFactor != 12){
274   #ifdef __DEBUG__
275                   printf("set to SF 12\n");
276   #endif // __DEBUG__
277                   LoRaNet_SetSpreadingFactor(12);
278                   LoRaNet_SendConfig();
279               }
280           }
281           //
282           // ->Bandwidth Switch
283           if(digitalRead(_DIP_2_PIN)){  //switch is off
284               if(LoRaNet_radioConfig.bandwidth != IMST_HCI_DEVMGMT_LORA_BW_125K){
285   #ifdef __DEBUG__
286                   printf("set to 125kHz\n");
287   #endif // __DEBUG__
288                   LoRaNet_SetBandwidth(IMST_HCI_DEVMGMT_LORA_BW_125K);
289                   LoRaNet_SendConfig();
290               }
291           } else {                          //switch is on
292               if(LoRaNet_radioConfig.bandwidth != IMST_HCI_DEVMGMT_LORA_BW_500K){
293   #ifdef __DEBUG__
294                   printf("set to 500kHz\n");
295   #endif // __DEBUG__
296                   LoRaNet_SetBandwidth(IMST_HCI_DEVMGMT_LORA_BW_500K);
297                   LoRaNet_SendConfig();
298               }
299           }
300
301           //----- BLINK LED -----
302           if(ledState){
303               digitalWrite(_LED_PIN, LOW);  //Normally on..
304               bool ledState = false;
305           } else {
306               digitalWrite(_LED_PIN, HIGH);
307               bool ledState = false;
308           }
309       }
310
311       //----- CLOSE THE UART -----
312   #ifdef __DEBUG__
313       printf("///////////////  END  ///////////////\n");
314   #endif // __DEBUG__
315       return 0;
316   }
317
```

**WiringPi: GPIO Pin Numbering Tables**

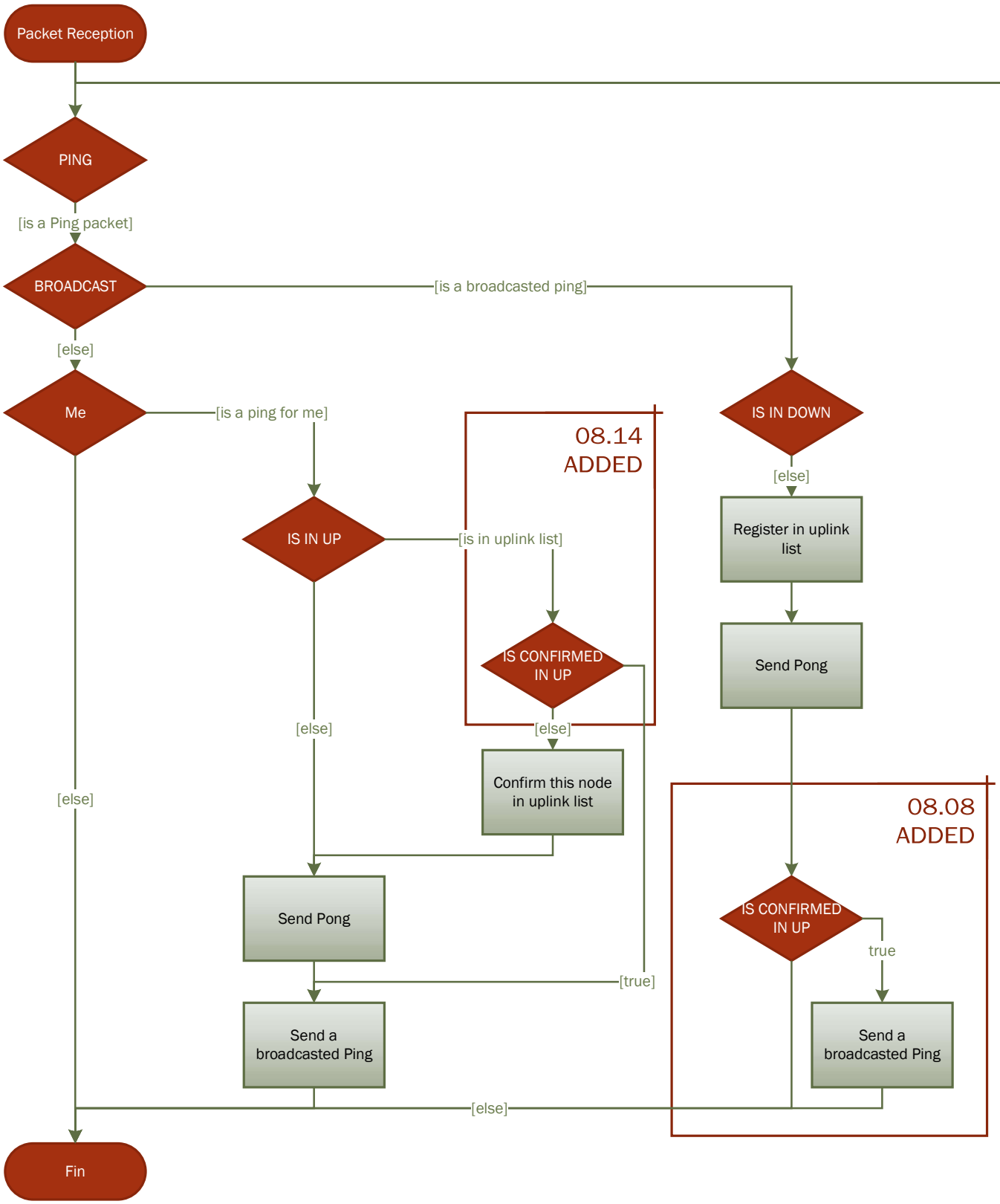
http://wiringpi.com/

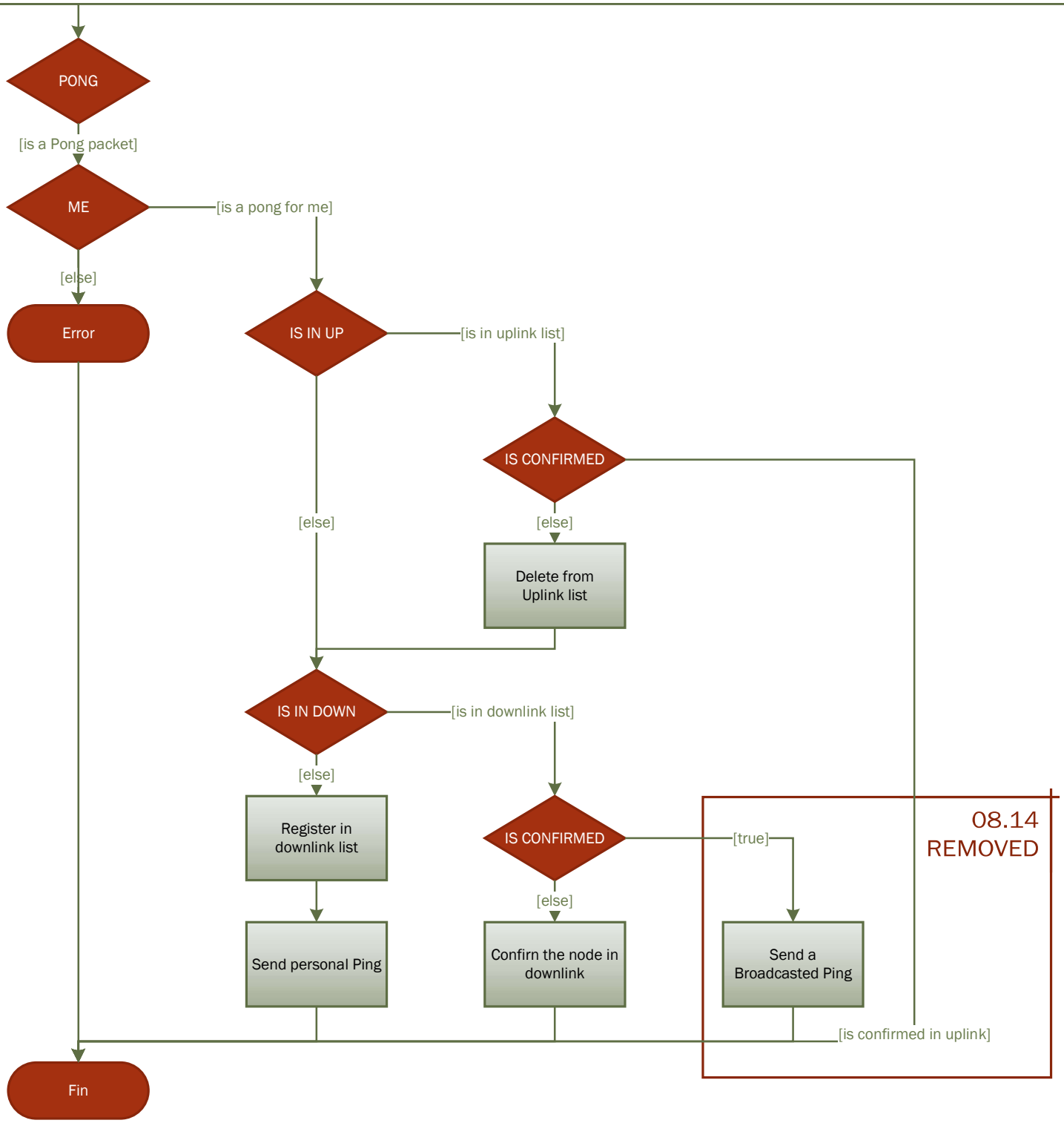
| P1: The Main GPIO connector | | | | | | | |
|---|---|---|---|---|---|---|---|
| WiringPi Pin | BCM GPIO | Name | Header | | Name | BCM GPIO | WiringPi Pin |
| | | 3.3v | 1 | 2 | 5v | | |
| 8 | Rv1:0 - Rv2:2 | SDA | 3 | 4 | 5v | | |
| 9 | Rv1:1 - Rv2:3 | SCL | 5 | 6 | 0v | | |
| 7 | 4 | GPIO7 | 7 | 8 | TxD | 14 | 15 |
| | | 0v | 9 | 10 | RxD | 15 | 16 |
| 0 | 17 | GPIO0 | 11 | 12 | GPIO1 | 18 | 1 |
| 2 | Rv1:21 - Rv2:27 | GPIO2 | 13 | 14 | 0v | | |
| 3 | 22 | GPIO3 | 15 | 16 | GPIO4 | 23 | 4 |
| | | 3.3v | 17 | 18 | GPIO5 | 24 | 5 |
| 12 | 10 | MOSI | 19 | 20 | 0v | | |
| 13 | 9 | MISO | 21 | 22 | GPIO6 | 25 | 6 |
| 14 | 11 | SCLK | 23 | 24 | CE0 | 8 | 10 |
| | | 0v | 25 | 26 | CE1 | 7 | 11 |
| WiringPi Pin | BCM GPIO | Name | Header | | Name | BCM GPIO | WiringPi Pin |

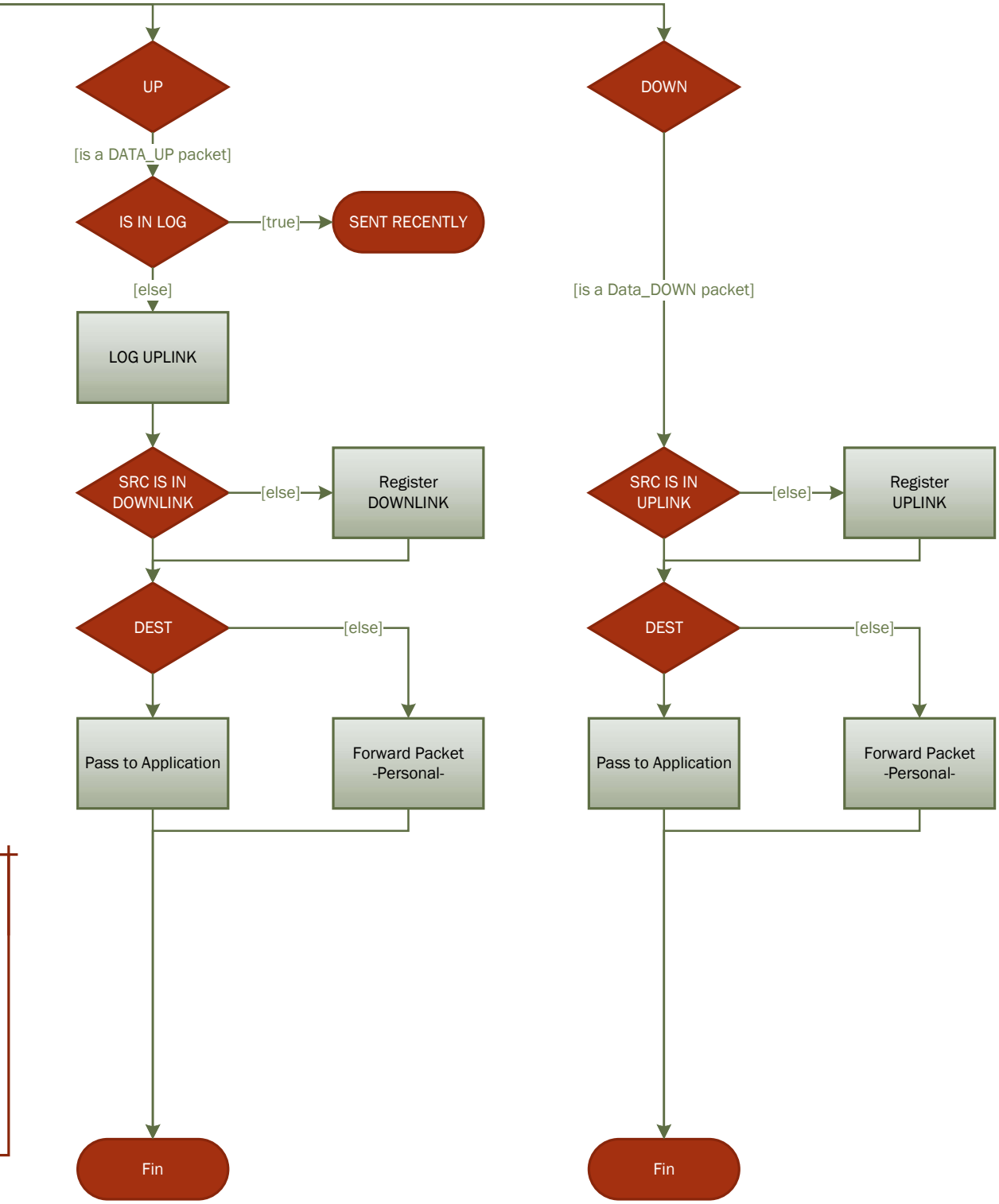
Note the differences between Revision 1 and Revision 2 Raspberry Pi's. The Revision 2 is readily identifiable by the presence of the 2 mounting holes.

The revision 2 Raspberry Pi has an additional GPIO connector, P5, which is next to the main P1 GPIO connector:

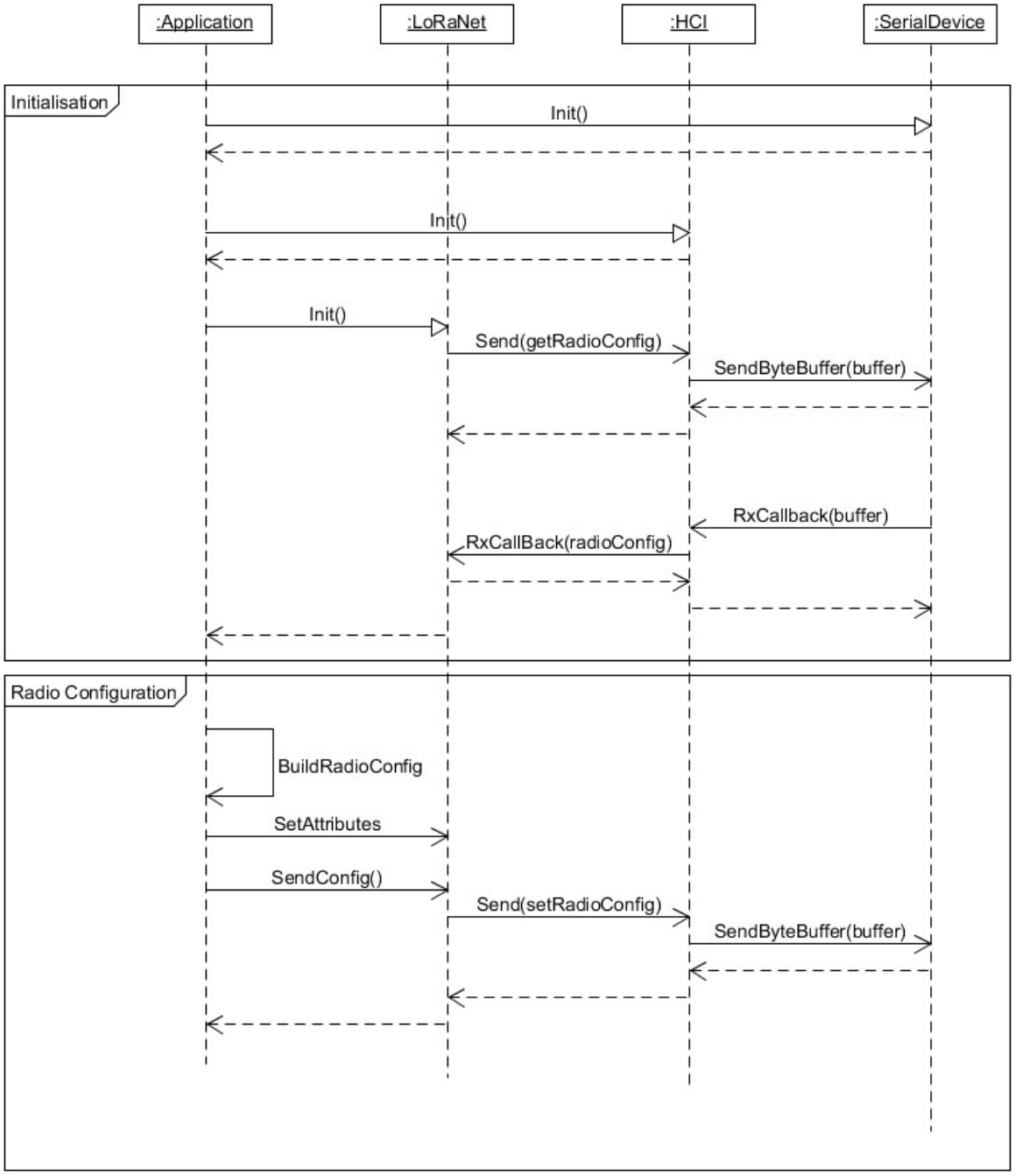
| P5: Secondary GPIO connector (Rev. 2 Pi only) | | | | | | | |
|---|---|---|---|---|---|---|---|
| WiringPi Pin | BCM GPIO | Name | Header | | Name | BCM GPIO | WiringPi Pin |
| | | 5v | 1 | 2 | 3.3v | | |
| 17 | 28 | GPIO8 | 3 | 4 | GPIO9 | 29 | 18 |
| 19 | 30 | GPIO10 | 5 | 6 | GPIO11 | 31 | 20 |
| | | 0v | 7 | 8 | 0v | | |
| WiringPi Pin | BCM GPIO | Name | Header | | Name | BCM GPIO | WiringPi Pin |


Packet Reception
PING
[is a Ping packet]
BROADCAST
[is a broadcasted ping]
[else]
Me
[is a ping for me]
[else]
IS IN UP
[is in uplink list]
[else]
Send Pong
Send a broadcasted Ping
08.14 ADDED
IS CONFIRMED IN UP
[else]
Confirm this node in uplink list
[true]
IS IN DOWN
[else]
Register in uplink list
Send Pong
08.08 ADDED
IS CONFIRMED IN UP
true
Send a broadcasted Ping
[else]
Fin

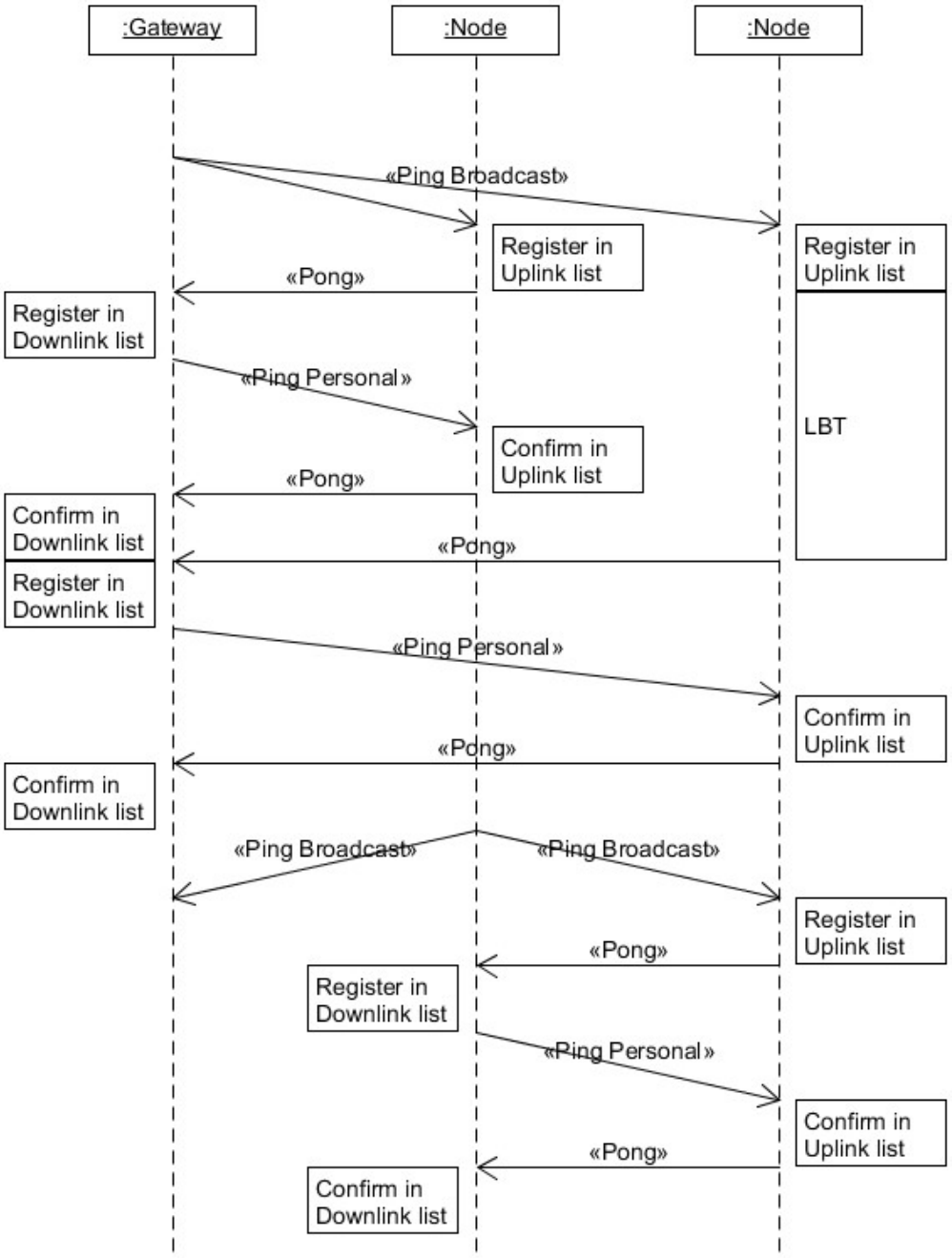

PONG
[is a Pong packet]
ME
[is a pong for me]
[else]
Error
IS IN UP
[is in uplink list]
IS CONFIRMED
[else]
[else]
Delete from Uplink list
IS IN DOWN
[is in downlink list]
[else]
Register in downlink list
IS CONFIRMED
[true]
[else]
Send personal Ping
Confirn the node in downlink
Send a Broadcasted Ping
08.14 REMOVED
[is confirmed in uplink]
Fin


UP
[is a DATA_UP packet]
IS IN LOG
[true]
SENT RECENTLY
[else]
LOG UPLINK
SRC IS IN DOWNLINK
[else]
Register DOWNLINK
DEST
[else]
Pass to Application
Forward Packet -Personal-
Fin
DOWN
[is a Data_DOWN packet]
SRC IS IN UPLINK
[else]
Register UPLINK
DEST
[else]
Pass to Application
Forward Packet -Personal-
Fin

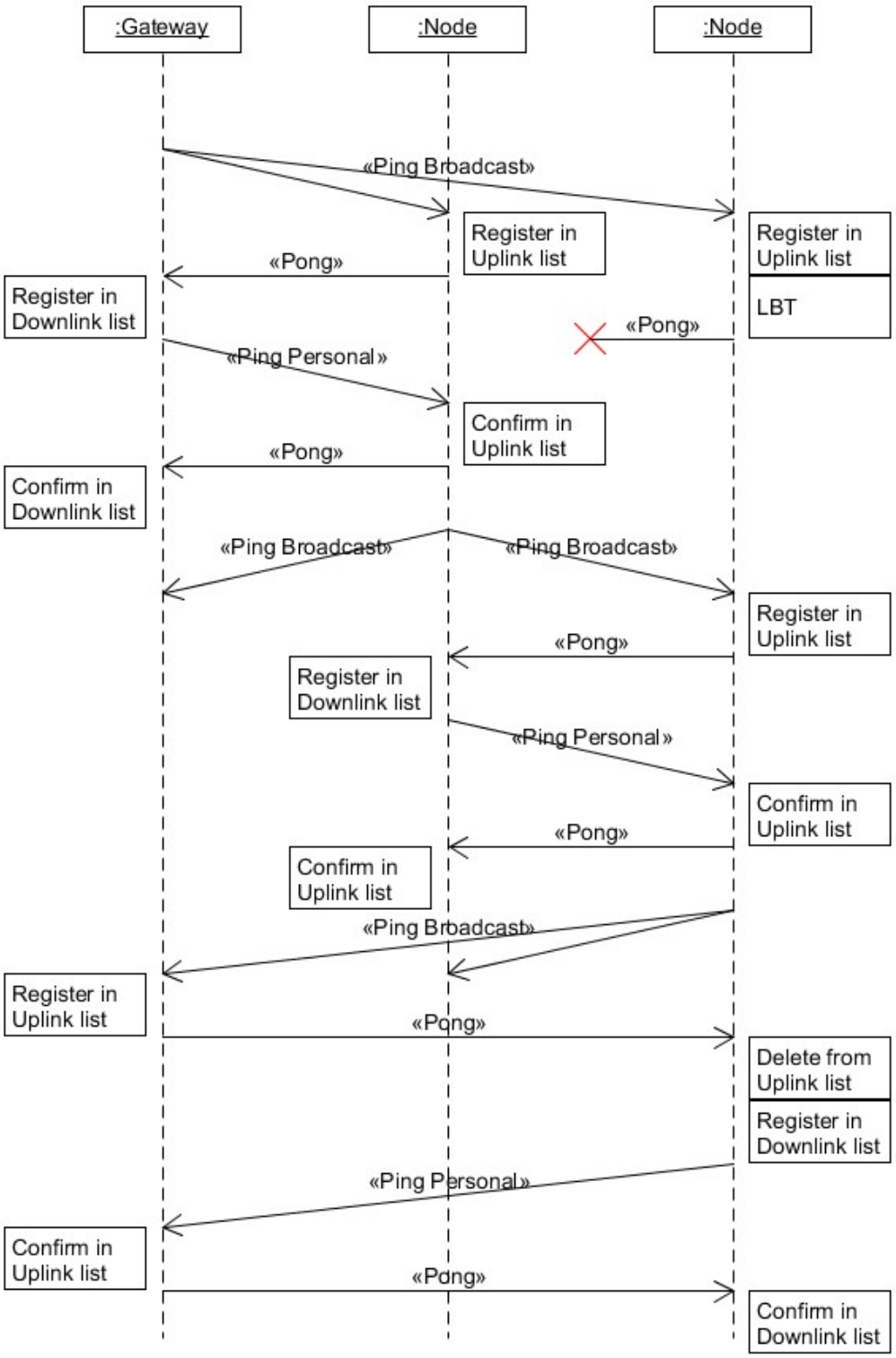
# INITIALISATIONS ET PARAMÉTRAGE DU MODULE RF



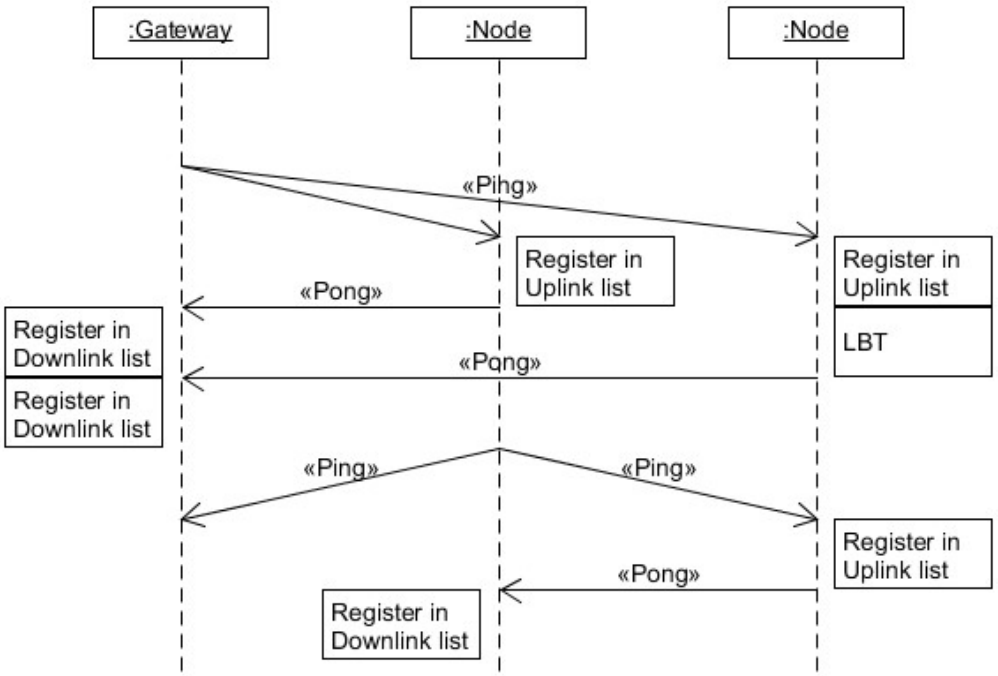
# SCÉNARIO FINAL, LBT FONCTIONNEL



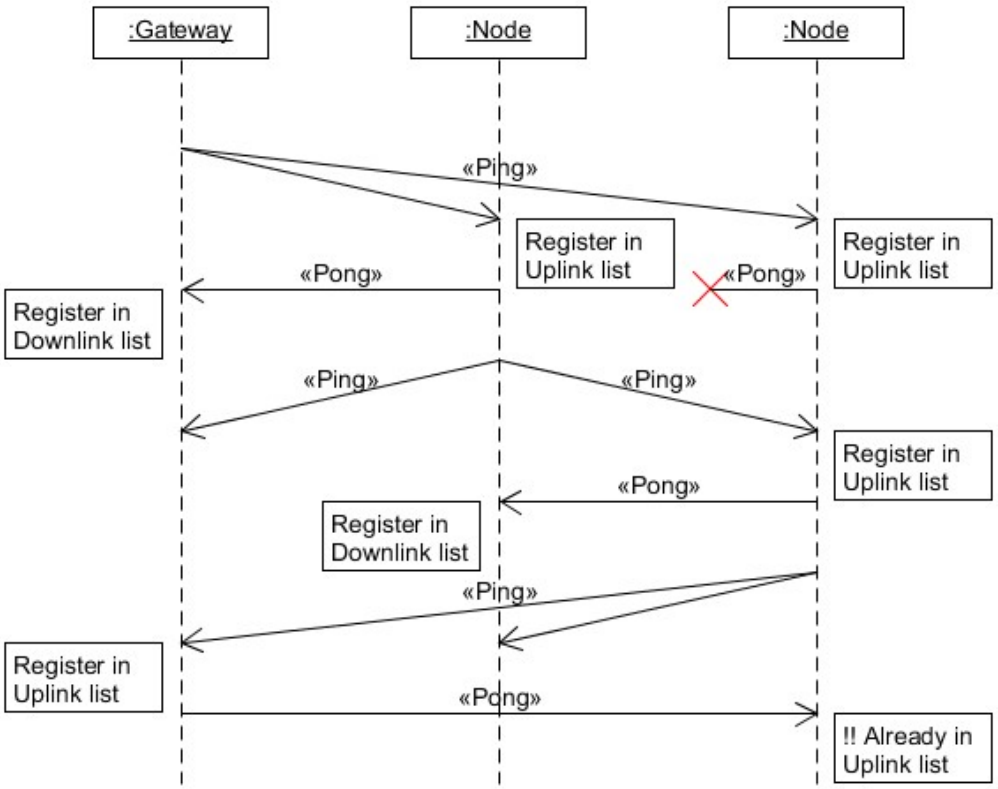
# SCÉNARIO FINAL, LBT DYSFONCTIONNEL



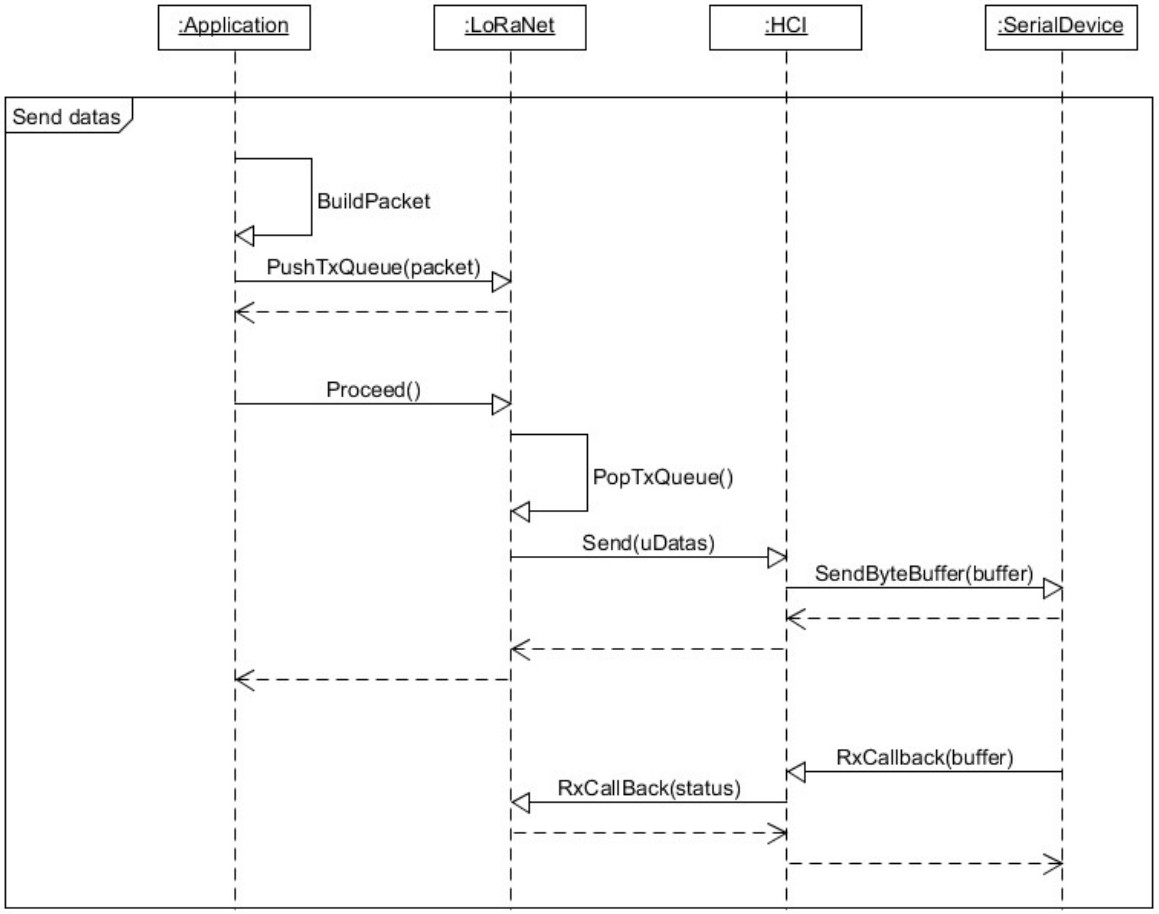
# SCÉNARIO INITIAL, LBT FONCTIONNEL



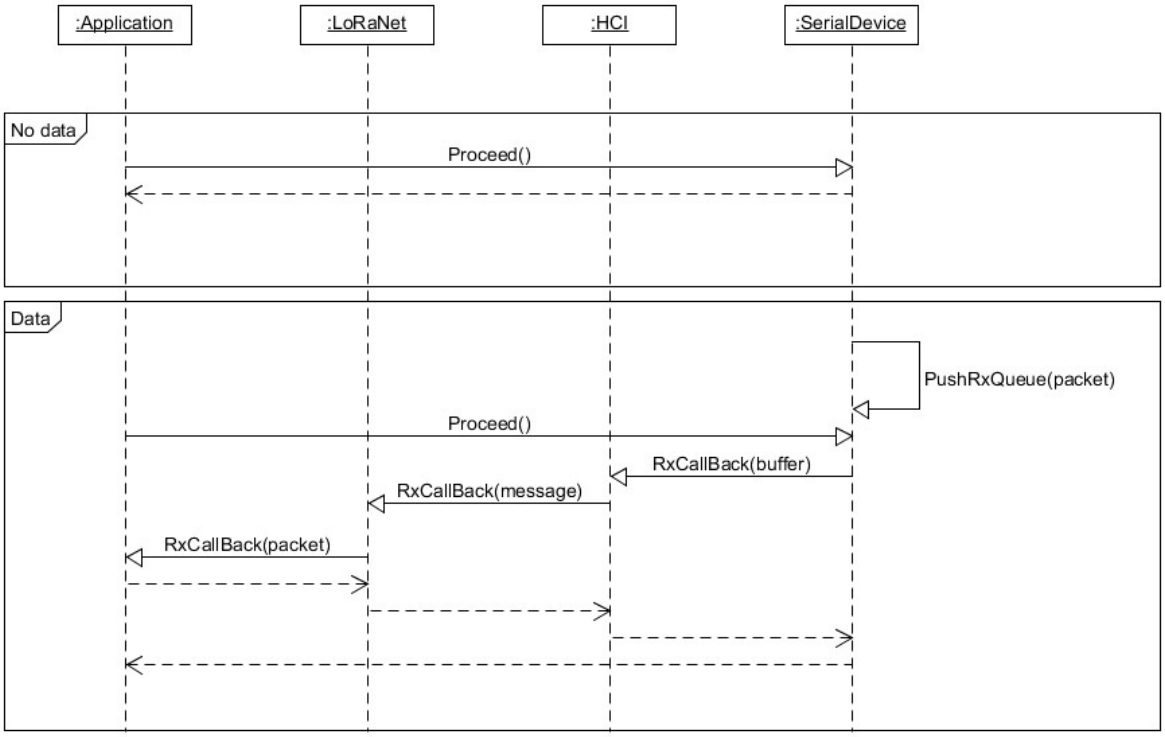
# SCÉNARIO INITIAL, LBT DYSFONCTIONNEL



# ENVOI DE DONNÉES DEPUIS L'APPLICATION



# RÉCEPTION DE DONNÉES VERS L'APPLICATION



```
SRC = main.c util/crc16.c util/time.c serial_device.c loranet.c hci.c
OBJS = $(subst util/,, $(patsubst %.c, %.o, $(SRC)))
CC = gcc
CFLAGS = -c
LIBS = -l wiringPi
EXEC = LoRaGateway

all : $(EXEC)
    rm $(OBJS)

$(OBJS) :
    $(CC) $(CFLAGS) $(SRC)

$(EXEC) : $(OBJS)
    $(CC) -o $@ $(OBJS) $(LIBS)

clean :
    rm $(OBJS) $(EXEC)
```

```
/**
 * \author       Pierre Mendicino @HES-SO Valais/Wallis\n
 *               pierre.mendicino@students.hevs.ch
 * \version      1.0
 * \mainpage     LoRaNode Project
 * \brief        This Documentation is about the project LoRaNode. This Project
 *               was made during a Bachelor named "Low-power networks relays"
 *               during the summer 2017, by Pierre Mendicino.
 * \brief        <table>
 *               <caption id="multi_row">Program layers</caption>
 *               <tr><td>Application Layer
 *               <tr><td>LoRaNet Layer
 *               <tr><td>HCI Layer
 *               <tr><td>SerialDevice Layer
 *               </table>
 *               This table show us an overview of the architecture of this
 *               project.
 */
//******************************************************************************
//
// Empty file
//
//******************************************************************************

```

```c
/**
  * \file        main.c
  * \author      Pierre Mendicino
  * \version 1.0
  * \date        2017.08.10
  * \brief       This file contains the application layer
  */
//*****************************************************************************
//
// Includes
//
//*****************************************************************************
#include "_conf.h"                ///< Device parameters

//Standards
#include <stddef.h>
#include <stdbool.h>

//Board specifics
#include "stm32f10x.h"
#include "STM32vldiscovery.h"

//Application specifics
#include "util/IMST_HCI.h"
#include "util/crc16.h"
#include "util/byteBuffer.h"
#include "util/time.h"
#include "serial_device.h"
#include "hci.h"
#include "loranet.h"
#ifdef __TEST__
#include "test.h"
#endif //__TEST__

//*****************************************************************************
//
// Definitions
//
//*****************************************************************************
#define MAX_MSG_SIZE      255     ///< Size of rxBuffer
#define MEASURE_TIME      2000    ///< Time to wait between two measures

//*****************************************************************************
//
// Variables
//
//*****************************************************************************
bool         isSent;
bool         tim2_flag;
uint32_t     tim2_value;
uint32_t tim2_end;

uint8_t      _rxBuffer[MAX_MSG_SIZE];  //Rx Buffer
uint8_t* _rxPtr = &_rxBuffer[0];

/**
 * \brief    Set the pin PB0 of the STM32VLDISCOVERY board to output, push-pull
 * \retval   void
 */
void setPB0_OPP(void){
    GPIO_InitTypeDef GPIO_InitStructure;

    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB, ENABLE);

    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_2MHz;

    GPIO_Init(GPIOB, &GPIO_InitStructure);
}

/**
 * \brief    Toggle the pin PB0 of the STM32VLDISCOVERY board
 * \retval   void
 */
void toggle_PB0(void){
    GPIO_WriteBit(GPIOB, GPIO_Pin_0, !GPIO_ReadOutputDataBit(GPIOB, GPIO_Pin_0));
}

/**
 * \brief    Initialize the ADC1 peripheral of the STM32VLDISCOVERY and link it
 *               to the pin PA1. The value can be read with the method readADC.
 * \retval   void
 */
```

```
 85  void InitializeADC(){
 86      GPIO_InitTypeDef GPIO_InitStructure;
 87      ADC_InitTypeDef ADC_InitStructure;
 88
 89      RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1, ENABLE);
 90      RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE);
 91
 92      GPIO_InitStructure.GPIO_Pin = GPIO_Pin_1;
 93      GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AIN;
 94      GPIO_InitStructure.GPIO_Speed = GPIO_Speed_2MHz;
 95
 96      GPIO_Init(GPIOA, &GPIO_InitStructure);
 97
 98      ADC_InitStructure.ADC_Mode = ADC_Mode_Independent;
 99      ADC_InitStructure.ADC_ScanConvMode = DISABLE;
100      ADC_InitStructure.ADC_ContinuousConvMode = ENABLE;
101      ADC_InitStructure.ADC_ExternalTrigConv = ADC_ExternalTrigConv_None;
102      ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;
103      ADC_InitStructure.ADC_NbrOfChannel = 1;
104
105      ADC_RegularChannelConfig(ADC1, ADC_Channel_1, 1, ADC_SampleTime_28Cycles5);
106      ADC_Init(ADC1, &ADC_InitStructure);
107
108      ADC_Cmd(ADC1, ENABLE);
109      ADC_SoftwareStartConvCmd(ADC1, ENABLE);
110  }
111
112  /**
113   * \brief    This function read the value of the ADC1 peripheral.
114   * \retval   The value measured
115   */
116  uint16_t readADC(){
117      return ADC_GetConversionValue(ADC1);
118  }
119
120  /**
121   * \brief    This function initialize the peripheral TIM2 of the
122   *           STM32VLDISCOVERY board at 1kHz.
123   * \retval   void
124   */
125  void InitializeTimer(){
126      RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2, ENABLE);
127
128      TIM_TimeBaseInitTypeDef timerInitStructure;
129      timerInitStructure.TIM_Prescaler = 7999;
130      timerInitStructure.TIM_CounterMode = TIM_CounterMode_Up;
131      timerInitStructure.TIM_Period = 2;
132      timerInitStructure.TIM_ClockDivision = TIM_CKD_DIV1;
133      timerInitStructure.TIM_RepetitionCounter = 0;
134      TIM_TimeBaseInit(TIM2, &timerInitStructure);
135      TIM_ITConfig(TIM2, TIM_IT_Update, ENABLE);
136      //TIM_Cmd(TIM2, ENABLE);
137
138      NVIC_EnableIRQ(TIM2_IRQn);
139
140      SysTick_Config(SystemCoreClock / 1000);
141  }
142
143  /**
144   * \brief    This function reset the tim2_xx attributes and launch the TIM2.
145   * \param param : The time to put the flag tim2_flag to 1
146   * \retval   void
147   */
148  void start_timer(uint32_t param){
149      tim2_end = param;
150      tim2_value = 0;
151      tim2_flag = false;
152      TIM_SetCounter(TIM2, 0); //reset
153      TIM_Cmd(TIM2, ENABLE);       //launch
154  }
155
156  /**
157   * \brief    This function handles the ticks from the TIM2
158   * \retval   void
159   */
160  void TIM2_IRQHandler(void){
161      if (TIM_GetITStatus(TIM2, TIM_IT_Update) != RESET){
162          TIM_ClearITPendingBit(TIM2, TIM_IT_Update);
163          if(++tim2_value == tim2_end){
164              tim2_flag = true;
165              TIM_Cmd(TIM2, DISABLE);       //stop
166          } else {
167              TIM_SetCounter(TIM2, 0); //reset
168              TIM_Cmd(TIM2, ENABLE);        //launch
```

```
169             }
170         }
171     }
172
173     /**
174      * \brief    This function should be passed to the LoRaNet layer and will be
175      *               called each time we receive a packet in witch we are the
176      *               destination.
177      * \param packet : The packet we received
178      * \retval   void
179      */
180     void LORA_RxCallBack(LoRaNet_Packet* packet){
181         uint16_t index = 0;
182         while(index < packet->data.length){
183             _rxBuffer[index] = packet->data.begin[index];
184             index++;
185         }
186         free(packet->data.begin);
187         free(packet);
188         //TOREMOVE
189         if(_rxBuffer[0] >= 0x31){//if value is >= 1000
190             LoRaNet_UnregisterAll(0x02, LORANET_DOWN);
191         }
192     }
193
194     /**
195      * \brief    Call this function to set all parameters in the radio modules
196      * \retval   void
197      */
198     void configureRadioModule(){
199         //Configure the command
200         LoRaNet_SetRadioMode(IMST_HCI_DEVMGMT_RADIO_MODE_STD);
201         LoRaNet_SetGroupAddress(0x10);
202     #ifdef __NODE_01__
203         LoRaNet_SetDeviceAddress(0x1234);
204     #endif //__NODE_01__
205     #ifdef __NODE_02__
206         LoRaNet_SetDeviceAddress(0x2345);
207     #endif //__NODE_02__
208     #ifdef __NODE_03__
209         LoRaNet_SetDeviceAddress(0x3456);
210     #endif //__NODE_03__
211     #ifdef __NODE_04__
212         LoRaNet_SetDeviceAddress(0x4567);
213     #endif //__NODE_04__
214         LoRaNet_SetModulation(IMST_HCI_DEVMGMT_MODULATION_LORA);
215         LoRaNet_SetCarrierFreq(IMST_HCI_DEVMGMT_CARRIER_FREQ_868_5);
216         LoRaNet_SetBandwidth(IMST_HCI_DEVMGMT_LORA_BW_125K);
217         LoRaNet_SetSpreadingFactor(IMST_HCI_DEVMGMT_LORA_SF_7);
218         LoRaNet_SetErrorCoding(IMST_HCI_DEVMGMT_LORA_RC_4_5);
219         LoRaNet_SetPowerLevel(17);
220         LoRaNet_SetTxControl(
221                 IMST_HCI_DEVMGMT_TX_CONTROL_LBT//|
222                 //IMST_HCI_DEVMGMT_TX_CONTROL_NF
223                 );
224         LoRaNet_SetRxControl(IMST_HCI_DEVMGMT_RX_CONTROL_ALWAYS_ON);
225         LoRaNet_SetRxWindowTime(3000);//in ms
226         LoRaNet_SetLedControl(
227                 IMST_HCI_DEVMGMT_LED_CONTROL_ALIVE_D4 |
228                 IMST_HCI_DEVMGMT_LED_CONTROL_RX_D3 |
229                 IMST_HCI_DEVMGMT_LED_CONTROL_TX_D2
230                 );
231         LoRaNet_SetMiscOpt(
232                 //IMST_HCI_DEVMGMT_MISC_OPTIONS_NOTHING |
233                 //IMST_HCI_DEVMGMT_MISC_OPTIONS_EXT_RF |
234                 //IMST_HCI_DEVMGMT_MISC_OPTIONS_RTC |
235                 //IMST_HCI_DEVMGMT_MISC_OPTIONS_TX_IND |
236                 IMST_HCI_DEVMGMT_MISC_OPTIONS_POWER_UP_IND//|
237                 //IMST_HCI_DEVMGMT_MISC_OPTIONS_BUTTON_IND |
238                 //IMST_HCI_DEVMGMT_MISC_OPTIONS_AES
239                 );
240         LoRaNet_SetFskDataRate(IMST_HCI_DEVMGMT_FSK_DATARATE_250_KBPS);
241         LoRaNet_SetLBTThresold(-90);
242     }
243
244     /**
245      * \brief : Main function : this function is supposed to have an infinite loop.
246      * \retval : Error
247      */
248     int main(void)
249     {
250         // Local variables
251     #ifdef __TEST__
252         //Don't delete : this variable is used by some tests
```

```
253         uint8_t testParam = 0;
254     #endif //__TEST__
255
256         // Initializations
257         InitializeTimer();
258         setPB0_OPP();
259         Time_Init();
260         InitializeADC();
261         SerialDevice_Init(HCI_RxCallBack);
262         HCI_Init(LoRaNet_RxCallBack);
263     #ifdef __TEST__
264         Test_Init();
265     #endif //__TEST__
266     #ifdef __NODE_01__
267         LoRaNet_Init(LORA_RxCallBack, 0x01);
268     #endif //__NODE_01__
269     #ifdef __NODE_02__
270         LoRaNet_Init(LORA_RxCallBack, 0x02);
271     #endif //__NODE_02__
272     #ifdef __NODE_03__
273         LoRaNet_Init(LORA_RxCallBack, 0x03);
274     #endif //__NODE_03__
275     #ifdef __NODE_04__
276         LoRaNet_Init(LORA_RxCallBack, 0x04);
277     #endif //__NODE_04__
278         STM32vldiscovery_LEDInit(LED3);
279         STM32vldiscovery_LEDInit(LED4);
280         STM32vldiscovery_PBInit(BUTTON_USER, BUTTON_MODE_GPIO);
281         STM32vldiscovery_LEDOff(LED3);
282         STM32vldiscovery_LEDOff(LED4);
283
284         // Configurations
285         configureRadioModule();
286         LoRaNet_SendConfig();
287
288         // One-time action : start the timer for the first time
289         start_timer(3000);
290
291         // Infinite loop
292         while (1)
293         {
294             // Heartbeat
295             STM32vldiscovery_LEDToggle(LED4);
296             // Look if a message have been arrived..
297             SerialDevice_Proceed();
298             // Send a message if tx queue is not empty..
299             LoRaNet_Proceed();
300             if(0 == STM32vldiscovery_PBGetState(BUTTON_USER))//button released
301             {
302                 // Turn On LED3
303                 isSent = false;
304             }
305             else
306             {
307                 if(!isSent){//On push
308     #ifdef __TEST__
309     #ifdef __TEST_01__
310                     Test_01(testParam++);
311     #endif //__TEST_01__
312     #ifdef __TEST_02__
313                     Test_02(testParam++);
314     #endif //__TEST_02__
315     #ifdef __TEST_03__
316                     Test_03(testParam++);
317     #endif //__TEST_03__
318     #ifdef __TEST_04__
319                     Test_04(testParam++);
320     #endif //__TEST_04__
321     #ifdef __TEST_05__
322                     Test_05(testParam++);
323     #endif //__TEST_05__
324                         //LoRaNet_PushTxQueue(LoRaNet_MakePingPongPacket(LORANET_TYPE_PING,
    LORANET_ID_BROADCAST));
325                     isSent = true;
326     #endif // __TEST__
327                     }
328                 }
329     #ifdef __TEST_06__
330             if(LoRaNet_IsAnUplinkConfirmed()){
331                 Test_06(readADC());
332                 //This lines permit to send data every seconds when the button is maintained
    down
333                 Time_wait(1000);
334                 isSent = false;
```

```
335                 testParam++; //Just to suppress "unused variable" during the test
336             }
337     #endif // __TEST_06__
338     #ifdef __TEST_07__
339             if(LoRaNet_IsAnUplinkConfirmed()){
340                 STM32vldiscovery_LEDOn(LED3);
341                 if(tim2_flag){
342                     Test_07(readADC());
343                     start_timer(3000);
344                     testParam++; //Just to suppress "unused variable" during the test
345                 }
346             } else {
347                 STM32vldiscovery_LEDOff(LED3);
348             }
349     #endif // __TEST_07__
350         }//_____End of infinite loop
351     }//_________End of main
352
353     /**
354      * \brief    Minimal __assert_func used by the assert() macro
355      */
356     void __assert_func(const char *file, int line, const char *func, const char *failedexpr)
357     {
358         while(1)
359         {}
360     }
361
362     /**
363      * \brief    Minimal __assert() uses __assert__func()
364      */
365     void __assert(const char *file, int line, const char *failedexpr)
366     {
367         __assert_func (file, line, NULL, failedexpr);
368     }
369
```

```
/**
  * \file        loranet.h
  * \author      Pierre Mendicino
  * \version 1.0
  * \date        2017.08.10
  * \brief       This file contains the headers of the LoRaNet layer
  */
#ifndef LORANET_H_
#define LORANET_H_

//*****************************************************************************
//
// Includes
//
//*****************************************************************************
#include <stdbool.h>
#include "util/byteBuffer.h"
#include "util/IMST_HCI.h"
#include "util/time.h"
#include "hci.h"
#include "_conf.h"

//*****************************************************************************
//
// Definitions
//
//*****************************************************************************
/// Max attempts for sending messages to the RF module without response
#define LORANET_MAX_ATTEMPTS  5       ///< Must be greater than 0
/// Maximum amount of nodes that can register in uplink and downlink lists
#define LORANET_MAX_NODES     20      ///< Must be greater than 0
/// Maximum amount of Tx messages in \ref uplink_log
#define LORANET_MAX_LOG           10      ///< Must be greater than 0
/// The size of the TxQueue
#define LORANET_QUEUE_SIZE        50      ///< Must be greater than 0

/**
 * \defgroup LORANET_OFFSET
 * \brief    This definitions should be used to reach a certain field in a
 *           seiralized LoRaNet packet
 * \{
 */
/// Offset to reach the type in a serialized packet
#define LORANET_OFFSET_TYPE       0x00    ///<
/// Offset to reach the destination Id in a serialized packet
#define LORANET_OFFSET_DST        0x01    ///<
/// Offset to reach the source Id in a serialized packet
#define LORANET_OFFSET_SRC        0x02    ///<
/// Offset to reach the packet Id in a serialized packet
#define LORANET_OFFSET_ID     0x03    ///<
/// Offset to reach the data in a serialized packet
#define LORANET_OFFSET_DATA       0x04    ///<
/// Useful to check if a certain type is acceptable
#define IS_LORANET_OFFSET(OFFSET) (((OFFSET) == LORANET_OFFSET_TYPE) || \
                                   ((OFFSET) == LORANET_OFFSET_DST)) || \
                                   ((OFFSET) == LORANET_OFFSET_SRC)) || \
                                   ((OFFSET) == LORANET_OFFSET_DATA))    ///<
/**
 * \}
 */

/**
 * \defgroup LORANET_TYPE
 * \brief    This definitions should be used to tag a LoRaNet packet with an
 *           acceptable type
 * \{
 */
/// Defines a type of LoRaNet packet : Unknown
#define LORANET_TYPE_UNKNOWN  0x00    ///<
/// Defines a type of LoRaNet packet : Data upstream
#define LORANET_TYPE_DATA_UP  0x01    ///<
/// Defines a type of LoRaNet packet : Data downstream
#define LORANET_TYPE_DATA_DOWN    0x02    ///<
/// Defines a type of LoRaNet packet : Ping packet
#define LORANET_TYPE_PING     0x03    ///<
/// Defines a type of LoRaNet packet : Pong packet
#define LORANET_TYPE_PONG     0x04    ///<
/// Useful to check if a certain type is acceptable
#define IS_LORANET_TYPE(TYPE) (((TYPE) == LORANET_TYPE_UNKNOWN) || \
                                   ((TYPE) == LORANET_TYPE_DATA_UP)) || \
                                   ((TYPE) == LORANET_TYPE_DATA_DOWN)) || \
                                   ((TYPE) == LORANET_TYPE_PING)) || \
                                   ((TYPE) == LORANET_TYPE_PONG)))   ///<
/**
```

```
 * \}
 */

/// Defines the broadcast Id
#define LORANET_ID_BROADCAST  0xFF     ///< This Id is used to send some
                                       ///< broadcast messages like "Ping"
/**
 * \defgroup LORANET_DIRECTION
 * \brief    This definitions should be used to tag a LoRaNet packet with an
 *           acceptable type
 * \{
 */
/// Defines a type of LoRaNet packet : Unknown
/// Defines an attribute that says if something is related to the uplink list
#define LORANET_UP                0x00     ///<
/// Defines an attribute that says if something is related to the downlink list
#define LORANET_DOWN          0x01     ///<
/// Useful to check if a certain direction is acceptable
#define IS_LORANET_DIRECTION(DIRECTION) (((DIRECTION) == LORANET_UP) || \
                                ((TYPE) == LORANET_DOWN))) ///<
/**
 * \}
 */

/**
 * \brief    This structure represent a LoRaNet node. It makes the link between
 *           the physical address and the id of a certain node.
 */
typedef struct
{
    /// The Id of the node
    uint8_t          nodeId;      ///< This value can't be :
                                  ///< 0x00 (Undefined)
                                  ///< 0xFF (Broadcast)
    /// The physical address where the id is reachable
    uint16_t     address; ///< This value can't be :
                                  ///< 0x0000 (Undefined)
                                  ///< 0xFFFF (Broadcast)
    /// The address of the group in witch the node is
    uint8_t          groupId; ///< This value can't be :
                                  ///< 0x00 (Undefined)
                                  ///< 0xFF (Broadcast)
    /// Attribute giving an information about the state of the node
    bool             confirmed;   ///< If the node is confirmed, it means that he
                                  ///< is physically reachable and that the link
                                  ///< with this node is established.
}LoRaNet_Node;

/**
 * \brief    This structure represent a LoRaNet packet. It's the easiest way to
 *           communicate with the LoRaNet layer.
 */
typedef struct
{
    /// The type of the packet.
    uint8_t          type;        ///< This value can be :
                                  ///< - LORANET_TYPE_UNKNOWN (Unknown type)
                                  ///< - LORANET_TYPE_DATA_UP (Data in upstream)
                                  ///< - LORANET_TYPE_DATA_DOWN (Data in
                                  ///< downstream)
                                  ///< - LORANET_TYPE_PING (Ping)
                                  ///< - LORANET_TYPE_PONG (Pong)
    /// The Id of the destination.
    uint8_t          dstId;       ///< When a new packet is created, normally
                                  ///< this value is the gateway Id.\n
                                  ///< For broadcast, use LORANET_ID_BROADCAST
    /// The Id of the source of the packet.
    uint8_t          srcId;       ///< When a new packet is created, this node is
                                  ///< the source (LoRaNet_myNode)
    /// The data contained in the packet
    byteBuffer_t data;        ///< If there is no data, please use malloc(0)
    /// The Id of the packet
    uint8_t          packetId;    ///< This allow the network to follow a
                                  ///< specific packet. The method getPacketId()
                                  ///< can give an id..
}LoRaNet_Packet;

//****************************************************************************
//
// Private variables
//
//****************************************************************************
/**
 * \vargroup Private variable
```

```
169    * \{
170    */
171    LoRaNet_Node*    LoRaNet_myNode;                        ///< This node
172    LoRaNet_Node*    LoRaNet_uplink[LORANET_MAX_NODES];     ///< Known nodes (uplink)
173    uint8_t          LoRaNet_uplink_cnt;                    ///< Occupation (uplink)
174    LoRaNet_Node*    LoRaNet_downLink[LORANET_MAX_NODES];///< Known nodes (downlink)
175    uint8_t          LoRaNet_downlink_cnt;                  ///< Occupation (downlink)
176    LoRaNet_Packet*  LoRaNet_uplink_log[LORANET_MAX_LOG];///< Tx Log (uplink)
177    uint8_t          LoRaNet_uplink_log_in;                 ///< Input Log (uplink)
178    LoRaNet_Packet*  LoRaNet_txQueue[LORANET_QUEUE_SIZE];///< Tx FIFO
179    uint8_t          LoRaNet_tx_in;                         ///< Input of Tx FIFO
180    uint8_t          LoRaNet_tx_out;                        ///< Output of Tx FIFO
181    void (*LoRaNet_CallBack)(LoRaNet_Packet* packet);  ///< Callback function
182    IMST_HCI_RADIO_CONFIG    LoRaNet_radioConfig;      ///< Radio configuration
183    bool             LoRaNet_isInit;                        ///< We got the config
184    /**
185     * \}
186     */
187
188    //****************************************************************************
189    //
190    // Public functions
191    //
192    //****************************************************************************
193    /**
194     * \fngroup  Public function
195     * \{
196     */
197    // Useful functions ********************************************************
198    void             LoRaNet_Init(void(*callBack)(LoRaNet_Packet* packet),
199            uint8_t myId);
200    IMST_HCI_RADIO_CONFIG*    LoRaNet_GetConfig();
201    void             LoRaNet_PushTxQueue(LoRaNet_Packet* packet);
202    void             LoRaNet_Proceed();
203    void             LoRaNet_SendConfig();
204    // Radio configuration functions*******************************************
205    void             LoRaNet_SetRadioMode(uint8_t radioMode);
206    void             LoRaNet_SetGroupAddress(uint8_t groupAddress);
207    //void           LoRaNet_SetTxGroupAddress(uint8_t groupAddress); Not used
208    void             LoRaNet_SetDeviceAddress(uint16_t deviceAddress);
209    //void           LoRaNet_SetTxDeviceAddress(uin16_t deviceAddress); Not used
210    void             LoRaNet_SetModulation(uint8_t modulation);
211    void             LoRaNet_SetCarrierFreq(uint32_t carrierFreq);
212    void             LoRaNet_SetBandwidth(uint8_t bandwidth);
213    void             LoRaNet_SetSpreadingFactor(uint8_t spreadingFactor);
214    void             LoRaNet_SetErrorCoding(uint8_t errorCoding);
215    void             LoRaNet_SetPowerLevel(uint8_t powerLevel);
216    void             LoRaNet_SetTxControl(uint8_t txControl);
217    void             LoRaNet_SetRxControl(uint8_t rxControl);
218    void             LoRaNet_SetRxWindowTime(uint16_t rxWindowTime);
219    void             LoRaNet_SetLedControl(uint8_t ledControl);
220    void             LoRaNet_SetMiscOpt(uint8_t miscOpt);
221    void             LoRaNet_SetFskDataRate(uint8_t fskDataRate);
222    void             LoRaNet_SetLBTThresold(int16_t lbtThreshold);
223    /**
224     * \}
225     */
226
227    //****************************************************************************
228    //
229    // Protected functions
230    //
231    //****************************************************************************
232    /**
233     * \fngroup  Protected function
234     * \{
235     */
236    void LoRaNet_RxCallBack(IMST_HCI_MSG* data);
237    /**
238     * \}
239     */
240
241    //****************************************************************************
242    //
243    // Private functions
244    //
245    //****************************************************************************
246    /**
247     * \fngroup  Private function
248     * \{
249     */
250    void             LoRaNet_AddInLog(LoRaNet_Packet* packet, uint8_t loranet_x);
251    LoRaNet_Node*    LoRaNet_GetFirst(uint8_t id, uint8_t loranet_x);
252    LoRaNet_Node*    LoRaNet_GetNum(uint8_t id, uint8_t num, uint8_t loranet_x);
```

```
253   LoRaNet_Node*    LoRaNet_GetNode(LoRaNet_Node* node, uint8_t loranet_x);
254   uint8_t      LoRaNet_GetPacketID(void);
255   // ADDED on 2017.08.08
256   bool             LoRaNet_IsAnUplinkConfirmed();
257   bool             LoRaNet_IsPresentInLog(LoRaNet_Packet* packet,
258           uint8_t loranet_x);
259   bool             LoRaNet_IsRegistered(LoRaNet_Node* node, uint8_t loranet_x);
260   bool             LoRaNet_IsTxQueueEmpty();
261   bool             LoRaNet_IsTxQueueFull();
262   LoRaNet_Node*    LoRaNet_MakeNode(IMST_HCI_MSG* msg);
263   LoRaNet_Packet* LoRaNet_MakePacket(IMST_HCI_MSG* msg);
264   LoRaNet_Packet* LoRaNet_MakePingPongPacket(uint8_t type, uint8_t destination);
265   LoRaNet_Packet* LoRaNet_PopTxQueue();
266   void             LoRaNet_RegConfig(IMST_HCI_MSG* msg);
267   void             LoRaNet_Register(LoRaNet_Node* node, uint8_t loranet_x);
268   bool             LoRaNet_ReqConfig();
269   void             LoRaNet_Send(LoRaNet_Packet* packet);
270   void             LoRaNet_SetPtrCnt();
271   void             LoRaNet_Unregister(LoRaNet_Node* node, uint8_t loranet_x);
272   void             LoRaNet_UnregisterAll(uint8_t id, uint8_t loranet_x);
273   //REMOVED on 2017.07.11
274   //LoRaNet_Packet* LoRaNet_MakePingPacket(uint8_t destination);
275   /**
276    * \}
277    */
278
279   #endif /* LORANET_H_ */
280
```

```
1   /**
2     * \file        loranet.c
3     * \author      Pierre Mendicino
4     * \version 1.0
5     * \date        2017.08.10
6     * \brief       This file contains the implementation of the LoRaNet layer
7     */
8   #include "loranet.h"
9
10  /**
11   * \brief    Initialize this layer. Actually, this function send a
12   *               requestRadioConfig to the lower layer and block the    program as
13   *               long as the module hasn't responded.
14   * \param    callback :   The function to call when some data arrives
15   * \param    myId :       The Id of this node
16   * \retval   void
17   */
18  void LoRaNet_Init(void(*callBack)(LoRaNet_Packet* packet), uint8_t myId){
19      LoRaNet_CallBack = callBack;
20      LoRaNet_myNode = malloc(sizeof(LoRaNet_Node));
21      LoRaNet_myNode->nodeId = myId;
22      LoRaNet_myNode->address = LoRaNet_radioConfig.deviceAddress;
23      LoRaNet_myNode->groupId = LoRaNet_radioConfig.groupAddress;
24      LoRaNet_uplink_cnt = 0;
25      LoRaNet_downlink_cnt = 0;
26      LoRaNet_tx_in = 0;
27      LoRaNet_uplink_log_in = 0;
28
29      LoRaNet_isInit = false;
30      LoRaNet_ReqConfig();
31      Time_LaunchTimeout(500);
32      while(!LoRaNet_isInit){
33          if(Time_TimeoutOccured()){
34              LoRaNet_ReqConfig();
35              Time_LaunchTimeout(500);
36          }
37          SerialDevice_Proceed();
38      }
39  }
40
41  /**
42   * \brief    Send the locally stored radioConfig to the module.
43   * \retval   void
44   */
45  void LoRaNet_SendConfig(){
46      IMST_HCI_MSG* toSend;
47      toSend = malloc(sizeof(IMST_HCI_MSG));
48      toSend->sapID = IMST_HCI_DEVMGMT_ID;
49      toSend->msgID = IMST_HCI_DEVMGMT_SET_RADIO_CONFIG_REQ;
50      toSend->payloadLength = 26;
51      toSend->payload = malloc(toSend->payloadLength);
52      toSend->payload[0] = 0x00;//NVM Flag
53      toSend->payload[1 + IMST_HCI_RCF_OFFSET_RADIO_MODE] = LoRaNet_radioConfig.radioMode;
54      toSend->payload[1 + IMST_HCI_RCF_OFFSET_GROUP_ADDRESS] = LoRaNet_radioConfig.
    groupAddress;
55      toSend->payload[1 + IMST_HCI_RCF_OFFSET_TX_GROUP_ADDRESS] = LoRaNet_radioConfig.
    txGroupAddress;
56      toSend->payload[1 + IMST_HCI_RCF_OFFSET_DEVICE_ADDRESS] = LoRaNet_radioConfig.
    deviceAddress & 0xFF;
57      toSend->payload[1 + IMST_HCI_RCF_OFFSET_DEVICE_ADDRESS + 1] = (LoRaNet_radioConfig.
    deviceAddress >> 8) & 0xFF;
58      toSend->payload[1 + IMST_HCI_RCF_OFFSET_TX_DEVICE_ADDRESS] = LoRaNet_radioConfig.
    txDeviceAddress &0xFF;
59      toSend->payload[1 + IMST_HCI_RCF_OFFSET_TX_DEVICE_ADDRESS + 1] = (LoRaNet_radioConfig.
    txDeviceAddress >> 8) & 0xFF;
60      toSend->payload[1 + IMST_HCI_RCF_OFFSET_MODULATION] = LoRaNet_radioConfig.modulation;
61      toSend->payload[1 + IMST_HCI_RCF_OFFSET_RF_CARRIER_FREQ_LSB] = LoRaNet_radioConfig.
    carrierFrequency &0xFF;
62      toSend->payload[1 + IMST_HCI_RCF_OFFSET_RF_CARRIER_FREQ_IMB] = (LoRaNet_radioConfig.
    carrierFrequency >> 8) & 0xFF;
63      toSend->payload[1 + IMST_HCI_RCF_OFFSET_RF_CARRIER_FREQ_MSB] = (LoRaNet_radioConfig.
    carrierFrequency >> 16) & 0xFF;
64      toSend->payload[1 + IMST_HCI_RCF_OFFSET_LORA_BW] = LoRaNet_radioConfig.bandwidth;
65      toSend->payload[1 + IMST_HCI_RCF_OFFSET_LORA_SF] = LoRaNet_radioConfig.spreadingFactor;
66      toSend->payload[1 + IMST_HCI_RCF_OFFSET_LORA_RC] = LoRaNet_radioConfig.errorCoding;
67      toSend->payload[1 + IMST_HCI_RCF_OFFSET_POWER_LEVEL] = LoRaNet_radioConfig.powerLevel;
68      toSend->payload[1 + IMST_HCI_RCF_OFFSET_TX_CONTROL] = LoRaNet_radioConfig.txControl;
69      toSend->payload[1 + IMST_HCI_RCF_OFFSET_RX_CONTROL] = LoRaNet_radioConfig.rxControl;
70      toSend->payload[1 + IMST_HCI_RCF_OFFSET_RX_WINDOW_TIME] = LoRaNet_radioConfig.
    rxWindowTime & 0xFF;
71      toSend->payload[1 + IMST_HCI_RCF_OFFSET_RX_WINDOW_TIME + 1] = (LoRaNet_radioConfig.
    rxWindowTime >> 8) & 0xFF;
72      toSend->payload[1 + IMST_HCI_RCF_OFFSET_LED_CONTROL] = LoRaNet_radioConfig.ledControl;
73      toSend->payload[1 + IMST_HCI_RCF_OFFSET_MISC_OPTIONS] = LoRaNet_radioConfig.
```

```
    miscOptions;
74      toSend->payload[1 + IMST_HCI_RCF_OFFSET_FSK_DATARATE] = LoRaNet_radioConfig.
    fskDataRate;
75      toSend->payload[1 + IMST_HCI_RCF_OFFSET_POWER_SAVING_MODE] = LoRaNet_radioConfig.
    powerSavingMode;
76      toSend->payload[1 + IMST_HCI_RCF_OFFSET_LBT_THRESHOLD] = LoRaNet_radioConfig.
    lbtThreshold & 0xFF;
77      toSend->payload[1 + IMST_HCI_RCF_OFFSET_LBT_THRESHOLD + 1] = (LoRaNet_radioConfig.
    lbtThreshold >> 8) & 0xFF;
78      HCI_Send(toSend);
79  }
80
81  /**
82   * \brief    Send a requestRadioConfig message to the lower layer.
83   * \retval   true :  The request have been received by the RF module
84   * \retval   false : Otherwise
85   */
86  bool LoRaNet_ReqConfig(){
87      uint8_t attempts = LORANET_MAX_ATTEMPTS;
88      IMST_HCI_MSG* toSend;
89      do{
90          toSend = malloc(sizeof(IMST_HCI_MSG));
91          toSend->sapID = IMST_HCI_DEVMGMT_ID;
92          toSend->msgID = IMST_HCI_DEVMGMT_GET_RADIO_CONFIG_REQ;
93          toSend->payloadLength = 0;
94          toSend->payload = malloc(toSend->payloadLength);
95          if(--attempts == 0){
96              return false;
97          }
98      }while(!HCI_Send(toSend));
99      return true;
100 }
101
102 /**
103  * \brief    Returns the locally stored radio configuration.
104  * \retval   A pointer to the local radio configuration
105  */
106 IMST_HCI_RADIO_CONFIG*    LoRaNet_GetConfig(){
107     return &LoRaNet_radioConfig;
108 }
109
110 /**
111  * \brief    Send a packet to the lower layer. To reach a certain node, this
112  *               layer need to know him before to call this function.
113  * \retval   void
114  */
115 void LoRaNet_Send(LoRaNet_Packet* packet){
116     IMST_HCI_MSG* toSend;
117     LoRaNet_Node* destination;
118     uint8_t index = 0;
119     uint8_t occurrence = 0;
120
121     switch(packet->type){
122     case LORANET_TYPE_DATA_UP:
123         if(LoRaNet_IsPresentInLog(packet, LORANET_UP)){
124             //We have sent this message recently..
125             break;
126         } else {
127             LoRaNet_AddInLog(packet, LORANET_UP);
128         }
129         //REPLACED ON 20.07
130         /*
131         while((destination  = LoRaNet_GetNum(packet->dstId, ++occurrence, LORANET_UP)) !=
    NULL){
132             //Do for all occurrences of this id in uplink...
133             toSend = malloc(sizeof(IMST_HCI_MSG));
134             toSend->sapID = IMST_HCI_RADIOLINK_ID;
135             toSend->msgID = IMST_HCI_RADIOLINK_SEND_U_DATA_REQ;
136             toSend->dstGrp = destination->groupId;
137             toSend->dstAdr = destination->address;
138             toSend->srcAdr = LoRaNet_myNode->address;
139             toSend->srcGrp = LoRaNet_myNode->groupId;
140             toSend->payloadLength = packet->data.length + LORANET_OFFSET_DATA;
141             toSend->payload = malloc(packet->data.length + LORANET_OFFSET_DATA);
142             while(index < packet->data.length){
143                 toSend->payload[index + LORANET_OFFSET_DATA] = packet->data.begin[index];
144                 index++;
145             }
146             toSend->payload[LORANET_OFFSET_TYPE] = packet->type;
147             toSend->payload[LORANET_OFFSET_DST] = packet->dstId;
148             toSend->payload[LORANET_OFFSET_SRC] = packet->srcId;
149             toSend->payload[LORANET_OFFSET_ID] = packet->packetId;
150             //Free space...
151             free(packet->data.begin); //Data sended by the application
```

```
152             free(packet);                //Packet
153             //Send
154             HCI_Send(toSend);
155         }
156         */
157         while(occurrence < LoRaNet_uplink_cnt){
158             destination = LoRaNet_uplink[occurrence++];
159             toSend = malloc(sizeof(IMST_HCI_MSG));
160             toSend->sapID = IMST_HCI_RADIOLINK_ID;
161             toSend->msgID = IMST_HCI_RADIOLINK_SEND_U_DATA_REQ;
162             toSend->dstGrp = destination->groupId;
163             toSend->dstAdr = destination->address;
164             toSend->srcAdr = LoRaNet_myNode->address;
165             toSend->srcGrp = LoRaNet_myNode->groupId;
166             toSend->payloadLength = packet->data.length + LORANET_OFFSET_DATA;
167             toSend->payload = malloc(packet->data.length + LORANET_OFFSET_DATA);
168             index = 0;
169             while(index < packet->data.length){
170                 toSend->payload[index + LORANET_OFFSET_DATA] = packet->data.begin[index];
171                 index++;
172             }
173             toSend->payload[LORANET_OFFSET_TYPE] = packet->type;
174             toSend->payload[LORANET_OFFSET_DST] = packet->dstId;
175             toSend->payload[LORANET_OFFSET_SRC] = packet->srcId;
176             toSend->payload[LORANET_OFFSET_ID] = packet->packetId;
177             /*
178              * MOVED ON 26.07
179             //Free space..
180             free(packet->data.begin); //Data sended by the application
181             free(packet);                //Packet
182             */
183             //Send
184             HCI_Send(toSend);
185         }
186         break;
187     case LORANET_TYPE_DATA_DOWN:
188         while((destination  = LoRaNet_GetNum(packet->dstId, ++occurrence, LORANET_DOWN))
    != NULL){
189             //Do for all occurrences of this id in uplink...
190             toSend = malloc(sizeof(IMST_HCI_MSG));
191             toSend->sapID = IMST_HCI_RADIOLINK_ID;
192             toSend->msgID = IMST_HCI_RADIOLINK_SEND_U_DATA_REQ;
193             toSend->dstGrp = destination->groupId;
194             toSend->dstAdr = destination->address;
195             toSend->srcAdr = LoRaNet_myNode->address;
196             toSend->srcGrp = LoRaNet_myNode->groupId;
197             toSend->payloadLength = packet->data.length + LORANET_OFFSET_DATA;
198             toSend->payload = malloc(packet->data.length + LORANET_OFFSET_DATA);
199             index = 0;
200             while(index < packet->data.length){
201                 toSend->payload[index + LORANET_OFFSET_DATA] = packet->data.begin[index];
202                 index++;
203             }
204             toSend->payload[LORANET_OFFSET_TYPE] = packet->type;
205             toSend->payload[LORANET_OFFSET_DST] = packet->dstId;
206             toSend->payload[LORANET_OFFSET_SRC] = packet->srcId;
207             toSend->payload[LORANET_OFFSET_ID] = packet->packetId;
208             /*
209              * MOVED ON 26.07
210              *
211             //Free space...
212             free(packet->data.begin); //Data sended by the application
213             free(packet);                //Packet
214             */
215             //Send
216             HCI_Send(toSend);
217         }
218         break;
219     case LORANET_TYPE_PING:
220         toSend = malloc(sizeof(IMST_HCI_MSG));
221         toSend->sapID = IMST_HCI_RADIOLINK_ID;
222         toSend->msgID = IMST_HCI_RADIOLINK_SEND_U_DATA_REQ;
223         toSend->dstGrp = IMST_HCI_BROADCAST_GROUP;
224         toSend->dstAdr = IMST_HCI_BROADCAST_ADDRESS;
225         toSend->srcAdr = LoRaNet_myNode->address;
226         toSend->srcGrp = LoRaNet_myNode->groupId;
227         toSend->payloadLength = LORANET_OFFSET_DATA;
228         toSend->payload = malloc(toSend->payloadLength);
229         toSend->payload[LORANET_OFFSET_TYPE] = packet->type;
230         toSend->payload[LORANET_OFFSET_DST] = packet->dstId;
231         toSend->payload[LORANET_OFFSET_SRC] = packet->srcId;
232         toSend->payload[LORANET_OFFSET_ID] = packet->packetId;
233         /*
234          * MOVED ON 26.07
```

```
235             *
236             //Free space...
237             free(packet->data.begin); //Data sended by the application
238             free(packet);                 //Packet
239             */
240             //Send
241             HCI_Send(toSend);
242             break;
243         case LORANET_TYPE_PONG:
244             toSend = malloc(sizeof(IMST_HCI_MSG));
245             toSend->sapID = IMST_HCI_RADIOLINK_ID;
246             toSend->msgID = IMST_HCI_RADIOLINK_SEND_U_DATA_REQ;
247             toSend->dstGrp = LoRaNet_GetFirst(packet->dstId, LORANET_UP)->groupId;
248             toSend->dstAdr = LoRaNet_GetFirst(packet->dstId, LORANET_UP)->address;
249             toSend->srcAdr = LoRaNet_myNode->address;
250             toSend->srcGrp = LoRaNet_myNode->groupId;
251             toSend->payloadLength = LORANET_OFFSET_DATA;
252             toSend->payload = malloc(toSend->payloadLength);
253             toSend->payload[LORANET_OFFSET_TYPE] = packet->type;
254             toSend->payload[LORANET_OFFSET_DST] = packet->dstId;
255             toSend->payload[LORANET_OFFSET_SRC] = packet->srcId;
256             toSend->payload[LORANET_OFFSET_ID] = packet->packetId;
257             /*
258              * MOVED ON 26.07
259              *
260             //Free space...
261             free(packet->data.begin); //Data sended by the application
262             free(packet);                 //Packet
263             */
264             //Send
265             HCI_Send(toSend);
266             break;
267         default:
268             break;
269         }
270         //Free space...
271         free(packet->data.begin); //Data sended by the application
272         free(packet);                 //Packet
273     }
274
275     /**
276      * \brief    Return a number on 8 bits. Every time this function is called, the
277      *               number is incremented.
278      * \retval   The number on 8 bits.
279      */
280     uint8_t LoRaNet_GetPacketID(void){
281         static uint8_t packetId = 0;
282         packetId += 1;
283         return packetId;
284     }
285
286     /**
287      * \brief    Transform an IMST_HCI_MSG to a new packet. This function conserves
288      *               the parameter.
289      * \param    msg :    A pointer on the message to transform
290      * \retval   A pointer to the newly created packet
291      */
292     LoRaNet_Packet* LoRaNet_MakePacket(IMST_HCI_MSG* msg){
293         LoRaNet_Packet* toReturn;
294         uint8_t index = 0;
295         //Create packet
296         toReturn = malloc(sizeof(LoRaNet_Packet));
297         //Reserve space for the datas
298         toReturn->data.length = msg->payloadLength - LORANET_OFFSET_DATA;
299         toReturn->data.begin = malloc(toReturn->data.length);
300         //Copy datas
301         while(index < toReturn->data.length){
302             toReturn->data.begin[index] = msg->payload[LORANET_OFFSET_DATA + index];
303             index++;
304         }
305         //Copy others attributes
306         toReturn->dstId = msg->payload[LORANET_OFFSET_DST];
307         toReturn->packetId = msg->payload[LORANET_OFFSET_ID];
308         toReturn->srcId = msg->payload[LORANET_OFFSET_SRC];
309         toReturn->type = msg->payload[LORANET_OFFSET_TYPE];
310         //Return
311         return toReturn;
312     }
313
314     /**
315      * \brief    Build a new packet with a given type and with the given
316      *               destination.
317      * \param    type :           The type of the packet
318      * \param    destination :    The id of the destination
```

```
319     * \retval   A pointer to the new packet
320     */
321    LoRaNet_Packet* LoRaNet_MakePingPongPacket(uint8_t type, uint8_t destination){
322        LoRaNet_Packet* toReturn;
323        //Create packet
324        toReturn = malloc(sizeof(LoRaNet_Packet));
325        toReturn->data.length = 0;
326        toReturn->data.begin = malloc(toReturn->data.length);
327        toReturn->dstId = destination;
328        toReturn->packetId = LoRaNet_GetPacketID();
329        toReturn->srcId = LoRaNet_myNode->nodeId;
330        toReturn->type = type;
331        return toReturn;
332    }
333
334    /**
335     * \brief    Build a new packet containing a ping message with the broadcast id.
336     * \retval                   a pointer to the new packet
337     */
338    /*LoRaNet_Packet* LoRaNet_MakePingPacket(uint8_t destination){
339        LoRaNet_Packet* toReturn;
340        //Create packet
341        toReturn = malloc(sizeof(LoRaNet_Packet));
342        toReturn->data.length = 0;
343        toReturn->data.begin = malloc(toReturn->data.length);
344        toReturn->dstId = destination;
345        toReturn->packetId = LoRaNet_getPacketID();
346        toReturn->srcId = LoRaNet_myNode->nodeId;
347        toReturn->type = LORANET_TYPE_PING;
348        return toReturn;
349    }*/
350
351    /**
352     * \brief    Build a new LoRaNet node from an IMST_HCI_MSG. This function
353     *               conserves the given parameter.
354     * \retval   A pointer to the newly created node
355     */
356    LoRaNet_Node* LoRaNet_MakeNode(IMST_HCI_MSG* msg){
357        LoRaNet_Node* toReturn;
358        toReturn = malloc(sizeof(LoRaNet_Node));
359        toReturn->address = msg->srcAdr;
360        toReturn->groupId = msg->srcGrp;
361        toReturn->nodeId = msg->payload[LORANET_OFFSET_SRC];
362        toReturn->confirmed = false;
363        return toReturn;
364    }
365
366    /**
367     * \brief    This function will send an element from txQueue. The upper layer
368     *               should call it periodically.
369     * \retval   void
370     */
371    void LoRaNet_Proceed(){
372        if(!LoRaNet_IsTxQueueEmpty()){
373            LoRaNet_Send(LoRaNet_PopTxQueue());
374            //MODIFIED ON 2017.08.10
375            //Time_wait(1000);
376            Time_LaunchTimeout(1000);
377            while(!Time_TimeoutOccured()){
378                SerialDevice_Proceed();
379            }
380            //END OF MODIFICATION
381        }
382    }
383
384    /**
385     * \brief    This function register the received configuration in radioConfig
386     * \retval   void
387     */
388    void LoRaNet_RegConfig(IMST_HCI_MSG* msg){
389        LoRaNet_radioConfig.radioMode = msg->payload[IMST_HCI_RCF_OFFSET_RADIO_MODE];
390        LoRaNet_radioConfig.groupAddress = msg->payload[IMST_HCI_RCF_OFFSET_GROUP_ADDRESS];
391        LoRaNet_radioConfig.txGroupAddress = msg->payload[IMST_HCI_RCF_OFFSET_TX_GROUP_ADDRESS];
392        LoRaNet_radioConfig.deviceAddress = msg->payload[IMST_HCI_RCF_OFFSET_DEVICE_ADDRESS];
393        LoRaNet_radioConfig.deviceAddress |= msg->payload[IMST_HCI_RCF_OFFSET_DEVICE_ADDRESS +
     1] << 8;
394        LoRaNet_radioConfig.txDeviceAddress = msg->payload[
    IMST_HCI_RCF_OFFSET_TX_DEVICE_ADDRESS];
395        LoRaNet_radioConfig.txDeviceAddress |= msg->payload[
    IMST_HCI_RCF_OFFSET_TX_DEVICE_ADDRESS + 1] << 8;
396        LoRaNet_radioConfig.modulation = msg->payload[IMST_HCI_RCF_OFFSET_MODULATION];
397        LoRaNet_radioConfig.carrierFrequency = msg->payload[
    IMST_HCI_RCF_OFFSET_RF_CARRIER_FREQ_LSB];
398        LoRaNet_radioConfig.carrierFrequency |= msg->payload[
```

```
    IMST_HCI_RCF_OFFSET_RF_CARRIER_FREQ_IMB] << 8;
399     LoRaNet_radioConfig.carrierFrequency |= msg->payload[
    IMST_HCI_RCF_OFFSET_RF_CARRIER_FREQ_MSB] << 16;
400     LoRaNet_radioConfig.bandwidth = msg->payload[IMST_HCI_RCF_OFFSET_LORA_BW];
401     LoRaNet_radioConfig.spreadingFactor = msg->payload[IMST_HCI_RCF_OFFSET_LORA_SF];
402     LoRaNet_radioConfig.errorCoding = msg->payload[IMST_HCI_RCF_OFFSET_LORA_RC];
403     LoRaNet_radioConfig.powerLevel = msg->payload[IMST_HCI_RCF_OFFSET_POWER_LEVEL];
404     LoRaNet_radioConfig.txControl = msg->payload[IMST_HCI_RCF_OFFSET_TX_CONTROL];
405     LoRaNet_radioConfig.rxControl = msg->payload[IMST_HCI_RCF_OFFSET_RX_CONTROL];
406     LoRaNet_radioConfig.rxWindowTime = msg->payload[IMST_HCI_RCF_OFFSET_RX_WINDOW_TIME];
407     LoRaNet_radioConfig.rxWindowTime |= msg->payload[IMST_HCI_RCF_OFFSET_RX_WINDOW_TIME +
    1] << 8;
408     LoRaNet_radioConfig.ledControl = msg->payload[IMST_HCI_RCF_OFFSET_LED_CONTROL];
409     LoRaNet_radioConfig.miscOptions = msg->payload[IMST_HCI_RCF_OFFSET_MISC_OPTIONS];
410     LoRaNet_radioConfig.fskDataRate = msg->payload[IMST_HCI_RCF_OFFSET_FSK_DATARATE];
411     LoRaNet_radioConfig.powerSavingMode = msg->payload[
    IMST_HCI_RCF_OFFSET_POWER_SAVING_MODE];
412     LoRaNet_radioConfig.lbtThreshold = msg->payload[IMST_HCI_RCF_OFFSET_LBT_THRESHOLD];
413     LoRaNet_isInit = true;
414 }
415
416 /**
417  * \brief    This function will be passed to the lower layer and will be called
418  *                  when a message has arrived.
419  * \retval   void
420  */
421 void LoRaNet_RxCallBack(IMST_HCI_MSG* msg){
422     LoRaNet_Node*    toRegister;
423     LoRaNet_Packet*  toForward;
424
425     switch(msg->sapID){
426     case IMST_HCI_DEVMGMT_ID:
427         switch(msg->msgID){
428         case IMST_HCI_DEVMGMT_PING_RSP:
429             break;
430         case IMST_HCI_DEVMGMT_GET_DEVICE_INFO_RSP:
431             break;
432         case IMST_HCI_DEVMGMT_GET_FW_INFO_RSP:
433             break;
434         case IMST_HCI_DEVMGMT_RESET_RSP:
435             break;
436         case IMST_HCI_DEVMGMT_SET_OPMODE_RSP:
437             break;
438         case IMST_HCI_DEVMGMT_GET_OPMODE_RSP:
439             break;
440         case IMST_HCI_DEVMGMT_SET_RTC_RSP:
441             break;
442         case IMST_HCI_DEVMGMT_GET_RTC_RSP:
443             break;
444         case IMST_HCI_DEVMGMT_SET_RADIO_CONFIG_RSP:
445             if(msg->formatStat == IMST_HCI_DEVMGMT_STATUS_OK){
446                 //Update config : send "Get radio config"
447                 LoRaNet_ReqConfig();
448             } else {
449                 //Re-send config : send "Set radio config"
450                 LoRaNet_SendConfig();
451             }
452             break;
453         case IMST_HCI_DEVMGMT_GET_RADIO_CONFIG_RSP:
454             if(msg->formatStat == IMST_HCI_DEVMGMT_STATUS_OK){
455                 //Register the received configuration
456                 LoRaNet_RegConfig(msg);
457             } else {
458                 //Re-send request : send "Request radio config"
459                 LoRaNet_ReqConfig();
460             }
461             break;
462         case IMST_HCI_DEVMGMT_RESET_RADIO_CONFIG_RSP:
463             break;
464         case IMST_HCI_DEVMGMT_GET_SYSTEM_STATUS_RSP:
465             break;
466         case IMST_HCI_DEVMGMT_SET_RADIO_MORE_RSP:
467             break;
468         case IMST_HCI_DEVMGMT_MSG_SET_PSV_MODE_RSP:
469             break;
470         case IMST_HCI_DEVMGMT_MSG_POWER_UP_IND:
471             break;
472         case IMST_HCI_DEVMGMT_MSG_SET_AES_KEY_RSP:
473             break;
474         case IMST_HCI_DEVMGMT_MSG_GET_AES_KEY_RSP:
475             break;
476         default:
477             //Unknown MSG Id
478             break;
```

```
479           }
480       case IMST_HCI_RADIOLINK_ID :
481           switch(msg->msgID){
482           case IMST_HCI_RADIOLINK_SEND_U_DATA_RSP :
483               //Message sent.. :  Something is already made with that
484               break;
485           case IMST_HCI_RADIOLINK_U_DATA_RX_IND :
486   //*****************************************************************************
487   //
488   // TEST 07 Specific Lines : Software filter
489   //
490   //*****************************************************************************
491   #ifdef __TEST_07__
492   #ifdef __NODE_01__
493               if(msg->srcAdr == 0x4567){
494                   break;   //The node 1 can't hear the node 4
495               }
496   #endif //__NODE_01__
497   #ifdef __NODE_02__
498               if(msg->srcAdr == 0x4567){
499                   break;   //The node 2 can't hear the node 4
500               }
501   #endif //__NODE_02__
502   #ifdef __NODE_03__
503                           //The node 3 can hear every node
504   #endif //__NODE_03__
505   #ifdef __NODE_04__
506               if(msg->srcAdr != 0x3456){
507                   break;
508               }
509   #endif //__NODE_04__
510   #endif //__TEST_07__
511   //*****************************************************************************
512               switch(msg->payload[LORANET_OFFSET_TYPE]){//switch type
513               case LORANET_TYPE_DATA_UP:
514                   toForward = LoRaNet_MakePacket(msg);
515                   //Register the node if we don't know it
516                   toRegister = LoRaNet_MakeNode(msg);
517                   LoRaNet_Register(toRegister, LORANET_DOWN);
518                   // -> Look if we are the destination of this message
519                   if(msg->payload[LORANET_OFFSET_DST] == LoRaNet_myNode->nodeId){
520                       LoRaNet_CallBack(toForward);
521                   } else {
522                       //Forward the message
523                       LoRaNet_PushTxQueue(toForward);
524                   }
525                   break;
526               case LORANET_TYPE_DATA_DOWN:
527                   // -> Look if we know the source of this message
528                   toRegister = LoRaNet_MakeNode(msg);
529                   if(!LoRaNet_IsRegistered(toRegister, LORANET_UP)){
530                       LoRaNet_Register(toRegister, LORANET_UP);
531                   } else {
532                       free(toRegister);
533                   }
534                   if(msg->payload[LORANET_OFFSET_DST] == LoRaNet_myNode->nodeId){
535
536                       //We are the final destination
537                       toForward = LoRaNet_MakePacket(msg);
538                       LoRaNet_CallBack(toForward);
539                   } else {
540
541                       //Forward the message
542                       toRegister = LoRaNet_MakeNode(msg);
543                       if(LoRaNet_IsRegistered(toRegister, LORANET_DOWN)){
544
545                           //Forward the message
546                           toForward = LoRaNet_MakePacket(msg);
547                           LoRaNet_PushTxQueue(toForward);
548                       } else {
549                           //We don't know the destination Id => Forget this message
550                       }
551                   }
552                   break;
553
554               case LORANET_TYPE_PING:
555
556                   //-> Prepare the node
557                   toRegister = LoRaNet_MakeNode(msg);
558
559                   if(msg->payload[LORANET_OFFSET_DST] == LORANET_ID_BROADCAST){
      //Broadcasted Ping
560
561                       if(LoRaNet_IsRegistered(toRegister, LORANET_DOWN)){//Is in down
```

```
562                         free(toRegister);
563                     } else {
564                         if(LoRaNet_IsRegistered(toRegister, LORANET_UP)){//Is in up
565 
566                             // ->Send a pong
567                             toForward = LoRaNet_MakePingPongPacket(LORANET_TYPE_PONG, msg
    ->payload[LORANET_OFFSET_SRC]);
568                             LoRaNet_PushTxQueue(toForward);
569 
570                             if(!LoRaNet_GetNode(toRegister, LORANET_UP)->confirmed){
571                                 LoRaNet_GetNode(toRegister, LORANET_UP)->confirmed = true;
572                             }
573 
574                             //->Send a broadcasted Ping
575                             toForward = LoRaNet_MakePingPongPacket(LORANET_TYPE_PING,
    LORANET_ID_BROADCAST);
576                             LoRaNet_PushTxQueue(toForward);
577                         } else {
578 
579                             // ->Register the node in uplink
580                             LoRaNet_Register(toRegister, LORANET_UP);
581 
582                             // ->Send a pong
583                             toForward = LoRaNet_MakePingPongPacket(LORANET_TYPE_PONG, msg
    ->payload[LORANET_OFFSET_SRC]);
584                             LoRaNet_PushTxQueue(toForward);
585                         }
586                         /*
587                          * REMOVED ON 2017.08.23
588                         // ->Register the node in uplink
589                         LoRaNet_Register(toRegister, LORANET_UP);
590 
591                         // ->Respond with a pong
592                         toForward = LoRaNet_MakePingPongPacket(LORANET_TYPE_PONG,
    msg->payload[LORANET_OFFSET_SRC]);
593                         LoRaNet_PushTxQueue(toForward);
594 
595                         if(!LoRaNet_GetNode(toRegister, LORANET_UP)->confirmed){
596                             LoRaNet_GetNode(toRegister, LORANET_UP)->confirmed = true;
597                             //->Send a broadcasted Ping
598                             toForward = LoRaNet_MakePingPongPacket(LORANET_TYPE_PING,
    LORANET_ID_BROADCAST);
599                             LoRaNet_PushTxQueue(toForward);
600                         } else {// If the node is registered but not confirmed
601                         }
602                         */
603                     }
604 
605                 } else if(msg->payload[LORANET_OFFSET_DST] == LoRaNet_myNode->nodeId){
    //Personal Ping
606                     if(LoRaNet_IsRegistered(toRegister, LORANET_UP)){
607                         if(LoRaNet_GetNode(toRegister, LORANET_UP)->confirmed){
608 
609                             //->Ping
610                             toForward = LoRaNet_MakePingPongPacket(LORANET_TYPE_PING,
    LORANET_ID_BROADCAST);
611                             LoRaNet_PushTxQueue(toForward);
612 
613                             break;
614                         } else {
615                             // ->Confirm the node
616                             LoRaNet_GetNode(toRegister, LORANET_UP)->confirmed = true;
617     #ifdef __DEBUG__
618                             printf("confirm node %i\n", msg->payload[LORANET_OFFSET_SRC]);
619     #endif // __DEBUG__
620                             free(toRegister);
621                         }
622                     }
623 
624                     // ->Respond with a pong
625                     toForward = LoRaNet_MakePingPongPacket(LORANET_TYPE_PONG, msg->
    payload[LORANET_OFFSET_SRC]);
626                     LoRaNet_PushTxQueue(toForward);
627 
628                     //->Ping
629                     toForward = LoRaNet_MakePingPongPacket(LORANET_TYPE_PING,
    LORANET_ID_BROADCAST);
630                     LoRaNet_PushTxQueue(toForward);
631                 }
632                 break;
633 
634             case LORANET_TYPE_PONG:
635 
636                 //-> Prepare the node
```

```
637                     toRegister = LoRaNet_MakeNode(msg);
638
639                     if(msg->payload[LORANET_OFFSET_DST] == LoRaNet_myNode->nodeId){ //We are
    the destination
640                         if(LoRaNet_IsRegistered(toRegister, LORANET_UP)){ //The node is
    registered in uplink
641                             if(LoRaNet_GetNode(toRegister, LORANET_UP)->confirmed){ //The
    node is confirmed
642                                 // ABORT
643                                 free(toRegister);
644                                 free(msg->payload);
645                                 free(msg);
646                                 return;
647                             } else { //The node is registered but not confirmed in uplink
648                                 LoRaNet_Unregister(toRegister, LORANET_UP);
649                             }
650                         }
651                         if(LoRaNet_IsRegistered(toRegister, LORANET_DOWN)){
652                             if(LoRaNet_GetNode(toRegister, LORANET_DOWN)->confirmed){
653
654                                 // ->Send broadcasted Ping
655                                 //toForward = LoRaNet_MakePingPongPacket(LORANET_TYPE_PING,
    LORANET_ID_BROADCAST);
656                                 //LoRaNet_PushTxQueue(toForward);7
657                                 //COMMENT ON 08.14
658
659                             } else {
660
661                                 // ->Confirm the node
662                                 LoRaNet_GetNode(toRegister, LORANET_DOWN)->confirmed = true;
663 #ifdef __DEBUG__
664                                 printf("confirm node %i\n", msg->payload[LORANET_OFFSET_SRC]);
665 #endif // __DEBUG__
666                             }
667                             free(toRegister);
668
669                         } else {
670                             // ->Register node in downlink
671                             LoRaNet_Register(toRegister, LORANET_DOWN);
672
673                             // ->Respond with a Ping
674                             toForward = LoRaNet_MakePingPongPacket(LORANET_TYPE_PING, msg->
    payload[LORANET_OFFSET_SRC]);
675                             LoRaNet_PushTxQueue(toForward);
676                         }
677                     } else {
678                         free(toRegister);
679                     }
680                     break;
681                 default:
682                     // Unknown type of paquet
683                     break;
684                 }
685                 break;
686             default :
687                 // Unknown MSG Id
688                 break;
689             }
690         break;
691         default :
692             // Unknown SAP Id
693             break;
694         }
695         //Free parameter space..
696         free(msg->payload);
697         free(msg);
698 }
699
700 /**
701  * \brief    This function is useful to know if the node is currently connected
702  *               with another node in upstream direction.
703  *
704  * \retval   true : at lest one node is confirmed in uplink list
705  * \retval   false : there is no confirmed node in uplink list
706  */
707 bool LoRaNet_IsAnUplinkConfirmed(){
708     uint8_t          index = 0;   //Index to iterate trough the array
709     while(index < LoRaNet_uplink_cnt){
710         if(LoRaNet_uplink[index++]->confirmed){
711             return true;
712         }
713     }
714     return false;
715 }
```

```
716
717 /**
718  * \brief    This function test if a certain node is registered in a certain
719  *              list.
720  * \param    node :       The node to test
721  * \param    loranet_x :  Can be LORANET_UP or LORANTE_DOWN
722  */
723 bool LoRaNet_IsRegistered(LoRaNet_Node* node, uint8_t loranet_x){
724     uint8_t          index = 0;   //Index to iterate trough the array
725     uint8_t*     cnt;         //Pointer to the counter of the array
726     LoRaNet_Node**   ptr;         //Pointer to the array itself
727
728     switch(loranet_x){            //Initialize variables...
729     case LORANET_UP:
730         cnt = &LoRaNet_uplink_cnt;
731         ptr = LoRaNet_uplink;
732         break;
733     case LORANET_DOWN:
734         cnt = &LoRaNet_downlink_cnt;
735         ptr = LoRaNet_downLink;
736         break;
737     default:
738         return NULL;          //ERROR in the argument loranet_x
739     }
740
741     while(index < *cnt){      //Find the right id...
742         if(ptr[index]->nodeId == node->nodeId &&
743                 ptr[index]->address == node->address &&
744                 ptr[index]->groupId == node->groupId){
745             return true;
746         }
747         index++;
748     }
749     return false;
750 }
751
752 /**
753  *  \brief   This function returns the first occurrence of a certain node in a
754  *              certain list.
755  *  \param id :      The Id of the node to find
756  *  \param loranet_x :   Can be LORANET_UP or LORANET_DOWN
757  *  \retval A pointer to the node we are looking for
758  *  \retval NULL if the node doesn't appear in that list
759  */
760 LoRaNet_Node* LoRaNet_GetFirst(uint8_t id, uint8_t loranet_x){
761     return LoRaNet_GetNum(id, 1, loranet_x);
762 }
763
764 /**
765  *  \brief This function returns a certain node, in a certain list.
766  *  \param node :        The node we are looking for
767  *  \param loranet_x :   Can be LORANET_UP or LORANET_DOWN
768  */
769 LoRaNet_Node* LoRaNet_GetNode(LoRaNet_Node* node, uint8_t loranet_x){
770     uint8_t          index = 0;   //Index to iterate trough the array
771     uint8_t*     cnt;         //Pointer to the counter of the array
772     LoRaNet_Node**   ptr;         //Pointer to the array itself
773
774     switch(loranet_x){    //Initialize variables...
775     case LORANET_UP:
776         cnt = &LoRaNet_uplink_cnt;
777         ptr = LoRaNet_uplink;
778         break;
779     case LORANET_DOWN:
780         cnt = &LoRaNet_downlink_cnt;
781         ptr = LoRaNet_downLink;
782         break;
783     default:
784         return NULL;          //ERROR in the argument loranet__x
785     }
786
787     while(index < *cnt){      //Find the right id...
788         if(ptr[index]->nodeId == node->nodeId &&
789                 ptr[index]->address == node->address &&
790                 ptr[index]->groupId == node->groupId){
791             return ptr[index];
792         }
793         index++;
794     }
795     return NULL;
796 }
797
798 /**
799  * \brief    Return a pointer to the desired occurrence of a certain node
```

```
800     * \param    id :         id of the node to search
801     * \param    num :        occurrence number
802     * \param    loranet_x :  can be LORANET_UP or LORANET_DOWN
803     * \retval   NULL :       the id is not present or the occurrence is too high
804     */
805   LoRaNet_Node* LoRaNet_GetNum(uint8_t id, uint8_t num, uint8_t loranet_x){
806       uint8_t          index = 0;   //Index to iterate trough the array
807       uint8_t          occ = 0;     //Occurrence already found
808       uint8_t*     cnt;         //Pointer to the counter of the array
809       LoRaNet_Node**   ptr;         //Pointer to the array itself
810
811       switch(loranet_x){   //Initialize variables...
812       case LORANET_UP:
813           cnt = &LoRaNet_uplink_cnt;
814           ptr = LoRaNet_uplink;
815           break;
816       case LORANET_DOWN:
817           cnt = &LoRaNet_downlink_cnt;
818           ptr = LoRaNet_downLink;
819           break;
820       default:
821           return NULL;          //ERROR in the argument loranet__x
822       }
823
824       while(index < *cnt){      //Find the right id...
825           if(ptr[index]->nodeId == id){
826               occ++;
827               if(occ == num){       //It's the right occurrence
828                   return ptr[index];   //Return the first element
829               }
830           }
831           index++;
832       }
833       return NULL;
834   }
835
836   /**
837    *  \brief   This function registers a node in a certain list
838    *  \param node : The node to register
839    *  \param loranet_x :   Can be LORANET_UP or LORANET_DOWN
840    *  \retval void
841    */
842   void LoRaNet_Register(LoRaNet_Node* node, uint8_t loranet_x){
843       uint8_t          index = 0;   //Index to iterate trough the array
844       uint8_t*     cnt;         //Pointer to the counter of the array
845       LoRaNet_Node**   ptr;         //Pointer to the array itself
846
847       //ADDED ON 2017.08.08
848       //to avoid self-registering..
849       if(node->nodeId == LoRaNet_myNode->nodeId){
850           return;
851       }
852
853       switch(loranet_x){   //Initialize variables...
854       case LORANET_UP:
855           cnt = &LoRaNet_uplink_cnt;
856           ptr = LoRaNet_uplink;
857           break;
858       case LORANET_DOWN:
859           cnt = &LoRaNet_downlink_cnt;
860           ptr = LoRaNet_downLink;
861           break;
862       default:
863           return;                   //ERROR in the argument loranet__x
864       }
865
866       while(index < *cnt){      //Find the right id...
867           if(ptr[index]->nodeId == node->nodeId &&
868                   ptr[index]->address == node->address &&
869                   ptr[index]->groupId == node->groupId){
870               //REMOVED ON 2017.08.23
871               //free(node);
872               return;               //Already in queue...
873           }
874           index++;
875       }
876       if(index < LORANET_MAX_NODES){//If the array is not full
877           ptr[index] = node;        //Add it
878           switch(loranet_x){   //Initialize variables...
879           case LORANET_UP:
880   #ifdef __DEBUG__
881               printf("register node %i with address %02x%02x in uplink\n",
882                       node->nodeId, node->address >> 8, node->address &0xFF);
883   #endif // __DEBUG__
```

```
884             break;
885         case LORANET_DOWN:
886 #ifdef __DEBUG__
887             printf("register node %i with address %02x%02x in downlink\n",
888                     node->nodeId, node->address >> 8, node->address &0xFF);
889 #endif // __DEBUG__
890             break;
891         default:
892             break;
893         }
894         *cnt += 1;              //Update the counter
895     } else {
896         return;                 //Error : the array is full...
897     }
898 }
899
900 /**
901  *  \brief   This function unregister a certain node in a certain list.
902  *  \param node :        The node to unregister
903  *  \param loranet_x :   Can be LORANET_UP or LORANET_DOWN
904  */
905 void LoRaNet_Unregister(LoRaNet_Node* node, uint8_t loranet_x){
906     uint8_t          index = 0;          //Index to iterate trough the array
907     uint8_t*     cnt;                //Pointer to the counter of the array
908     LoRaNet_Node**   ptr;                //Pointer to the array itself
909     bool             removed = false; //True if a node has been removed
910
911     switch(loranet_x){   //Initialize variables...
912     case LORANET_UP:
913         cnt = &LoRaNet_uplink_cnt;
914         ptr = LoRaNet_uplink;
915         break;
916     case LORANET_DOWN:
917         cnt = &LoRaNet_downlink_cnt;
918         ptr = LoRaNet_downLink;
919         break;
920     default:
921         return;                 //ERROR in the argument loranet__x
922     }
923
924     while(index < *cnt){     //Find the right id...
925         if(ptr[index]->nodeId == node->nodeId &&
926                 ptr[index]->address == node->address &&
927                 ptr[index]->groupId == node->groupId){
928             free(ptr[index]);
929             cnt--;
930             removed = true;
931             switch(loranet_x){   //Initialize variables...
932             case LORANET_UP:
933 #ifdef __DEBUG__
934                 printf("unregister node %i with address %02x%02x from uplink\n",
935                         node->nodeId, node->address >> 8, node->address &0xFF);
936 #endif // __DEBUG__
937                 break;
938             case LORANET_DOWN:
939 #ifdef __DEBUG__
940                 printf("unregister node %i with address %02x%02x from downlink\n",
941                         node->nodeId, node->address >> 8, node->address &0xFF);
942 #endif // __DEBUG__
943                 break;
944             default:
945                 break;
946             }
947         }
948         if(removed){
949             ptr[index] = ptr[index + 1];
950         }
951         index++;
952     }
953     if(removed){
954         cnt--;
955     }
956 }
957
958 void LoRaNet_UnregisterAll(uint8_t id, uint8_t loranet_x){
959     uint8_t          index = 0;          //Index to iterate trough the array
960     uint8_t          removed = 0;     //Number of nodes unregisterd
961     uint8_t*     cnt;                //Pointer to the counter of the array
962     LoRaNet_Node**   ptr;                //Pointer to the array itself
963
964     switch(loranet_x){   //Initialize variables...
965     case LORANET_UP:
966         cnt = &LoRaNet_uplink_cnt;
967         ptr = LoRaNet_uplink;
```

```
968             break;
969         case LORANET_DOWN:
970             cnt = &LoRaNet_downlink_cnt;
971             ptr = LoRaNet_downLink;
972             break;
973         default:
974             return;                  //ERROR in the argument loranet__x
975         }
976
977         while (index < *cnt){        //Find the right id...
978             if(ptr[index]->nodeId == id){
979     #ifdef __DEBUG__
980                 switch(loranet_x){   //Initialize variables...
981                 case LORANET_UP:
982                     printf("unregister node %i with address %02x%02x from uplink\n",
983                             id, ptr[index]->address >> 8, ptr[index]->address &0xFF);
984                     break;
985                 case LORANET_DOWN:
986                     printf("unregister node %i with address %02x%02x from downlink\n",
987                             id, ptr[index]->address >> 8, ptr[index]->address &0xFF);
988                     break;
989                 default:
990                     break;
991                 }
992     #endif // __DEBUG__
993                 free(ptr[index]);
994                 removed++;
995             } else {                 //It's not the right Id
996                 ptr[index - removed] = ptr[index];
997             }
998             *cnt -= removed;
999             index++;
1000        }
1001    }
1002
1003    /**
1004     *  \brief  This function pop the Tx Queue.
1005     *  \retval The next packet to send
1006     */
1007    LoRaNet_Packet* LoRaNet_PopTxQueue(){
1008        uint8_t oldIndex = LoRaNet_tx_out;
1009        if(LoRaNet_IsTxQueueEmpty()){
1010            return NULL;//Error : queue is empty
1011        } else {
1012            //Update index
1013            LoRaNet_tx_out = (LoRaNet_tx_out + 1) % LORANET_QUEUE_SIZE;
1014            //Return value
1015            return LoRaNet_txQueue[oldIndex];
1016        }
1017    }
1018
1019    /**
1020     *  \brief This function push a packet in the Tx Queue.
1021     *  \param  packet :     The packet to push
1022     *  \retval void
1023     */
1024    void LoRaNet_PushTxQueue(LoRaNet_Packet* packet){
1025        if(LoRaNet_IsTxQueueFull()){
1026            return;//Error : queue is full
1027        } else {
1028            //Register value
1029            LoRaNet_txQueue[LoRaNet_tx_in] = packet;
1030            //Update index
1031            LoRaNet_tx_in = (LoRaNet_tx_in + 1) % LORANET_QUEUE_SIZE;
1032        }
1033    }
1034
1035    /**
1036     *  \brief  This function test if the Tx queue is empty.
1037     *  \retval true : The queue is empty
1038     *  \retval false : The queue is not empty
1039     */
1040    bool LoRaNet_IsTxQueueEmpty(){
1041        if(LoRaNet_tx_in == LoRaNet_tx_out){
1042            return true;
1043        } else {
1044            return false;
1045        }
1046    }
1047
1048    /**
1049     *  \brief  This function test if the Tx queue is full.
1050     *  \retval true : The queue is full
1051     *  \retval false : The queue is not full
```

```
1052    */
1053   bool LoRaNet_IsTxQueueFull(){
1054       if((LoRaNet_tx_in + 1) % LORANET_QUEUE_SIZE == LoRaNet_tx_out){
1055           return true;
1056       }
1057       return false;
1058   }
1059
1060   /**
1061    *  \brief   This function test if a packet is present in a log list
1062    *  \param packet :       The packet we are looking for
1063    *  \parma loranet_X :    The log-direction [NOT USED]
1064    */
1065   bool LoRaNet_IsPresentInLog(LoRaNet_Packet* packet, uint8_t loranet_x){
1066       uint8_t              index = 0;   //Index to iterate trough the array
1067       LoRaNet_Packet** ptr;         //Pointer to the array itself
1068
1069       /*
1070       switch(loranet_x){            //Initialize variables...
1071       case LORANET_UP:
1072       */
1073           ptr = LoRaNet_uplink_log;
1074           /*
1075           break;
1076       case LORANET_DOWN:
1077           //ptr = LoRaNet_downLink_log;
1078           break;
1079       default:
1080           return NULL;          //ERROR in the argument loranet__x
1081       }
1082       */
1083
1084       while(index < LoRaNet_uplink_log_in){
1085           if(ptr[index]->dstId == packet->dstId &&
1086                   ptr[index]->packetId == packet->packetId &&
1087                   ptr[index]->srcId == packet->srcId &&
1088                   ptr[index]->type == packet->type){
1089               return true;
1090           }
1091           index = (index + 1) % LORANET_MAX_LOG;
1092       }
1093       return false;
1094   }
1095
1096   /**
1097    *  \brief   This function add a packet in a certain log list
1098    *  \param packet :       The packet to add in the log list
1099    *  \param loranet_x :    The log-direction [NOT USED]
1100    */
1101   void LoRaNet_AddInLog(LoRaNet_Packet* packet, uint8_t loranet_x){
1102       /*
1103       switch(loranet_x){            //Initialize variables...
1104       case LORANET_UP:
1105       */
1106           free(LoRaNet_uplink_log[LoRaNet_uplink_log_in]); //Free the oldest packet
1107           LoRaNet_uplink_log[LoRaNet_uplink_log_in] = malloc(sizeof(LoRaNet_Packet));
1108           LoRaNet_uplink_log[LoRaNet_uplink_log_in]->dstId = packet->dstId;
1109           LoRaNet_uplink_log[LoRaNet_uplink_log_in]->packetId = packet->packetId;
1110           LoRaNet_uplink_log[LoRaNet_uplink_log_in]->srcId = packet->srcId;
1111           LoRaNet_uplink_log[LoRaNet_uplink_log_in]->type = packet->type;
1112           LoRaNet_uplink_log_in = (LoRaNet_uplink_log_in + 1) % LORANET_MAX_LOG;
1113           /*
1114           break;
1115       case LORANET_DOWN:
1116           // NOT USED...
1117           break;
1118       default:
1119           return;          //ERROR in the argument loranet__x
1120       }
1121       */
1122   }
1123
1124   /**
1125    *  \brief   This function set the corresponding attribute in the local radio
1126    *               configuration. Actually, the parameter is not check.
1127    *  \param radioMode :    The new value of the attribute
1128    *  \retval void
1129    */
1130   void LoRaNet_SetRadioMode(uint8_t radioMode){
1131       LoRaNet_radioConfig.radioMode = radioMode;
1132   }
1133
1134   /**
1135    *  \brief   This function set the corresponding attribute in the local radio
```

```
1136    *              configuration. Actually, the parameter is not check.
1137    *   \param groupAddress : The new value of the attribute
1138    *   \retval void
1139    */
1140   void LoRaNet_SetGroupAddress(uint8_t groupAddress){
1141       LoRaNet_radioConfig.groupAddress = groupAddress;
1142       LoRaNet_myNode->groupId = LoRaNet_radioConfig.groupAddress;
1143   }
1144
1145   /**
1146    *   \brief   This function set the corresponding attribute in the local radio
1147    *              configuration. Actually, the parameter is not check.
1148    *   \param deviceAddress :    The new value of the attribute
1149    *   \retval void
1150    */
1151   void LoRaNet_SetDeviceAddress(uint16_t deviceAddress){
1152       LoRaNet_radioConfig.deviceAddress = deviceAddress;
1153       LoRaNet_myNode->address = LoRaNet_radioConfig.deviceAddress;
1154   }
1155
1156   /**
1157    *   \brief   This function set the corresponding attribute in the local radio
1158    *              configuration. Actually, the parameter is not check.
1159    *   \param modulation :   The new value of the attribute
1160    *   \retval void
1161    */
1162   void LoRaNet_SetModulation(uint8_t modulation){
1163       LoRaNet_radioConfig.modulation = modulation;
1164   }
1165
1166   /**
1167    *   \brief   This function set the corresponding attribute in the local radio
1168    *              configuration. Actually, the parameter is not check.
1169    *   \param carrierFreq :  The new value of the attribute
1170    *   \retval void
1171    */
1172   void LoRaNet_SetCarrierFreq(uint32_t carrierFreq){
1173       LoRaNet_radioConfig.carrierFrequency = carrierFreq;
1174   }
1175
1176   /**
1177    *   \brief   This function set the corresponding attribute in the local radio
1178    *              configuration. Actually, the parameter is not check.
1179    *   \param bandwidth :    The new value of the attribute
1180    *   \retval void
1181    */
1182   void LoRaNet_SetBandwidth(uint8_t bandwidth){
1183       LoRaNet_radioConfig.bandwidth = bandwidth;
1184   }
1185
1186   /**
1187    *   \brief   This function set the corresponding attribute in the local radio
1188    *              configuration. Actually, the parameter is not check.
1189    *   \param spreadingFactor :  The new value of the attribute
1190    *   \retval void
1191    */
1192   void LoRaNet_SetSpreadingFactor(uint8_t spreadingFactor){
1193       LoRaNet_radioConfig.spreadingFactor = spreadingFactor;
1194   }
1195
1196   /**
1197    *   \brief   This function set the corresponding attribute in the local radio
1198    *              configuration. Actually, the parameter is not check.
1199    *   \param errorCoding :  The new value of the attribute
1200    *   \retval void
1201    */
1202   void LoRaNet_SetErrorCoding(uint8_t errorCoding){
1203       LoRaNet_radioConfig.errorCoding = errorCoding;
1204   }
1205
1206   /**
1207    *   \brief   This function set the corresponding attribute in the local radio
1208    *              configuration. Actually, the parameter is not check.
1209    *   \param powerLevel :   The new value of the attribute
1210    *   \retval void
1211    */
1212   void LoRaNet_SetPowerLevel(uint8_t powerLevel){
1213       LoRaNet_radioConfig.powerLevel = powerLevel;
1214   }
1215
1216   /**
1217    *   \brief   This function set the corresponding attribute in the local radio
1218    *              configuration. Actually, the parameter is not check.
1219    *   \param txControl :    The new value of the attribute
```

```
1220     *  \retval void
1221     */
1222    void LoRaNet_SetTxControl(uint8_t txControl){
1223        LoRaNet_radioConfig.txControl = txControl;
1224    }
1225
1226    /**
1227     *  \brief   This function set the corresponding attribute in the local radio
1228     *              configuration. Actually, the parameter is not check.
1229     *  \param rxControl :   The new value of the attribute
1230     *  \retval void
1231     */
1232    void LoRaNet_SetRxControl(uint8_t rxControl){
1233        LoRaNet_radioConfig.rxControl = rxControl;
1234    }
1235
1236    /**
1237     *  \brief   This function set the corresponding attribute in the local radio
1238     *              configuration. Actually, the parameter is not check.
1239     *  \param rxWindowTime : The new value of the attribute
1240     *  \retval void
1241     */
1242    void LoRaNet_SetRxWindowTime(uint16_t rxWindowTime){
1243        LoRaNet_radioConfig.rxWindowTime = rxWindowTime;
1244    }
1245
1246    /**
1247     *  \brief   This function set the corresponding attribute in the local radio
1248     *              configuration. Actually, the parameter is not check.
1249     *  \param ledControl :  The new value of the attribute
1250     *  \retval void
1251     */
1252    void LoRaNet_SetLedControl(uint8_t ledControl){
1253        LoRaNet_radioConfig.ledControl = ledControl;
1254    }
1255
1256    /**
1257     *  \brief   This function set the corresponding attribute in the local radio
1258     *              configuration. Actually, the parameter is not check.
1259     *  \param miscOpt : The new value of the attribute
1260     *  \retval void
1261     */
1262    void LoRaNet_SetMiscOpt(uint8_t miscOpt){
1263        LoRaNet_radioConfig.miscOptions = miscOpt;
1264    }
1265
1266    /**
1267     *  \brief   This function set the corresponding attribute in the local radio
1268     *              configuration. Actually, the parameter is not check.
1269     *  \param fskDataRate :  The new value of the attribute
1270     *  \retval void
1271     */
1272    void LoRaNet_SetFskDataRate(uint8_t fskDataRate){
1273        LoRaNet_radioConfig.fskDataRate = fskDataRate;
1274    }
1275
1276    /**
1277     *  \brief   This function set the corresponding attribute in the local radio
1278     *              configuration. Actually, the parameter is not check.
1279     *  \param lbtTreshold :  The new value of the attribute
1280     *  \retval void
1281     */
1282    void LoRaNet_SetLBTThresold(int16_t lbtThreshold){
1283        LoRaNet_radioConfig.lbtThreshold = lbtThreshold;
1284    }
1285
```

```
/**
  * \file        IMST_HCI.h
  * \author      Pierre Mendicino
  * \version 1.0
  * \date        2017.08.10
  * \brief       This file contains all the definition needed to communicate
  *              withe the iM880 series from IMST GmbH.
  */
#ifndef IMST_HCI_H_
#define IMST_HCI_H_

//*****************************************************************************
//
// Includes
//
//*****************************************************************************
#include <stdint.h>

//*****************************************************************************
//
// Constants
//
//*****************************************************************************
/// Frame Check Sequence field size
#define IMST_HCI_FCS_SIZE             2       ///< 2 bytes for CRC
/// Maximum HCI message payload size
#define IMST_HCI_MAX_PAY_SIZE         220     ///< TODO : Check that value
/// Header size : 1 byte for SAP ID, 1 byte for MSG ID
//
#define IMST_HCI_HEADER_SIZE          2       ///< - SAP Id (1 byte)
                                              ///< - MSG Id (1 byte)
/// Minimal HCI serialized frame (2*slip + 2*crc + sapID + msgID + status)
#define IMST_HCI_MINIMAL_SERIAL_SIZE  7       ///< - SLIP (2 bytes)
                                              ///< - FCS (2 bytes)
                                              ///< - SAP Id (1 byte)
                                              ///< - MSG Id (1 byte)
                                              ///< - Status (1 byte)
///Slip byte : at the begin and the end of the serialized HCI message
#define IMST_HCI_SLIP                     0xC0    ///< Actually 0xC0
/// Broadcast address
#define IMST_HCI_BROADCAST_ADDRESS        0xFFFF  ///< Actually 0xFFFF
/// Broadcast group Id
#define IMST_HCI_BROADCAST_GROUP      0xFF    ///< Actually 0xFF
//
// Device Management
// -> SAP ID
#define IMST_HCI_DEVMGMT_ID                             0x01
// -> MSG ID
#define IMST_HCI_DEVMGMT_PING_REQ                   0x01
#define IMST_HCI_DEVMGMT_PING_RSP                   0x02
#define IMST_HCI_DEVMGMT_GET_DEVICE_INFO_REQ        0x03
#define IMST_HCI_DEVMGMT_GET_DEVICE_INFO_RSP        0x04
#define IMST_HCI_DEVMGMT_GET_FW_INFO_REQ            0x05
#define IMST_HCI_DEVMGMT_GET_FW_INFO_RSP            0x06
#define IMST_HCI_DEVMGMT_RESET_REQ                      0x07
#define IMST_HCI_DEVMGMT_RESET_RSP                      0x08
#define IMST_HCI_DEVMGMT_SET_OPMODE_REQ                 0x09
#define IMST_HCI_DEVMGMT_SET_OPMODE_RSP                 0x0A
#define IMST_HCI_DEVMGMT_GET_OPMODE_REQ                 0x0B
#define IMST_HCI_DEVMGMT_GET_OPMODE_RSP                 0x0C
#define IMST_HCI_DEVMGMT_SET_RTC_REQ                0x0D
#define IMST_HCI_DEVMGMT_SET_RTC_RSP                0x0E
#define IMST_HCI_DEVMGMT_GET_RTC_REQ                0x0F
#define IMST_HCI_DEVMGMT_GET_RTC_RSP                0x10
#define IMST_HCI_DEVMGMT_SET_RADIO_CONFIG_REQ       0x11
#define IMST_HCI_DEVMGMT_SET_RADIO_CONFIG_RSP       0x12
#define IMST_HCI_DEVMGMT_GET_RADIO_CONFIG_REQ       0x13
#define IMST_HCI_DEVMGMT_GET_RADIO_CONFIG_RSP       0x14
#define IMST_HCI_DEVMGMT_RESET_RADIO_CONFIG_REQ         0x15
#define IMST_HCI_DEVMGMT_RESET_RADIO_CONFIG_RSP         0x16
#define IMST_HCI_DEVMGMT_GET_SYSTEM_STATUS_REQ      0x17
#define IMST_HCI_DEVMGMT_GET_SYSTEM_STATUS_RSP      0x18
#define IMST_HCI_DEVMGMT_SET_RADIO_MODE_REQ             0x19
#define IMST_HCI_DEVMGMT_SET_RADIO_MORE_RSP             0x1A
#define IMST_HCI_DEVMGMT_MSG_SET_PSV_MODE_REQ       0x1B    // Obsolete in v1.9
#define IMST_HCI_DEVMGMT_MSG_SET_PSV_MODE_RSP       0x1C    // Obsolete in v1.9
#define IMST_HCI_DEVMGMT_MSG_POWER_UP_IND           0x20    // Firmware v1.6
#define IMST_HCI_DEVMGMT_MSG_SET_AES_KEY_REQ        0x21    // Firmware v1.10
#define IMST_HCI_DEVMGMT_MSG_SET_AES_KEY_RSP        0x22    // Firmware v1.10
#define IMST_HCI_DEVMGMT_MSG_GET_AES_KEY_REQ        0x23    // Firmware v1.10
#define IMST_HCI_DEVMGMT_MSG_GET_AES_KEY_RSP        0x24    // Firmware v1.10
// -> RET VAL
#define IMST_HCI_DEVMGMT_STATUS_OK                      0x00
#define IMST_HCI_DEVMGMT_STATUS_ERROR               0x01
```

```
#define IMST_HCI_DEVMGMT_STATUS_CMD_NOT_SUPPORTED  0x02
#define IMST_HCI_DEVMGMT_STATUS_WRONG_PARAMETER        0x03
#define IMST_HCI_DEVMGMT_STATUS_WRONG_DEVICE_MODE  0x04
// -> PARAMS
#define IMST_HCI_DEVMGMT_RADIO_MODE_STD                0x00
#define IMST_HCI_DEVMGMT_RADIO_MODE_ECHO           0x01
#define IMST_HCI_DEVMGMT_RADIO_MODE_SNIFFER            0x02
// --
#define IMST_HCI_DEVMGMT_MODULATION_LORA           0x00
#define IMST_HCI_DEVMGMT_MODULATION_FSK                0x01
// --
#define IMST_HCI_DEVMGMT_LORA_BW_125K              0x00
#define IMST_HCI_DEVMGMT_LORA_BW_250K              0x01
#define IMST_HCI_DEVMGMT_LORA_BW_500K              0x02
// --
#define IMST_HCI_DEVMGMT_LORA_SF_7                     0x07
#define IMST_HCI_DEVMGMT_LORA_SF_8                     0x08
#define IMST_HCI_DEVMGMT_LORA_SF_9                     0x09
#define IMST_HCI_DEVMGMT_LORA_SF_10                    0x0A
#define IMST_HCI_DEVMGMT_LORA_SF_11                    0x0B
#define IMST_HCI_DEVMGMT_LORA_SF_12                    0x0C
// --
#define IMST_HCI_DEVMGMT_LORA_RC_4_5               0x01
#define IMST_HCI_DEVMGMT_LORA_RC_4_6               0x02
#define IMST_HCI_DEVMGMT_LORA_RC_4_7               0x03
#define IMST_HCI_DEVMGMT_LORA_RC_4_8               0x04
// --
// Formula :  2^19  * Fc =
//           -------
//          32*10^6
#define IMST_HCI_DEVMGMT_CARRIER_FREQ_868_5            0xD92000
// -- LBT : listen before talk, NF : narrow filter
#define IMST_HCI_DEVMGMT_TX_CONTROL_NOTHING            0x00
#define IMST_HCI_DEVMGMT_TX_CONTROL_NF             0x01
#define IMST_HCI_DEVMGMT_TX_CONTROL_LBT                0x02
#define IMST_HCI_DEVMGMT_TX_CONTROL_ALL                0x03
// --
#define IMST_HCI_DEVMGMT_RX_CONTROL_ALWAYS_OFF     0x00
#define IMST_HCI_DEVMGMT_RX_CONTROL_ALWAYS_ON      0x01
#define IMST_HCI_DEVMGMT_RX_CONTROL_ON_WINDOWS     0x02
// --
#define IMST_HCI_DEVMGMT_LED_CONTROL_NOTHING       0x00
#define IMST_HCI_DEVMGMT_LED_CONTROL_RX_D3         0x01
#define IMST_HCI_DEVMGMT_LED_CONTROL_TX_D2         0x02
#define IMST_HCI_DEVMGMT_LED_CONTROL_ALIVE_D4      0x04
// --
#define IMST_HCI_DEVMGMT_MISC_OPTIONS_NOTHING      0x00
#define IMST_HCI_DEVMGMT_MISC_OPTIONS_EXT_RF       0x01
#define IMST_HCI_DEVMGMT_MISC_OPTIONS_RTC          0x02
#define IMST_HCI_DEVMGMT_MISC_OPTIONS_TX_IND       0x04
#define IMST_HCI_DEVMGMT_MISC_OPTIONS_POWER_UP_IND 0x08
#define IMST_HCI_DEVMGMT_MISC_OPTIONS_BUTTON_IND   0x10
#define IMST_HCI_DEVMGMT_MISC_OPTIONS_AES          0x20
// --
#define IMST_HCI_DEVMGMT_FSK_DATARATE_50_KBPS      0x00
#define IMST_HCI_DEVMGMT_FSK_DATARATE_100_KBPS     0x01
#define IMST_HCI_DEVMGMT_FSK_DATARATE_250_KBPS     0x02
// -- ON -> requires rtc enabled !
#define IMST_HCI_DEVMGMT_POWER_SAVING_MODE_OFF     0x00
#define IMST_HCI_DEVMGMT_POWER_SAVING_MODE_ON      0x01
//
// RadioLink Test
// -> SAP ID
#define IMST_HCI_RLT_ID                                0x02
// -> MSG ID
#define IMST_HCI_RLT_START_REQ                         0x01
#define IMST_HCI_RLT_START_RSP                         0x02
#define IMST_HCI_RLT_STOP_REQ                      0x03
#define IMST_HCI_RLT_STOP_RSP                      0x04
#define IMST_HCI_RLT_STATUS_IND                        0x06
// -> RET VAL
#define IMST_HCI_RLT_STATUS_OK                         0x00
#define IMST_HCI_RLT_STATUS_ERROR                  0x01
#define IMST_HCI_RLT_STATUS_CMD_NOT_SUPPORTED      0x02
#define IMST_HCI_RLT_STATUS_WRONG_PARAMETER            0x03
#define IMST_HCI_RLT_STATUS_WRONG_RADIO_MODE       0x04
//
// RadioLink
// -> SAP ID
#define IMST_HCI_RADIOLINK_ID                      0x03
// -> MSG ID
#define IMST_HCI_RADIOLINK_SEND_U_DATA_REQ         0x01
#define IMST_HCI_RADIOLINK_SEND_U_DATA_RSP         0x02
#define IMST_HCI_RADIOLINK_U_DATA_RX_IND           0x04
```

```
169  #define IMST_HCI_RADIOLINK_U_DATA_TX_IND           0x06    // Firmware v1.6
170  #define IMST_HCI_RADIOLINK_RAW_DATA_RX_IND         0x08
171  #define IMST_HCI_RADIOLINK_SEND_C_DATA_REQ         0x09
172  #define IMST_HCI_RADIOLINK_SEND_C_DATA_RSP         0x0A
173  #define IMST_HCI_RADIOLINK_C_DATA_RX_IND           0x0C
174  #define IMST_HCI_RADIOLINK_C_DATA_TX_IND           0x0E
175  #define IMST_HCI_RADIOLINK_ACK_RX_IND              0x10
176  #define IMST_HCI_RADIOLINK_ACK_TIMEOUT_IND         0x12
177  #define IMST_HCI_RADIOLINK_ACK_TX_IND              0x14
178  #define IMST_HCI_RADIOLINK_SET_ACK_DATA_REQ            0x15
179  #define IMST_HCI_RADIOLINK_SET_ACK_DATA_RSP            0x16
180  // -> RET VAL
181  #define IMST_HCI_RADIOLINK_STATUS_OK               0x00
182  #define IMST_HCI_RADIOLINK_STATUS_ERROR                0x01
183  #define IMST_HCI_RADIOLINK_STATUS_CMD_NOT_SUPPORTED    0x02
184  #define IMST_HCI_RADIOLINK_STATUS_WRONG_PARAMETER  0x03
185  #define IMST_HCI_RADIOLINK_STATUS_WRONG_RADIO_MODE 0x04
186  #define IMST_HCI_RADIOLINK_STATUS_MEDIA_BUSY       0x05
187  #define IMST_HCI_RADIOLINK_STATUS_DEVICE_BUSY      0x06
188  #define IMST_HCI_RADIOLINK_STATUS_BUFFER_FULL      0x07
189  #define IMST_HCI_RADIOLINK_STATUS_LENGTH_ERROR     0x08
190  //
191  // Remote Control
192  // -> SAP ID
193  #define IMST_HCI_REM_ID                                0x04
194  // -> MSG ID
195  #define IMST_HCI_REM_BTN_PRESSED_IND               0x02
196
197  // Hardware Test
198  // -> SAP ID
199  #define IMST_HCI_HWTEST_ID                             0xA1
200  // -> MSG ID
201  #define IMST_HCI_HWTEST_RADIO_TEST_REQ             0x01
202  #define IMST_HCI_HWTEST_RADIO_TEST_RSP             0x02
203  // -> RET VAL
204  #define IMST_HCI_HWTEST_STATUS_OK                  0x00
205  #define IMST_HCI_HWTEST_STATUS_ERROR               0x01
206  #define IMST_HCI_HWTEST_STATUS_CMD_NOT_SUPPORTED   0x02
207  #define IMST_HCI_HWTEST_STATUS_WRONG_PARAMETER     0x03
208  //
209  // Device Information Element
210  #define IMST_HCI_DIE_OFFSET_MODULE_TYPE                0x00     // 1 byte
211  #define IMST_HCI_DIE_OFFSET_DEVICE_ADDRESS         0x01     // 2 byte
212  #define IMST_HCI_DIE_OFFSET_GROUP_ADDRESS          0x03     // 1 byte
213  #define IMST_HCI_DIE_OFFSET_RESERVED               0x04     // 1 byte
214  #define IMST_HCI_DIE_OFFSET_32_BIT_DEVICE_ID       0x05     // 4 bytes
215  //
216  // Firmware Information Element
217  #define IMST_HCI_FIE_OFFSET_MAJOR_FW_VERSION       0x00     // 1 byte
218  #define IMST_HCI_FIE_OFFSET_MINOR_FW_VERSION       0x01     // 1 byte
219  #define IMST_HCI_FIE_OFFSET_BUILD_COUNT                0x02     // 2 byte
220  #define IMST_HCI_FIE_OFFSET_FIRMWARE_IMAGE         0x04     // m bytes
221  //
222  // Radio Configuration Field
223  #define IMST_HCI_RCF_OFFSET_RADIO_MODE             0x00     // 1 byte
224  #define IMST_HCI_RCF_OFFSET_GROUP_ADDRESS          0x01     // 1 byte
225  #define IMST_HCI_RCF_OFFSET_TX_GROUP_ADDRESS       0x02     // 1 byte    NaN
226  #define IMST_HCI_RCF_OFFSET_DEVICE_ADDRESS         0x03     // 2 bytes
227  #define IMST_HCI_RCF_OFFSET_TX_DEVICE_ADDRESS      0x05     // 2 bytes   NaN
228  #define IMST_HCI_RCF_OFFSET_MODULATION             0x07     // 1 byte
229  #define IMST_HCI_RCF_OFFSET_RF_CARRIER_FREQ_LSB        0x08     // 1 byte
230  #define IMST_HCI_RCF_OFFSET_RF_CARRIER_FREQ_IMB        0x09     // 1 byte
231  #define IMST_HCI_RCF_OFFSET_RF_CARRIER_FREQ_MSB        0x0A     // 1 byte
232  #define IMST_HCI_RCF_OFFSET_LORA_BW                    0x0B     // 1 byte
233  #define IMST_HCI_RCF_OFFSET_LORA_SF                    0x0C     // 1 byte
234  #define IMST_HCI_RCF_OFFSET_LORA_RC                    0x0D     // 1 byte
235  #define IMST_HCI_RCF_OFFSET_POWER_LEVEL                0x0E     // 1 byte
236  #define IMST_HCI_RCF_OFFSET_TX_CONTROL             0x0F     // 1 byte
237  #define IMST_HCI_RCF_OFFSET_RX_CONTROL             0x10     // 1 byte
238  #define IMST_HCI_RCF_OFFSET_RX_WINDOW_TIME         0x11     // 2 bytes
239  #define IMST_HCI_RCF_OFFSET_LED_CONTROL                0x13     // 1 byte
240  #define IMST_HCI_RCF_OFFSET_MISC_OPTIONS           0x14     // 1 byte
241  #define IMST_HCI_RCF_OFFSET_FSK_DATARATE           0x15     // 1 byte
242  #define IMST_HCI_RCF_OFFSET_POWER_SAVING_MODE      0x16     // 1 byte v1.9
243  #define IMST_HCI_RCF_OFFSET_LBT_THRESHOLD          0x17     // 1 byte v1.10
244
245  //****************************************************************************
246  //
247  // Structures definitions
248  //
249  //****************************************************************************
250  /**
251   * \brief    This structure represent an HCI message. It can be a message that
252   *           will be sent or a received one. Please refer to the file IMST_HCI.h
```

```
253     */
254     typedef struct
255     {
256         /// Format field or status field depending if the msg is a TX or a RX msg
257         uint8_t      formatStat;      ///< This value can be for example :
    IMST_HCI_RADIOLINK_STATUS_OK
258         /// The Id of the SAP
259         uint8_t      sapID;           ///<
260         /// The Id of the message
261         uint8_t      msgID;           ///<
262         /// The group of the source
263         uint8_t      srcGrp;          ///< This value can't be :
264                                       ///< - 0x00 (Undefined)
265                                       ///< - 0xFF (Broadcast)
266         /// The physical address of the source
267         uint16_t srcAdr;          ///< This value can't be :
268                                       ///< - 0x0000 (Undefined)
269                                       ///< - 0xFFFF(Broadcast)
270         /// The group of the destination
271         uint8_t      dstGrp;      ///< This value can't be :
272                                       ///< - 0x00 (Undefined)
273                                       ///< - 0xFF (Broadcast)
274         /// The physical address of the destination
275         uint16_t dstAdr;          ///< This value can't be :
276                                       ///< - 0x0000 (Undefined)
277                                       ///< - 0xFFFF(Broadcast)
278         /// A pointer to the first byte of the data
279         uint8_t* payload;     ///< TODO : replace this with a byteBuffer_t
280         /// The length of the data in bytes
281         uint16_t payloadLength;   ///< TODO : replace this with a byteBuffer_t
282                                       ///< This value is not transmitted
283         /// Receive signal strength in dBm
284         uint16_t rssi;            ///< ~Optional
285         /// Signal to noise ratio in dB
286         uint8_t      snr;             ///< ~Optional
287         /// Timestamp from RTC
288         uint32_t rxTime;          ///< ~Optional
289     }IMST_HCI_MSG;
290
291     /**
292      * \brief    This structure represent the radio configuration for the IMST GmbH
293      *           LoRa module, series iM880. For more explanations, please refer to :
294      *           \n "AN1200.22 LoRa(TM) Modulation Basics", (c)Semtech 2015
295      *           \n "WiMOD LR Base Host Controller Interface", (c)IMST GmbH 2011
296      */
297     typedef struct
298     {
299         /// The radio mode in witch the module is.
300         uint8_t      radioMode;           ///< This value can be :
301                                           ///< - IMST_HCI_DEVMGMT_RADIO_MODE_STANDARD (Standard)
302                                           ///< - IMST_HCI_DEVMGMT_RADIO_MODE_ECHO (Echo)
303                                           ///< - IMST_HCI_DEVMGMT_RADIO_MODE_SNIFFER (Sniffer)
304         /// The address of the group in witch the device is.
305         uint8_t      groupAddress;        ///< This value can't be :
306                                           ///< - 0x00 (Undefined)
307                                           ///< - 0xFF (Broadcast)
308         /// Reserved for future use
309         uint8_t      txGroupAddress;      ///<
310         /// The physical address of the device.
311         uint16_t deviceAddress;       ///< This value can't be :
312                                           ///< - 0x0000 (Undefined)
313                                           ///< - 0xFFFF(Broadcast)
314         /// Reserved for future use
315         uint16_t txDeviceAddress; ///<
316         /// The type of modulation used by the device.
317         uint8_t      modulation;          ///< This value can be :
318                                           ///< - IMST_HCI_DEVMGMT_MODULATION_LORA (LoRa)
319                                           ///< - IMST_HCI_DEVMGMT_MODULATION_FSK (FSK, 50kbps)
320         /// The carrier Frequency used by the device.
321         uint32_t carrierFrequency;    ///< The formula to obtain the value is :
322                                           ///< 2^19  * Fc
323                                           ///< ________
324                                           ///<  32*10^6
325                                           ///< This value can be :
326                                           ///< - IMST_HCI_DEVMGMT_CARRIER_FREQ_868_5 (868.5MHz)
327         /// The bandwidth used by the device.
328         uint8_t      bandwidth;           ///< This value can be :
329                                           ///< - IMST_HCI_DEVMGMT_LORA_BW_125K (125kHz)
330                                           ///< - IMST_HCI_DEVMGMT_LORA_BW_250K (250kHz)
331                                           ///< - IMST_HCI_DEVMGMT_LORA_BW_500K (500kHz)
332         /// The spreading-factor used by the device.
333         uint8_t      spreadingFactor; ///< This value can be :
334                                           ///< - 0x00 -> 0x07 (SF = 7)
335                                           ///< - ...
```

```
336                                     ///< - 0x0C (SF = 12)
337     /// The error-coding value used by the device.
338     uint8_t      errorCoding;     ///< This value can be :
339                                     ///< - IMST_HCI_DEVMGMT_LORA_RC_4_5 (4/5)
340                                     ///< - IMST_HCI_DEVMGMT_LORA_RC_4_6 (4/6)
341                                     ///< - IMST_HCI_DEVMGMT_LORA_RC_4_7 (4/7)
342                                     ///< - IMST_HCI_DEVMGMT_LORA_RC_4_8 (4/8)
343     /// The transmit power-level used by the device.
344     uint8_t      powerLevel;        ///< This value can be :
345                                     ///< - 0x00 -> 0x05 (5dBm)
346                                     ///< - ...
347                                     ///< - 0x14 (20dBm)
348     /// Tx attributes.
349     uint8_t      txControl;         ///< This value can be :
350                                     ///< - IMST_HCI_DEVMGMT_TX_CONTROL_NOTHING (LBT and
    NF OFF)
351                                     ///< - IMST_HCI_DEVMGMT_TX_CONTROL_NF (Narrow filter
    ON)
352                                     ///< - IMST_HCI_DEVMGMT_TX_CONTROL_LBT
    (Listen-before-transmit ON) [From v1.10 extension]
353                                     ///< - IMST_HCI_DEVMGMT_TX_CONTROL_ALL (LBT and NF
    ON) [From 1.10 extension]
354     /// Rx attributes.
355     uint8_t      rxControl;         ///< This value can be :
356                                     ///< - IMST_HCI_DEVMGMT_RX_CONTROL_ALWAYS_OFF (Off)
357                                     ///< - IMST_HCI_DEVMGMT_RX_CONTROL_ALWAYS_ON (On)
358                                     ///< - IMST_HCI_DEVMGMT_RX_CONTROL_ON_WINDOWS (On on
    windows)
359     /// Time for radio receive-mode after transmit in milliseconds.
360     uint16_t rxWindowTime;        ///< This value can be :
361                                     ///< - 0x0000 (Rx off)
362                                     ///< - ...
363                                     ///< - 0xFFFF (65535ms)
364     /// Bit field to configure LED control options.
365     uint8_t      ledControl;        ///< - bit[0] 0 = no GPIO access, 1 = ld3
    "rxIndicator"
366                                     ///< - bit[1] 0 = no GPIO access, 1 = ld2
    "txIndicator"
367                                     ///< - bit[2] 0 = no GPIO access, 1 = ld4
    "aliveIndicator"
368                                     ///< - bit[3] 0 = no GPIO access, 1 = ld1
    "buttonPressedIndicator" [From v1.6 extension]
369     /// Bit field to configure further radio firmware options.
370     uint8_t      miscOptions;     ///< - bit[0] 0 = Standard RF packet format, 1 =
    Extended RF packet format
371                                     ///< - bit[1] 0 = RTC disabled, 1 = RTC enabled (RTC
    = real-time-clock)
372                                     ///< - bit[2] HCI Tx-indication : 0 = Disabled, 1 =
    Enabled [From v1.6 extensions]
373                                     ///< - bit[3] HCI power-up indication : 0 =
    Disabled, 1 = Enabled [From v1.6 extensions]
374                                     ///< - bit[4] HCI button pressed [From v1.6
    extensions]
375                                     ///< - bit[5] AES encryption : 0 = Off, 1 = On [From
    v1.10 extensions]
376     /// Determine the data-rate if FSK is enabled
377     uint8_t      fskDataRate;     ///< This value can be :
378                                     ///< - IMST_HCI_DEVMGMT_FSK_DATARATE_50_KBPS (50kbps)
379                                     ///< - IMST_HCI_DEVMGMT_FSK_DATARATE_100_KBPS
    (100kbps)
380                                     ///< - IMST_HCI_DEVMGMT_FSK_DATARATE_250_KBPS
    (250kbps)
381     /// Defines the power-saving mode
382     uint8_t      powerSavingMode; ///< [From firmware v1.9 extension] This value can be :
383                                     ///< - IMST_HCI_DEVMGMT_POWER_SAVING_MODE_OFF (Off)
384                                     ///< - IMST_HCI_DEVMGMT_POWER_SAVING_MODE_ON (auto)
    - require RTC On
385     /// Defines the LBT threshold in dBm
386     int16_t      lbtThreshold;      ///< [From firmware v1.10 extension] This value can
    be :
387                                     ///< - -120 to 0 (LSB first)
388 }IMST_HCI_RADIO_CONFIG;
389
390 #endif /* IMST_HCI_H_ */
391
```

```c
/**
  * \file        hci.h
  * \author      Pierre Mendicino
  * \version 1.0
  * \date        2017.08.10
  * \brief       This file contains the headers of the HCI layer
  */
#ifndef HCI_H_
#define HCI_H_

//*****************************************************************************
//
// Includes
//
//*****************************************************************************
#include <stdlib.h>
#include "serial_device.h"
#include "util/byteBuffer.h"
#include "util/crc16.h"
#include "util/IMST_HCI.h"
#include "util/time.h"
#include "_conf.h"

//*****************************************************************************
//
// Private variables
//
//*****************************************************************************
/**
 * \vargroup Private variable
 * \{
 */
/// The call-back function
void (*HCI_callBack)(IMST_HCI_MSG* msg);   ///< Set by the Init function
/// Attribute that say if the last message have been confirmed by the module
bool HCI_isConfirmed;                           ///< True if confirmed /*NEW*/
/**
 * \}
 */

//*****************************************************************************
//
// Public functions
//
//*****************************************************************************
/**
 * \fngroup Public function
 * \{
 */
void HCI_Init(void (*callBack)(IMST_HCI_MSG* msg));
bool HCI_Send(IMST_HCI_MSG* msg);
void HCI_RxCallBack(byteBuffer_t* buffer);
/**
 * \}
 */

//*****************************************************************************
//
// Private functions
//
//*****************************************************************************
/**
 * \fngroup Private function
 * \{
 */
bool HCI_WaitForConfirmation();               /*NEW*/
/**
 * \}
 */

#endif /* HCI_H_ */

```

```
1   /**
2     * \file        hci.c
3     * \author      Pierre Mendicino
4     * \version 1.0
5     * \date        2017.08.10
6     * \brief       This file contains the implementation of the HCI layer
7     */
8   #include "hci.h"
9
10  /**
11   * \brief    Initialize this layer. This function is intended to be called at
12   *               the startup, just after the lower layer initialization
13   * \param    callback :   The function this layer should call when he have to
14   *                        send some data.
15   * \retval   void
16   */
17  void HCI_Init(void (*callBack)(IMST_HCI_MSG* buffer)){
18      HCI_callBack = callBack;
19      HCI_isConfirmed = true;
20  }
21
22  /**
23   * \brief    Build the serialized message and send it to the lower layer. This
24   *                        function deletes the parameter when she returns.
25   * \param    msg :        the message to send
26   * \retval   true :   The RF module has confirmed that the message have been sent
27   *           false : Error with the message or the RF module doesn't respond
28   */
29  bool HCI_Send(IMST_HCI_MSG* msg){
30      uint8_t offset = 0;
31      uint8_t loop_int = 0;
32      uint16_t crc16;
33      uint8_t serializedSize;
34      byteBuffer_t* toSend = malloc(sizeof(byteBuffer_t));
35
36      // -> Serialize the message
37      switch(msg->sapID){
38      case IMST_HCI_DEVMGMT_ID:
39          switch(msg->msgID){
40          case IMST_HCI_DEVMGMT_SET_RADIO_CONFIG_REQ:
41              //Length = 2*slip + sapId + msgId + 2*crc = 6
42              serializedSize = msg->payloadLength + 6;
43              toSend->begin = malloc(serializedSize);
44              toSend->begin[offset++] = IMST_HCI_SLIP;
45              toSend->begin[offset++] = msg->sapID;
46              toSend->begin[offset++] = msg->msgID;
47              while(loop_int < msg->payloadLength){
48                  toSend->begin[offset++] = msg->payload[loop_int++];
49              }
50              //crc length : sapId + msgId = 2
51              crc16 = ~CRC16_Calc(&toSend->begin[1], msg->payloadLength + 2,
    CRC16_INIT_VALUE);
52              break;
53          case IMST_HCI_DEVMGMT_GET_RADIO_CONFIG_REQ:
54              //Length = 2*slip + sapId + msgId + 2*crc = 6
55              serializedSize = msg->payloadLength + 6;
56              toSend->begin = malloc(serializedSize);
57              toSend->begin[offset++] = IMST_HCI_SLIP;
58              toSend->begin[offset++] = msg->sapID;
59              toSend->begin[offset++] = msg->msgID;
60              //No payload
61              //crc length : sapId + msgId = 2
62              crc16 = ~CRC16_Calc(&toSend->begin[1], msg->payloadLength + 2,
    CRC16_INIT_VALUE);
63              break;
64          default:
65              return false;//Unsupported MSG ID
66          }
67          break;
68      case IMST_HCI_RADIOLINK_ID:
69          switch(msg->msgID){
70          case IMST_HCI_RADIOLINK_SEND_U_DATA_REQ:
71              //PayloadLength + 2*slip + 2*adr + grp + sapId + msgId + 2*crc
72              serializedSize = msg->payloadLength + 9;
73              toSend->begin = malloc(serializedSize);
74              toSend->begin[offset++] = IMST_HCI_SLIP;
75              toSend->begin[offset++] = msg->sapID;
76              toSend->begin[offset++] = msg->msgID;
77              toSend->begin[offset++] = msg->dstGrp;
78              toSend->begin[offset++] = msg->dstAdr & 0xFF;
79              toSend->begin[offset++] = (msg->dstAdr >> 8) & 0xFF;
80              while(loop_int < msg->payloadLength){
81                  toSend->begin[offset++] = msg->payload[loop_int++];
82              }
```

```
83              //crc length : payloadLength + grp + 2*adr + sapId + msgId
84              crc16 = ~CRC16_Calc(&toSend->begin[1], msg->payloadLength + 5,
    CRC16_INIT_VALUE);
85              HCI_isConfirmed = false;
86              break;
87          default:
88              return false;//Unsupported MSG ID
89          }
90          break;
91      default:
92          return false;//Unsupported SAP ID
93      }
94      toSend->begin[offset++] = crc16 & 0xFF;
95      toSend->begin[offset++] = (crc16 >> 8) & 0xFF;
96      toSend->begin[offset++] = IMST_HCI_SLIP;
97      toSend->length = serializedSize;//DEBUG : must be equal to serializedSize
98
99      //Free space
100     free(msg->payload);
101     free(msg);
102
103     //Send msg..
104     SerialDevice_SendByteBuffer(toSend);
105     return HCI_WaitForConfirmation();
106 }
107
108 /**
109  * \brief    Callback function called by the lower layer. This function build
110  *                      an IMST_HCI_MSG and sent it to the upper layer. This
111  *                      function deletes the parameter when whe returns.
112  * \param    data :      The data to send
113  * \retval   void
114  */
115 void HCI_RxCallBack(byteBuffer_t* buffer){
116     //Reserve space for the header
117     IMST_HCI_MSG* receivedMsg = malloc(sizeof(IMST_HCI_MSG));
118     //Iterator
119     uint8_t* bytePtr = buffer->begin;
120     uint8_t offset = 0;
121     uint16_t dataLength = 0;
122
123     //Verify minimal size
124     if(buffer->length < IMST_HCI_MINIMAL_SERIAL_SIZE){
125         free(buffer->begin);
126         free(buffer);
127         return;//Error : the frame is too short...
128     }
129     //Verify if the data begin with a slip character
130     if(buffer->begin[0] != IMST_HCI_SLIP){
131         free(buffer->begin);
132         free(buffer);
133         return;//Error : the frame is not correctly encoded
134     }
135     //Verify CRC
136     if(!CRC16_Check(++bytePtr, buffer->length - 2, CRC16_INIT_VALUE)){
137         free(buffer->begin);
138         free(buffer);
139         return;//Error : crc
140     }
141     //Build IMST msg...
142     receivedMsg->sapID = *bytePtr++;
143     receivedMsg->msgID = *bytePtr++;
144     receivedMsg->formatStat = *bytePtr++;
145     switch(receivedMsg->sapID){//Switches the SapId
146     case IMST_HCI_DEVMGMT_ID:
147         //sapID + msgID + status + 2 * slip + 2 * crc = 7
148         dataLength = buffer->length - 7;
149         receivedMsg->payloadLength = dataLength;
150         receivedMsg->payload = malloc(dataLength);
151         while(offset < dataLength){
152             receivedMsg->payload[offset++] = *bytePtr++;
153         }
154         HCI_callBack(receivedMsg);
155         //Nothing to do here..?
156         switch(receivedMsg->msgID){
157         case IMST_HCI_DEVMGMT_PING_RSP:
158         case IMST_HCI_DEVMGMT_GET_DEVICE_INFO_RSP:
159         case IMST_HCI_DEVMGMT_GET_FW_INFO_RSP:
160         case IMST_HCI_DEVMGMT_RESET_RSP:
161         case IMST_HCI_DEVMGMT_SET_OPMODE_RSP:
162         case IMST_HCI_DEVMGMT_GET_OPMODE_RSP:
163         case IMST_HCI_DEVMGMT_SET_RTC_RSP:
164         case IMST_HCI_DEVMGMT_GET_RTC_RSP:
165         case IMST_HCI_DEVMGMT_SET_RADIO_CONFIG_RSP:
```

```
166             case IMST_HCI_DEVMGMT_GET_RADIO_CONFIG_RSP:
167             case IMST_HCI_DEVMGMT_RESET_RADIO_CONFIG_RSP:
168             case IMST_HCI_DEVMGMT_GET_SYSTEM_STATUS_RSP:
169             case IMST_HCI_DEVMGMT_SET_RADIO_MORE_RSP:
170             case IMST_HCI_DEVMGMT_MSG_SET_PSV_MODE_RSP:
171             case IMST_HCI_DEVMGMT_MSG_POWER_UP_IND:
172             case IMST_HCI_DEVMGMT_MSG_SET_AES_KEY_RSP:
173             case IMST_HCI_DEVMGMT_MSG_GET_AES_KEY_RSP:
174             default:
175                 break;
176             }
177             break;
178         case IMST_HCI_RADIOLINK_ID:
179             switch(receivedMsg->msgID){//Switches the MsgId
180             //Verify status
181             if(receivedMsg->formatStat != IMST_HCI_RADIOLINK_STATUS_OK){
182                 return;//Error..?
183             }
184             case IMST_HCI_RADIOLINK_SEND_U_DATA_RSP:
185                 receivedMsg->payload = NULL;
186                 HCI_isConfirmed = true;
187                 break;
188             case IMST_HCI_RADIOLINK_U_DATA_RX_IND:
189                 //sapID + msgID + status + 2*grp + 2*2*adr + 2*crc + 2*slip = 13
190                 dataLength = buffer->length - 13;
191                 receivedMsg->payload = malloc(dataLength);
192                 receivedMsg->payloadLength = dataLength;
193                 receivedMsg->dstGrp = *bytePtr++;
194                 receivedMsg->dstAdr = *bytePtr++;
195                 receivedMsg->dstAdr |= (*bytePtr++ << 8);
196                 receivedMsg->srcGrp = *bytePtr++;
197                 receivedMsg->srcAdr = *bytePtr++;
198                 receivedMsg->srcAdr |= (*bytePtr++ << 8);
199                 while(offset < dataLength){
200                     receivedMsg->payload[offset++] = *bytePtr++;
201                 }
202                 HCI_callBack(receivedMsg);
203                 break;
204             case IMST_HCI_RADIOLINK_RAW_DATA_RX_IND:
205                 //in sniffer mode..
206                 break;
207             default:
208                 break;
209             }
210         default:
211             break;
212         }
213         //Free space
214         free(buffer->begin);
215         free(buffer);
216     }
217
218     /**
219      * \brief    Wait here while the RF module respond with a confirmation message.
220      *           This function take care to call the proceed method of the
221      *           SerialDevice layer.
222      * \retval   true :  The confirmation is arrived in time
223      *           false : Otherwise
224      */
225     bool HCI_WaitForConfirmation(){
226         Time_LaunchTimeout(500);
227         while(!HCI_isConfirmed){
228             if(Time_TimeoutOccured()){
229                 return false;
230             }
231             SerialDevice_Proceed();
232         }
233         return true;
234     }
235
```

```
/**
  * \file        serial_device.h
  * \author      Pierre Mendicino
  * \version 1.0
  * \date        2017.08.10
  * \brief       This file contains the headers of the SerialDevice layer.
  */
#ifndef SERIAL_DEVICE_H_
#define SERIAL_DEVICE_H_

//*****************************************************************************
//
// Includes
//
//*****************************************************************************
#include <stdbool.h>
#include <stdlib.h>           //for NULL
#include "util/IMST_HCI.h"
#include "util/byteBuffer.h"
#include "_conf.h"

//*****************************************************************************
//
// Definitions
//
//*****************************************************************************
/// The Rx buffer for the bytes received by the USART peripheral
#define RX_BUFFER_SIZE    255 ///< TODO : Check thi value
/// The size of the Rx fifo
#define RX_FIFO_SIZE 10  ///< Increase this value if some messages are lost

//*****************************************************************************
//
// Private variables
//
//*****************************************************************************
/**
 * \vargroup Private variable
 * \{
 */
void (*SerialDevice_callBack)(byteBuffer_t* buffer);
uint8_t      SerialDevice_rxBuffer[RX_BUFFER_SIZE];
uint8_t*         SerialDevice_rxPtr;
uint16_t         SerialDevice_frameSize;
byteBuffer_t*    SerialDevice_rxFifo[RX_FIFO_SIZE];
uint8_t      SerialDevice_rxFifo_in;
uint8_t      SerialDevice_rxFifo_out;
/**
 * \}
 */

//*****************************************************************************
//
// Public functions
//
//*****************************************************************************
/**
 * \vargroup Public function
 * \{
 */
void SerialDevice_Init(void (*callBack)(byteBuffer_t* buffer));
void SerialDevice_Proceed();
void SerialDevice_SendByteBuffer(byteBuffer_t* buffer);
void SerialDevice_SendFrame(uint8_t* data, uint16_t length);
/**
 * \}
 */

//*****************************************************************************
//
// Private functions
//
//*****************************************************************************
/**
 * \vargroup Private function
 * \{
 */
bool SerialDevice_IsRxFifoEmpty();
bool SerialDevice_IsRxFifoFull();
void SerialDevice_PushRxFifo(byteBuffer_t* buffer);
byteBuffer_t* SerialDevice_PopRxFifo();
void SerialDevice_Reset();

#ifdef __STM32F10X__
```

```
85   void USART3_IRQHandler(void);
86   #endif //__STM32F10X
87   /**
88    * \}
89    */
90
91   #endif /* SERIAL_DEVICE_H_ */
92
```

```
/**
  * \file        serial_device.c
  * \author      Pierre Mendicino
  * \version 1.0
  * \date        2017.08.10
  * \brief       This file contains the implementation of the SerialDevice layer
  *                  for the STM32F10X device.
  */
#include <serial_device.h>
#include <stm32f10x.h>

/**
  * \brief   Initialize STM32 for USART3 (Pin P10 & P11) and interrupt
  * \param   callBack : Function to call when something when a frame is received
  * \retval  void
  */
void SerialDevice_Init(void (*callBack)(byteBuffer_t* buffer)){

    //Initialize some variables
    SerialDevice_callBack = callBack; //store the callBack function
    SerialDevice_rxFifo_in = 0;           //input index for the rxFifo
    SerialDevice_rxFifo_out = 0;      //output index for the rxFifo
    SerialDevice_Reset();                 //indexes for the rxBuffer

    //Declaration of the initialization structures
    USART_InitTypeDef USART_InitStructure;     //UART
    GPIO_InitTypeDef GPIO_InitStructure;  //GPIO

    //Enable clocks
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_USART3, ENABLE); //USART
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB, ENABLE);  //GPIO

    //Configure Tx on pin B10
    GPIO_InitStructure.GPIO_Pin =     GPIO_Pin_10;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_InitStructure.GPIO_Mode =    GPIO_Mode_AF_PP;
    GPIO_Init(GPIOB, &GPIO_InitStructure);

    //Configure Rx on pin B11
    GPIO_InitStructure.GPIO_Pin =     GPIO_Pin_11;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_InitStructure.GPIO_Mode =    GPIO_Mode_IN_FLOATING;
    GPIO_Init(GPIOB, &GPIO_InitStructure);

    //Configure UART
    USART_InitStructure.USART_BaudRate =           115200;
    USART_InitStructure.USART_WordLength =             USART_WordLength_8b;
    USART_InitStructure.USART_StopBits =           USART_StopBits_1;
    USART_InitStructure.USART_Parity =                 USART_Parity_No;
    USART_InitStructure.USART_HardwareFlowControl = USART_HardwareFlowControl_None;
    USART_InitStructure.USART_Mode =               USART_Mode_Rx | USART_Mode_Tx;
    USART_Init(USART3, &USART_InitStructure);
    //
    USART_ITConfig(USART3, USART_IT_RXNE, ENABLE);

    //Start the UART and enable the interrupts
    USART_Cmd(USART3, ENABLE);
    NVIC_EnableIRQ(USART3_IRQn);
}

/**
 * \brief    Send some bytes
 * \param    data : A pointer to the first byte
 * \param    length : The amount of bytes
 * \retval   void
 */
void SerialDevice_SendFrame(uint8_t* data, uint16_t length)
{
    uint8_t* char_ptr = data;
    uint8_t index = 0;
    while(index++ < length)
    {
        USART_SendData(USART3, (uint16_t) *char_ptr++);
        while(USART_GetFlagStatus(USART3, USART_FLAG_TC) == RESET){
            //Wait for byte transmitted...
        }
    }
}

/**
 * \brief    Send a byteBuffer
 * \param    buffer : The buffer to send
 * \retval   void
 */
```

```
85  void SerialDevice_SendByteBuffer(byteBuffer_t* buffer){
86      uint8_t* char_ptr = buffer->begin;
87      uint8_t index = 0;
88      while(index++ < buffer->length)
89      {
90          USART_SendData(USART3, (uint16_t) *char_ptr++);
91          while(USART_GetFlagStatus(USART3, USART_FLAG_TC) == RESET){
92              //Wait for byte transmitted...
93          }
94      }
95
96      free(buffer->begin); //Give back the data pointed by the buffer
97      free(buffer);            //Give back the buffer itself
98  }
99
100 /**
101  * \brief    This function is called by the system when a byte has been received
102  *               by the USART3 peripheral.
103  * \retval   void
104  */
105 void USART3_IRQHandler(void)
106 {
107     uint8_t i = 0;
108     byteBuffer_t* frame;
109     //Receive a character
110     if(USART_GetFlagStatus(USART3, USART_FLAG_RXNE) != RESET)
111     {
112         *SerialDevice_rxPtr = USART_ReceiveData(USART3);
113
114         if(*SerialDevice_rxPtr == IMST_HCI_SLIP &&
115                 SerialDevice_rxPtr == &SerialDevice_rxBuffer[0]){//Begin of frame
116             SerialDevice_rxPtr++;
117             SerialDevice_frameSize++;
118         } else if(*SerialDevice_rxPtr == IMST_HCI_SLIP &&
119                 SerialDevice_rxPtr != &SerialDevice_rxBuffer[0]){//End of frame
120             SerialDevice_frameSize++;
121             frame = malloc(sizeof(byteBuffer_t));
122             frame->begin = malloc(SerialDevice_frameSize);
123             while(i < SerialDevice_frameSize){
124                 frame->begin[i] = SerialDevice_rxBuffer[i];
125                 i++;
126             }
127             frame->length = SerialDevice_frameSize;
128             // REMOVED because of the add of the rxFifo
129             //SerialDevice_callBack(frame);
130             SerialDevice_PushRxFifo(frame);
131             SerialDevice_Reset();
132         } else if(SerialDevice_rxPtr != &SerialDevice_rxBuffer[0]){//Other char
133             SerialDevice_rxPtr++;
134             SerialDevice_frameSize++;
135         } else {
136             //Nothing
137         }
138     }
139 }
140
141 /**
142  * \brief    This function is used to reset some attributes
143  * \retval   void
144  */
145 void SerialDevice_Reset(){
146     SerialDevice_rxPtr = &SerialDevice_rxBuffer[0];
147     SerialDevice_frameSize = 0;
148 }
149
150 /**
151  * \brief    This function test if the Rx fifo is empty
152  * \retval   true : The fifo is empty
153  * \retval   false : Otherwise
154  */
155 bool SerialDevice_IsRxFifoEmpty(){
156     if(SerialDevice_rxFifo_in == SerialDevice_rxFifo_out){
157         return true;
158     }
159     return false;
160 }
161
162 /**
163  * \brief    This function test if the Rx fifo is full
164  * \retval   true : The fifo is full
165  * \retval   false : Otherwise
166  */
167 bool SerialDevice_IsRxFifoFull(){
168     if((SerialDevice_rxFifo_in + 1) % RX_FIFO_SIZE == SerialDevice_rxFifo_out){
```

```
169             return true;
170         }
171         return false;
172     }
173 
174     /**
175      * \brief    This function is called by the ISR to push a byteBuffer_t when the
176      *               frame that he contains is fully arrived.
177      * \param    buffer : A pointer on the buffer to push
178      * \retval   void
179      */
180     void SerialDevice_PushRxFifo(byteBuffer_t* buffer){
181         if(SerialDevice_IsRxFifoFull()){
182             return;//Error : queue is full
183         } else {
184             //Register value
185             SerialDevice_rxFifo[SerialDevice_rxFifo_in] = buffer;
186             //Update index
187             SerialDevice_rxFifo_in = (SerialDevice_rxFifo_in + 1) % RX_FIFO_SIZE;
188         }
189     }
190 
191     /**
192      * \brief    This function pop the Rx fifo.
193      * \retval   A pointer to the buffer popped
194      * \retval   NULL if the queue is empty
195      */
196     byteBuffer_t* SerialDevice_PopRxFifo(){
197         uint8_t oldIndex = SerialDevice_rxFifo_out;
198         if(SerialDevice_IsRxFifoEmpty()){
199             return NULL;//Error : queue is empty
200         } else {
201             //Update index
202             SerialDevice_rxFifo_out = (SerialDevice_rxFifo_out + 1) % RX_FIFO_SIZE;
203             //Return value
204             return SerialDevice_rxFifo[oldIndex];
205         }
206     }
207 
208     /**
209      * \brief    This function use the callBack function to push the data in the
210      *               upper layer. You should call-it periodically!
211      * \retval   void
212      */
213     void SerialDevice_Proceed(){
214         byteBuffer_t* tmp;
215         if(!SerialDevice_IsRxFifoEmpty()){
216             tmp = SerialDevice_PopRxFifo();
217             SerialDevice_callBack(tmp);
218         }
219     }
220 
```

```
/**
  * \file        _conf.h
  * \author      Pierre Mendicino
  * \version 1.0
  * \date        2017.08.10
  * \brief       This file contains some useful definitions to choose a target
  *                  device or a certain test to do.
  */
#ifndef CONF_H_
#define CONF_H_

//
// CHOOSE THE TEST TO LOAD IN THE TARGET => UNCOMMENT ONLY ONE
//
#define __TEST__       //Uncomment to any test
//#define __TEST_01__      //Uncomment to launch the test 01
//#define __TEST_02__      //Uncomment to launch the test 02
//#define __TEST_03__      //Uncomment to launch the test 03
//#define __TEST_04__      //Uncomment to launch the test 04
//#define __TEST_05__      //Uncomment to launch the test 05
//#define __TEST_06__      //Uncomment to launch the test 06
#define __TEST_07__   //Uncomment to launchn the test 07
//
// CHOOSE THE ID OF THE TARGET => UNCOMMENT ONLY ONE
//
//#define __NODE_01__          //Uncomment to programm the node 01
#define __NODE_02__         //Uncomment to programm the node 02
//#define __NODE_03__          //Uncomment to programm the node 03
//#define __NODE_04__          //Uncomment to programm the node 04
//
// DEVICE SPECIFICS
//
// -> Raspberry pi 3
//#define __RPI_3__            //Uncomment when use with Raspberry pi 3
//#define __DEBUG__            //Uncomment for trace in RPI3
//#define __DEBUG_SERIAL__     //Uncomment for advanced trace in RPI3
//
// -> STM32F10x
#define __STM32F10X__          //Uncomment when use with STM32F10x

////////////////////////////////////////////////////////////////////////////////
//
//  Don't edit this part of the file !
//
////////////////////////////////////////////////////////////////////////////////
#ifdef __RPI_3__           //if __RPI_3__ is defined
#ifdef __DEBUG_SERIAL__            //if __DEBUG_SERIAL__ is defined
#ifndef __DEBUG__                      //if __DEBUG is not defined
#define __DEBUG__                          //define __DEBUG__//
#endif // __DEBUG__                    //
#endif // __DEBUG_SERIAL__         //
#else                          //if __RPI_3__ is not defined
#ifdef __DEBUG__               //if __DEBUG__ is defined
#undef __DEBUG__                   //undef __DEBUG__//
#endif // __DEBUG__                //
#ifdef __DEBUG_SERIAL__            //if __DEBUG_SERIAL__ is defined
#undef __DEBUB_SERIAL__                //undef __DEBUG_SERIAL//
#endif // __DEBUG_SERIAL__         //
#endif // __RPI_3__            //

#ifdef __DEBUG__           //if __DEBUG__ is defined
#include <stdio.h>                //Include (for printf function)//
#endif // __DEBUG__            //
////////////////////////////////////////////////////////////////////////////////

#endif /* CONF_H_ */

```

```
/**
  * \file        test.h
  * \author      Pierre Mendicino
  * \version 1.0
  * \date        2017.08.10
  * \brief       This file contains the headers of some tests
  */
#ifndef TEST_H_
#define TEST_H_

#include "loranet.h"
#include "util/IMST_HCI.h"

LoRaNet_Node Test_node1;
LoRaNet_Node Test_node2;
LoRaNet_Node Test_node3;
LoRaNet_Node Test_node4;

IMST_HCI_MSG* msgToReceive;
LoRaNet_Packet* packetToSend;
uint8_t loop_int;

void Test_Init();
void Test_RegisterNode(LoRaNet_Node* node, uint8_t direction);
void Test_Receive();

void Test_01(uint16_t param);
void Test_02(uint16_t param);
void Test_03(uint16_t param);
void Test_04(uint16_t param);
void Test_05(uint16_t param);
void Test_06(uint16_t param);
void Test_07(uint16_t param);

#endif /* TEST_H_ */

```

```
/**
  * \file        test.c
  * \author      Pierre Mendicino
  * \version 1.0
  * \date        2017.08.10
  * \brief       This file contains the implementation and the description of
  *                  some tests
  */
#include "test.h"
#include "_conf.h"

void Test_Init(){
    //
    //Node 1
    Test_node1.address =     0x1234;
    Test_node1.groupId =     0x10;
    Test_node1.nodeId =  0x01;
    //
    //Node 2
    Test_node2.address =     0x2345;
    Test_node2.groupId =     0x10;
    Test_node2.nodeId =  0x02;
    //
    //Node 3
    Test_node3.address =     0x3456;
    Test_node3.groupId =     0x10;
    Test_node3.nodeId =  0x03;
    //
    //Node 4
    Test_node4.address =     0x4567;
    Test_node4.groupId =     0x10;
    Test_node4.nodeId =  0x04;
}

void Test_RegisterNode(LoRaNet_Node* node, uint8_t direction){
    LoRaNet_Node* toRegister;
    toRegister = malloc(sizeof(LoRaNet_Node));
    toRegister->address = node->address;
    toRegister->groupId = node->groupId;
    toRegister->nodeId = node->nodeId;
    LoRaNet_Register(toRegister, direction);
}

/**
 * ------------------------------------------------------
 * SITUATION
 * ------------------------------------------------------
 * +----+     +----+
 * | 04 |---->| 01 |
 * +----+     +----+
 * -> We are the node (04)
 * -> We know the gateway (01)
 * ------------------------------------------------------
 * SCENARIO
 * ------------------------------------------------------
 * -> We send a packet to the gateway
 * ------------------------------------------------------
 * EXPECTED
 * ------------------------------------------------------
 * -> The Gateway receive the packet
 */
void Test_01(uint16_t param){
    loop_int = 0;

    // -> Register the nodes we know
    Test_RegisterNode(&Test_node1, LORANET_UP);

    packetToSend =                                  malloc(sizeof(LoRaNet_Packet));
    packetToSend->dstId =                           Test_node1.nodeId;
    packetToSend->data.length =                     6;
    packetToSend->data.begin =                      malloc(packetToSend->data.length);
    loop_int = param;
    while(loop_int < packetToSend->data.length){
        packetToSend->data.begin[loop_int] =  loop_int;
        loop_int++;
    }
    packetToSend->srcId =                           LoRaNet_myNode->nodeId;
    packetToSend->type =                            LORANET_TYPE_DATA_UP;
    packetToSend->packetId =                    LoRaNet_GetPacketID();
    LoRaNet_PushTxQueue(packetToSend);
}

/**
 * ------------------------------------------------------
```

```
 85    * SITUATION
 86    * ------------------------------------------------------
 87    *           +----+
 88    *         ->| 05 |-------.
 89    *        /  +----+       v
 90    * +----+ /   +----+     +----+     +----+
 91    * | 08 |---->| 06 |---->| 04 |-//->| 01 |
 92    * +----+ \   +----+     +----+     +----+
 93    *         \  +----+       ^
 94    *          ->| 07 |-------'
 95    *           +----+
 96    * -> We are the node (04)
 97    * -> We don't know the gateway (01)
 98    * -> We know the nodes (05), (06), (07)
 99    * ------------------------------------------------------
100    * SCENARIO
101    * ------------------------------------------------------
102    * -> The node 08 try to send a message to the gateway
103    * -> The node 05, 06 and 07 forward this message
104    * ------------------------------------------------------
105    * EXPECTED
106    * ------------------------------------------------------
107    * -> The gateway need to send a ping otherwise we don't
108    *    know it..
109    * -> The node 04 forward only one time the message
110    * -> When the gateway send something to the node 08, he
111    *    can now do it trough the node 04
112    */
113   void Test_02(uint16_t param){
114       // -> Register the nodes we know
115       // -> Not this one.. Test_RegisterNode(&Test_node1, LORANET_UP);
116       Test_RegisterNode(&Test_node2, LORANET_DOWN);
117       Test_RegisterNode(&Test_node3, LORANET_DOWN);
118       Test_RegisterNode(&Test_node4, LORANET_DOWN);
119
120       // -> Simulate the reception of the different messages
121       msgToReceive =                                      malloc(sizeof(IMST_HCI_MSG));
122       msgToReceive->dstAdr =                              0xBBBB;
123       msgToReceive->dstGrp =                              0x10;
124       msgToReceive->formatStat =                          0x00;
125       msgToReceive->msgID =                               IMST_HCI_RADIOLINK_U_DATA_RX_IND;
126       msgToReceive->payloadLength =                       LORANET_OFFSET_DATA + 1;
127       msgToReceive->payload =                           malloc(msgToReceive->payloadLength);
128       msgToReceive->payload[LORANET_OFFSET_DATA] =    0x32;
129       msgToReceive->payload[LORANET_OFFSET_DST] =         Test_node1.nodeId;
130       msgToReceive->payload[LORANET_OFFSET_ID] =          param;
131       msgToReceive->payload[LORANET_OFFSET_SRC] =         0x08;
132       msgToReceive->payload[LORANET_OFFSET_TYPE] =    LORANET_TYPE_DATA_UP;
133       msgToReceive->sapID =                               IMST_HCI_RADIOLINK_ID;
134       msgToReceive->srcAdr =                              0x5678;
135       msgToReceive->srcGrp =                              0x10;
136       LoRaNet_RxCallBack(msgToReceive);
137
138       msgToReceive =                                      malloc(sizeof(IMST_HCI_MSG));
139       msgToReceive->dstAdr =                              0xBBBB;
140       msgToReceive->dstGrp =                              0x10;
141       msgToReceive->formatStat =                          0x00;
142       msgToReceive->msgID =                               IMST_HCI_RADIOLINK_U_DATA_RX_IND;
143       msgToReceive->payloadLength =                       LORANET_OFFSET_DATA + 1;
144       msgToReceive->payload =                           malloc(msgToReceive->payloadLength);
145       msgToReceive->payload[LORANET_OFFSET_DATA] =    0x32;
146       msgToReceive->payload[LORANET_OFFSET_DST] =         Test_node1.nodeId;
147       msgToReceive->payload[LORANET_OFFSET_ID] =          param;
148       msgToReceive->payload[LORANET_OFFSET_SRC] =         0x08;
149       msgToReceive->payload[LORANET_OFFSET_TYPE] =    LORANET_TYPE_DATA_UP;
150       msgToReceive->sapID =                               IMST_HCI_RADIOLINK_ID;
151       msgToReceive->srcAdr =                              0x6789;
152       msgToReceive->srcGrp =                              0x10;
153       LoRaNet_RxCallBack(msgToReceive);
154
155       msgToReceive =                                      malloc(sizeof(IMST_HCI_MSG));
156       msgToReceive->dstAdr =                              0xBBBB;
157       msgToReceive->dstGrp =                              0x10;
158       msgToReceive->formatStat =                          0x00;
159       msgToReceive->msgID =                               IMST_HCI_RADIOLINK_U_DATA_RX_IND;
160       msgToReceive->payloadLength =                       LORANET_OFFSET_DATA + 1;
161       msgToReceive->payload =                           malloc(msgToReceive->payloadLength);
162       msgToReceive->payload[LORANET_OFFSET_DATA] =    0x32;
163       msgToReceive->payload[LORANET_OFFSET_DST] =         Test_node1.nodeId;
164       msgToReceive->payload[LORANET_OFFSET_ID] =          param;
165       msgToReceive->payload[LORANET_OFFSET_SRC] =         0x08;
166       msgToReceive->payload[LORANET_OFFSET_TYPE] =    LORANET_TYPE_DATA_UP;
167       msgToReceive->sapID =                               IMST_HCI_RADIOLINK_ID;
168       msgToReceive->srcAdr =                              0x789a;
```

```
169       msgToReceive->srcGrp =                              0x10;
170       LoRaNet_RxCallBack(msgToReceive);
171   }
172
173   /**
174    * ------------------------------------------------------
175    * SITUATION
176    * ------------------------------------------------------
177    *           +----+
178    *         ->| 05 |-------.
179    *        /  +----+       v
180    * +----+ /   +----+     +----+     +----+
181    * | 08 |---->| 06 |---->| 04 |-//->| 01 |
182    * +----+ \   +----+     +----+     +----+
183    *        \  +----+       ^
184    *         ->| 07 |-------'
185    *           +----+
186    * -> We are the node (04)
187    * -> We don't know the gateway (01)
188    * -> We know the nodes (05), (06), (07)
189    * ------------------------------------------------------
190    * SCENARIO
191    * ------------------------------------------------------
192    * -> The Gateway send a ping. Now we know it.s
193    * -> The txQueue contains some messages and we need to
194    *    send it.
195    * ------------------------------------------------------
196    * EXPECTED
197    * ------------------------------------------------------
198    * -> The node 04 send the messages and no crash occurs
199    * -> The gateway receive three messages from 5, 6 and 7
200    * -> The gateway register this nodes in downlink
201    */
202   void Test_03(uint16_t param){
203       // -> Register the nodes we know
204       Test_RegisterNode(&Test_node2, LORANET_DOWN);
205       Test_RegisterNode(&Test_node3, LORANET_DOWN);
206       Test_RegisterNode(&Test_node4, LORANET_DOWN);
207
208       // -> Fill the queue
209       packetToSend =                                  malloc(sizeof(LoRaNet_Packet));
210       packetToSend->dstId =                           Test_node1.nodeId;
211       packetToSend->data.length =               10;
212       packetToSend->data.begin =                      malloc(packetToSend->data.length);
213       loop_int = param;
214       while(loop_int < packetToSend->data.length){
215           packetToSend->data.begin[loop_int] =  loop_int;
216           loop_int++;
217       }
218       packetToSend->srcId =                           Test_node2.nodeId;
219       packetToSend->type =                            LORANET_TYPE_DATA_UP;
220       packetToSend->packetId =                  LoRaNet_GetPacketID();
221       LoRaNet_PushTxQueue(packetToSend);
222
223       packetToSend =                                  malloc(sizeof(LoRaNet_Packet));
224       packetToSend->dstId =                           Test_node1.nodeId;
225       packetToSend->data.length =               10;
226       packetToSend->data.begin =                      malloc(packetToSend->data.length);
227       loop_int = param;
228       while(loop_int < packetToSend->data.length){
229           packetToSend->data.begin[loop_int] =  loop_int;
230           loop_int++;
231       }
232       packetToSend->srcId =                           Test_node3.nodeId;
233       packetToSend->type =                            LORANET_TYPE_DATA_UP;
234       packetToSend->packetId =                  LoRaNet_GetPacketID();
235       LoRaNet_PushTxQueue(packetToSend);
236
237       packetToSend =                                  malloc(sizeof(LoRaNet_Packet));
238       packetToSend->dstId =                           Test_node1.nodeId;
239       packetToSend->data.length =               10;
240       packetToSend->data.begin =                      malloc(packetToSend->data.length);
241       loop_int = param;
242       while(loop_int < packetToSend->data.length){
243           packetToSend->data.begin[loop_int] =  loop_int;
244           loop_int++;
245       }
246       packetToSend->srcId =                           Test_node4.nodeId;
247       packetToSend->type =                            LORANET_TYPE_DATA_UP;
248       packetToSend->packetId =                  LoRaNet_GetPacketID();
249       LoRaNet_PushTxQueue(packetToSend);
250   }
251
252   /**
```

```
253   * ------------------------------------------------------
254   * SITUATION
255   * ------------------------------------------------------
256   *            +----+
257   *          ->| 05 |-------.
258   *         /  +----+       v
259   * +----+ /   +----+     +----+     +----+
260   * | 08 |---->| 06 |---->| 04 |-//->| 01 |
261   * +----+ \   +----+     +----+     +----+
262   *         \  +----+       ^
263   *          ->| 07 |-------'
264   *            +----+
265   * -> We are the gateway, with id (04)
266   * -> We don't know the node (01)
267   * ------------------------------------------------------
268   * SCENARIO
269   * ------------------------------------------------------
270   * -> We send a ping, the node 01 respond to us with a pong
271   * ------------------------------------------------------
272   * EXPECTED
273   * ------------------------------------------------------
274   * -> The node 01 send a pong
275   */
276  void Test_04(uint16_t param){
277      // -> Send Ping
278      packetToSend = LoRaNet_MakePingPongPacket(LORANET_TYPE_PING, LORANET_ID_BROADCAST);
279      LoRaNet_PushTxQueue(packetToSend);
280  }
281
282  /**
283   * ------------------------------------------------------
284   * SITUATION
285   * ------------------------------------------------------
286   * +----+     +----+
287   * | 04 |---->| 03 |-------+
288   * +----+     +----+       v
289   *              |        +----+
290   *              |        | 01 |
291   *              V        +----+
292   *            +----+       ^
293   *            | 02 |-------+
294   *            +----+
295   * -> This test can be loaded in each nodes
296   * -> The node (04) knows the node (03) in uplink
297   * -> The node (03) knows the nodes (02) and (01) in uplink
298   * -> The node (03) knows the node (04) in downlink
299   * -> The node (02) knows the node (01) in uplink
300   * -> The node (02) knows the node (03) in downlink
301   * -> The node (01) knows the nodes (02) and (03) in downli
302   * ------------------------------------------------------
303   * SCENARIO
304   * ------------------------------------------------------
305   * -> The node (04) send some data to the node (01)
306   * ------------------------------------------------------
307   * EXPECTED
308   * ------------------------------------------------------
309   * -> The node (01) respond to this message
310   * -> The nodes (02) and (03) forward this messages
311   */
312  void Test_05(uint16_t param){
313      // -> Register the nodes we know
314  #ifdef __NODE_02__
315      Test_RegisterNode(&Test_node3, LORANET_DOWN);
316      Test_RegisterNode(&Test_node1, LORANET_UP);
317  #endif //__NODE_02__
318  #ifdef __NODE_03__
319      Test_RegisterNode(&Test_node1, LORANET_UP);
320      Test_RegisterNode(&Test_node2, LORANET_UP);
321      Test_RegisterNode(&Test_node4, LORANET_DOWN);
322  #endif //__NODE_03__
323  #ifdef __NODE_04__
324      Test_RegisterNode(&Test_node3, LORANET_UP);
325
326      // -> Prepare the packet
327      packetToSend =                                  malloc(sizeof(LoRaNet_Packet));
328      packetToSend->dstId =                           Test_node1.nodeId;
329      packetToSend->data.length =               10;
330      packetToSend->data.begin =                      malloc(packetToSend->data.length);
331      loop_int = 0;
332      while(loop_int < packetToSend->data.length){
333          packetToSend->data.begin[loop_int] =  loop_int + 48;
334          loop_int++;
335      }
336      packetToSend->srcId =                           Test_node4.nodeId;
```

```
337       packetToSend->type =                            LORANET_TYPE_DATA_UP;
338       packetToSend->packetId =                    LoRaNet_GetPacketID();
339       LoRaNet_PushTxQueue(packetToSend);
340   #endif //__NODE_04__
341   }
342
343   /**
344    * ------------------------------------------------------
345    * SITUATION
346    * ------------------------------------------------------
347    * +----+     +----+
348    * | 04 |---->| 03 |-------+
349    * +----+     +----+       v
350    *              |        +----+
351    *              |        | 01 |
352    *              V        +----+
353    *            +----+       ^
354    *            | 02 |-------+
355    *            +----+
356    * -> Nobody knows nobody
357    * ------------------------------------------------------
358    * SCENARIO
359    * ------------------------------------------------------
360    * -> Only the gateway and the nodes (02) and (03) are on
361    * -> The node (01) send a ping the two other node responds
362    * -> The node (04) is now turned on
363    * -> The node (04) can only hear the node (03)
364    * -> The gateway send another ping
365    * -> The node (04) can now send some datas
366    * ------------------------------------------------------
367    * EXPECTED
368    * ------------------------------------------------------
369    * -> The node (01) respond to this message
370    * -> The nodes (02) and (03) forward this messages
371    */
372   byteBuffer_t Test_06_makeByteBuff(uint16_t param){
373       byteBuffer_t toReturn;
374       uint8_t index = 0;
375       uint8_t buffer = 0;
376
377       //Determine the length of the data
378       if(param > 9){
379           if(param > 99){
380               if(param > 999){
381                   if(param > 9999){
382                       toReturn.length = 5;
383                   } else {
384                       toReturn.length = 4;
385                   }
386               } else {
387                   toReturn.length = 3;
388               }
389           } else {
390               toReturn.length = 2;
391           }
392       } else {
393           toReturn.length = 1;
394       }
395
396       //Reserve space
397       toReturn.begin = malloc(toReturn.length);
398
399       //Build the buffer
400       switch(toReturn.length){
401       case 5:
402           buffer = param / 10000;
403           toReturn.begin[index++] = buffer + 48;
404           param -= buffer * 10000;
405       case 4:
406           buffer = param / 1000;
407           toReturn.begin[index++] = buffer + 48;
408           param -= buffer * 1000;
409       case 3:
410           buffer = param / 100;
411           toReturn.begin[index++] = buffer + 48;
412           param -= buffer * 100;
413       case 2:
414           buffer = param / 10;
415           toReturn.begin[index++] = buffer + 48;
416           param -= buffer * 10;
417       case 1:
418           toReturn.begin[index++] = param + 48;
419       default:
420           break;
```

```
421         }
422
423         //Return
424         return toReturn;
425     }
426
427     void Test_06(uint16_t param){
428         // -> Register the nodes we know
429     #ifdef __NODE_02__
430     #endif //__NODE_02__
431     #ifdef __NODE_03__
432     #endif //__NODE_03__
433     #ifdef __NODE_04__
434     #endif //__NODE_04__
435
436         // -> Prepare the packet
437         packetToSend =                                  malloc(sizeof(LoRaNet_Packet));
438         packetToSend->dstId =                           Test_node1.nodeId;
439         packetToSend->data =                            Test_06_makeByteBuff(param);
440         packetToSend->srcId =                           LoRaNet_myNode->nodeId;
441         packetToSend->type =                            LORANET_TYPE_DATA_UP;
442         packetToSend->packetId =                    LoRaNet_GetPacketID();
443         LoRaNet_PushTxQueue(packetToSend);
444     }
445
446     /**
447      * ------------------------------------------------------
448      * SITUATION
449      * ------------------------------------------------------
450      * UPLINK                          DOWNLINK
451      * +----+     +----+           +----+     +----+
452      * | 04 |---->| 03 |-------+   | 04 |<----| 03 |<------+
453      * +----+     +----+       v   +----+     +----+       |
454      *              |        +----+             ^        +----+
455      *              |        | 01 |             |        | 01 |
456      *              V        +----+             |        +----+
457      *            +----+       ^              +----+       |
458      *            | 02 |-------+              | 02 |<------+
459      *            +----+                      +----+
460      * -> This test can be loaded in each nodes
461      * -> The node (04) use a software filter to simulate the
462      *       fact that he can only hear the node (03)
463      * -> The node (02) use a software filter to simulate the
464      *       fact that he can't hear the node (04)
465      * -> The node (01) use a software filter to simulate the
466      *       fact that he can't hear the node (04)
467      * ------------------------------------------------------
468      * SCENARIO
469      * ------------------------------------------------------
470      * -> The node (04) want to send some data to the node (01)
471      * ------------------------------------------------------
472      * EXPECTED
473      * ------------------------------------------------------
474      * -> The node (01) receive the messages, forwarded by the
475      *       two others nodes
476      */
477     void Test_07(uint16_t param){
478         // -> Prepare the packet
479         packetToSend =                                  malloc(sizeof(LoRaNet_Packet));
480         packetToSend->dstId =                           Test_node1.nodeId;
481         packetToSend->data =                            Test_06_makeByteBuff(param);
482         packetToSend->srcId =                           LoRaNet_myNode->nodeId;
483         packetToSend->type =                            LORANET_TYPE_DATA_UP;
484         packetToSend->packetId =                    LoRaNet_GetPacketID();
485         LoRaNet_PushTxQueue(packetToSend);
486     }
487
```

```c
/**
  * \file        byteBuffer.h
  * \author      Pierre Mendicino
  * \version 1.0
  * \date        2017.08.10
  * \brief       This file contains the definition of a byteBuffer
  */
#ifndef UTIL_BYTEBUFFER_H_
#define UTIL_BYTEBUFFER_H_

//******************************************************************************
//
// Includes
//
//******************************************************************************
#include <stdint.h>

//******************************************************************************
//
// Structures definitions
//
//******************************************************************************
/**
 * \brief    This structure represent a "buffer", containing a certain number of
 *           bytes.
 */
typedef struct
{
    /// A pointer to the first byte
    uint8_t* begin;   ///<
    /// The number of byte this buffer have
    uint16_t length; ///<
}byteBuffer_t;

#endif /* UTIL_BYTEBUFFER_H_ */

```

```c
/**
  * \file        time.h
  * \author      Pierre Mendicino
  * \version 1.0
  * \date        2017.08.10
  * \brief       This file contains the headers of the time tools
  */
#ifndef TIME_H_
#define TIME_H_

//*****************************************************************************
//
// Includes
//
//*****************************************************************************
#include <stdint.h>
#include <stdbool.h>
#include "../_conf.h"

//*****************************************************************************
//
// Private variables
//
//*****************************************************************************
/**
 * \fngroup Private variable
 * \{
 */
uint32_t     Time_timeoutValue;
/**
 * \}
 */

//*****************************************************************************
//
// Public functions
//
//*****************************************************************************
/**
 * \fngroup Pulic function
 * \{
 */
/**
 * \brief    Initialize the time tools
 */
void Time_Init();
/**
 * \brief    Blocking wait
 * \param time_ms : The time to wait in this function
 */
void Time_wait(uint32_t time_ms);
/**
 * \brief    Launch the single timeout
 * \param    time_ms : The time to load
 */
void Time_LaunchTimeout(uint32_t time_ms);
/**
 * \brief    Test the timeout
 * \retval true : The timeout have occured
 * \retval false : Otherwise
 */
bool Time_TimeoutOccured();

#ifdef STM32F10X_MD_VL
/**
 * \brief Called every millisecond by the system
 */
void SysTick_Handler(void);
#endif // STM32F10X_MD_VL
/**
 * \}
 */

#endif /* TIME_H_ */

```

```c
/**
  * \file        time.c
  * \author      Pierre Mendicino
  * \version 1.0
  * \date        2017.08.10
  * \brief       This file contains the implementation of the time tools.
  *              Actually, this implementation supports two devices :
  * \brief       - RPI 3
  * \brief       - STM32F10X
  */
#include "time.h"

#ifdef STM32F10X_MD_VL ////////////////////////////////////////////////////
#include "stm32f10x.h"

uint32_t     Time_ticksTimeout;
uint32_t     Time_ticksCounter;

void Time_Init(){
    //SYSTICK
    SysTick_Config(SystemCoreClock / 1000);
}

void Time_wait(uint32_t time_ms){
    Time_ticksCounter = 0;
    while(Time_ticksCounter < time_ms){
        //Wait
    }
}

void Time_LaunchTimeout(uint32_t time_ms){
    Time_timeoutValue = time_ms;
    Time_ticksTimeout = 0;
}

bool Time_TimeoutOccured(){
    if(Time_ticksTimeout >= Time_timeoutValue){
        return true;
    }
    return false;
}

void SysTick_Handler(void){
    Time_ticksCounter++;
    Time_ticksTimeout++;
}

#endif // STM32F10X_MD_VL ////////////////////////////////////////////////////

#ifdef __RPI_3__ ////////////////////////////////////////////////////////////

uint32_t Time_ticksTimeout;

#include <stdio.h>
#include <wiringPi.h>
PI_THREAD (timer){
    while(1){
        Time_ticksTimeout++;
        delay(1);
    }
}

void Time_Init(){
    Time_ticksTimeout = 0;
    if(piThreadCreate (timer) != 0){
        printf("Error creating thread..\n");
    }
}

void Time_wait(uint32_t time_ms){
    delay(time_ms);
}

void Time_LaunchTimeout(uint32_t time_ms){
    Time_timeoutValue = time_ms;
    Time_ticksTimeout = 0;
}

bool Time_TimeoutOccured(){
    if(Time_ticksTimeout >= Time_timeoutValue){
        return true;
    }
    return false;
}
```

```
85
86  #endif // __RPI_3__ ////////////////////////////////////////////////////////
87
88
```

```
/**
  * \file        crc16.h
  * \author      Kai vorm Walde (KvW), IMST
  * \version 0.2
  * \date        2011.09.14
  * \brief       Copyright (c) 2009\n
  *              IMST GmbH\n
  *              Carl-Friedrich Gauss Str. 2\n
  *              47475 Kamp-Lintfort
  */
#ifndef    __CRC16_H__
#define    __CRC16_H__

//------------------------------------------------------------------------------
//
//  Section Include Files
//
//------------------------------------------------------------------------------
#include <inttypes.h>
#include <stdbool.h>

typedef uint8_t     UINT8;
typedef uint16_t    UINT16;

//------------------------------------------------------------------------------
//
//  Section Defines
//
//------------------------------------------------------------------------------

#define CRC16_INIT_VALUE    0xFFFF    //!< initial value for CRC algorithem
#define CRC16_GOOD_VALUE    0x0F47    //!< constant compare value for check
#define CRC16_POLYNOM       0x8408    //!< 16-BIT CRC CCITT POLYNOM

//------------------------------------------------------------------------------
// C++ Extensions
//------------------------------------------------------------------------------
#ifdef   __cplusplus
extern  "C" {
#endif
//------------------------------------------------------------------------------
//
//  Section Prototypes
//
//------------------------------------------------------------------------------

//------------------------------------------------------------------------------
//! Calc CRC16
UINT16
CRC16_Calc  (UINT8*     data,
             UINT16     length,
             UINT16     initVal);
//------------------------------------------------------------------------------
//! Calc & Check CRC16
bool
CRC16_Check (UINT8*     data,
             UINT16     length,
             UINT16     initVal);

//------------------------------------------------------------------------------
// C++ Extensions
//------------------------------------------------------------------------------
#ifdef    __cplusplus
}
#endif
//------------------------------------------------------------------------------

#endif // __CRC16_H__
//------------------------------------------------------------------------------
// end of file
//------------------------------------------------------------------------------

```

```
/**
  * \file        crc16.c
  * \author      Kai vorm Walde (KvW), IMST
  * \version 0.2
  * \date        2011.09.14
  * \brief       Copyright (c) 2009\n
  *              IMST GmbH\n
  *              Carl-Friedrich Gauss Str. 2\n
  *              47475 Kamp-Lintfort
  */

//------------------------------------------------------------------------------
//
//  Section Include Files
//
//------------------------------------------------------------------------------
#include "crc16.h"

// use fast table algorithm
#define __CRC16_TABLE__
//------------------------------------------------------------------------------
//
//  Section CONST
//
//------------------------------------------------------------------------------

#ifdef    __CRC16_TABLE__
//------------------------------------------------------------------------------
//
//  Lookup Table for fast CRC16 calculation
//
//------------------------------------------------------------------------------

UINT16 CRC16_Table[] =
{
    0x0000, 0x1189, 0x2312, 0x329B, 0x4624, 0x57AD, 0x6536, 0x74BF,
    0x8C48, 0x9DC1, 0xAF5A, 0xBED3, 0xCA6C, 0xDBE5, 0xE97E, 0xF8F7,
    0x1081, 0x0108, 0x3393, 0x221A, 0x56A5, 0x472C, 0x75B7, 0x643E,
    0x9CC9, 0x8D40, 0xBFDB, 0xAE52, 0xDAED, 0xCB64, 0xF9FF, 0xE876,
    0x2102, 0x308B, 0x0210, 0x1399, 0x6726, 0x76AF, 0x4434, 0x55BD,
    0xAD4A, 0xBCC3, 0x8E58, 0x9FD1, 0xEB6E, 0xFAE7, 0xC87C, 0xD9F5,
    0x3183, 0x200A, 0x1291, 0x0318, 0x77A7, 0x662E, 0x54B5, 0x453C,
    0xBDCB, 0xAC42, 0x9ED9, 0x8F50, 0xFBEF, 0xEA66, 0xD8FD, 0xC974,
    0x4204, 0x538D, 0x6116, 0x709F, 0x0420, 0x15A9, 0x2732, 0x36BB,
    0xCE4C, 0xDFC5, 0xED5E, 0xFCD7, 0x8868, 0x99E1, 0xAB7A, 0xBAF3,
    0x5285, 0x430C, 0x7197, 0x601E, 0x14A1, 0x0528, 0x37B3, 0x263A,
    0xDECD, 0xCF44, 0xFDDF, 0xEC56, 0x98E9, 0x8960, 0xBBFB, 0xAA72,
    0x6306, 0x728F, 0x4014, 0x519D, 0x2522, 0x34AB, 0x0630, 0x17B9,
    0xEF4E, 0xFEC7, 0xCC5C, 0xDDD5, 0xA96A, 0xB8E3, 0x8A78, 0x9BF1,
    0x7387, 0x620E, 0x5095, 0x411C, 0x35A3, 0x242A, 0x16B1, 0x0738,
    0xFFCF, 0xEE46, 0xDCDD, 0xCD54, 0xB9EB, 0xA862, 0x9AF9, 0x8B70,
    0x8408, 0x9581, 0xA71A, 0xB693, 0xC22C, 0xD3A5, 0xE13E, 0xF0B7,
    0x0840, 0x19C9, 0x2B52, 0x3ADB, 0x4E64, 0x5FED, 0x6D76, 0x7CFF,
    0x9489, 0x8500, 0xB79B, 0xA612, 0xD2AD, 0xC324, 0xF1BF, 0xE036,
    0x18C1, 0x0948, 0x3BD3, 0x2A5A, 0x5EE5, 0x4F6C, 0x7DF7, 0x6C7E,
    0xA50A, 0xB483, 0x8618, 0x9791, 0xE32E, 0xF2A7, 0xC03C, 0xD1B5,
    0x2942, 0x38CB, 0x0A50, 0x1BD9, 0x6F66, 0x7EEF, 0x4C74, 0x5DFD,
    0xB58B, 0xA402, 0x9699, 0x8710, 0xF3AF, 0xE226, 0xD0BD, 0xC134,
    0x39C3, 0x284A, 0x1AD1, 0x0B58, 0x7FE7, 0x6E6E, 0x5CF5, 0x4D7C,
    0xC60C, 0xD785, 0xE51E, 0xF497, 0x8028, 0x91A1, 0xA33A, 0xB2B3,
    0x4A44, 0x5BCD, 0x6956, 0x78DF, 0x0C60, 0x1DE9, 0x2F72, 0x3EFB,
    0xD68D, 0xC704, 0xF59F, 0xE416, 0x90A9, 0x8120, 0xB3BB, 0xA232,
    0x5AC5, 0x4B4C, 0x79D7, 0x685E, 0x1CE1, 0x0D68, 0x3FF3, 0x2E7A,
    0xE70E, 0xF687, 0xC41C, 0xD595, 0xA12A, 0xB0A3, 0x8238, 0x93B1,
    0x6B46, 0x7ACF, 0x4854, 0x59DD, 0x2D62, 0x3CEB, 0x0E70, 0x1FF9,
    0xF78F, 0xE606, 0xD49D, 0xC514, 0xB1AB, 0xA022, 0x92B9, 0x8330,
    0x7BC7, 0x6A4E, 0x58D5, 0x495C, 0x3DE3, 0x2C6A, 0x1EF1, 0x0F78,
};
#endif
//------------------------------------------------------------------------------
//
//  Section Code
//
//------------------------------------------------------------------------------

//------------------------------------------------------------------------------
//
//  CRC16_Calc
//
//------------------------------------------------------------------------------
//!
//! @brief   calculate CRC16
//!
//------------------------------------------------------------------------------
```

```
85  //!
86  //! This function calculates the one's complement of the standard
87  //! 16-BIT CRC CCITT polynomial G(x) = 1 + x^5 + x^12 + x^16
88  //!
89  //! <!-------------------------------------------------------------------------->
90  //! @param[in]      data        pointer to data block
91  //! @param[in]      length      number of bytes
92  //! @param[in]      initVal     CRC16 initial value
93  //! <!-------------------------------------------------------------------------->
94  //! @retVal         crc16       crc
95  //------------------------------------------------------------------------------
96  #ifdef    __CRC16_TABLE__
97  UINT16
98  CRC16_Calc  (UINT8*             data,
99               UINT16             length,
100              UINT16             initVal)
101 {
102     // init crc
103     UINT16    crc = initVal;
104
105     // iterate over all bytes
106     while(length--)
107     {
108         // calc new crc
109         crc = (crc >> 8) ^ CRC16_Table[(crc ^ *data++) & 0x00FF];
110     }
111
112     // return result
113     return crc;
114 }
115
116 #else
117
118
119 UINT16
120 CRC16_Calc  (UINT8*             data,
121              UINT16             length,
122              UINT16             initVal)
123 {
124     // init crc
125     UINT16    crc = initVal;
126
127     // iterate over all bytes
128     while(length--)
129     {
130         int     bits    = 8;
131         UINT8   byte    = *data++;
132
133         // iterate over all bits per byte
134         while(bits--)
135         {
136             if((byte & 1) ^ (crc & 1))
137             {
138                 crc = (crc >> 1) ^ CRC16_POLYNOM;
139             }
140             else
141                 crc >>= 1;
142
143             byte >>= 1;
144         }
145     }
146
147     // return result
148     return crc;
149 }
150 #endif
151
152 //------------------------------------------------------------------------------
153 //
154 //  CRC16_Check
155 //
156 //------------------------------------------------------------------------------
157 //!
158 //! @brief   calculate & test CRC16
159 //!
160 //------------------------------------------------------------------------------
161 //!
162 //! This function checks a data block with attached CRC16
163 //!
164 //! <!-------------------------------------------------------------------------->
165 //! @param[in]      data        pointer to data block
166 //! @param[in]      length      number of bytes (including CRC16)
167 //! @param[in]      initVal     CRC16 initial value
168 //! <!-------------------------------------------------------------------------->
```

```
169   //! @retVal          true        CRC16 ok -> data block ok
170   //! @retVal          false       CRC16 failed -> data block corrupt
171   //-----------------------------------------------------------------------------
172
173   bool
174   CRC16_Check     (UINT8*                   data,
175                    UINT16                   length,
176                    UINT16                   initVal)
177   {
178       UINT16 crc = ~CRC16_Calc(data, length, initVal);
179
180       if( crc == CRC16_GOOD_VALUE)
181           return true;
182
183       return false;
184   }
185   //-----------------------------------------------------------------------------
186   // end of file
187   //-----------------------------------------------------------------------------
188
189
190
191
```