DECAESTEKER Théophile

RAPPORT DE STAGE :

Implémentation d'un protocole robuste de collecte de données dans un réseau de capteurs sans fil

REMERCIEMENT

Nous remercions chaleureusement les membres de l’équipe du projet et les encadrants de ce stage, M. Antonio Freitas et M. Michel Misson, qui ont apporté leur soutien tout au long du stage et de la rédaction de ce rapport.

Nous remercions également toutes les personnes qui ont participé par le passé au projet, notamment Mme. Marie-Françoise Servajean, M. Aurélien Surier, M. Raphaël Bidaud et M. Léo Essique.

SOMMAIRE

Table des matières

TABLE DES FIGURES

I - INTRODUCTION

Ce stage s'inscrit dans une collaboration en cours liant le Centre National des Etudes Spatiales (CNES), et le Laboratoire d'Informatique, de Modélisation et d'Optimisation des Systèmes (LIMOS). Etant mise en pause en octobre 2019, l'activité sur ce projet a été reprise en début 2021, et se poursuit encore grâce aux travaux accomplis par différents ingénieurs et chercheurs. La spécification fonctionnelle de la solution a fait l'objet de plusieurs travaux (thèse de doctorat [1], stages et activités de codage). Ce rapport de licence professionnelle décrit l'aboutissement d'un travail réalisé au cours de trois occasions différentes, un travail à temps plein correspondant à deux stages lors du premier semestre de 2021 et de 2022, et un projet tutoré lors de l'année scolaire 2021-2022. Le travail décrit dans ce rapport correspond à une troisième collaboration dont la durée dépasse celle qui encadre ce stage.

Ce projet vise à spécifier et implémenter un système informatique de contrôle commande s'appuyant sur un réseau de capteurs sans fil embarqué à bord de l'unité de pilotage, de localisation, de guidage et de sauvegarde d'un lanceur spatial : la case à équipement. L'objectif de ce projet est de remplacer les précédentes installations filaires de communication. En tant que projet de recherche, le travail consiste de mesurer les contraintes et les avantages d'une série de choix techniques ou technologiques. Une partie de ce travail étant déjà réalisée, notre apport à ce projet sera de s'assurer du bon fonctionnement de ce qui a été fait par le passé, et de poursuivre le développement de notre solution.

Au regard de l'environnement d'exploitation, notre solution repose sur un ordonnancement déterministe des actions de contrôle commande avec des contraintes temporelles strictes, des contraintes de robustesses des communications, ainsi que des contraintes matérielles. Ce stage étant une suite directe au projet tutoré et au stage de 2021 évoqués plus tôt, nous possédons une maîtrise importante du sujet et du travail précédemment accompli. Notre problématique est, en nous basant sur l'existant, de poursuivre l'implémentation et vérifier le bon fonctionnement de notre solution, tout en respectant les contraintes évoquées plus tôt. De plus, nous souhaitons pouvoir vérifier le bon fonctionnement de notre solution à l'aide de tests pratiqués sur le matériel dont nous disposons.

Dans un premier temps, nous parlerons de manière générale de l'existant, du contexte de ce stage et des différents acteurs impliqués.

Suite à cela, nous vous donnerons une explication détaillée de notre projet. Nous commencerons par vous présenter le projet en lui-même et les différents concepts qui lui sont associés dans une présentation fonctionnelle. Nous parlerons ensuite de nos travaux réalisés et de nos tests dans une présentation de notre implémentation.

Enfin, nous mettrons en avant les résultats accomplis lors de ce stage, ainsi que les différentes difficultés que nous avons pu rencontrer dans une conclusion. Vous trouverez aussi un résumé en anglais, un glossaire, et notre annexe dans la fin de ce document.

II - ENVIRONNEMENT DU STAGE

Notre travail porte donc, comme nous l'avons évoqué précédemment, sur la création d'un réseau de capteurs sans fil au sein d'une case à équipement d'un lanceur spatial, ainsi que la création d'une solution adaptée aux contraintes du milieu. Commençons par présenter les organisations qui encadrent ce projet :

Le CNES [2] a été fondé par Charles de Gaulle le 19 décembre 1961. Il s'agit d'un établissement public de recherche qui propose aux pouvoirs publics la politique spatiale de la France, situé au 2, Place Maurice-Quentin, à Paris. Le CNES intervient dans différentes activités [3] telles que :

- La défense, à l'aide de télécommunications sécurisées et l'écoute dans une optique d'espionnage.
- Les télécommunications, avec des nouveaux moyens de transmissions pouvant être de nouvelles antennes ou de nouvelles normes de télécommunications.
- L'observation, avec des nombreux satellites en orbite dans l'atmosphère terrestre, qui nous permettent d'étudier notre planète.
- Les sciences, dans lesquelles des recherches sur de nouvelles technologies sont utiles à l'exploration spatiale.
- Les lanceurs, pour lesquels sont recherchées une plus grande légèreté et une plus grande fiabilité, afin de posséder une plus grande autonomie dans l'accès à l'espace.

Le LIMOS [4] est une unité mixte de recherche en informatique. Elle est rattachée à l'Institut des Sciences de l'Information et de leurs Interactions (INS2I) du CNRS et aussi, de manière moins importante, à l'Institut des Sciences de l'Ingénierie et des Systèmes. Les tutelles académiques du LIMOS sont l'Université Clermont Auvergne (UCA) et Les Mines de Saint-Etienne (MSE) [5]. Notre activité répond à une demande du CNES faite au LIMOS afin d'améliorer l'infrastructure de télécommunication au sein de leurs lanceurs.

La raison de notre recrutement vient d'un besoin du LIMOS de faire reprendre le travail fait sur le long terme par les équipes précédentes, et de poursuivre ce qui a pu être fait. De plus, un travail réalisé en trois temps sur cette collaboration nous y apporte une grande maîtrise qui joue un facteur déterminant. Ceci vient notamment de la familiarité avec le matériel, le code, et les concepts que nous avons étudiés.

Comme dit précédemment, le réseau sur lequel nous travaillons sera déployé au sein d'une case à équipement, que vous pouvez voir sur la figure II.1. Ce compartiment des lanceurs généralement rassemble la majorité des équipements qui assurent les fonctions de guidage, de pilotage, de sauvegarde et de localisation.

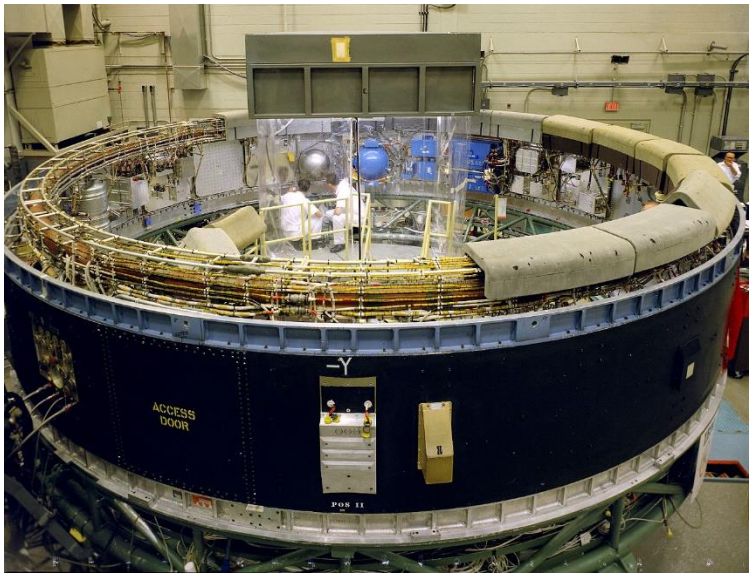

Figure II.1 Case à équipement de la fusée Saturn V

La vocation d'un lanceur est d'amener une charge utile dans l'espace, pour la placer en orbite autour de la Terre, ou dans l'espace interplanétaire. Cet environnement particulier résulte en des réseaux sans fil avec un taux d'erreur bits particulièrement important.

Notre solution doit répondre à un certain nombre de contraintes techniques. Tout d'abord, pour répondre à une contrainte de robustesse, nous proposons une solution logicielle afin de renforcer le taux des trames utilisables sur notre réseau. Nous proposons aussi, aux vues de la topologie de notre réseau définie dans la première partie du développement de ce rapport, une redondance matérielle de son élément central.

Notre réseau pour capteurs sans fil a pour objectif de remplacer les installations filaires électriques utilisées habituellement dans les lanceurs. Notre nouvelle installation sans fil doit satisfaire au minimum les besoins auxquels répond l'installation filaire en termes de performances et de débit. Ces besoins constituent notre contrainte temporelle. Les contraintes matérielles sont liées aux capteurs sans fil que nous utilisons, définis dans la partie suivante.

Il est important de noter que notre réseau a une durée de vie limitée au temps nécessaire au lanceur pour accomplir sa mission, dû à une contrainte qui dépasse ce projet qui concerne les lanceurs. Dans l'industrie spatiale, la majorité des lanceurs souffrent d'un problème majeur : ils ne sont, à l'heure actuelle, pas encore réutilisables, et doivent être détruits une fois que leur charge utile a été larguée. Il est possible que cela change au cours des prochaines décennies, mais pour l'instant, nous partons du principe que notre réseau sera fonctionnel jusqu'à ce que le lanceur soit détruit.

Le besoin du CNES de remplacer les installations filaires dans les lanceurs par des installations sans fil est lié à une question très importante dans le monde de l'aéronautique, qui influence la conception des engins concernés : le besoin de perte de poids. Facteur négatif très marquant pour ce type d'appareil, leur poids influencera la quantité de carburant à embarquer et l'espace de stockage nécessaire

dans ces engins. Plus un lanceur est lourd, plus la force nécessaire à son vol sera conséquente. Plus la force nécessaire au vol est grande, plus la quantité de carburant sera importante, ce qui renforce donc inévitablement la contrainte de poids par celui dudit carburant. Remplacer des dizaines de mètres de câble par une solution sans fil va permettre d'alléger de plusieurs centaines de kilos le poids total du lanceur. De plus, la présence de câbles nécessite de les placer au sein du lanceur, ce qui va créer des contraintes de places et des installations prévues pour. Ceci, fatalement, alourdira l'appareil.

Comme ce stage fait suite à de nombreuses collaborations, nous disposons d'un existant riche et complexe. Nous pouvons inclure dans cet existant le rapport de projet tutoré [8] réalisé dans les mois précédents le stage. Nous pouvons faire de cet existant le résumé suivant :

- Une partie de l'existant est théorique, et on y retrouve les rapports, les livrables, et la thèse faites sur le sujet ;
- L'autre partie de l'existant est matérielle, et on y retrouve les prototypes d'antenne et surtout l'implémentation logicielle de notre solution, qui est ce sur quoi nous avons principalement travaillé.

De par la maîtrise de l'ensemble du sujet que nous avons acquise lors des précédentes collaborations, nous avons pu établir dès le début du stage une liste d'objectifs approximative. Notre solution propose un protocole en plusieurs phases, qui se suivent, afin de répondre à nos besoins. La première phase se nomme phase de découverte du voisinage, et elle permet d'identifier tous les éléments dans notre réseau. Lors de cette phase, nous utilisons un mécanisme de détection de collision, qui permet de s'assurer que plusieurs messages n'ont pas été reçus en même temps. La phase suivante, la phase de collecte, consiste à récupérer les données mesurées par nos capteurs. L'implémentation de la phase de collecte sert de base pour celle de la robustesse de nos données et des actionneurs, qui sont alors appliquées à notre phase de collecte.

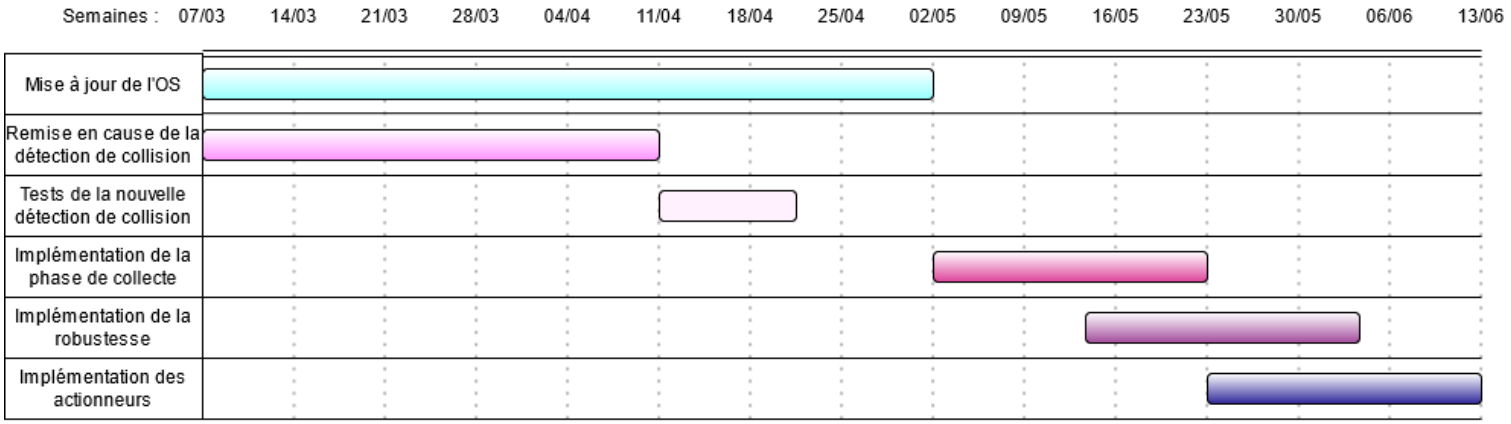

Figure II.2 Schéma des Objectifs

La remise en cause de la détection de collision consiste à revoir ce qui a été fait pour pouvoir s'assurer de son efficacité, puis

d’effectuer des tests de détections de collision pour le confirmer. La remise en cause demande de revoir de nombreuses options, ce qui explique sa durée. Parallèlement, la mise à jour du système d’exploitation (Operating System) est un travail nécessaire au commencement de l’implémentation de la phase de collecte. Par conséquent, elle peut être réalisée n’importe quand avant la phase de collecte. Une fois les tests et la mise à jour finis, nous pourrons commencer l’implémentation de la phase de collecte, durant laquelle il faudra aussi commencer à implémenter la robustesse de nos échanges. Une fois que nous aurons fini l’implémentation de la phase de collecte, nous pourrons rajouter les actionneurs, qui vont exécuter des ordres dans notre réseau, ou au moins commencer à les rajouter dans notre code.

Dans le développement qui va suivre, nous commencerons par une présentation fonctionnelle de notre projet. Nous ferons de la vulgarisation sur les différentes notions importantes du sujet, en vous présentant nos solutions et les différents aspects théoriques. Suite à cela, nous vous ferrons une présentation de notre implémentation, où vous pourrez trouver tout le travail qui a été effectué. Il est important de noter que de par la nature du projet, un important travail a été effectué pour maîtriser le contexte riche et complexe de ce projet de Recherche et de Développement, et qui est représenté dans la présentation fonctionnelle.

III – DÉVELOPPEMENT

PARTIE 1 : Problématiques et Présentation Fonctionnelle de la solution proposée

A) Informations générales sur les réseaux de capteurs sans fils.

Afin de pouvoir comprendre le travail fourni lors de ce stage, une présentation fonctionnelle de notre solution est nécessaire. Quand on parle de réseau informatique, on parle de communications entre plusieurs entités. Ces communications reposent sur des protocoles standardisés pour permettre le développement des entités communicantes quel que soit le constructeur. La majorité des solutions réseaux respectent, ou au moins se basent, sur les normes réseaux définies par l'IEEE. Notre réseau appartient à la famille des réseaux locaux (LAN), qui est définie par les normes IEEE 802.x. Dans le domaine des réseaux de capteurs sans fil, il existe plusieurs normes 802 qui pourraient convenir à notre solution. Par exemple, la technologie sur laquelle se base le bluetooth s'appuie sur la norme 802.15.1, et est adapté à un réseau d'entités communicantes sans fil. Cependant, le bluetooth est limité par son nombre d'entités possibles sur le réseau, pour un maximum de 7 en fonctionnement optimal. Notre équipe a donc par le passé décidé d'utiliser la norme IEEE 802.15.4 [17], qui correspond à la norme dédiée à un réseau sans fil avec faible débit, porté, et faible consommation énergétique.

Pour déployer un tel réseau, il est nécessaire d'identifier deux types d'appareils différents : Les coordinateurs et les capteurs/actionneurs. Les premiers vont avoir pour rôle de faire transiter les informations et les ordres sur le réseau, tandis que les seconds récolteront des informations et les fourniront aux coordinateurs, ou réagiront aux ordres transmis par ces derniers, qui contrôlent donc le réseau.

Au vu des besoins de notre solution, nous avons décidés d'utiliser un réseau avec une topologie dite en étoile.

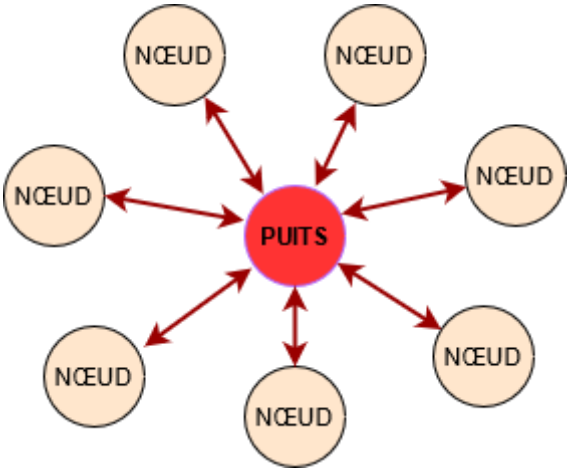

Figure III.1 Topologie en étoile

Dans ce genre de réseaux sans fil, on ne dispose que d'un seul coordinateur, et tous les capteurs / actionneurs en sont placés à un saut, c'est-à-dire qu'ils sont tous à portée du coordinateur. Comme on peut le voir dans la figure III.1, toutes les communications passent donc par l'entité au centre de notre réseau. De par la nature de notre réseau, et comme toutes les informations sont regroupées en un point en son centre, le coordinateur de notre réseau est donc considéré comme un puits. Durant la suite de ce document, le terme de puits sera utilisé pour parler du coordinateur central de notre réseau, et les capteurs / actionneurs seront désignés à travers le terme générique nœud.

Nous possédons deux types de cartes embarquées différentes : les Openmote-B [9], illustrées dans la figure III.2 qui joueront le rôle de nœuds, et la SAMR21-XPRO [10] , que vous pouvez voir dans la figure III.3 qui jouera le rôle de puits.

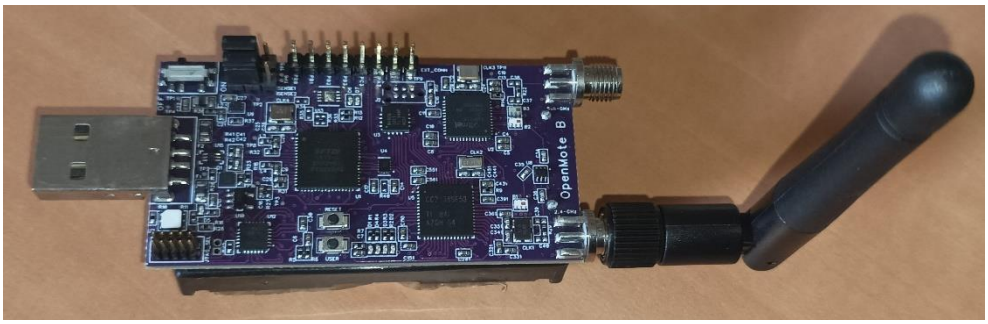

Figure III.2 Photo d'une Openmote-B

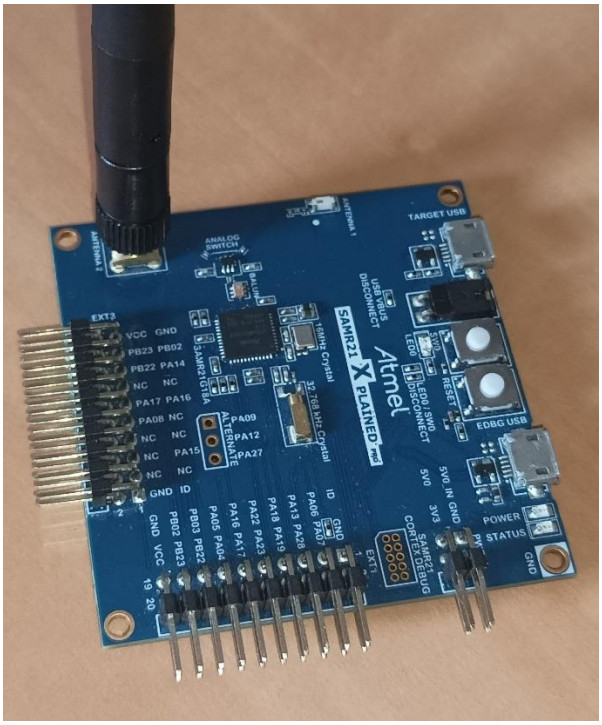

Figure III.3 Photo d'une SAMR21-XPRO

Nous avons décidé d'utiliser la topologie en étoile car elle offre des avantages conséquents liés à la robustesse de notre réseau. En effet, comme toutes les communications sont orchestrées par le puits, les nœuds ne pourront pas créer de collisions en envoyant des

trames en même temps. De plus, si un nœud est endommagé ou ne répond pas au puits, le réseau ne cessera pas de fonctionner. Cependant, le désavantage principal de cette topologie réside dans la notion de point de défaillance unique : si notre puits rencontre un problème, alors tout notre réseau cessera de fonctionner. Nous ne l'évoquerons que partiellement dans ce document, mais pour contrer ce problème, nous proposons une redondance du puits, qui consiste à placer un autre puits en tandem qui, si le puits original rencontre un problème, pourra poursuivre les opérations sur notre réseau en attendant le redémarrage du puits original ou la résolution du problème.

De manière générale, un capteur qui appartient à un réseau de capteurs sans fil est composée de plusieurs modules, ou unité, comme indiqué dans la figure III.5 :

- L'unité d'acquisition permet de mesurer des grandeurs physiques.
- L'unité de source d'énergie fournit l'énergie nécessaire au bon fonctionnement de la carte.
- L'unité de traitement est constituée d'un microprocesseur pour traiter les données et d'une mémoire de stockage pour les sauvegarder.
- L'unité de communication contient le module radio qui permet la transmission et la réception des données via un lien radio.

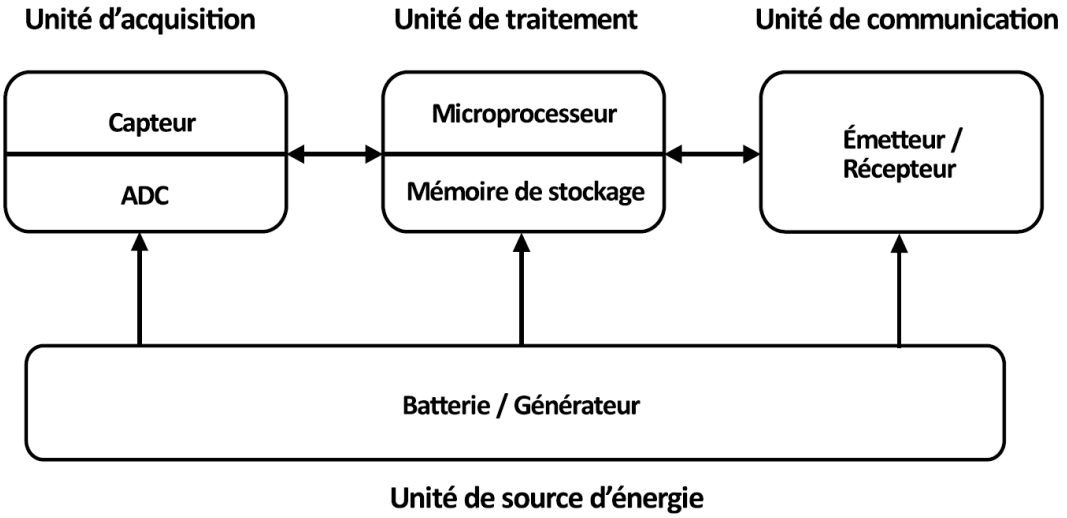

Figure III.4 Schéma des modules d'un capteur sans fil

B) Secteurs et phases dans notre solution.

Notre solution algorithmique est décomposée en trois états que l'on peut voir dans l'automate ci-dessous.

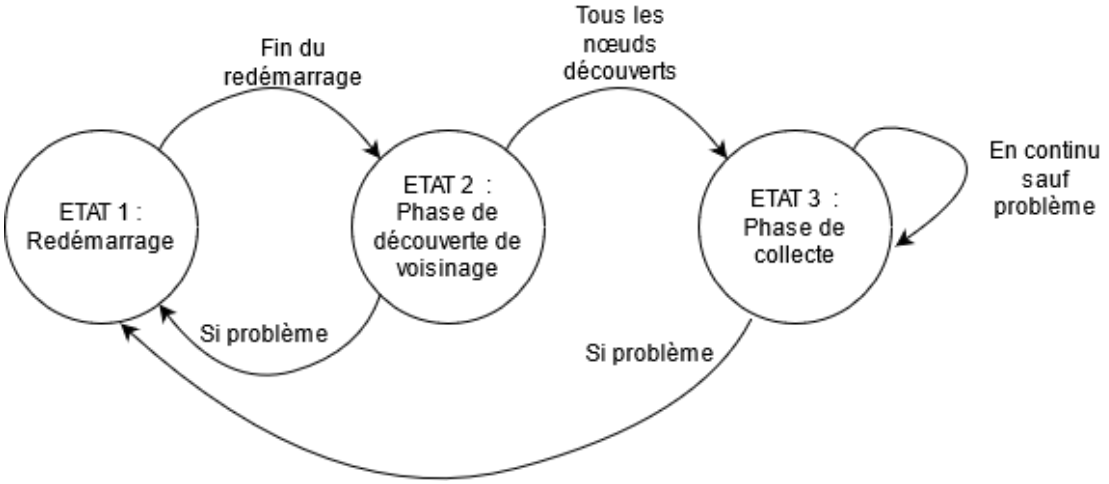

Figure III.5 Schéma des états dans notre protocole

L'état de redémarrage est explicite, dans le sens où l'on retourne à cet état en cas de problème, ou lors du démarrage de notre réseau. Le puits va s'initialiser et se configurer avant de pouvoir démarrer la phase suivante : la phase de découverte du voisinage.

Dans l'état phase de découverte, le puits va constituer notre réseau et renseigner, dans un tableau que l'on appelle table de connectivité, tous les nœuds qui sont membres de notre réseau.

L'état phase de collecte est atteint quand tous les nœuds de notre réseau ont été découverts. Le puits va appeler les nœuds en groupe en fonction des informations nécessaires et des ordres qu'il veut fournir.

Nous allons détailler ces deux phases dans cette partie, et nous nous intéresserons à leur implémentation dans la deuxième partie. Mais avant cela, il est nécessaire d'évoquer certaines particularités explorées et implémentées dans l'existant de ce projet. Compte tenu du domaine d'application temps réel, la solution consiste en un ordonnancement déterministe.

Géographiquement, le puits se trouvera au centre de la case à équipement, et nous pouvons considérer que les nœuds sont placés de manière aléatoire autours du puits. Chaque entité est équipée d'une antenne afin de communiquer avec les autres, et bien que les nœuds soient équipés d'antennes omnidirectionnelles, nous avons décidé de doter le puits d'une antenne directionnelle à rayon commutatif. En effet, comparé à une antenne omnidirectionnelle, l'antenne directionnelle permet de couvrir une zone plus large pour un coût énergétique moindre.

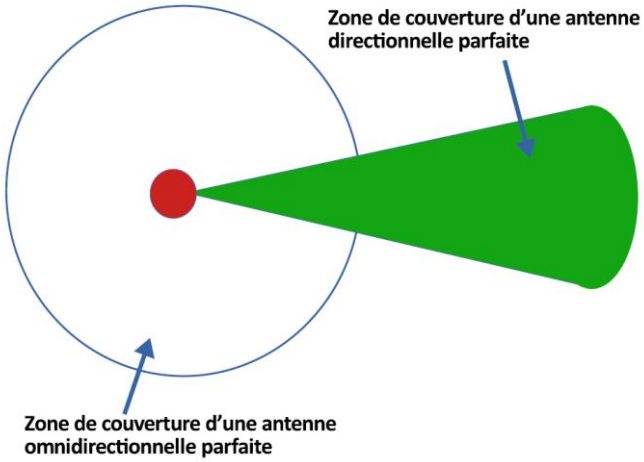

Figure III.6 Comparaison des portées d'une antenne omnidirectionnelle et directionnelle

Ceci permet d'augmenter la portée de la transmission, d'augmenter la durée de vie et la capacité du réseau en termes de débit. L'avantage d'une antenne directive à rayon commutatif réside aussi dans le fait que l'implémentation est simple et correspond aux besoins de notre réseau, et avec son utilisation, nous avons défini la notion de secteurs.

Les secteurs représentent les zones géographiques dans lesquelles chaque antenne de notre antenne directive à rayon commutatif émet, comme indiqué dans la figure III.7.

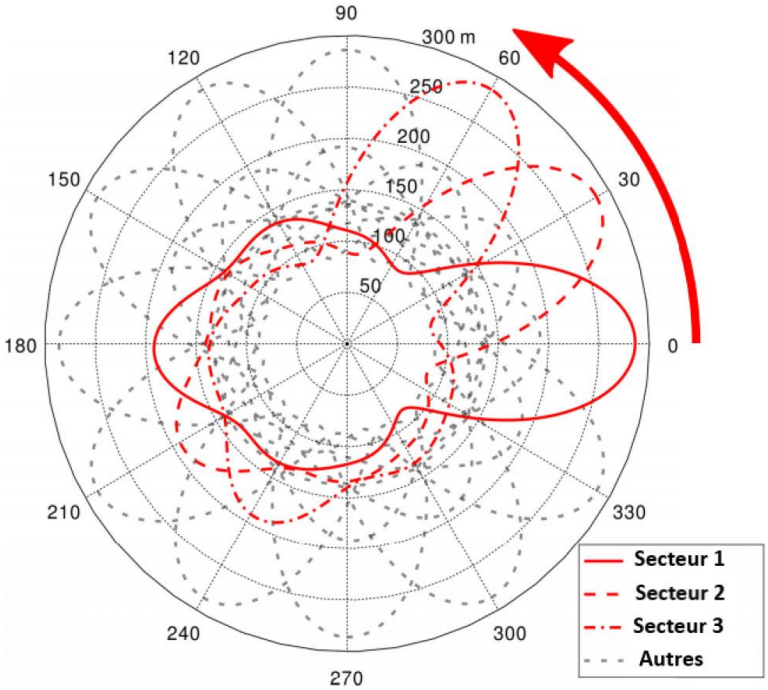

Figure III.7 Représentation des faisceaux de l'antenne couvrants les secteurs

L'addition de tous les secteurs nous donne donc une couverture égale à celle d'une antenne omnidirectionnelle, tout en possédant les avantages d'une antenne directionnelle. Par conséquent, chaque nœud pourrait être placé géographiquement dans un secteur. Cependant, il

est possible qu'un nœud soit découvert par plusieurs secteurs de l'antenne. Pour pallier cela, dans la table de connectivité, chaque nœud découvert dans chacun des secteurs est renseigné, et à l'aide de mesures énergétiques, nous pouvons déterminer quel est le meilleur secteur pour communiquer avec chacun des nœuds. Dans la figure III.8, nous vous présentons une photo vue de dessus de notre solution matérielle. Au centre, vous pouvez voir l'antenne commutative, avec les six secteurs indiqués en bleu. Nous pouvons remarquer qu'à l'exception du nœud 2, tous les autres nœuds sont distinctement dans un seul et unique secteur. Concernant le nœud 2, il recevra les messages qui transitent dans le secteur 1 et le secteur 2, mais le puits lui assignera un secteur parmis les deux en fonction de la qualité du signal entre le nœud 2 et chaque secteur concernés.

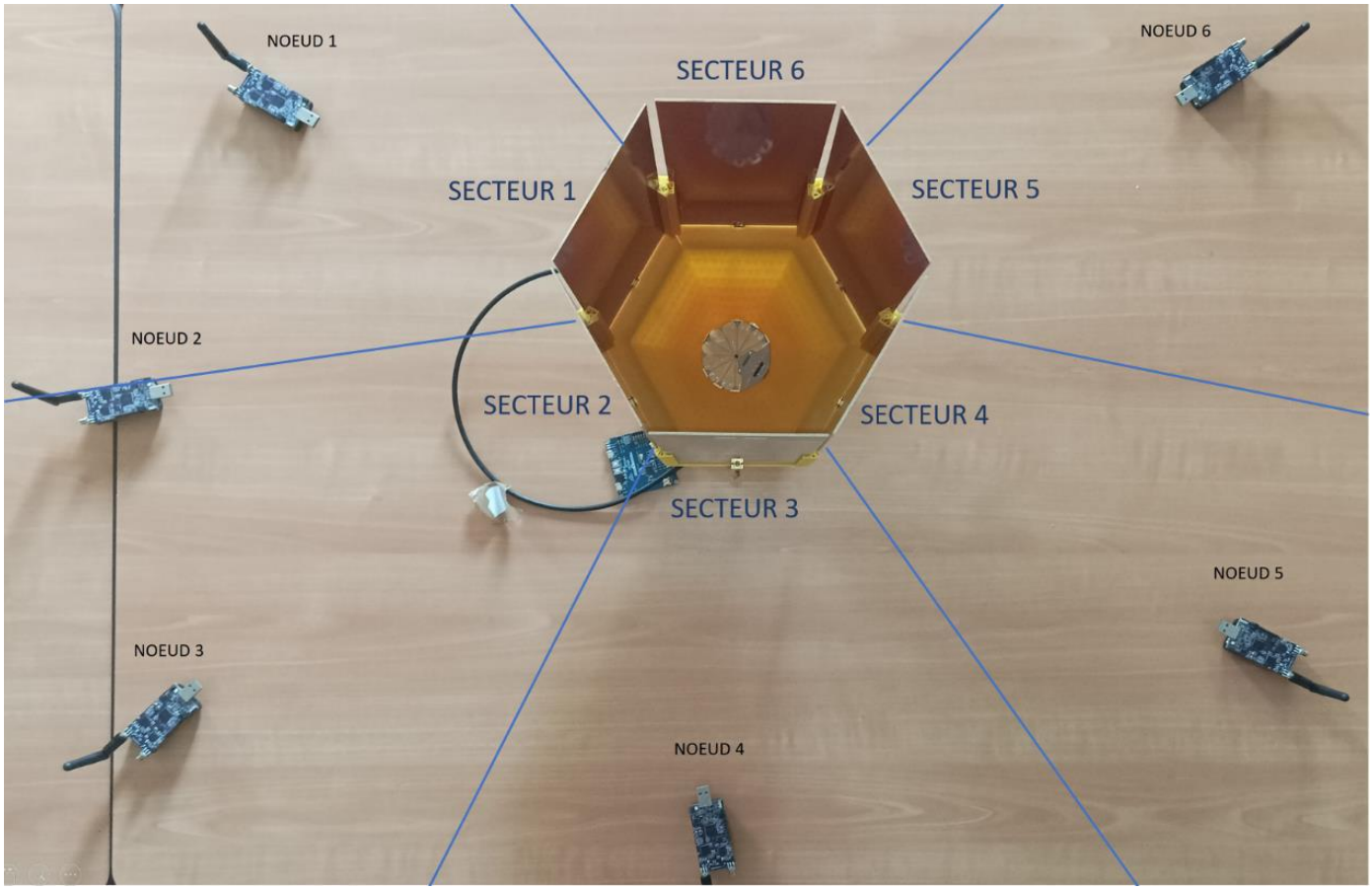

Figure III.8 Vue du dessus de la répartition géographique de notre matériel lors des phases de tests

C) Présentation théorique de notre algorithme.

La notion de secteur étant maintenant définie, nous pouvons nous attarder sur la phase de découverte de voisinage. Dans cette dernière, le puits va renseigner dans sa table de connectivité chaque nœud détecté dans chacun des secteurs. Pour ce faire, nous avons opté pour un mécanisme de POLLING SELECTING, c'est-à-dire un système dans lequel le puits va diffuser un message dans un secteur auquel les nœuds vont répondre. Le mécanisme de cet échange sont montrés dans le schéma ci-dessous :

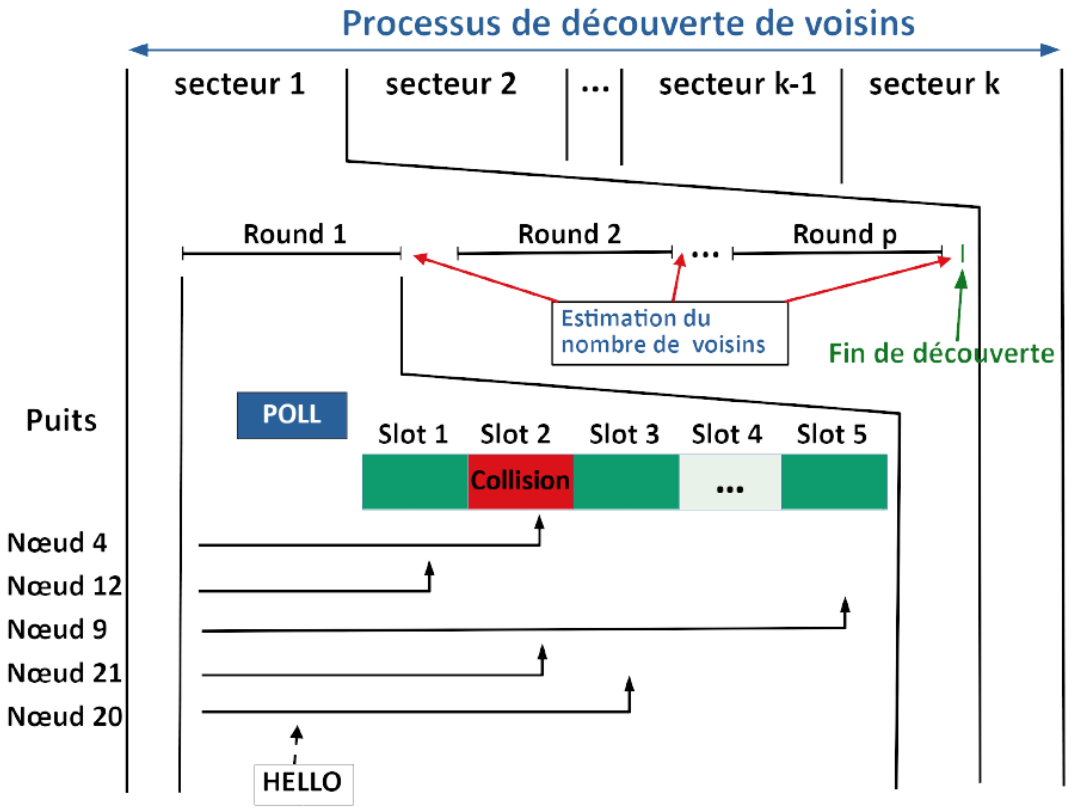

Figure III.9 Processus général de découverte du voisinage et d'estimation du nombre de voisins

Comme vous pouvez le voir dans la figure III.9, le puits va diffuser un message de poll (polling, en français sondage) à tous les nœuds situés dans un secteur. Suite à la diffusion du poll, le puits va attendre pendant un certain nombre de slots, dont le nombre dépend du nombre de nœuds dans notre réseau. Les nœuds vont, à la réception du message, tirer aléatoirement un slot et répondre au puits dans le slot temporel qu'ils ont sélectionné. Le puits va alors écouter les réponses des nœuds et renseigner chaque nœud et leur relevé d'énergie associé dans la table de connectivité. Cette suite d'action est appelée un round, et le puits fera autant de rounds que nécessaire dans chaque secteur pour s'assurer que tous les nœuds présents ont été découverts. Dans un même secteur, les nœuds qui ont été découverts lors de rounds précédents ne répondent plus au poll du puits.

Cependant, de par l'aléatoire dans le choix du slot correspondant, il est possible que plusieurs nœuds répondent dans le même slot. Pour différencier les différents cas de figures qui

pourraient se produire lors d'un slot, nous avons incorporé une notion d'état des slots.

- Si lors du slot le puits reçoit un message correct, c'est-à-dire un message avec un début et une fin qui respecte la norme 802.15.4, alors il est dit **valide** ;
- Si lors du slot le puits ne reçoit aucun message et ne détecte aucun relevé d'énergie, le slot est dit **vide** ;
- Si lors du slot le puits détecte un message avec un début qui respecte la norme 802.15.4 mais qu'elle est incomplète, relève un niveau d'énergie sur le canal supérieur à un seuil, ou qu'il reçoit un message valide ainsi qu'au moins autre message, valide ou non, le slot est alors dit **en collision** (concernant ce dernier cas, le slot sera dit en collision, mais le message valide sera malgré tout traité et rajouté à la table de connectivité).

Les nombres de slot valides, vides, et en collision sont indiqués respectivement par les variables S1, S0 et S2*. A l'aide de ces variables, le puits pourra alors faire une estimation du nombre de slots nécessaire pour découvrir tous les nœuds lors du round suivant, et mettre en place la fin de la découverte de voisinage dans le secteur. En effet, si le puits ne détecte plus aucune réponse des nœuds (si S1 = 0, S2* = 0 et S0 = le nombre de slot prévu pour ce round), il fera un potentiel dernier round dans ce secteur avec un seul et unique slot pour s'assurer que tous les nœuds ont bien été découverts. Comme la probabilité que les nœuds qui n'ont pas été découvert dans ce secteur choisissent ce slot est de 100%, s'il n'y a ni collision, ni message valide reçu lors de ce round, alors on peut considérer que tous les nœuds ont été découverts, et qu'on peut passer au secteur suivant.

Le travail fourni lors de ce stage a dans un premier temps été concentré sur la détection des collisions, et plus particulièrement sur la détection d'énergie sur le canal. En effet, comme dit dans [12], de nombreuses manières de détecter les collisions lors d'une réception ont été développées au cours des années, et il a été nécessaire de tester la solution la plus fonctionnelle. Nous en avons étudié trois, que nous détaillerons dans la deuxième partie de ce rapport. Leurs utilisations, toutes trois identiques, étaient les suivantes : dès l'instant où le puits a fini d'envoyer son poll, il va mesurer l'énergie sur le canal tout le long du round. Si la mesure d'énergie dépasse un certain seuil, alors on peut considérer qu'il y a de l'activité sur le médium lors du slot. Si dans le même slot on a un message correct, alors le slot sera valide, sinon le slot sera en collision.

Une fois tous les nœuds découverts, la phase de collecte pourra commencer. Il est à noter qu'il est possible à tout moment de retourner à la phase de découverte de voisinage si l'ordre est donné.

Dans la phase de collecte, le puits possède deux modes de fonctionnements, liés à deux files d'attentes d'ordres distinctes. Les ordres peuvent être à destination des nœuds actionneurs comme des nœuds capteurs. Le premier mode est dit au fil de l'eau : dans une liste d'ordre en attente de type FIFO (premier arrivé, premier servi), nous stockerons les ordres liés à chaque secteur. On va passer de

secteurs en secteurs, et donner des ordres aux nœuds concernés dans le secteur actuel. C'est le mode de fonctionnement de base, et on y revient une fois qu'on a quitté le second mode, dit urgent : si la liste d'ordre en attente du mode urgent n'est pas vide, alors le puits, dès la fin de la requête courante, videra cette liste d'urgence. Pour ce faire, le puits changera de secteur directement, peu importe son emplacement géographique, et enverra les ordres urgents un à un. Une fois ce processus fait, si la liste d'urgence n'est pas vide, on passera au prochain ordre urgent, sinon, on retournera en mode au fil de l'eau et dans le dernier secteur dans lequel on se trouvait avant qu'on traite l'urgence.

Afin d'optimiser le temps des transactions dans notre protocole, nous avons aussi introduit le concept d'agrégation de données aux nœuds capteurs, au niveau de la source. En effet, pour minimiser le nombre de transactions, les nœuds stockent les mesures qu'ils récupèrent dans une liste d'attente, et lorsqu'ils reçoivent la requête de récolte de ces données du puits, ils agrègent ces données en une seule trame avant de l'envoyer au puits.

Les dites données ont aussi des contraintes de fraîcheur, et il n'est donc pas nécessaire de les fournir au puits si elles ne sont plus valides. Afin de gérer cela, au moment où les nœuds vont faire leurs mesures, ils vont aussi récupérer un horodatage, utile pour les dater. Si l'horodatage d'une mesure dépasse une certaine durée de validité (durée variable en fonction du type de données), alors elle est jetée (retirée de la file d'attente).

Nous avons aussi pris en compte les interférences qui seront fréquentes dans le milieu dans lequel notre réseau sera déployé. Ainsi, nous proposons de protéger certaines informations par des code correcteurs d'erreur, que nous définirons dans la deuxième partie de ce développement.

L'exécution d'un ordre envoyé par le puits lors de la phase de collecte se déroulera de la manière suivante, en accord avec la figure III.10.

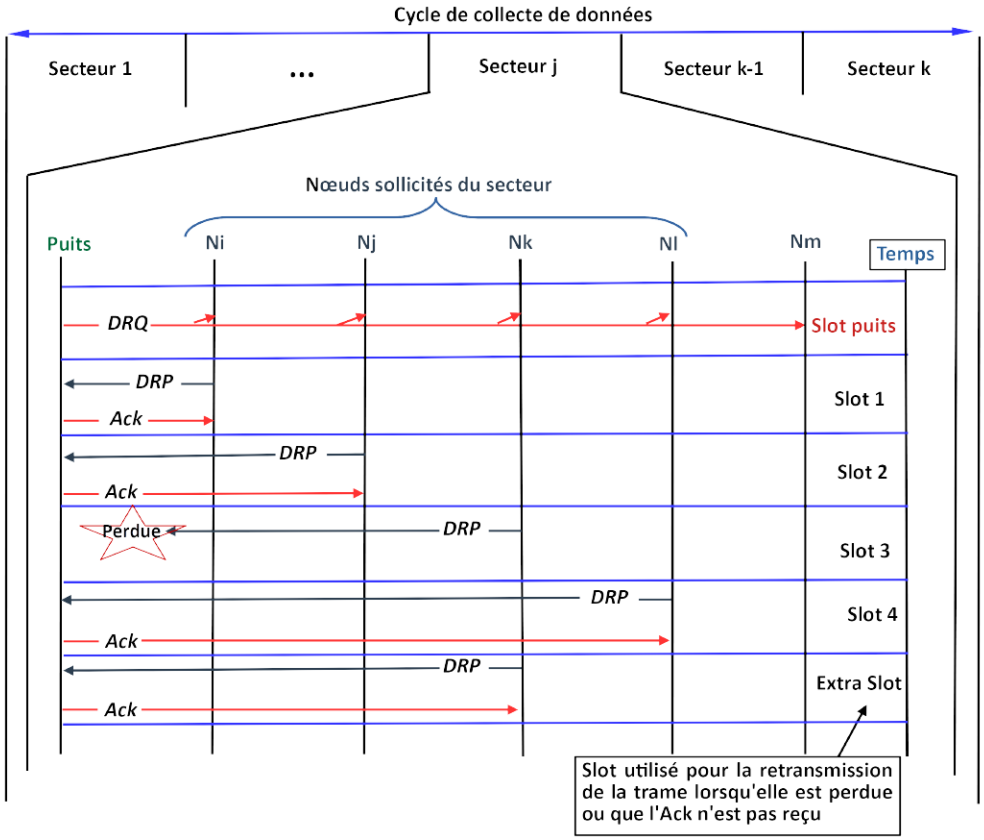

Figure III.10 Exemple d'exécution de la méthode de polling groupé avec extra slot

Dans un premier temps, le puits va diffuser aux nœuds dans un secteur une requête de données (Data ReQuest, DRQ). Le puits se mettra alors en écoute pendant un nombre de slots égal au nombre de nœuds concernés plus un, afin de mettre en place l'extra slot, un mécanisme que nous détaillerons ci-dessous. Bien que diffusé, il est possible de sélectionner des nœuds spécifiques en rajoutant dans notre trame une liste d'adresse des nœuds concernés. Lorsque que les nœuds vont recevoir cette DRQ, ils vont vérifier si leur adresse est incluse dans la trame, et si oui, sa position dans la liste. En fonction de cette indication, ils pourront déterminer dans quel slot temporel ils répondront au puits. Les nœuds attendront alors leur tour, et avant d'envoyer leurs réponses (Data ResPonse, DRP). Ils protégeront leurs mesures et certaines informations utiles de la trame des interférences en appliquant nos codes correcteurs d'erreurs.

Lorsque le puits recevra cette DRP, comme indiqué dans la figure III.11, il vérifiera alors le CRC de cette trame. Le CRC, ou code de redondance cyclique, est une indication mise dans le pied de trames pour vérifier leurs intégrités. Lors de l'envoi, les octets transmis seront passés à travers un polynôme, et le résultat de cette opération, le CRC, sera alors transmis. En recevant une trame, un appareil qui utilise le CRC va appliquer le même polynôme sur les octets qu'il reçoit. L'appareil pourra alors, en comparant les deux CRC, savoir si

l’intégrité de la trame a été respecté, c’est-à-dire que la trame telle qu’on l’a reçu correspond à celle que l’on a envoyé. S’il est valide, il enverra un acquittement en réponse au nœud capteur pour lui faire savoir que les données ont été correctement reçues. Un acquittement est une courte trame que l’on envoie en réponse à un message correctement reçu pour informer l’envoyeur de la bonne réception de la trame. Ainsi, le nœud pourra vider sa file d’attente de mesures. Si le CRC n’est pas valide, alors, avant de jeter la trame, le puits va essayer de la récupérer en utilisant nos codes correcteurs d’erreurs. Si le résultat de cette opération indique que la trame a été correctement récupérée, le puits pourra envoyer au nœud un acquittement, qui aura le même rôle qu’évoqué précédemment. Sinon, le puits jettera la trame, et le nœud, n’ayant pas reçu d’acquittement, ne videra pas sa liste d’attente de mesure et attendra alors l’extra-slot.

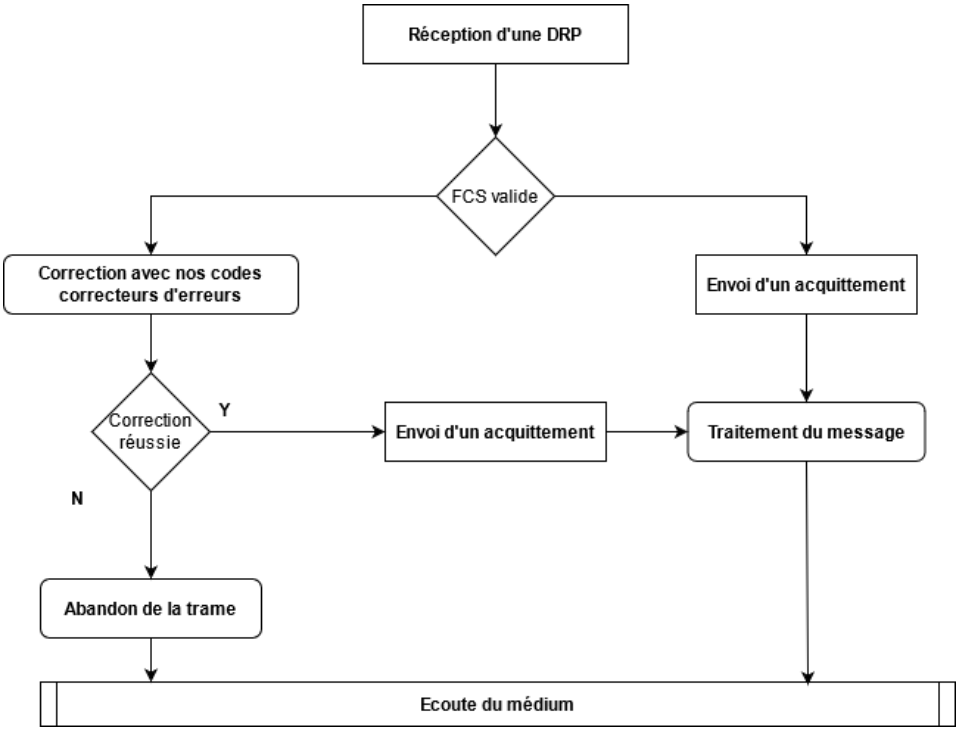

Figure III.11 Automate d’état de notre puits lors de la réception d’une DRP

L’extra-slot est un mécanisme que nous avons introduit dans le cas où un problème ait lieu lors de la transaction avec le puits. Les nœuds concernés par l’extra-slot, à savoir ceux qui n’ont pas vidés leur file d’attente de mesure, vont alors renvoyer leurs messages dans ce nouveau slot prévu à cet effet. Cette méthode est limitée, dû au fait que si 2 nœuds répondent dans l’extra-slot, des collisions vont être créées dans cet extra-slot. On remarque cependant [1] que le rajout de cet extra-slot augmente bien les performances de notre réseau. De plus, en rajoutant la protection contre les interférences avec nos codes correcteurs d’erreurs, on réduit considérablement le taux de transactions qui échouent. Les nœuds qui n’auront cependant pas réussi

leurs transactions avec le puits lors de l'extra-slot conserveront leurs mesures en attendant la prochaine DRQ du puits les concernant.

Ceci marque la fin de la présentation fonctionnelle de notre solution. Comme nous l'avons vu, notre solution répond à la contrainte de robustesse en faisant une redondance du puits, ainsi que l'utilisation de code correcteur d'erreurs afin de protéger nos trames des interférences du milieu. Notre protocole nous permet de respecter les contraintes temporelles strictes qui nous sont indiqués, à l'aide d'une phase de découverte du voisinage complète et efficace. De plus, notre mécanisme de collecte de données est performant, et son adaptation suivant les situations permet en théorie de respecter nos contraintes temporelles. Dans la partie qui va suivre, nous allons voir en détail le travail qui a été effectué et l'implémentation de la solution théorique.

PARTIE 2 : PRESENTATION DE L'IMPLEMENTATION

A) Les trames et les couches réseaux.

Avant de pouvoir pleinement nous plonger dans le travail de codage effectué lors de ce stage, il nous est d'abord nécessaire d'expliquer quelques notions importantes quant aux aspects techniques de l'implémentation. Nous parlerons d'abord des trames sous tous leurs aspects, puis nous fournirons les informations nécessaires pour comprendre le concept d'interruptions et des rôles qu'elles jouent dans notre implémentation, et enfin, nous donnerons quelques mesures temporelles pour permettre une meilleure compréhension de la contrainte temporelle de notre solution.

Comme nous l'avons précisé plus haut, notre solution, bien qu'elle en diverge, se base sur la norme IEEE 802.15.4 de 2006 [17]. Cette dernière définit un grand nombre de scénarios et de concepts sur lesquels se basent les constructeurs de cartes embarquées pour offrir un comportement standardisé et optimal, comme les primitives de réseaux ou encore les trames et leurs formats. Cette norme propose plusieurs formats de trames de tailles et de contenu variable, que nous allons expliciter ci-dessous.

Un concept commun que l'on retrouve dans de multiples disciplines en réseau est celui d'encapsulation des trames, dont une représentation schématique peut être visualisée dans la figure III.12.

<table>
<tr><td>PHY</td><td>SHR</td><td>PHR</td><td colspan="3">Charge utile PHY</td></tr>
<tr><td>MAC</td><td colspan="2" rowspan="2"></td><td>MHR</td><td>Charge utile MAC</td><td>MFR</td></tr>
<tr><td>Couches applicatives</td><td colspan="2">Notre trame</td><td></td></tr>
</table>

Figure III.12 Encapsulation des trames

En effet, pour qu'une trame soit lisible et compréhensible par les différents appareils, elle ne peut juste consister des données que nous souhaitons faire transiter. Ainsi, les trames sont encapsulées d'une manière à ce que les différentes couches de nos appareils puissent traiter les informations qui leur sont associées. Dans notre solution, nous comptons trois couches différentes sur chaque entité, et donc trois encapsulations différentes, comme vous pouvez le constater sur la figure III.13.

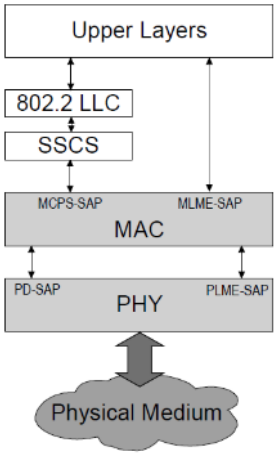

Figure III.13 Représentation des couches réseaux dans la norme IEEE 802.15.4

En premier nous avons la couche PHY (physique), qui transforme les informations envoyées ou reçues respectivement en un signal radio ou en bits. Elle ne considère les informations que comme des bits, c'est-à-dire qu'elle ne traite pas ce qu'elle reçoit, mais le transmet juste à la couche supérieure, la couche MAC (Medium Access Control). Au niveau de la couche physique, la trame est divisée en 3 groupes : le SHR (Synchronisation HeadeR), qui est utilisé pour synchroniser le module radio avec la trame ; le PHR (Physical HeadeR), qui est utile pour déterminer la taille totale de la trame (dans la norme IEEE 802.15.4, la taille maximale d'une trame est 127 octets) ; la charge utile physique (PSDU), qui contient toutes les autres données.

		Octets		
		1		**variable**
Preamble	SFD	Frame length (7 bits)	Reserved (1 bit)	PSDU
SHR		PHR		PHY payload

Figure III.14 Représentation d'une trame au niveau de la couche physique

Ensuite, nous avons la couche MAC. Cette couche contient les informations principales que nous allons utiliser pour traiter la trame. On peut remarquer que les adresses de destination et de sources peuvent avoir plusieurs tailles. En effet, dans la norme 802.15.4, il est défini la notion d'adresses courtes, une adresse sur 2 octets qui est un créée à partir de l'adresse longue sur 8 octets d'une carte embarquée.

<table>
<tr><td>Octets: 2</td><td>1</td><td>0/2</td><td>0/2/8</td><td>0/2</td><td>0/2/8</td><td>0/5/6/10/ 14</td><td>variable</td><td>2</td></tr>
<tr><td rowspan="2">Frame Control</td><td rowspan="2">Sequence Number</td><td>Destination PAN Identifier</td><td>Destination Address</td><td>Source PAN Identifier</td><td>Source Address</td><td rowspan="2">Auxiliary Security Header</td><td rowspan="2">Frame Payload</td><td rowspan="2">FCS</td></tr>
<tr><td colspan="4">Addressing fields</td></tr>
<tr><td colspan="7">MHR</td><td>MAC Payload</td><td>MFR</td></tr>
</table>

Figure III.15 Représentation d'une trame au niveau de la couche MAC

Dans notre solution, tous les champs ne nous intéressent pas. De manière générale, notre MHR (MAC HeadeR) a une taille de 9 octets, sauf pour les messages de poll du puits lors de la phase de découverte du voisinage, qui a une taille de 7 octets. On y retrouve : le Frame Control, qui nous donnera des informations sur comment la trame doit être traitée - par exemple si la trame doit être acquittée, ou encore dans le cas où le PAN de destination et de source sont égaux, si l'on écrit une seule fois l'identifiant PAN dans le champs Destination PAN Identifier et pas dans le Source PAN Identifier ; le Sequence Number, qui est propre à l'appareil, et qui permet de recevoir des acquittements et d'identifier la trame ; les champs PAN Identifier, qui permettent de savoir dans quel réseau l'on se trouve – dans notre cas, seul le champ Source PAN Identifier n'est donc pas renseigné - ; l'adresse de destination, qui est soit égale à l'adresse d'un nœud, soit en diffusion (0xFFFF) ; l'adresse source, qui est toujours égale à l'adresse de l'appareil envoyeur, sauf pour le puits dans la phase de découverte du voisinage, qui dans ce cas est nulle (car le puits est le coordinateur du réseau).

Dans la figure III.15, nous pouvons aussi voir que la charge utile MAC correspond à notre charge utile de couche applicative. De manière générale, on appelle la charge utile de la couche MAC le MSDU (MAC Service Data Unit).

Enfin, nous avons la couche applicative, où notre application va tourner. C'est dans cette couche qu'on exploite habituellement la charge utile de la couche MAC, et c'est dans cette charge utile que l'on va stocker nos données. Elles auront alors des formats spécifiques définis [1] propre à leur utilité (poll et réponse au poll pour la phase de découverte du voisinage, DRQ et DRP pour la phase de collecte) et à ce format sera associé dans notre code une structure prédéfinie. Tout ce que nous avons alors à faire, c'est de définir une structure dans notre code qui sera toujours accessible, et que l'on va configurer lors des différents traitements. Quand un nœud capteur voudra répondre à un ordre du puits, il suffira de charger la structure dans la charge utile MAC.

B) Les interruptions au niveau de notre puits.

Tout cela étant précisé, nous pouvons à présent définir le concept d'interruptions et d'exceptions, et expliquer son utilité dans notre code. En informatique, un programme est défini comme un fichier contenant des lignes de code. Lorsqu'un programme s'exécute, il devient un processus qui va suivre une liste de commandes et d'opérations jusqu'à la fin de son exécution. On dit alors que le processus suit son flot d'exécution. Cependant, il est possible qu'un événement externe ou interne vienne perturber l'exécution du programme, que l'on appelle une interruption. Ainsi, pour qu'une interruption n'empêche pas l'exécution attendu de notre programme, le processus sera suspendu et le gestionnaire d'interruptions s'exécutera afin de s'occuper de l'événement qui a provoqué le problème dans notre processus. Une fois que le gestionnaire d'interruptions aura fini de s'exécuter, il rendra la main au processus. Il est important de noter que lorsqu'on gère une interruption, c'est-à-dire que nous sommes dans un contexte d'interruptions, de nombreux problèmes peuvent avoir lieu, car la pause du processus principal peut amener de nombreux problèmes d'accès à la mémoire. Par conséquent, lors d'une interruption, on va chercher à passer un temps minime dans le contexte d'interruptions avant de rendre la main au programme. Plus de détail à ce sujet peut être lu dans [11].

Les interruptions peuvent être liées à de nombreux événements, et pas nécessairement à un problème. Elles sont généralement soit prévues pour empêcher une opération de s'exécuter, comme une division par zéro, ou pour prévenir qu'un événement a eu lieu, comme un changement de processus, ou encore la réception d'une trame. C'est sur cet aspect précisément que les interruptions jouent un rôle important dans notre programme.

En effet, comme nous l'avons expliqué plus haut, l'unité de traitement de nos cartes et l'unité de communication sont deux entités différentes. Elles communiquent l'une avec l'autre à l'aide de broches et de circuits électriques, qui ont de multiples fonctions. Une de ces broches a pour rôle de prévenir l'unité de traitements que des interruptions ont été générées par l'unité radio, et notre CPU va alors lire un registre de notre module radio pour identifier les interruptions qui ont eu lieu et ainsi pouvoir les traiter. Tous les modèles d'unités radio n'ont pas nécessairement les mêmes interruptions, mais on en retrouve généralement quelques-unes peu importe le modèle. Dans notre cas, le système d'exploitation sur lequel nous programmons, RIOT-OS, propose des événements associés à des interruptions, ce qui permet de reconnaître ces interruptions sous un même nom.

Avant de nous y intéresser, nous devons définir le concept d'état pour un composant. Dans le cas du module radio, il possède généralement plusieurs états dont un pour recevoir les trames, et un autre pour les transmettre. Le module radio ne peut en temps normal pas être dans deux états différents à la fois. Pour les différencier, l'état de transmission d'une trame a été nommé 'TX' (Transmit), l'état de réception d'une trame 'RX' (Receive) et l'état neutre 'idle'. Parfois, si l'on parle simultanément des aspects émetteur et récepteurs d'un module, on emploiera le terme 'TRX'.

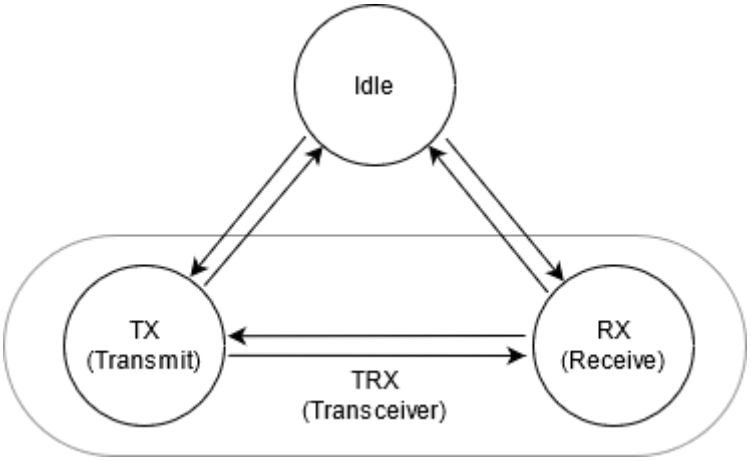

Figure III.16 Schéma des états que peuvent prendre nos cartes embarquées

Pour revenir sur nos interruptions, celles que nous utilisons sont au nombre de 4, et elles sont toutes liées à l'aspect émetteur-récepteur de nos modules radio, à l'envoi ou à la réception d'une trame :

L'interruption TX_START est levée lorsque le module radio commence à émettre un message.

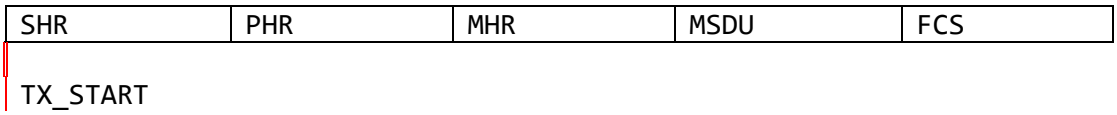

Figure III.17 Timing de l'interruption TX_START

L'interruption TX_END est levée lorsque le module radio a fini d'émettre un message s'il n'y a pas eu de problème lors de l'envoi.

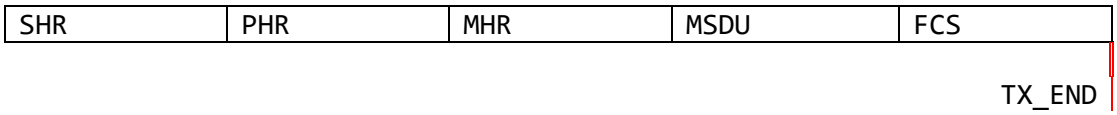

Figure III.18 Timing de l'interruption TX_END

L'interruption RX_START est levée lorsque le module radio a commencé à recevoir un message. Plus spécifiquement, elle est levée après la détection du PHR, si le SHR et le PHR sont valides.

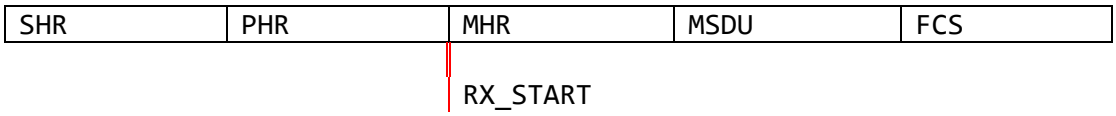

Figure III.19 Timing de l'interruption RX_START

L'interruption RX_END est levée lorsque le module radio a fini de recevoir un message, et que le message est correct selon nos critères de validation.

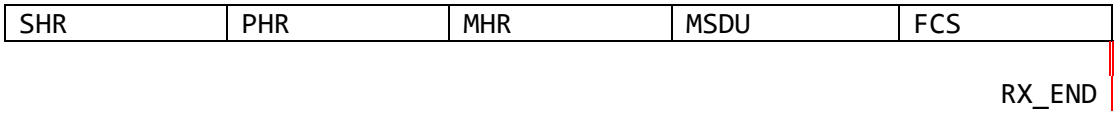

Figure III.20 Timing de l'interruption RX_END

Grâce à ces interruptions, nous pouvons maîtriser le flot d'exécution de notre processus, et elles jouent un rôle capital dans le bon fonctionnement de notre solution.

Enfin, il est intéressant d'étudier certaines durées théoriques et pratiques afin de vous donner une idée des contraintes temporelles qui pèsent sur notre projet. Tout d'abord, revenons à son fonctionnement théorique : notre réseau a un débit théorique de 250 kbits/s. En excluant le temps de redémarrage du puits, car il est dépendant du matériel, et le temps de redémarrage des nœuds, car ils sont négligeables dans notre solution, lors de la phase de découverte du voisinage, chaque slot aura une durée du 5 ms. Lors de la phase de collecte, le temps de chaque slot a été estimé entre 5 et 10 ms, et est fixé à 7 ms dans notre code pour l'instant. Pour ce qu'il en est de la réception d'une trame, la fiche technique de notre puits dans la section 36.1.3 [10] indique un temps de réception TR = 192 + (9 + n) x 32, avec n = le nombre d'octets dans le MSDU.

C) Travail sur la détection de collision.

Maintenant que ces concepts ont été définis, nous pouvons parler plus en profondeur du travail qui a été effectué, en commençant par le travail sur la phase de découverte du voisinage.

Cette partie de notre solution était réalisée dans l'existant lorsque ce stage a commencé. La majorité de cette implémentation était fonctionnelle, et en plaçant les nœuds autour du puits, tous étaient découverts. Le premier travail qui a été effectué consistait à corriger certains problèmes mineurs, tel que les adresses courtes qui n'était pas implémenté automatiquement. Cependant, notre découverte de voisinage, bien que fonctionnel, semblait pouvoir être optimisée, en particulier sur la détection de collision du côté récepteur. C'est donc sur cet aspect que nous nous sommes focalisés.

Une collision a lieu lorsqu'un récepteur ne peut correctement recevoir un message dû à une interférence. Une interférence peut être définie comme :

- Un autre message reçu en provenance d'un nœud.
- Un signal radio reçu en provenance d'un autre appareil.
- Un problème avec l'antenne ou l'appareil lors de la réception.

Notre solution incorporait précédemment un système afin de détecter ces collisions. Ce système est le suivant :

Lorsque l'on a fini d'envoyer un poll, notre puits va mettre une alarme qui correspondra à la fin de notre round. A partir de là, le

puits va entrer dans une boucle qui s'interrompra soit si une interruption en provenance du module radio est levée, soit si le round est fini. Nous pouvons voir cette boucle dans la figure III.21, ainsi que les moments où la boucle de détection de collision est activée ou inactivée. Dans cette figure, le temps des interruptions est exagéré pour noter les différences, il se compte en micro secondes en temps normal.

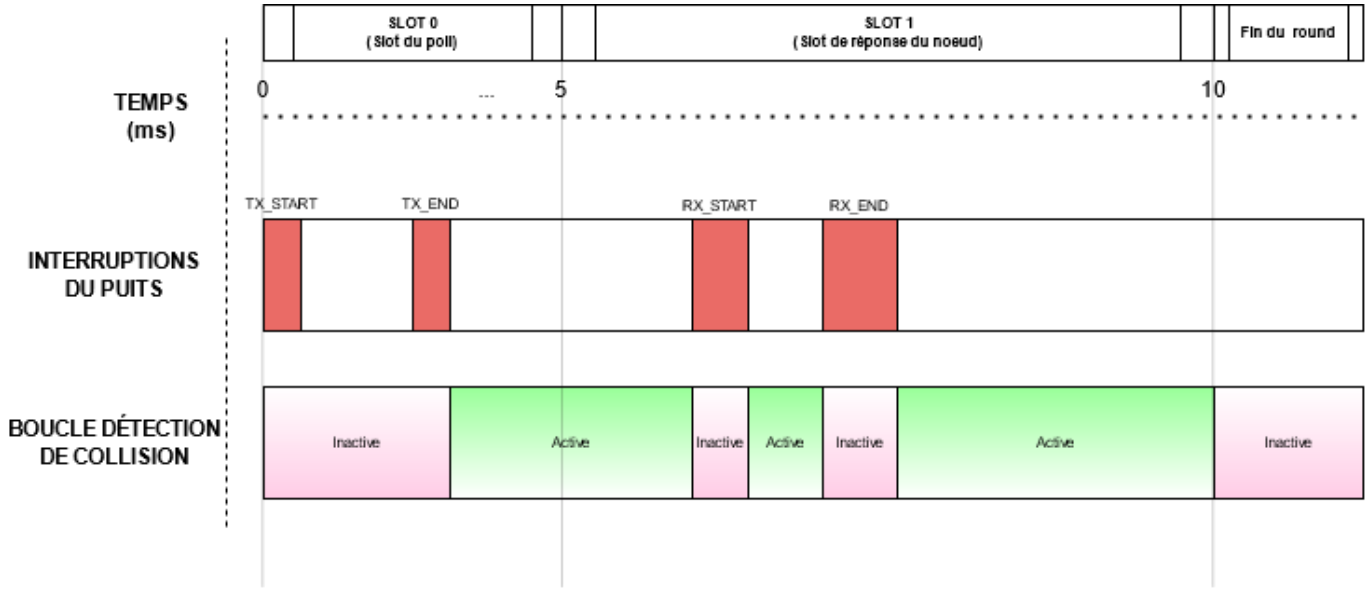

Figure III.21 Représentation du fonctionnement de la boucle de collision avec un seul slot, pour prévu pour un seul nœud

Cette boucle est sensée permettre de détecter des collisions, et notre programme va alors l'utiliser pour décider de l'état des slots afin de pouvoir obtenir une estimation du nombre de slot nécessaire précise. Son fonctionnement est le suivant, et est résumé dans la figure III.22 :

Nous disposons d'un nombre de slot par round, qui, à la du dit round, sont tous dit vides par défauts.

A chaque fois que le puits détecte une interruption RX_END, la vérification de l'intégrité de la trame a été validé et rajouté dans la table de connectivité, et notre slot est donc valide si un seul RX_END a été levé lors du slot.

Si nous avons au moins un deuxième RX_END dans un seul slot, le slot est alors en collision, cependant comme les trames ont été validées, on peut les rajouter dans notre table de connectivité.

Si un RX_START a été levé lors d'un slot, on va regarder l'état de ce slot. S'il a été mis en valide précédemment, alors nous pouvons en conclure que le RX_START correspond au RX_END qui a été validé plus tôt. Cependant, si le slot est toujours vide, cela implique qu'un RX_START a eu lieu mais qu'aucun RX_END, qui confirme donc la validité du message, n'a été levé. Par conséquent, une collision a eu lieu.

Si lors du round, une collision a été détecté du côté récepteur dans un slot, alors on mettra le slot dans l'état en collision s'il était vide originellement.

	Valeur de S0	Valeur de S1	Valeur de S2*
Fin du round	Nombre de slot	0	0
UN RX_END	S0 - 1	S1 + 1	S2*
DEUX OU PLUS RX_END	S0	S1 - 1	S2* + 1
RX_START sans RX_END	S0 - 1	S1	S2* + 1
Collision slot vide	S0 - 1	S1	S2* + 1

Figure III.22 Tableau représentant le nombre de nœuds dans différents états à la fin d'un round

Nous allons maintenant pouvoir expliquer les différents moyens pour faire de la détection de collision du côté récepteur. La thèse [12] décrit trois possibles méthodes efficaces afin de détecter les collisions. Notre solution à ce moment utilisait l'une de ces méthodes pour détecter un relevé d'énergie supérieur à la normale à l'aide d'un mécanisme que l'on appelle RSSI. Les autres méthodes proposées par la thèse sont d'utiliser un mécanisme nommé CCA, ou d'utiliser le code de redondance cyclique pour déterminer s'il y a eu collision ou non. Cette dernière solution a été rejeté, car elle ne permet pas de traiter les cas où l'on ne reçoit pas de RX_START. Les solutions qui nous sont alors accessibles d'après cette thèse sont le CCA et le RSSI. Cependant, nos recherches nous ont amenés à tester une troisième solution, que l'on appelle l'ED. Ce travail a été fait sur le puits, et par conséquent c'est avec les valeurs définies dans sa fiche technique que nous allons l'expliquer.

Commençons par définir ce en quoi consistent ces solutions. Le RSSI (Received Signal Strengh Indicator) est une valeur qui indique la puissance reçue sur un canal, généralement en décibel. Elle représente une capture instantanée du niveau d'énergie, et nous pouvons y accéder simplement en lisant un registre qui lui est associé si le module radio est en train d'écouter le canal, c'est-à-dire s'il est en état RX. Pour notre puits, cette valeur est mise à jour automatiquement toutes les 2 µs, et appartient à l'intervalle [-94 dBm ; -10 dBm] avec un pas de 3 dB. La valeur -94 dBm est celle de base de notre puits lorsqu'il n'y a aucune activité sur le médium, et elle augmente si de l'énergie est détectée. Le temps nécessaire pour récupérer la valeur du RSSI est par conséquent simplement égale au temps qu'il faut au microcontrôleur pour demander la lecture d'un registre au module radio.

La deuxième solution à notre disposition est l'utilisation de l'ED (Energy Detection), un mécanisme qui consiste à faire la moyenne de huit valeurs de RSSI. En temps normal, il est utilisé dans les algorithmes de sélection de canal d'un appareil afin de choisir le canal le plus performant pour une communication avec un autre appareil. Comme pour le RSSI, il est nécessaire d'être dans l'état RX pour pouvoir être s'en servir, et les deux méthodes possèdent le même intervalle, à la différence que celle-ci a un pas de 1 dB. Pour faire une mesure avec l'ED, nous disposons de deux manières : automatique ou manuel. Dans ce second cas, il nous faut d'abord écrire une valeur dans un registre, puis attendre que les mesures se fassent, avec une durée de 128 µs, et enfin récupérer la valeur - cependant, nous en reparlerons plus tard, récupérer cette valeur peut poser problème, car il est nécessaire de notifier le microcontrôleur de la fin de la mesure.

Dans le premier cas, notre puits va automatiquement faire une mesure de l'ED lorsque le SHR d'une trame reçue est détectée. Le résultat de cette mesure sera disponible à la fin de la réception de notre message, lors de l'interruption RX_END.

La troisième solution envisagée a été d'utiliser le CCA (Clear Channel Assessment) afin de détecter de l'énergie ou un signal qui respecte la norme IEEE 802.15.4. Il s'agit d'un mécanisme utilisé en temps normal par le CSMA/CA, une méthode pour savoir si le médium est libre lors de l'envoi d'une trame. En écoutant le médium un certain temps pour s'assurer qu'aucune entité n'essaye de communiquer, le CSMA/CA permettra d'éviter certaines collisions. Si les solutions que nous avons amenées sont des méthodes de détection de collision du côté récepteur, le CSMA/CA est une méthode de détection de collision du côté envoyeur. Le CSMA/CA n'est pas compatible avec notre méthode de découverte de voisinage quasi-déterministe, mais nous pouvons malgré tout utiliser le CCA afin de détecter des collisions. Le CCA utilise l'ED pour détecter une activité sur le médium, et son utilisation nécessite donc la lecture de deux registres : l'un des registres sera le résultat booléen du CCA – soit le canal est libre, soit le canal est occupé -, et l'autre sera le registre de l'ED, dans lequel on aura le résultat du calcul de l'énergie. Le CCA, tel que défini dans 6.9 de [17] possède 4 modes différents :

- Détection d'énergie au-dessus d'un seuil : le calcul de l'énergie mesurée sur le médium est au-dessus d'un seuil arbitrairement défini ;
- Détection de porteuse : le module radio a détecté un signal avec les mêmes caractéristiques que ceux de la couche physique de l'appareil ;
- Détection d'énergie au-dessus d'un seuil ET détection de porteuse : combinaison logique AND des deux autres modes désignés ci-dessus ;
- Détection d'énergie au-dessus d'un seuil OU détection de porteuse : Combinaison logique OR des deux autres modes désignés ci-dessus.

Pour faire une mesure avec le CCA, le processus est similaire à celui d'une mesure d'ED, mais avec quelques étapes supplémentaires.

Dans la thèse [12], il est dit que la meilleure méthode pour détecter une collision consiste à utiliser le CCA avec le matériel dont ils disposent. Pour rappel, l'utilisation de l'une de ces trois méthodes se fera en boucle jusqu'à certains événements. Notre solution incorporait déjà l'utilisation du RSSI, mais au vu de la thèse, cela ne semblait pas pertinent : en effet, en théorie, l'utilisation répétée du RSSI entraîne une surcharge du processeur, ce qui pourrait ralentir le nombre de mesures possibles. Nous avons donc pesé le pour et le contre de chacune des solutions à notre disposition.

Commençons par le CCA. Comme nous l'avons dit plus tôt, le processus d'une mesure avec le CCA est similaire à celui d'une mesure d'ED, mais avec certaines étapes supplémentaires afin de ne pas gêner le comportement normal du puits. C'est donc la solution la plus lente en termes de nombre de mesures, mais en théorie elle devrait être la plus efficace. Lors de nos tests, nous avons testé les modes de détection d'énergie au-dessus d'un seuil et de détection de porteuses

séparément. Très vite, nous avons remarqué que le mode détection de porteuses n'est que peu compatible avec notre solution :

Afin de pouvoir recevoir correctement une porteuse, notre puits va écouter le médium à la recherche d'un signal qui ressemble au SHR d'une de trames envoyées par les nœuds. Dès que le SHR a été correctement détecté, le CCA va pouvoir être indiqué au puits à l'aide d'une nouvelle interruption dont nous vous parlerons ci-dessous, qui indique la fin d'une mesure d'ED ou de CCA. Cependant, la réception du SHR ne levait pas l'interruption RX_START, car une fois que le SHR était identifié comme un signal valide qui correspond à nos configurations, il était jeté car l'opération était considérée finie. Par conséquent, nous ne pouvions pas aisément recevoir nos trames sans devoir changer comment fonctionnement physiquement la carte.

Cependant le point qui nous a fait mettre de côté la détection de porteuse peut être vu dans la figure III.23 :

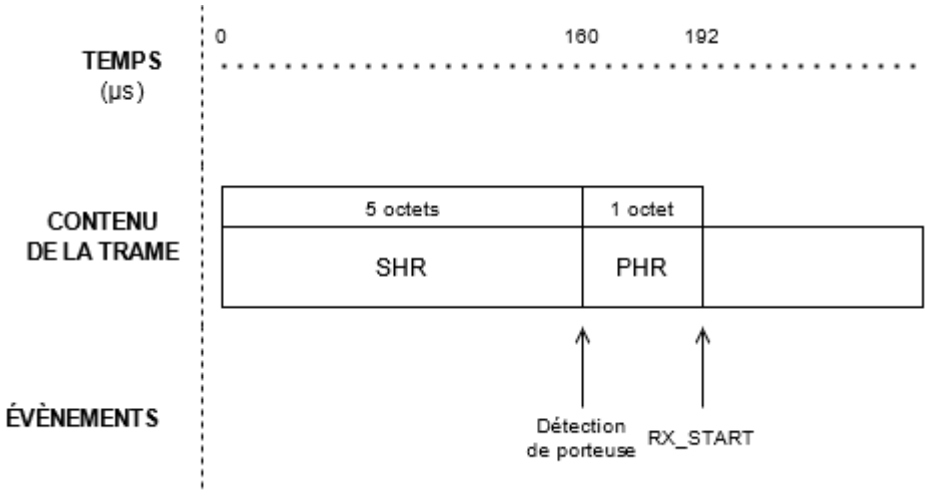

Figure III.23 Timing de l'interruption RX START par rapport à la détection de porteuse

En effet, entre l'instant où notre détection de porteuse peut commencer à notifier le microcontrôleur qu'un signal valide respectant les normes IEEE 802.15.4 a été détecté et l'instant où l'interruption RX_START, qui est donc l'interruption levée lorsqu'un SHR et un PHR respectant les normes IEEE 802.15.4 ont été détectés, le delta est de 32µs. Ceci implique que naturellement, avec l'utilisation de RX_START, nous avons le même fonctionnement que la détection de porteuse avec 32µs de délai. Cette durée est suffisamment négligeable pour que l'on puisse définitivement rejeter la solution d'utiliser le CCA en mode détection de porteuse.

Par conséquent, seul le mode détection d'énergie du CCA nous intéresse. Nous avons alors remis en question l'utilisation même du CCA, car il imposait notamment des contraintes sur l'implémentation de notre solution, ne permettant pas au puits d'envoyer des acquittements automatiquement. Voyant cela, nous avons décidé de nous pencher sur les autres solutions, notamment l'ED.

L'ED est une solution qui permet normalement d'avoir des résultats plus précis et plus stables que des simples lectures du registre du RSSI. Cependant, l'ED, comme le CCA, n'a pas été conçu pour être utilisé pour faire de la détection de collisions. Nos tests

nous ont donc demandé de modifier grandement notre implémentation afin de pouvoir obtenir ces résultats. Le problème principal de cette implémentation est que nous ne pouvons pas aisément faire savoir à notre programme de la fin de nos mesures d'ED. En effet, en utilisation normale de l'ED, nous allons simplement attendre qu'un registre, qui indique si la mesure d'ED est finie, passe à un. Comme nous n'attendons pas de recevoir de messages, ça ne pose pas de problème pour son utilisation. Cependant, dans notre cas, nous ne pouvions pas juste lire en boucle ce registre, car cela empêchait la bonne réception de certains messages.

Nous avons alors décidé d'utiliser une interruption supplémentaire que nous avons évoqué plus haut, à savoir CCA_ED_DONE. Cette interruption de notre module radio prévient le puits quand une mesure de CCA ou d'ED est finie. Cependant elle n'est pas enclenchée par défaut, et pour l'utiliser, il nous a fallu modifier quelles interruptions étaient reconnues par le puits.

Cependant, comme nous faisions un grand nombre de mesure d'ED alors que nous voulions aussi recevoir des trames, le surplus d'interruptions a posé un problème pour les bus qui relient le module radio et le microcontrôleur - pour rappel, chaque mesure d'ED génère une interruption quand elle est finie.

L'ED semble finalement ne pas être une solution veine, mais elle nous aurait demandé de passer plus de temps sur cette partie de notre solution, qui semblait finalement être plus fonctionnelle que les autres options qui étaient à notre disposition. Ceci nous a été confirmé quand une étude poussée du code et des changements qui y ont été fait ont montré que des anciens collaborateurs avaient cherché à utiliser le CCA par le passé, pour se rabattre sur le RSSI.

La manière dont notre solution utilise le RSSI est montrée dans la figure III.24. La valeur de base du RSSI est définie d'après notre module radio, et le seuil de collision a été choisi dans l'existant car il convenait au bruit de notre milieu, qui n'était pas supérieur à -85dBm.

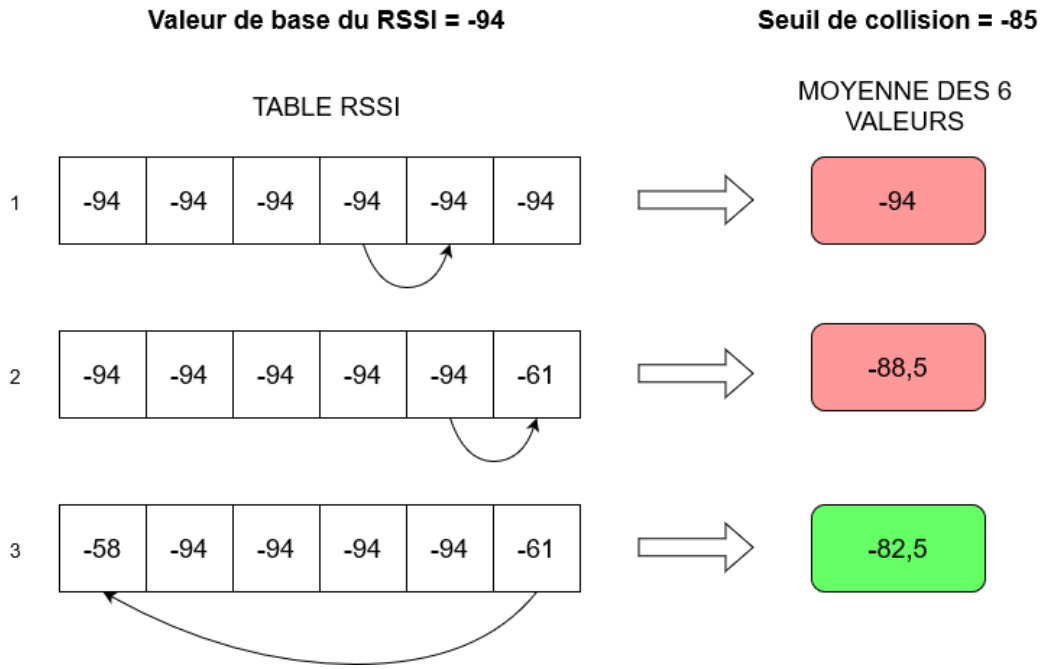

Figure III.24 Exemple de trois mesures de RSSI succinctes lors de la boucle de détection de collision

Lorsque la moyenne des 6 dernières valeurs de RSSI dépassent un certain seuil, on peut conclure qu'une activité a eu lieu sur le médium dans ce slot. Dans notre code, cela se traduit par la prise d'un horodatage, que nous renseignons dans un tableau. A la fin du round, nous pourrons nous servir de cet horodatage pour savoir dans quel slot une collision a été détectée.

Bien que d'après la thèse citée plus haut, de multiples requêtes du RSSI surchargent le processeur, cela ne posait pas de problème à notre équipement - cela doit sans doute être dû à l'équipement dont ils disposaient pour leurs tests. Pour faire le tour de nos secteurs à vide, nous comptons plus de 10 000 valeurs de RSSI réalisées. De plus son utilisation est très simple, comme il suffit de récupérer la valeur d'un registre.

Cependant, pour pouvoir prouver l'efficacité de notre solution, nous avons décidé de faire des tests que nous vous présenterons par la suite.

D) Mise à jour du système d'exploitation.

Après avoir fini nos tests pour la détection d'énergie sur le médium lors de la phase de découverte du voisinage, nous avons commencé à travailler sur la phase de collecte. Contrairement à l'existant sur la phase de de découverte, très peu de travail avait été accompli sur la phase de collecte. Les types de messages étaient ainsi que la base du fonctionnement du puits avait été implémentés, mais le reste était à faire, et c'est lors de l'implémentation de la phase de collecte sur les nœuds que nous avons rencontré le problème représenté dans le figure III.25.

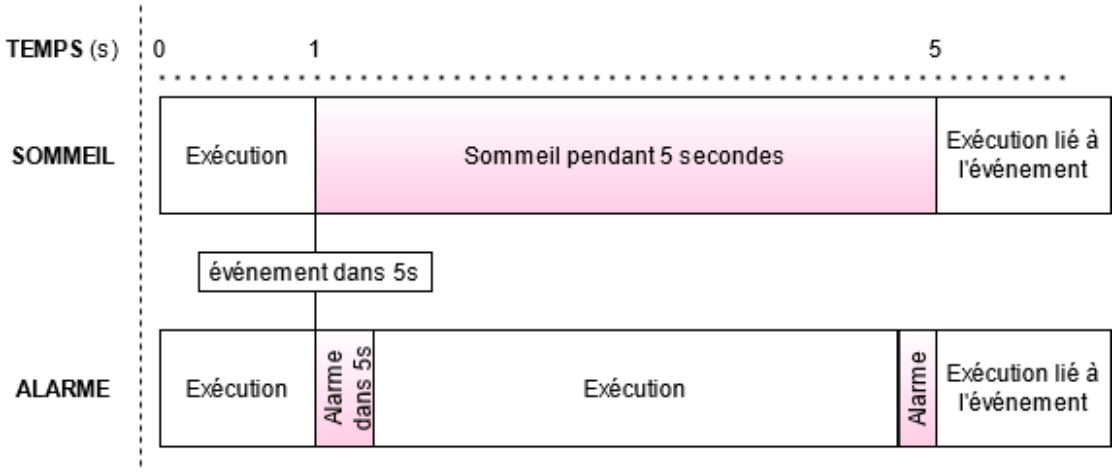

Figure III.25 Mise en sommeil des nœuds et mise en place d'une alarme

Comme vous pouvez le voir, lorsqu'on met en sommeil un nœud, ce dernier ne peut plus poursuivre son exécution pendant un certain temps, ce qui rend par exemple impossible la réception de messages. La solution à ce problème serait d'installer un système d'alarme, qui, une fois fixée, permettrait au nœud de poursuivre son exécution initiale en attendant l'enclenchement de l'alarme. Cependant, pour faire cela, il nous a fallu faire une mise à jour de notre système d'exploitation, RIOT OS [13].

RIOT OS est un système d'exploitation pour cartes embarquées de réseaux sans fil. Gratuit, libre de droit, avec un code source ouvert, et mis à jour très régulièrement par une communauté active notamment sur GITHUB [14], il permet de faire de la programmation de haut comme de bas niveaux sur des systèmes embarqués. Il consiste en un système d'exploitation à micro-noyau, c'est-à-dire que l'OS installé sur une carte ne contient que le minimum de code nécessaire à son exécution. Selon nos choix et le firmware qui tourne sur ladite carte, grâce à un système de modules qui permet de sélectionner les programmes que l'on veut spécifiquement utiliser, le système d'exploitation va se compiler avec le strict minimum pour son bon fonctionnement. Ecrit à plus de 90% en C, qui est donc le langage dans lequel nous programmons, RIOT OS nous offre un nombre conséquent de documentation, et l'utilisation de GITHUB permet notamment d'avoir un historique de toutes les versions de l'OS.

Etant donné la nature code source ouvert du système d'exploitation, de nombreuses options et optimisations ont été faites jusqu'à aujourd'hui. Parmi ces options, nous retrouvons la possibilité de mettre en place une alarme sur nos nœuds capteurs. De plus, le système d'exploitation est plus performant, et de nouveaux pilotes ont été rajoutés. C'est donc sur cet aspect que nous avons poursuivi nos travaux.

La mise à jour nous a demandé de récupérer toutes les modifications que nous avions réalisées à l'échelle du système d'exploitation depuis la version 2019. Après avoir récupéré tous nos fichiers et les avoir plus ou moins adaptés, nous avons rencontré quelques problèmes liés à la manière dont les pilotes sont initialisés.

Dans notre code, l'initialisation de nos pilotes radio commencent par l'utilisation d'un module que l'on appelle auto_init (Automatic Initialisation). Ce dernier va configurer nos pilotes, avant de créer une interface réseau entre notre module radio et le microcontrôleur. C'est après avoir créé cette interface que les différences entre notre ancienne version de RIOT OS et la nouvelle commence à se voir. En effet, dans la première, l'initialisation de l'interface réseau amenait celle de notre module radio puis celle de notre programme, alors que dans la seconde, seul notre programme était initialisé après l'interface réseau. Il a donc fallu rajouter l'initialisation de notre pilote radio lors de celle de notre programme.

Un autre problème que nous avons rencontré était lié à la manière dont les threads sont appelés à l'échelle du système d'exploitation. La manière dont nous stockions les identifiants des processus dans notre ancienne version (PID) ne fonctionnait plus dans la nouvelle version de RIOT OS, et lorsque l'on avait une interruption, notre programme s'arrêtait. Il nous a fallu changer spécifiquement certains appellent pour que le PID correct de nos threads soit utilisé.

Enfin, nous l'avons évoqué plus haut, la nouvelle version de RIOT OS a rajouté des pilotes à notre disposition. Parmi ces derniers, on retrouve un des pilotes radio de nos nœuds : l'at86rf215. Pour ce projet, les deux types de cartes que nous utilisons possèdent des modules radio ATMEL, respectivement l'at86rf233 [15] pour le puits et l'at86rf215 [16] pour les nœuds capteurs. Bien que dans l'idéal, chaque module radio de cette famille devrait avoir un pilote qui lui est propre, l'ancienne version de RIOT OS proposait un seul pilote global pour toute cette catégorie de module radio, appelé at86rf2xx. Cependant, comme nous l'avons évoqué, dans la nouvelle version, bien que notre puits utilise encore le pilote at86rf2xx, le pilote at86rf215 a été rajouté, et nous pouvons l'utiliser spécifiquement pour nos nœuds plutôt qu'utiliser celui global. Ce nouveau pilote permet une meilleure manipulation du matériel et rajoute de nombreuses possibilités de débogage, que nous avons notamment utilisé lors de la phase de développement.

E) Génération de données aléatoires.

Après avoir réussi à faire notre mise à jour, nous avons alors pu tenter, avec succès, d'utiliser les alarmes sur nos nœuds. Ceci nous a permis de les utiliser dans notre phase de collecte afin de pouvoir mettre en place les slots dans lesquels ils doivent répondre.

La première étape pour implémenter notre phase de collecte a été de rajouter un cycle de collecte pour les nœuds : à partir du moment où les nœuds sont informés du début de la phase de collecte – c'est-à-dire, au moment où les nœuds reçoivent le premier message de poll -, les nœuds vont créer un thread qui va récupérer des mesures. Dans notre code, ces mesures ne sont que des données générées aléatoirement, afin de servir d'éléments de substitution. Associé à chacune de ces mesures se trouve être un horodatage, qui nous permet de connaître le degré de fraîcheur de nos données. Ces informations – les données et leurs horodatages- sont stockées dans un tableau dont la taille est un paramètre système, c'est-à-dire un paramètre modifiable en fonction des besoins de l'utilisateur. Si l'horodatage des données dépassent

une certaine durée, les données sont retirées du tableau. Nous avons implémenté ce système de manière simple, avec un système similaire à la figure III.25, où notre tableau boucle sur lui-même, et remplace automatiquement les données trop vieilles par des nouvelles. Les données ont une taille d'un octet, et les horodatages une taille de deux octets, comme vous pouvez le voir dans la figure III.26.

Taille en octets :	1	2
Contenu :	Mesure	Horodatage

Figure III.26 Structure de nos données

F) Implémentation de la robustesse.

Après cette étape, nous avons travaillé sur la robustesse de nos communications, à l'aide de codes correcteurs d'erreurs. Le travail sur le choix de la robustesse a été fait lors du stage de 2021 [18], où nous avons étudié différents algorithmes et solutions pour répondre à nos contraintes de robustesses. Celle que nous avons retenu consiste à utiliser deux codes correcteurs différents afin de protéger nos données. Ces deux codes sont Golay et l'entrelacement.

- Golay :

Un code de Golay est un code correcteur d'erreur utilisé lors d'une transmission de donnée, et plus spécifiquement des données binaires, ou tout autre alphabet de taille deux. Un alphabet en cryptologie est le nombre de valeurs que peut prendre chaque élément d'un mot (ici, information que l'on veut envoyer). Dans notre cas, il s'agit de bits pouvant être égaux à 0 ou 1. Le mot en question, dans un code de Golay, doit avoir une taille égale à 12 bits, ou un multiple de 12 bits. Une fois que le mot a été chiffré, sa taille se voit doublée, et on appelle ce nouvel élément un bloc. Si l'information que l'on transmet est composée de plusieurs mot, alors le résultat que l'on obtient après chiffrement sera composé d'autant de bloc.

Comme en informatique, manipuler des messages de 12 bits est compliqué, le consensus veut que nous utilisions simplement des informations d'une taille double, à savoir des messages de 24 bits - soit 3 octets –. Ainsi, lorsque nous chiffrons avec l'algorithme de Golay, nous le faisons sur des groupes de 3 octets, et Golay va être appliqué sur deux messages de 24 bits. Comme vous pouvez le constater dans la figure III.26, nos propres données ont une taille égale à 3 octets, ce qui les rends chiffrable avec l'algorithme. Vous pouvez voir une explication schématique de la notion de mots et de blocs avec Golay dans la figure III.27.

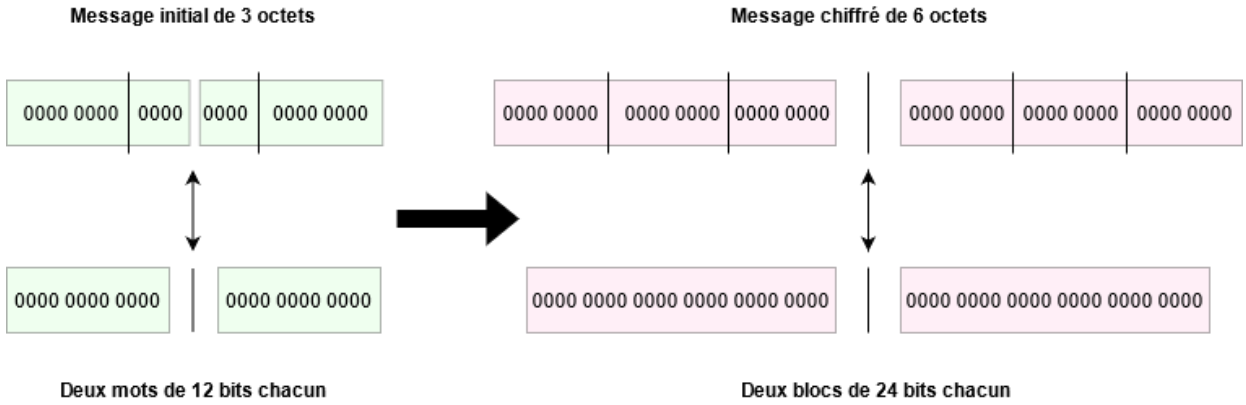

Figure III.27 Equivalence entre la taille des octets et des mots dans Golay

Nous pouvons ensuite envoyer nos messages chiffrés, et lorsque le puits le recevra, il pourra le déchiffrer simplement en réutilisant Golay. C'est lors du déchiffrement que Golay va pouvoir corriger, au moins détecter les erreurs. Si sur tout un bloc, 3 bits ou moins ont pris une autre valeur que celle initiale, l'algorithme va reconstruire les mots tels qu'ils étaient originalement. Si Golay ne parvient pas à déchiffrer le message, alors au moins nous aurons une indication que le message n'a pas pu être corrigé.

- Entrelacement

En prenant en compte notre environnement, il est plus que probable que lorsque des interférences brouillent nos trames, les erreurs ne se retrouvent pas aléatoirement réparties sur toute la trame. Elles seraient plutôt regroupées à plusieurs, modifiant la valeur de plusieurs bits l'un à la suite de l'autre. Si un groupe de 4 bits ou plus est brouillé par une interférence, Golay ne sera pas capable de déchiffrer le bloc et nous perdrons les données que nous souhaitions recevoir. Cependant, si nous pouvions faire en sorte de répartir les erreurs sur toutes la trame plutôt que concentrée sur un ou plusieurs bloc, Golay pourrait les déchiffrer et les corriger. C'est à cette fin que nous avons décidé d'utiliser, en complément de Golay, l'entrelacement.

Il s'agit d'un autre code correcteur d'erreur qui, comparé au code précédent, n'a aucune contrainte quant à son utilisation. Son résultat dépend uniquement de la taille du mot que l'on veut envoyer et d'un paramètre qui contrôle la complexité du chiffrement que l'on appelle la profondeur. L'entrelacement consiste à mélanger la position des éléments dans un mot afin de pouvoir répartir une erreur sur l'intégralité du mot plutôt que sur une seule partie. En temps normal, l'objectif de ce mélange est de pouvoir déterminer le sens d'une information malgré les éléments manquants, plutôt que de perdre le sens d'une information à cause d'un mot manquant.

Ainsi, en utilisant dans un premier temps un code de Golay sur nos données utiles, puis en y appliquant l'entrelacement, nous obtenons une trame renforcée. Il sera alors possible pour le puits de corriger les erreurs qui touchent plusieurs bits à la suite, ce qui n'était pas possible avec la simple utilisation de Golay. Nous l'avons prouvé dans [18], et vous pouvez trouver ces résultats dans la figure III.28.

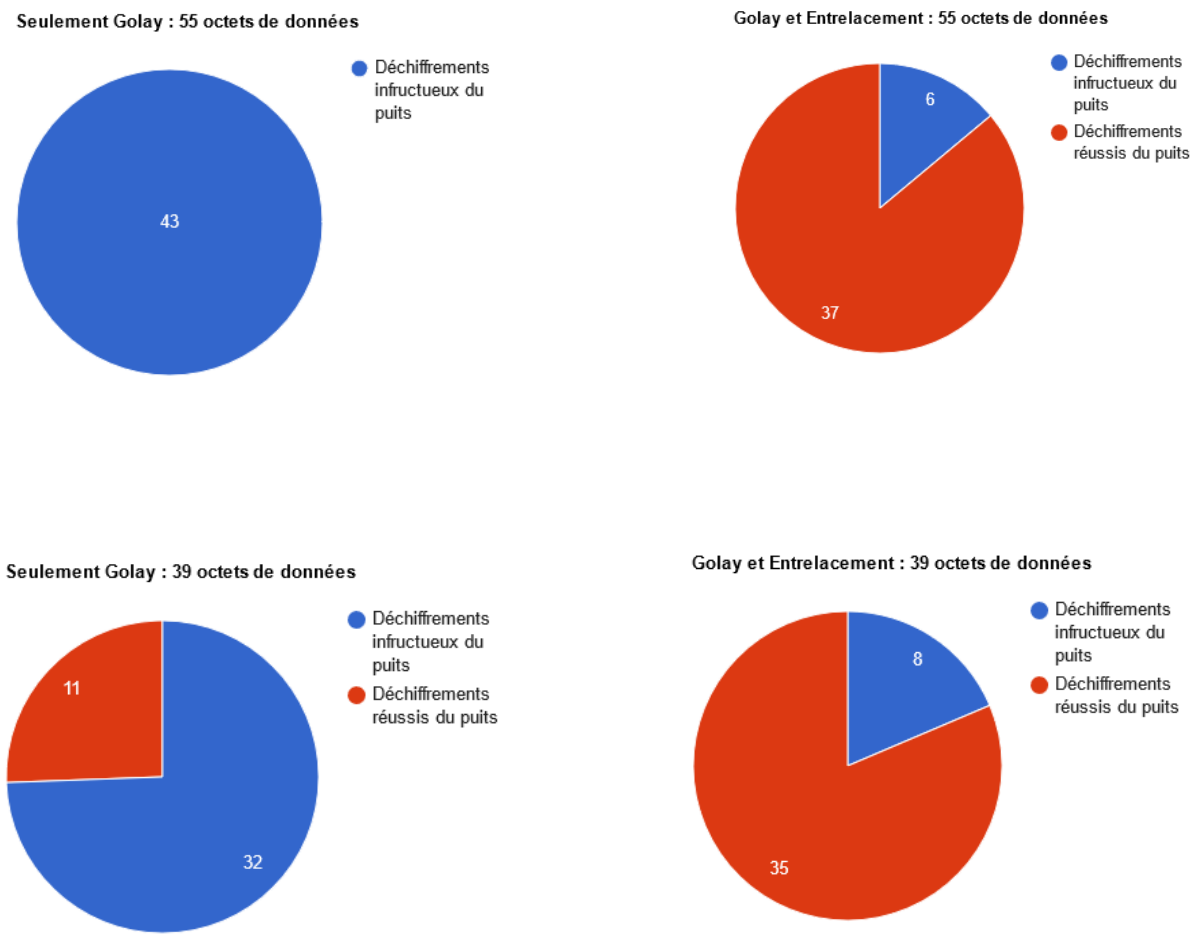

Figure III.28 Amélioration de la validité des trames avec l'utilisation seule de Golay et l'utilisation de Golay ainsi que de l'entrelacement

La dernière étape a consisté à travailler sur les acquittements que va envoyer le puits aux nœuds en réponse à leurs DRP. Comme montré dans la figure III.11, notre puits aura trois comportements possibles lors de la réception :

- Si le CRC de la trame reçue est valide, alors le puits envoie un acquittement qui sera reconnu par le nœud qui a envoyé la DRP, puis déchiffre la trame avant de la traiter ;

- Si le CRC est faux, alors nous allons lire la zone mémoire où est stockée la trame reçue et la déchiffrer avec l'entrelacement puis Golay ;
    - Si le résultat ainsi obtenu indique que la trame a été corrigée, alors le puits envoie un acquittement au nœud, puis traite la trame ;
    - Si le résultat ainsi obtenu indique que la trame n'a pas pu être corrigée, alors le puits vide la zone mémoire et n'envoie pas d'acquittement.

Les acquittements ne sont cependant pas activés par défaut, et il nous a fallu nous référer à la norme 802.15.4 – 2006 [17] pour les utiliser, ainsi qu'a la fiche technique du puits [8]. En lisant cette dernière, nous y avons trouvé un problème lié à l'utilisation des acquittements.

En temps normal, le puits est sensé capter tous les messages qui transitent sur notre réseau. De plus, nous souhaitons que le puits reçoive les trames même si ces dernières sont fausses. Par conséquent, nous avons cherché à mettre notre puits en mode promiscuous, une fonctionnalité propre aux réseaux sans fil qui consiste à accepter les messages reçus sans prendre en compte leur provenance ou leur validité.

Le problème vient du fait qu'en temps normal, si un appareil est en mode promiscuous, il n'est pas censé répondre aux messages avec des acquittements. Si c'était le cas, le réseau se retrouverait surchargé à cause des acquittements du puits, qu'il enverrait à chaque message reçu, peu importe l'appareil de provenance, tant que son préambule est compréhensible par notre puits. Notre solution étant déterministe, le puits n'est sensé envoyer un acquittement que si le CRC est valide ou que le CRC est faux mais que le déchiffrage est valide. Nous ne devrions donc pas avoir de problème si nous utilisions le mode promiscuous et les acquittements ensemble, mais le pilote radio du puits ne nous proposait pas cette option, car elle n'est pas respectueuse de la norme IEEE 802.15.4 – 2006.

Ce problème a été résolu en forçant le mode promiscuous et les acquittements à être activés en même temps, ces deux options étant liées à des registres. Après plusieurs tests, nous avons pu confirmer que les acquittements fonctionnaient correctement alors que notre puits était en mode promiscuous.

Cependant, utiliser les acquittements a apporté une contrainte importante pour notre matériel. Nous avons indiqué plus haut que les acquittements étaient envoyés si le CRC d'une trame est valide ou si le CRC est incorrect mais que le déchiffrement arrive à corriger la trame. Dans le premier cas, l'acquittement est fait automatiquement par notre module radio, mais nous n'avons pas réussi à utiliser ces mêmes acquittements pour le second cas. Afin de pouvoir répondre dans les temps nous avions initialement prévu d'utiliser ces mêmes acquittements automatiques, mais en décalant leurs envoient pour qu'ils soient transmis si la correction de la trame est un succès. Mais parvenir à ce résultat nous demandait de manipuler l'état des broches et cela avait pour conséquence de bloquer notre algorithme. Nous avons alors décidé d'utiliser nos propres acquittements, des trames avec une structure très simple, que le puits enverra aux nœuds

si le déchiffrement est valide. Nous pouvons résumer cette explication avec la figure III.29.

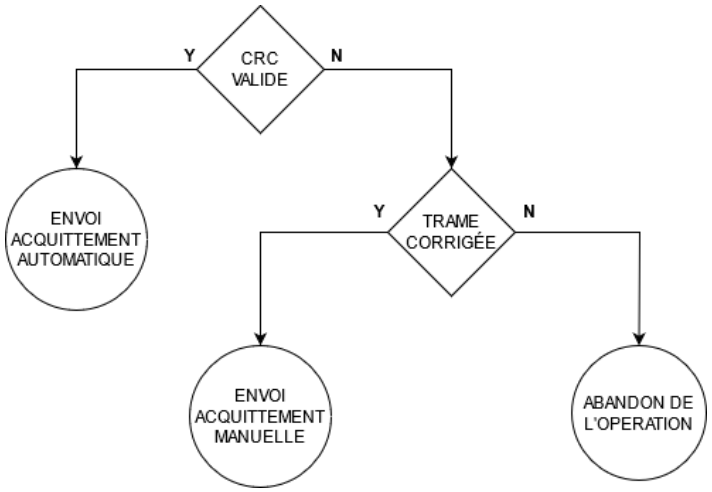

Figure III.29 Représentation des différents cas possibles lors de la réception d'une trame dans la phase de collecte

Nous n'avons pas encore réussi à totalement finir cette implémentation, et ceci constituait nos dernières semaines de stages.

PARTIE 3 : Résultats

A) Résultats de la détection de collisions.

Avant de passer à la conclusion de ce rapport, nous allons vous présenter les différents résultats que nous avons pu obtenir lors de nos travaux. Tout d'abord, parlons de la phase de découverte du voisinage et de la détection de collision.

Après avoir finalisé la détection d'énergie sur le médium, il nous a fallu avoir des preuves de son fonctionnement et de ce qu'elle nous apportait. Pour rappel, sans notre détection d'énergie, les slots lors de la phase de découverte étaient dits en collision si deux messages valides étaient reçus dans un seul slot, ou si une interruption RX_START était levée mais pas l'interruption RX_END correspondante. La solution réalisée dans l'existant et modifiée lors de ce stage devrait théoriquement pouvoir détecter, en plus de cela, les cas où le RX_START n'est pas reçu, mais où de l'énergie significative sur le médium a été mesurée.

Pour pouvoir faire nos tests, lors de la boucle de détection de collision, en plus de lire le registre RSSI, nous avons rajouté la lecture d'un temporisateur afin de pouvoir dater toutes nos mesures de RSSI. Ceci réduit considérablement le nombre de mesure de RSSI que nous réalisons, mais nous pouvons désormais lier les mesures de RSSI à un horodatage. Nous les avons alors saisies dans un tableur EXCEL, afin de pouvoir avoir une représentation graphique des mesures d'énergies faites lors de la phase de découverte. Afin de pouvoir aisément représenter nos valeurs, nous avons aussi récupéré les horodatages qui correspondent à la fin des rounds, ainsi que ceux liés aux interruptions RX_START.

Enfin, nous avons modifié le comportement des nœuds. En temps normal, notre protocole amène nos nœuds à sélectionner un slot aléatoire parmi ceux disponibles. Pour tester l'efficacité de notre détection de collision, lors des tests que nous avons accomplis, nous avons forcé tous les nœuds à répondre dans un même slot. Ce faisant, tous les messages envoyés au même moment créait une collision importante qui rendait la réception des messages compliqué au niveau du puits. Bien que théoriquement, le puits ne devrait recevoir aucun message en provenance de nœuds si tous créent des collisions, certaines trames arrivaient à être correctement reçues, dû à l'effet capture. En termes de transmission radio, on parle d'effet capture quand, lorsqu'une entité reçoit plusieurs messages au même instant, elle arrive à recevoir un de ces messages car son signal est plus que celui des autres messages. Par exemple, si plusieurs nœuds à une distance différentes du puits envoient leur message en même temps, les nœuds les plus proches du puits auront une plus grande chance de voir leur message reçu par rapport à ceux les plus loin.

Maintenant que nous avons pu vous présenter nos méthodes de test, nous allons vous montrer nos résultats. Lors de nos tests, nous disposions de cinq nœuds autour du puits, et ils envoient tous leur message dès l'instant où commence le round. Aussi, le puits commence chaque secteur avec 6 slots différents.

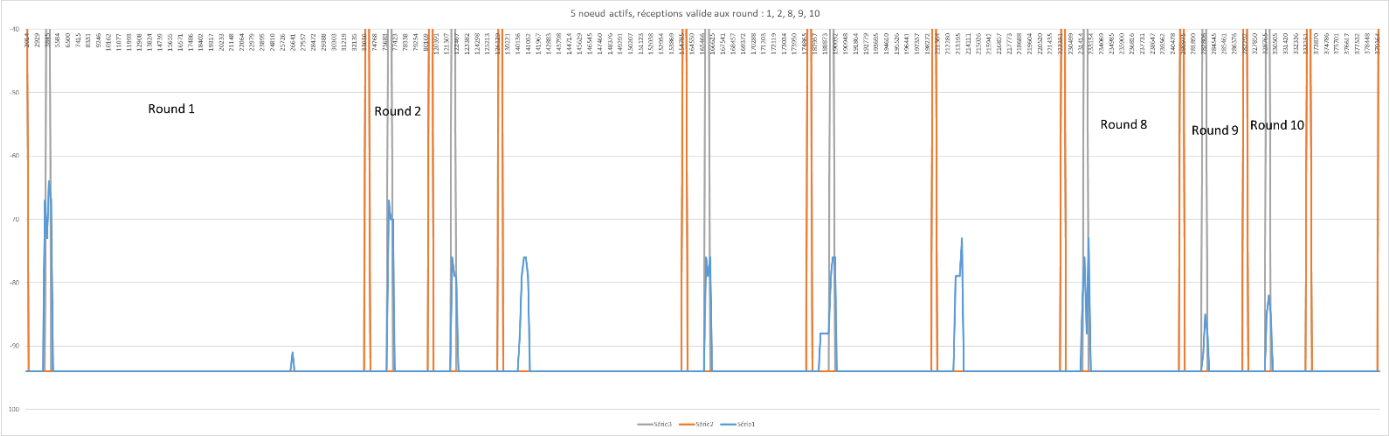

Figure III.30 Graphique représentant l'énergie mesurée sur le médium lors des rounds de la phase de découverte

Sur la figure III.30, on peut voir de nombreuses informations.

- Les barres en oranges désignent le début et la fin d'un round. L'espace blanc qui les sépare indique une durée indéterminée entre chaque round.
- Les barres en grises indiquent une durée indéterminée durant laquelle l'interruption RX_START a été levée.
- La série en bleu représentent le niveau d'énergie sur le médium. Par défaut, il est à -94 dBm.
- Lorsque que dans un round, l'indication 'Round x' est visible, un message a été correctement reçu par le puits.

Nous allons maintenant analyser cette figure afin de pouvoir prouver le fonctionnement de notre détection de collision.

Le premier round est plus grand que les autres, car le nombre de slot est de six pour le puits, soit 30 ms. Nous pouvons y voir un pic d'énergie à -43 dBm, un RX_START est levé et une bonne réception d'une trame. Comme nous l'avons précédemment dit, le message d'un de nos nœuds a été correctement reçu grâce à l'effet de capture.

Lors du deuxième round, nous ne disposons que d'un seul slot, car comme le premier slot du round précédent était valide et les cinq autres vides, l'estimation du nombre de nœuds restant considère que tous les nœuds ont potentiellement été découvert. Notre estimation va donc faire un dernier round pour s'assurer qu'aucun nœud ne veut répondre dans ce slot. Les quatre nœuds restants répondent dans ce même slot, et un des messages est reçu par le puits. Nous constatons que le pic d'énergie est à -49 dBm.

Le puits ne reçoit pas de message valide lors du troisième round, cependant l'interruption RX_START est levée. Nous avons donc ici un des cas de collision qui est détecté par notre puits, sans prendre en compte le niveau d'énergie. Ici, le pic d'énergie est à -58 dBm, et seul trois nœuds répondent en même temps.

Le quatrième round possède trois slots, car le slot du round précédent est en collision. Nous pouvons remarquer que aucune interruption RX_START n'a été levée, mais nous pouvons voir un pic d'énergie à -58 dBm. Nous pouvons donc supposer que le premier slot

de ce round est en collision si notre algorithme de détection d'énergie sur le médium est bien fonctionnel.

Nous avons la confirmation de cela en regardant le cinquième round. Comme nous pouvons le voir, ce round possède deux slots. Si tous les slots du quatrième round étaient vides, alors le cinquième round ne devrait posséder qu'un seul slot. Par conséquent, nous constatons que la détection d'énergie sur le médium permet effectivement de rajouter de la performance lors de notre phase de découverte de voisinage.

Nous pouvons aussi constater que tout le long du graphique, à chaque nœud découvert, comme moins de nœuds répondent, la valeur de RSSI de plus en plus faible.

B) Résultats sur la phase de collecte.

Sur la phase de collecte de données, nous n'avons pas de résultats tel que celui présenté précédemment. Cependant, nous pouvons malgré tout vous montrer ce qui a été accompli à l'aide d'organigramme. Dans les deux organigrammes qui vont suivre, nous avons représentés les multiples étapes que nous avons implémentés après l'envoi d'une DRQ par le puits. Les nœuds attendent donc leurs tours avant de s'exécuter, et le puits continue d'écouter pour recevoir des DRP jusqu'à la fin des slots qu'il a accordés.

Dans la figure III.31, on peut voir l'exécution d'un nœud lorsque son adresse est indiquée dans la DRQ. L'indication 'Dans l'extra slot' est égale à non lors du premier passage, et à oui lors du second. Elle correspond simplement à si le nœud répond dans l'extra slot ou non.

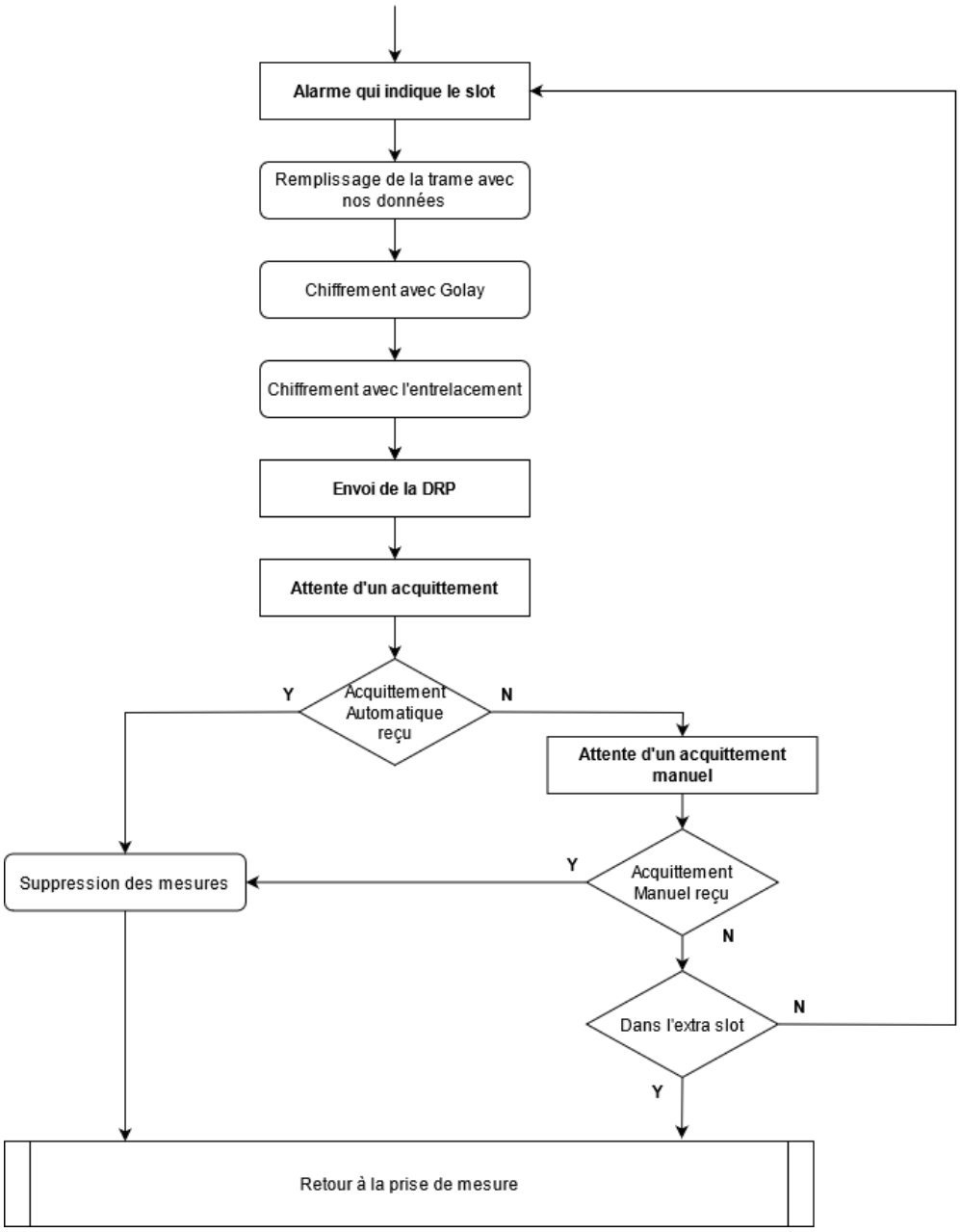

Figure III.31 Organigramme des nœuds à partir de la réception d'une DRQ

Dans la figure III.32, on peut voir l'organigramme du puits, à partir du moment où il a envoyé la DRQ. Cette implémentation utilise celle qu'on trouve par défaut dans le pilote du module radio du puits, notamment sur la réception de la DRP, l'interruption RX_END et l'événement RX_COMPLETE associé. Pour le reste, nous avons rajouté et modifié son fonctionnement original afin de pouvoir réduire au minimum les temps de traitement lors de la réception.

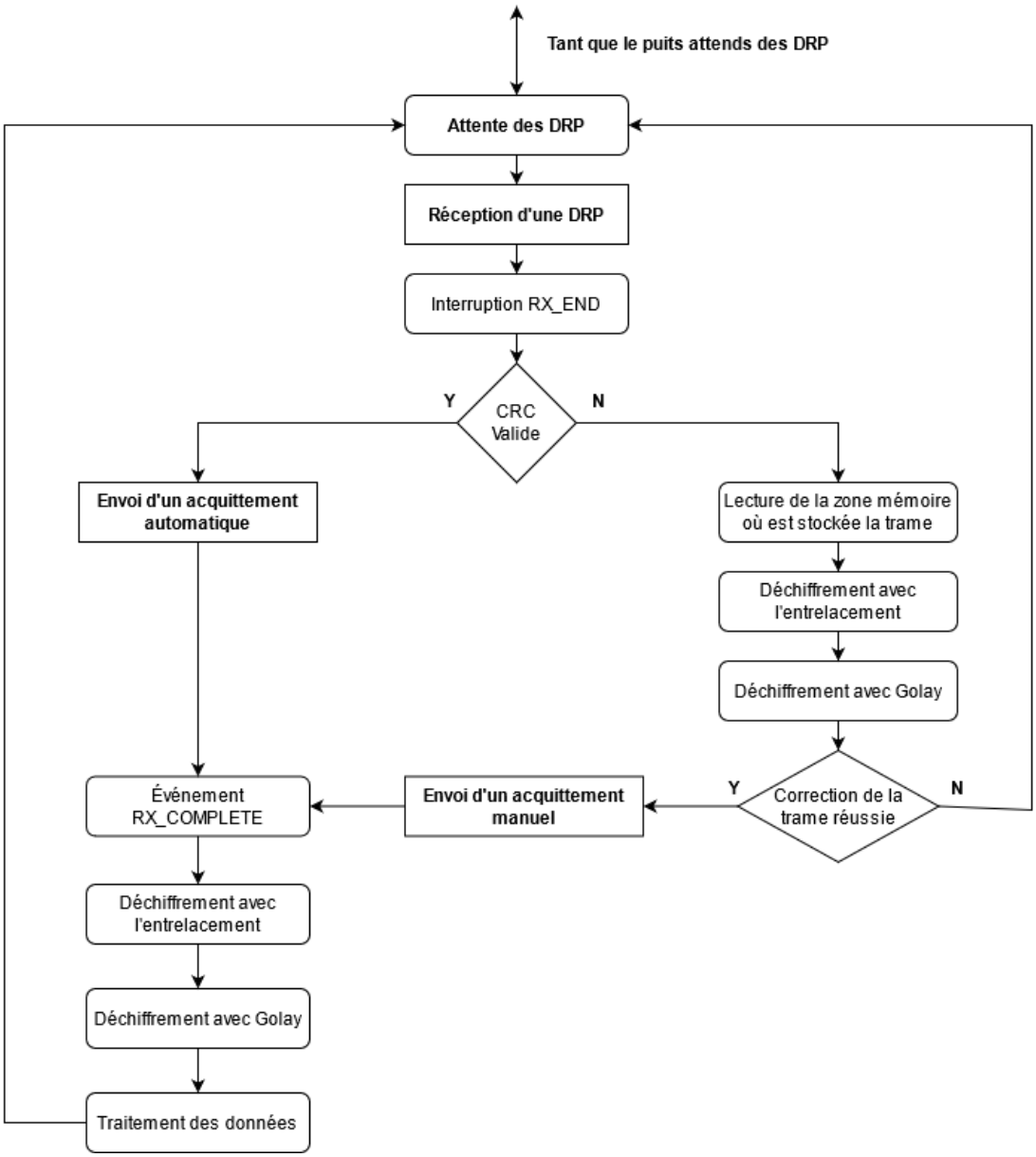

Figure III.32 Organigramme du puits à partir de l'envoi de la DRQ

IV – BILAN

PARTIE 1 : Finalité du projet

Nous allons finir ce rapport en faisant un point sur les objectifs réalisés.

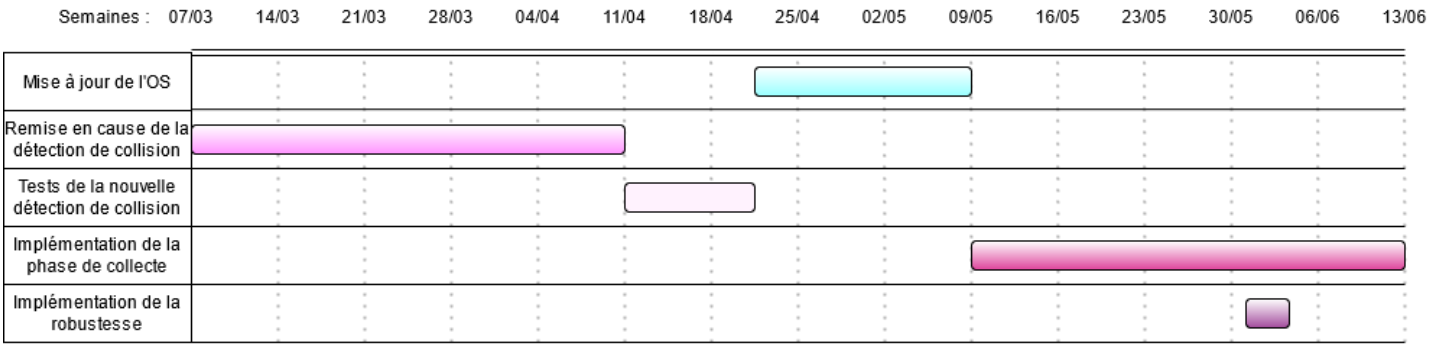

Figure IV.1 Schéma des objectifs finaux

Comme nous l'avions suspecté, la remise en cause de la détection de collision, qui a demandé des longues semaines de recherches sur le CCA, le RSSI et la détection d'énergie, a pris plus d'un mois. Ceci est dû à la complexité qu'a amené la manipulation des pilotes et du matériel, qui ont demandés de manier des registres et des éléments très bas niveaux pour pouvoir réaliser notre solution.

Les tests de détection de collision ont une durée à peu près similaire à ce que nous avions prévu, mais nous l'avons un peu étendu car nous avons réalisé de nombreux tests pour avoir des résultats satisfaisants. La mise à de RIOT OS nous a demandé un temps conséquent, notamment car nous avons rencontrés un problème avec notre matériel qui a mis en pause le travail pendant une semaine. Nous avions étendu cet objectif sur toute la première partie du stage car nous pensions qu'il serait simple de faire cette mise à jour lors des temps mort, mais il s'est avéré que la tâche était bien plus ardue.

Enfin, l'implémentation de la phase de collecte a pris un temps considérable, et à l'heure où nous rendons ce rapport, nous n'avons pas pu finir complétement cet objectif. Malgré cela, nous avons implémenté facilement la robustesse dans notre solution, car l'ayant déjà utilisé par le passé avec un ancien matériel, quelques jours on suffit pour la finir.

Nous n'avons pas pu finir tous les objectifs que nous nous sommes fixés, mais cela était déjà pris en compte au début du stage, et nous n'étions pas sensé finir ce projet lors de ce dernier. En effet, la complexité du sujet rend l'avancement sur les travaux très irrégulier. Il nous est arrivé de passer des semaines entières de recherches intensives et de tests afin de faire fonctionner nos implémentations. De plus, nous travaillons à un niveau très rigoureux et particulier, et toutes les erreurs ne sont pas parfaitement indiquées. Il nous a fallu à plusieurs reprises lire quelques morceaux de code en assembleur afin de comprendre ce qu'il se passait lors de l'exécution.

Si nous voulions finir ce projet, il faudrait pouvoir continuer l'implémentation de la phase de collecte, puis rajouter celle des

actionneurs. La redondance du puits est aussi un objectif que nous avons évoqué dans ce rapport mais sur lequel nous ne nous sommes pas étendus : il faudrait l'implémenter lors des travaux futurs. Enfin, un travail spécifique de documentation de ce qui a été réalisé est nécessaire pour pouvoir satisfaire les demandes du CNES, ainsi que la réalisation de nombreux tests pour pouvoir confirmer la performance de nos travaux.

En termes d'apprentissage, ce stage a été une véritable mine d'or. La manipulation des registres, des pilotes, des éléments très bas niveaux et du matériel physique m'a permis d'apprendre énormément sur la programmation bas niveau et embarquée. Les contraintes temporelles très strictes ont amenées une envie constante d'améliorer le code, et malgré la difficulté et la présence de morceau du code en assembleur, maîtriser toutes ces notions à été une véritable passion pendant les derniers mois.

Le niveau de complexité de ce stage nous a aussi grandement invité à réfléchir de manière différente et en dehors des sentiers battus.

Au niveau des résultats à obtenir dans le futur, nous souhaitons pouvoir respecter nos diverses contraintes. Nous souhaitons pouvoir présenter au CNES une solution aussi efficace et pointue que possible. Les résultats que nous devrons avoir dans le futur serait un test de la phase de collecte dans lequel nous mettrions en évidence nos contraintes temporelles.

PARTIE 2 : Conclusion

L’objectif de ce stage était de poursuivre l’implémentation des solutions théoriques développées pour ce projet. Ce projet consiste à remplacer les installations filaires au sein des équipements des lanceurs de fusée par des installations sans fil. Notre solution constitue un protocole qui concerne les réseaux de capteurs sans fil. Ordonnancé par un coordinateur que l’on appelle puits, au centre de notre réseau, ce dernier va contrôler l’intégralité des transmissions qui auront lieu.

La première partie de ce rapport de stage porte sur la présentation fonctionnelle de notre solution. Nous y définissons de nombreux concepts liés à notre solution, ainsi que le matériel que nous utilisons. Nous pouvons notamment y retrouver la norme sur laquelle se base notre projet, les cartes embarquées que nous utilisons pour tester l’efficacité de notre solution, ou encore les différentes phases qui composent notre algorithme.

La seconde partie de ce rapport se focalise sur la présentation de l’implémentation de notre solution. Dans un premier temps, nous parlons de la phase de découverte du voisinage, durant laquelle le puits va découvrir tous les éléments qui appartiennent à notre réseau. Cette phase fait partie de nos existants et est fonctionnelle, mais nous avons étudié une partie spécifique de cet existant afin de pouvoir faire des tests sur son efficacité. Le travail qui nous a été demandé consiste à revoir comment notre puits détecte que des messages ont transités sur notre réseau alors que la réception de ces messages n’a pas été correcte. Ce mécanisme se nomme la détection de collision. Pour accomplir cette tâche, nous avons étudié diverses méthodes de mesure de l’énergie sur un médium, dans l’objectif de détecter une activité d’un des éléments de notre réseau que nous n’aurions pas remarqué autrement.

Après avoir étudié le travail fourni sur la phase de découverte du voisinage, ce rapport se focalise sur la phase de collecte. C’est une phrase dans laquelle les éléments du réseau vont prendre des mesures à intervalle régulier et les stocker en attendant de les envoyer aux puits quand il fera une requête. Il a été nécessaire de programmer entièrement cette partie, qui repose sur de nombreuses mécaniques. Parmi celles-ci, on retrouve notamment l’utilisation de codes correcteurs d’erreurs pour permettre de corriger les messages erronés qui sont transmis sur notre réseau, ainsi que la manipulation des acquittements lors de la réception d’un envoi.

PARTIE 3 : Bilan humain

Sur le plan de l'insertion professionnelle, ce stage m'a été d'une très grande utilité. En effet, d'un commun accord avec mes encadrants, ce stage devrait se poursuivre en emploi à plein temps pour une durée déterminée en septembre. Ce stage m'a aussi appris à utiliser mes compétences pointues dans le domaine des réseaux de capteurs sans fil, et pourra certainement m'être utile pour être employé dans des entreprises spécifiques qui travaillent dans ce milieu.

Sur le plan de la communication, ce stage m'a appris à communiquer avec des chercheurs dont les connaissances sont pointues dans le domaine des réseaux, ainsi qu'à commencer à comprendre comment aborder des notions très pointues avec des interlocuteurs qui ne sont pas spécialisés dans ces notions.

Comme c'est la troisième itération de mes travaux sur ce projet, je possède dorénavant une connaissance très forte sur notre matériel et sur le sujet, ce qui en fait une expérience très intéressante. En effet, l'occasion de travailler sur un sujet si exigeant est rare, et j'apprécie grandement les connaissances techniques que j'ai pu y acquérir.

Durant ce stage, j'ai aussi appris à faire de la documentation de mes travaux en temps réel. Chaque jour, j'ai renseigné dans des fichiers mes avancements et mes questions, ce qui m'a permis de pouvoir suivre avec attention ce projet, malgré le fait que j'étais seul à travailler sur l'implémentation.

PARTIE 4 : Résumé en anglais

This internship is focused on an implementation of a theoretical protocol developed by former team members. It is a wireless sensor network that will be deployed inside a space rocket. To this end, this project possesses strict constraints that we must respect, such as temporal and robustness constraints.

This document is split in two major parts. In the first one, we expand on some theoretical aspect that needs to be understood if we want to comprehend what has been done in the second part, where we'll talk about what we've done and how we implemented our solution.

In term of theoretical aspects, it is important to understand that our project follows a particular topology, named star topology. This means that, in the geographical center of our network, we'll put a device called sink that will schedule the entirety of the communications that will take place. Our protocol describes multiple phases that will take place one after the other, controlled by the sink: one makes the sink discover every node that are parts of our network, named the discovery phase, the other request the said nodes to transmit data to the sink, called the collection phase. To ensure that no data is lost when we transmit them in the collection phase, we will secure them with forward error correction code.

Our implementation is composed of two parts, each corresponding to one of our phases. At first, we'll discuss the implementation of our discovery phase, more particularly the aspects of collision, which happens when two or more nodes sends their messages at the same moment. We considered different solutions to detect the energetic trace of our messages, namely using the RSSI, the CCA, or other ways of measuring a mediums' activity. The second part of our implementation leans towards the collection phase, and in it, we explain how and why we made numerous choices when it came to coding our solution, such as our choice in forward error correcting code.

This is an in-depth project, which asks us to modify directly the hardware of our nodes and sink, and that relies on low level programming. To this end, we had to do a lot of research, and understand the devices we were using and the specificity of the operating system we were using. We needed to change register values, to modify the code of some drivers, all while making sure that we didn't break our algorithm by inadvertence.

However, we indeed managed to make it through, and we even were able to put our results inside graphics and bring the project close to its end, although there is still a lot of work to do on some specific parts, such as implementing actuators inside our network.

Glossaire :

Acquittement : Court message envoyé par l'appareil A en réponse à une réception valide en provenance de l'appareil B pour faire savoir à l'appareil B la bonne réception de sa trame.
Case à équipement : Compartiment de lanceur dans lequel on retrouve les modules de pistage, de guidage, de sauvegarde et de localisation. Notre réseau est déployé à l'intérieur d'une case à équipement.
CCA : Clear Channel Assessement, écoute du canal sur une période de 128 µs afin de détecter de l'activité.
Charge utile : Partie d'une trame qui contient les données à destination des couches supérieures ou pour l'application.
Code correcteur d'erreur : Regroupement d'un grand nombre d'algorithme dont le principe est de pouvoir corriger des transmissions erronées dans le cas où celles-ci auraient subies des interférences.
CRC : Contrôle de Redondance Cyclique, permet de vérifier l'intégrité d'une trame, et de savoir si elle a été interférée lors de la transmission.
DRP : Data ResPonse, réponse des nœuds au puits lors de la phase de collecte.
DRQ : Data ReQuest, demande du puits aux nœuds pour récupérer leurs mesures.
ED : Energy Detection, moyenne de plusieurs mesures de RSSI, permettant normalement d'avoir de résultats plus stables.
IEEE 802.15.4 : Norme définie par le comité IEEE 802, qui portent sur les réseaux locaux à faible portée, débit, et consommation énergétique.
MHR : MAC HeadeR, en-tête de la trame contenue dans la charge utile physique.
Module radio : Composant physique de notre matériel qui contient les éléments nécessaires à la transmission et la réception de trames.
Nœuds : Entité dans notre réseau qui récolte des données ou joue un rôle d'actionneur. Commandé par le puits.
Phase de collecte : Phase de notre protocole dans laquelle le puits va parcourir chaque secteur pour y interroger les nœuds qui sont indiqués dans sa file d'attente.
Phase de découverte du voisinage : Phase de notre protocole dans laquelle le puits va interroger chaque secteur pour découvrir les nœuds qui s'y trouvent.
PHR : Physical HeadeR, en-tête de la trame transmise.
PHY : PHYsical Layer, couche physique.
Pilote radio : Fichier modifiable qui permet de configurer le module radio.
Puits : Éléments central de notre réseau, ordonnanceur, qui va gérer tout ce qui transite sur notre réseau. Tous les nœuds communiquent uniquement si le puits les a interrogés.
Round : Cycle temporel dans lequel le puits va d'abord envoyer un message à tous les nœuds dans un secteur, puis attendre leurs réponses dans des slots. Dans chaque secteur, notre protocole va faire autant de round que nécessaire pour y découvrir tous les nœuds.
RSSI : Received Strength Signal Indicator, mesure de l'énergie sur le médium, pour notre puits faites toutes les 2 µs.

Secteur : Zones géographiques autour du puits qui correspondent chacune à une orientation de nos antennes commutatives.
SHR : Synchronisation HeadeR, préambule de nos trames qui servent à nos appareils pour démoduler les messages qu'ils reçoivent.

References :

[1] Guéréguin Der Sylvestre SIDIBE, Architecture matérielle et protocolaire pour réseaux de capteurs sans fil adaptés à la surveillance environnementale

[2] Site web du CNES : URL
https://cnes.fr/fr

[3] Domaines d'applications du CNES : URL
https://cnes.fr/fr/web/CNES-fr/3358-les-5-themes-dapplications.php

[4] Site web du LIMOS : URL
https://limos.fr/

[5] Le travail du LIMOS sur les Mines de Saint Etienne : URL
https://www.mines-stetienne.fr/recherche/unites-de-recherche/umr-cnrs-6158-limos-laboratoire-informatique-modelisation-et-optimisation-des-systemes/

[6] Source d'information sur les lanceurs : URL
https://cnes.fr/fr/un-peu-de-vulgarisation-le-lanceur

[7] Raphaël Bidaud, Marie-Françoise Servajean, Guéréguin Der Sylvestre Sidibé, Aurélien Surieur, Michel Misson, Spécification de la topologie et des protocoles de niveau 1 et 2 retenus, Livrable numéro 4 de la collaboration précédente avec le CNES

[8] Théophile Decaesteker, Léo Essique, Implémentation d'une solution théorique de collecte de données, Rapport de projet tutoré 2021-2022

[9] Fiche technique des cartes Openmote-B : URL
https://openmote.com/wp-content/uploads/2021/06/content.pdf

[10] Fiche technique des cartes SAMR21-XPRO : URL
http://ww1.microchip.com/downloads/en/devicedoc/sam-r21_datasheet.pdf

[11] Leçon sur les interruptions : URL
https://linux-kernel-labs.github.io/refs/heads/master/lectures/interrupts.html

[12] Murat Demirbas, Onur Soysal, Muzammil Hussain, A Singlehop Collaborative Feedback Primitive for Wireless Sensor Networks : URL
http://ubicomp.cse.buffalo.edu/pollcast/Pollcast%20a%20Singlehop%20Collaborative%20Feedback%20Primitive%20for%20Wireless%20Sensor%20Networks.pdf

[13] Emmanuel Baccelli, Cenk Gündoğan, Oliver Hahm, Peter Kietzmann, Martine Lenders, Hauke Petersen, Kaspar Schleiser, Thomas C. Schmidt, Matthias Wählisch, IEEE Internet of Things Journal, Vol. 5, No. 6, pp. 4428-4440, December 2018. : URL

RIOT: An Open Source Operating System for Low-End Embedded Devices in the IoT

[14] Dépôt GITHUB du système d'exploitation RIOT OS : URL https://github.com/RIOT-OS/RIOT

[15] Fiche technique du module radio at86rf233 : URL https://ww1.microchip.com/downloads/en/DeviceDoc/atmel-8351-mcu_wireless-at86rf233_datasheet.pdf

[16] Fiche technique du module radio at86rf215 : URL https://www.mouser.com/datasheet/2/268/atmel-42415-wireless-at86rf215_datasheet-1368619.pdf

[17] IEEE Std 802.15.4 - 2006 : IEEE Standard for Information technology – Local and metropolitan area networks – Specific Requirements – Part 15.4 : Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low Rate Wireless Personal Area Networks (WPANs)

[18] Théophile Decaesteker, Création d'un réseau de capteurs sans fil robuste, Rapport de stage

4ème de couverture

Dans le monde de l’industrie aéronautique, le poids joue un rôle très important. Dans ce rapport de stage, nous allons vous expliquer comment, en utilisant un réseau de capteurs sans fil, nous proposons une solution à la question du poids. Nous vous détaillerons notre protocole, les apports techniques sur lesquels nous avons contribués, et comment nous comptons mettre en place un système de contrôle commande sur un système embarqué.