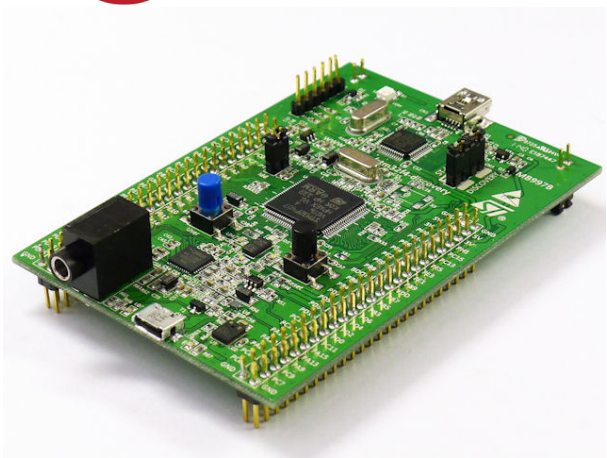
Rapport de projet de 5ème année

Informatique, Microélectronique, Automatique

Réseau de capteurs temps réel



Élève: Edmur Lopes
Encadrants: Alexandre Boé - Xavier Redon - Thomas Vantroys

Année 2017-2018

Contents

1 Introduction

Dans le cadre de la cinquième année au sein du département Informatique, Microélectronique et Automatique de Polytech Lille, j'ai choisi de travailler sur un des sujets de projets proposés par des personnes intérieures à l'école. L'intitulé de ce sujet est "Réseau de capteur temps réel", avec le but d'introduire sur un réseau sans fil non temps réel le notion d'un réseau temps réel.

Je tiens à remercier toutes les personnes qui participent de près ou de loin à l'élaboration de mon projet et plus particulièrement: Monsieur BOE, Monsieur REDON, Monsieur VANTROYS, encadrants de mon projet à Polytech Lille, pour leurs conseils et remarqu es concernant mon travail.

2 Présentation du projet

2.1 Contexte

L'industrie 4.0 propose d'organiser les moyens de production de façon agile et autonome. Pour cela, on peut envisager de créer des ilots de travail interconnectés. Chaque ilot pourrait être déplacé pour optimiser soit l'utilisation des ressources soit la production. Dans le cas d'ilots identiques, pour faciliter la mise à jour des systèmes et le contrôle des machines, il peut être intéressant de faire la régulation sur un système "serveur". Les ilots deviennent alors les clients et envoient leurs données au serveur qui renvoie par la suite la commande.

2.2 Description du projet

Ce projet consiste à évaluer la possibilité de contrôler en temps réel un robot par liaison sans fil. Cela sera possible avec la mise en place d'un réseau sans fil de capteurs temps réel, afin d'échanger des informations entre les nœuds. Ce réseau va devoir être capable d'effectuer le chemin le plus court vers la cible dans le but de diminuer le temps de propagation de l'information, comme montre la figure 2.

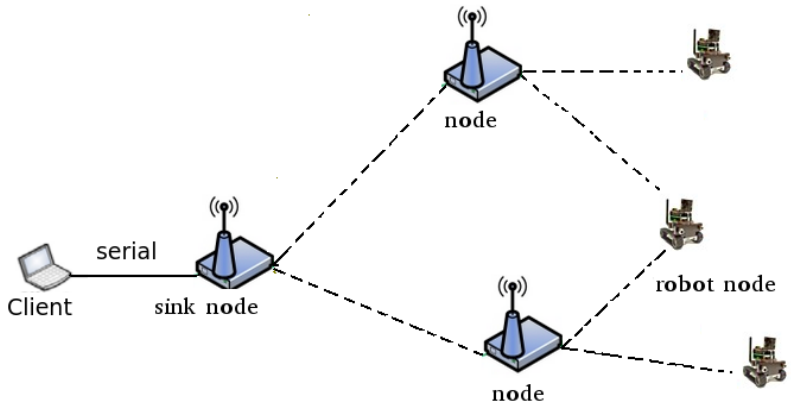


Figure 2: Réseau de capteurs sans fil

2.2.1 Objectif du projet

L'objectif est donc la mise en place d'un réseau avec des cartes stm32 équipées d'un module radio at86rf231, plus précisément:

1. Développer un réseau non temps réel basé sur des cartes STM32 et des radios à 868 MHz, embarquant un système d'exploitation Riot OS
2. Évaluer les temps de propagation de l'information en fonction de l'environnement (perturbations, distance, taille du message...)
3. Modifier la partie MAC de Riot OS pour garantir les temps de latence

3 Travail accompli

Ici je vais vous présenter le travail que j'ai effectué déjà sur ce projet. Je vais tout d'abord commencer à vous présenter la prise en main du matériel que j'utilise pour aboutir aux objectifs attendus, la simple et belle raison de cette prise main c'est qu'il est la première fois que j'utilise **RIOT OS** et la carte **STM32F4discovery**, pour ensuite présenter en détailles les méthodes appliquées afin de mettre en place le réseau ainsi que les tests faits sur le même.

3.1 Prise en main de RIOT OS

RIOT est un système d'exploitation construit autour d'un micro-noyau temps-réel, développé pour les besoins de l'Internet des objets et des réseaux sans fil. Initialement lancé en Europe par INRIA, Freie Universität Berlin, et l'Université de Hambourg en 2013, RIOT est actuellement développé par une large communauté du logiciel libre rassemblant des développeurs du monde entier. Son code source ce trouve sur Github.

Pour utiliser RIOT avec la carte STM32 il faut tout d'abord installer les outils nécessaires à la compilation et au débogage de la famille de microcontrôleurs ARM ainsi que le paquetage g++-multilib si le système hôte est 64 bits, ensuite il faut récupérer son code source depuis git.

3.1.1 Changement d'état d'une led avec les interruptions gpio

J'ai commencé par réaliser une application basé sur l'exemple "hello_world" de RIOT: cette simple application "gpio_to_serial" est capable d'allumeréteindre une led dès qu'une interruptions est générée lors qu'on appuie sur le bouton "user" et affiche l'état de la led au terminal. Cette application m'a permit de comprendre comment fonctionne le système RIOT et de passer à l'étape suivante, qui est la mise en place d'un premier réseau avec les modules radios.

3.2 Mise en place d'un réseau statique à 3 nœuds

Après avoir compris comment marche vraiment le système RIOT, Je suis passé à la réalisation d'un premier réseau multi-point avec 3 nœuds (2 sauts); Pour que cela soit possible j'ai défini pour chaque carte ayant un module radio son adresse ipv6 global qui va permettre d'échanger des paquets entre elles. J'ai défini tout d'abord une adresse global du type **baad:a555::Hwaddr**. Ce type de réseau se comporte comme un réseau linéaire avec un nœud central se chargeant uniquement de router les paquets vers la destination, comme montre la figure 3.

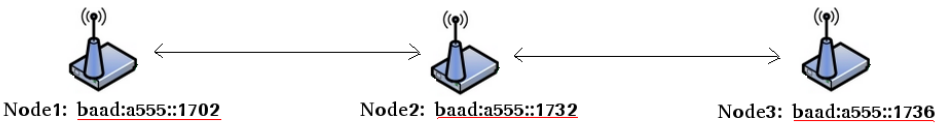


Figure 3: Réseau static à 3 nœuds

RIOT OS dispose des nombreuses fonctions qui nos permet de tester ou configurer les interfaces réseau, telles que:

1. **ifconfig:** permet de configurer l'interface réseau
2. **fibroute:** table dynamique qui lie les adresses MAC aux ports
3. **pingv6:** permet de tester l'accessibilité d'une autre machine à travers un réseau IP en envoyant de paquet ICMPv6
4. **udp:** permet de démarrer un serveur udp et d'envoyer des paquets via le protocole udp sur un port prédéfini

L'exemple gnrc_networking de RIOT, permet de mettre en place la pile réseau GNRC IPv6/6LoWPAN qui est dédié à l'internet des objets. Cet exemple agit comme un routeur et permet de définir des routes statiques ainsi que d'utiliser RPL. J'ai donc utilisé cet exemple afin de mettre en place le réseau, il suffit de rajouter le module at86rf231 dans le Makefile du projet et configurer ses paramètres.

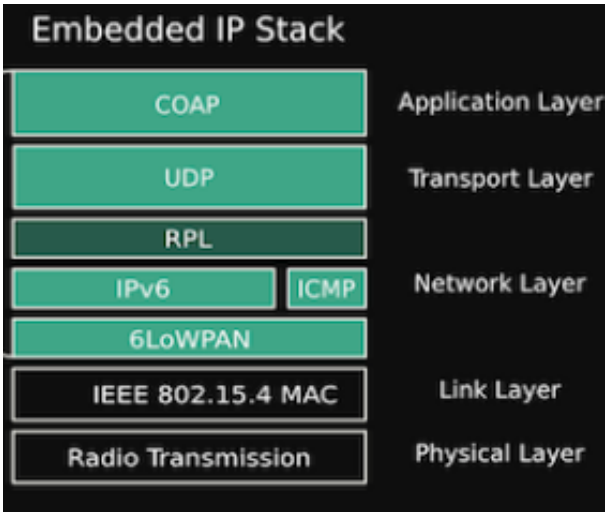


Figure 4: Pile réseau pour IoT

Une fois le réseau mis en place, on a fait des tests afin de voir si les cartes arrivaient à communiquer entre-elles. Les résultats ont étés satisfaisants comme le montre la figure 5.

```
ping6  baad:a555::1736
12 bytes from baad:a555::1736: id=86 seq=1 hop limit=63 time = 18.595 ms
12 bytes from baad:a555::1736: id=86 seq=2 hop limit=63 time = 18.917 ms
12 bytes from baad:a555::1736: id=86 seq=3 hop limit=63 time = 16.996 ms
--- baad:a555::1736 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2.0673703 s
rtt min/avg/max = 16.996/18.169/18.917 ms
```

Figure 5: ping entre les nœuds

3.3 Routage dynamique avec le protocole RPL

Les progrès technologiques permettent maintenant d'envisager la connexion des objets du quotidien à l'Internet. Des solutions ouvertes et interopérables doivent cependant être utilisées pour garantir une communication optimum entre ces objets. Le protocole de routage est un élément clé de cet objectif, car il permet pour chaque objet de décider comment joindre un autre objet. Les contraintes s'appliquant aux objets (faible puissance, communications instables) doivent être prises en compte pour le développement de protocole de routages adaptés.

3.3.1 Introduction au protocole RPL

Les périphériques réseau exécutant le protocole RPL sont connectés de manière acyclique. Ainsi, un graphe acyclique DODAG (Destination-Oriented Directed Acyclic Graph) est crée comme montre la figure 6.

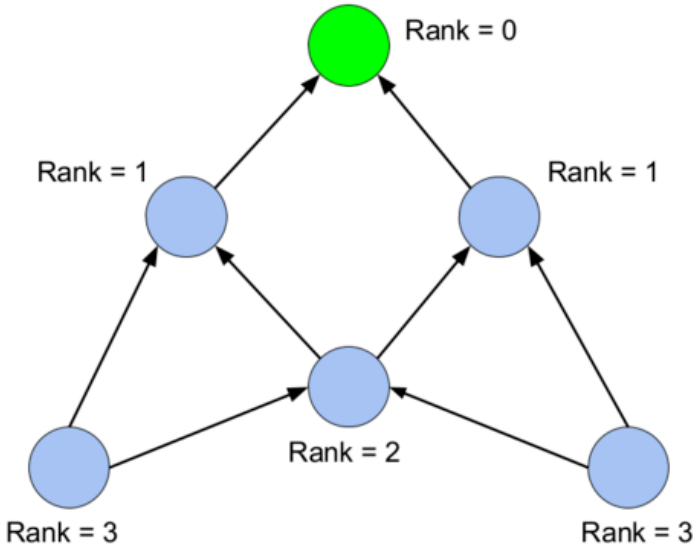


Figure 6: DODAG RPL

Chaque nœud choisit le meilleur chemin pour arriver jusqu'au nœud racine, chaque nœud utilise un objectif function (OF) afin de trouver le mieux chemin pour arriver au nœud racine, cette fonction définie le metric de routage qui va être appliqué.

Le processus de routage commence avec la création d'un DAG, le nœud racine envoie son DODAG à tout les noeuds accecible. A chaque niveau de l'arbre de routage les nœuds font le routage en se basant sur l'OF. Une fois qu'un noeud joint le DODAG, il choisit une racine comme son parent et calcule son rank, ce rank correnspond au metric qui indique les coordonnées du nœud dans la hiérarchie du réseau. Les autres noeuds vont répéter ce processus de sélection des parents et de notification des informations DODAG pour de nouveaux appareils possibles. Lorsque ce processus se stabilise, le routage des données peut alors commencer. Le processus crée des routes ascendantes (des nœuds à une racine). Pour construire les routes descendantes, le protocole RPL envoie des messages ICMPv6 telles que:

1. **DAG Information Object (DIO):** transmet des informations qui permettent à un nœud de découvrir une instance RPL, d'apprendre ses paramètres de configuration et de choisir des parents dans le DODAG

2. **DAG Information Solicitation (DIS):** sollicite un DIO à partir d'un noeud RPL

3. **Destination Advertisement Object (DAO):** utilisé pour propager les informations de destination vers le haut le long du DODAG

3.3.2 Application dans RIOT OS

Le protocole RPL est présent dans RIOT OS. On s'est basé sur le travail de nos collègues de l'année dernier (PFE 7) pour mettre en place ce réseau dynamique. En effet ils ont changé les fichiers de base de l'implémentation du protocole RPL dans RIOT OS afin d'améliorer le choix du parent avec lequel un nœud communique, en utilisant les notions de metrics.

Le rang de chaque noeud, dans ce projet, est calculé en fonction de deux principaux indicateurs: le LQI et RSSI.

LQI (Link Quality Indicator) est une métrique de la qualité actuelle du signal reçu. Le LQI donne une estimation de la facilité avec laquelle un signal reçu peut être démodulé en accumulant l'amplitude de l'erreur entre les constellations idéales et le signal reçu sur les 64 symboles immédiatement après le mot de synchronisation. LQI est mieux utilisé en tant que mesure relative de la qualité de la liaison (une valeur faible indique une meilleure liaison qu'une valeur élevée), puisque la valeur dépend du format de modulation.

RSSI (Received Signal Strength Indication) est une indication de force du signal. Il ne se soucie pas de la «qualité» ou de la «correction» du signal. LQI ne se soucie pas de la puissance réelle du signal, mais la qualité du signal est souvent liée à la puissance du signal. En effet, un signal fort est susceptible d'être moins affecté par le bruit et sera donc perçu comme "plus propre" ou plus "correct" par le récepteur.

Il y a quatre à cinq «cas extrêmes» qui peuvent être utilisés pour illustrer comment RSSI et LQI fonctionnent:

- Un signal faible en présence de bruit peut donner un RSSI faible et un LQI élevé.
- Un signal faible en absence totale de bruit peut donner un RSSI faible et un LQI faible.
- Un fort bruit (provenant généralement d'un interféreur) peut donner un RSSI élevé et un IQI élevé.
- Un signal fort sans trop de bruit peut donner un RSSI élevé et un faible LQI.
- Un signal très fort qui provoque la saturation du récepteur peut donner un RSSI élevé et un LQI élevé.

La fonction qui calcul le rang d'un noeud, en fonction des indicateurs LQI et le RSSI:

```
//Avec base_rank=256, rssi <= 28 et <=255
add = 256 - (parent->mc.rssi + parent->mc.lqi)/10;
my_rank = base_rank + add;
```

Figure 7: fonction pour calculer le rang

De cette manière, on se rendre mieux compte de la distance d'un nœud par rapport au DODAG racine diminuant son rang en fonction de ces deux facteurs.

Pour lancer le routage RPL sur le cartes il suffit de choisir un nœud comme racine (avec la fonction **gnrc_rpl_root_init** dans un thread) lors du flashage du code.

```
 rpl
instance table:      [X]
parent table:        [X]     [ ]     [ ]

instance [1 | Iface: 7 | mop: 2 | ocp: 0 | mhri: 229 | mri 0]
     dodag [baad:a555::1702 | R: 485 | OP: Router | PIO: on | CL: 0s | TR(I=[8,20], k=10, c=11, TC=4088s)]
              parent [addr: fe80::1210:642c:1432:1702 | rank: 256 | lifetime: 256s]
```

Figure 8: nœud robot choix du parent

Une telle solution permet de choisir le parent avec le rang le plus faible et autorise un choix plus objectif du parent.

3.4 Implémentation d'un driver pour le pilotage des moteurs

Les cartes STM32 n'ont pas la puissance nécessaire pour piloter les deux moteurs DC utilisés sur le robot. Pour cela on utilise un module driver de moteur afin d'accomplir cette tâche. Le driver TB6612FNG est un driver de moteur à courant continue, qui possède deux entrées IN1 et IN2 permettant de choisir l'un de quatre mode suivant: mode CW (tourner dans le sens des aiguilles de la montre), mode CCW (tourner dans le sens anti horaire), frein court et mode d'arrêt.

Dans RIOT OS, on trouve plusieurs drivers déjà implémentés mais le TB6612FNG n'en fait pas partie. Alors j'ai mis en place un module que j'ai appelé **robotcar** incluant le *driver de moteur + un codeur infrarouge*. Le codeur infrarouge va permettre, avec la roue codeuse, de donner la distance que le robot va devoir parcourir.

Les drivers sont trouvés dans les répertoires: RIOT/drivers/ pour la définition des fonctions (archive .c) et RIOT/drivers/include pour le prototypage de fonctions (archive .h). j'ai inclut un fichier robotcar.c et robotcar.h. Pour utiliser ce driver, il suffit d'ajouter dans le fichier Makefile la ligne suivante: **USEMODULE += robotcar**.

3.4.1 Les fonctions implémentées

- **robot_init**(robot_t *dev, pwm_t pwm, int pwm_channel, gpio_t in1, gpio_t in2, gpio_t stby, gpio_t encoder,int id) : cette fonction sert à configurer les moteurs qui se trouvent sur le robot, elle prend en paramètre le moteur, la PWM avec son numéro de canal, les ports sur lesquels il est connecté ainsi que le port de son codeur incrémental

- **robot_drive**(robot_t *dev, uint16_t speed, int direction) : permet de guider le robot en choisissant la direction souhaité (CW/CCW)

- **robot_stop**(robot_t *dev1, robot_t *dev2) : arrêt des moteurs

- **cm_to_steps**(uint8_t cm) : convertit des centimètres en pas

- **robot_move_forward**(robot_t *dev1, robot_t *dev2, uint16_t speed,int steps)

- **robot_move_reverse**(robot_t *dev1, robot_t *dev2, uint16_t speed,int steps)

- **robot_spin_left**(robot_t *dev1, robot_t *dev2, uint16_t speed,int steps)

- **robot_spin_right**(robot_t *dev1, robot_t *dev2, uint16_t speed,int steps)

Les quatre dernières fonction permettent au robot, soit de tourner à droite ou gauche, soit d'aller tout droit ou en arrière avec une vitesse donnée et la distance à parcourir en cm.

Le robot peut être piloté manuel au automatiquement. A ce qui concerne le pilotage manuel, on a utilisé le shell de RIOT OS et on a créé des lignes de commandes. Par exemple pour aller tout droit il suffit de taper la commande sur le shell côté client: **motor move forward 500 50**. Ici on a demandé au robot d'aller tout droit avec une vitesse de 500 jusqu'à 50 cm.

3.5 Calcul du temps de latence du réseau

La latence, fait référence au RTT (Round Trip Time/Delay) d'une requête c'est à dire le temps que met un signal pour parcourir l'ensemble d'un circuit fermé. En particulier ça correspond à l'heure à laquelle le paquet doit atteindre la destination. La machine de destination doit répondre à cette requête et cette réponse doit parvenir au demandeur.

Chaque réseau a une latence. La quantité totale de latence, que l'on obtient lors de la connexion à un hôte distant donné, peut varier considérablement en fonction des conditions du réseau. C'est alors que on s'intéresse dans cette partie à l'étude de la latence du notre réseau (le temps de transmission d'un paquet à partir de la couche MAC jusqu'au récepteur en fonction de la taille du paquet ou de la distance émetteur/récepteur).

Il faut savoir qu'il y a plusieurs autres facteurs qui peuvent ajouter une latence supplémentaire en plus de la latence de la couche MAC:

- les délais de routage: Si les paquets doivent passer par un saut de réseau très encombré, ce délai de buffer peut ajouter une latence considérable.

- la retransmission des paquets
- retards de propagation du signal dans le matériel qui reçoit, transmet ou répète des paquets

Afin de calculer la latence du réseau j'ai procédé de la manière suivante:

1. Lors de l'envoi d'un paquet, le client calcule le temps T1 et attend la réponse du serveur (temps d'attente max = 1 seconde)
2. Le serveur reçoit le message et le retransmet au client
3. Lors que le client reçoit la réponse du client il calcule le temps de réception T2 du paquet. Si la réponse arrive après 1 seconde, on considère que le paquet a été perdu
4. Le round-trip time est calculé donc de cette manière: rtt = T2 - T1

A la fin de tout le processus le client affiche le message suivant:

```
Packets: Size: 32 bytes, Sent = 100, Received = 100, Lost = 0
Round-trip delay: min/avg/max = 10.057000/12.173100/14.198999 ms, time = 1.217735 s
```

Figure 9: réponse client

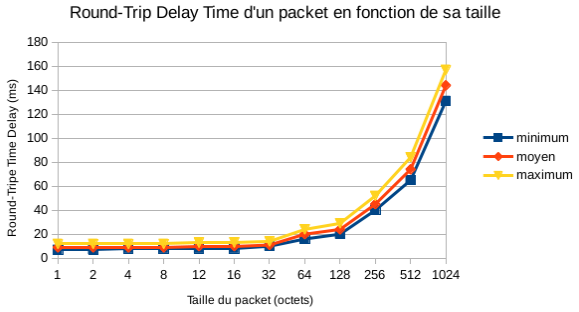


Figure 10: latence d'un réseau 2 nœuds en fonction de la taille du message

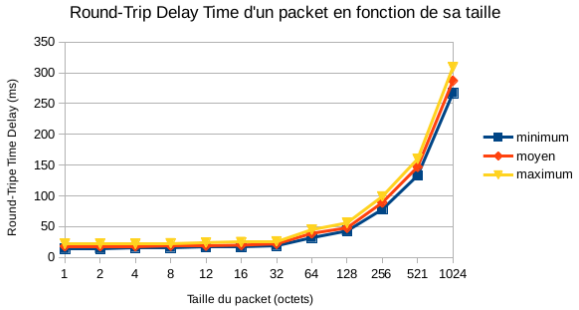


Figure 11: latence d'un réseau3 nœuds en fonction de la taille du message

D'après le résultat ci-dessus, on constate que le temps de latence augmente selon la taille du message. La distance entre le nœuds n'a pas vraiment d'influence sur le temps de propagation de messages.

On compte envoyer des requêtes avec une taille de 32 octets, ayant le message suivant: [cmd,speed,distance], la commande (cmd) a une taille de 8 bytes, la vitesse (speed) 16 octets et la distance que le robot doit parcourir, a une taille de 8 octets. Le temps total qu'un paquet de 32 octets met pour faire le tour d'un réseau avec de 2 nœuds est compris entre 10 et 14 millisecondes; Pour un réseau avec 3 nœuds le temps de latence est le double de celui avec 2 nœuds, si un rajoute un quatrième noued il va falloir multiplier le temps de base par trois et ainsi de suite.

Temps N nœuds = (N-1) x Temps de 2 nœuds

3.6 Conception des nouveaux modules RF

Les modules radios que l'on utilise maintenant, ne sont pas directement fixés sur les cartes stm32 et on doit à chaque fois faire le câblage lors de tests. Les cartes stm32 n'ont pas aussi un trou permettant de la fixer sur le robot. Alors pour rendre les tâches moins difficiles, on a décidé d'en concevoir des nouveaux.

3.6.1 Réalisation du schematic sous KiCad

On s'est basé, sur la datasheet du composant AT86RF231, pour réaliser le schéma électrique de la carte. On peut y trouver plusieurs notes d'application ainsi qu'un BOM pour la réalisation du module radio. On a choisit de réaliser la carte avec KiCad car, est un logiciel facile à utiliser et gratuit.

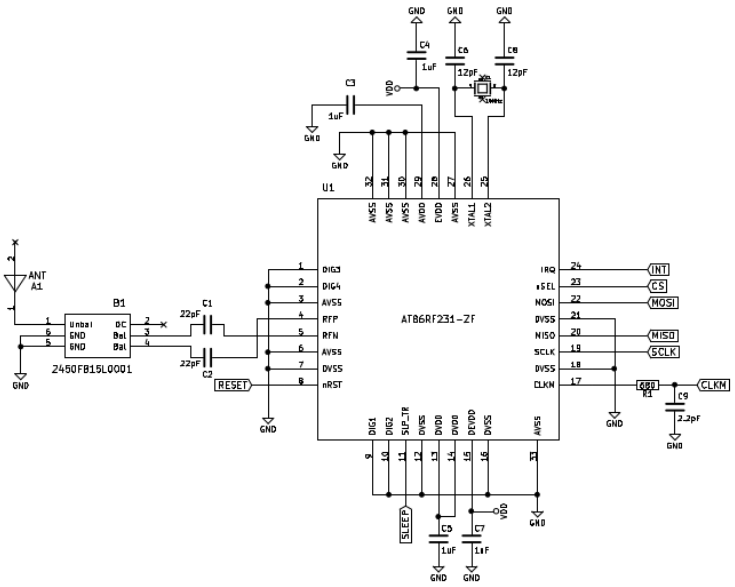


Figure 12: schematic du module sous kicad

3.6.2 Routage de la carte

Le tramsmetteur radio AT86RF231 envoie les données modulées via les pins RFP et RFN. La bande utilisée pour les liaisons Zigbee est à 2,45 GHz ce qui correspond aux hyperfréquences.L'antenne utilisée a une impédance d'entrée de 50 Ohms et il de même pour l'impédance de sortie du balun. Pour adapter ces deux éléments, il faut une ligne d'impédance caractéristique de 50 Ohms.Pour une ligne microstrip, l'impédance caractéristiques dépend fortement de ses dimensions et du matériau isolant. Pour calculer la taille de piste nécessaire, on a utilisé un outil en ligne, figure 13.

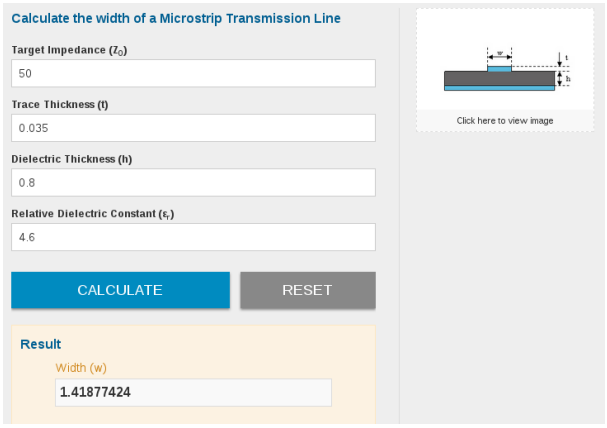


Figure 13: Calcul de la largeur de piste

Il nous faut donc l'épaisseur t de la piste, l'épaisseur h du diélectrique ainsi que sa permittivité. A Polytech, il est possible de réaliser des cartes avec des pistes d'épaisseur 35 um avec un diélectrique verre-epoxy FR4 d'épaisseur 0,8 mm. Avec ces informations, on trouve une largeur de piste égale à 1,42 mm. Selon la datasheet de l'antenne, il était conseillé d'ajouter un guide d'onde coplanaire constitué de trous afin d'améliorer le transport de l'énergie jusqu'à l'antenne. De plus, l'ajout de vias permet d'atténuer l'effet capacitif des plans de masses situés de part et d'autres du diélectrique. Le rôle du balun est de transformer une impédance symétrique en impédance asymétrique et vice versa. Le balun réalise également une fonction d'adaptation d'impédance entre les ports RFP et RFN et l'antenne. Enfin les condensateurs à l'entrée des ports RFP/RFN sont utilisés pour supprimer la composante continue de l'entrée RF provenant de l'antenne.

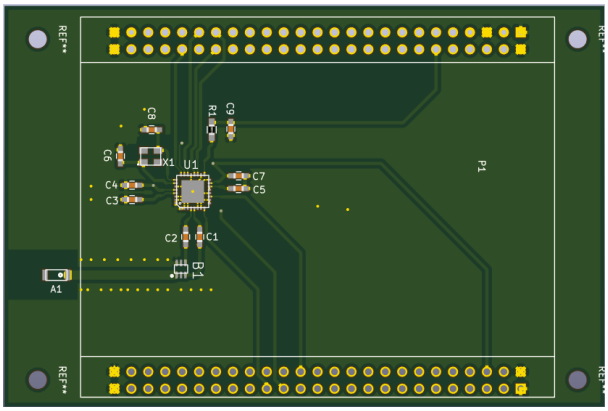


Figure 14: PCB module radio

3.6.3 Soudure et test du module radio

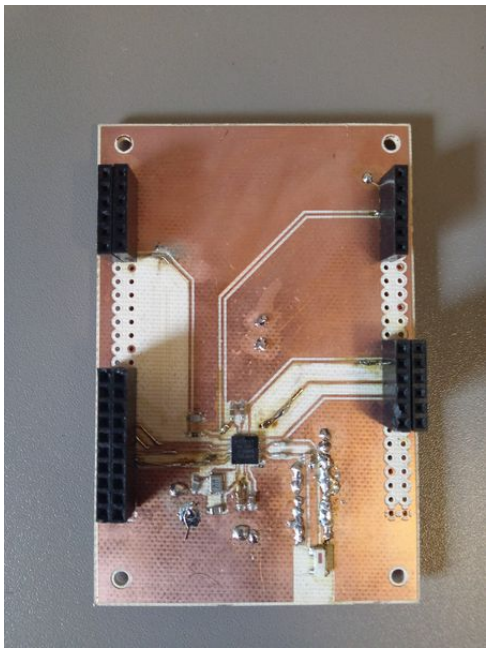


Figure 15: Module radio soudé

Une fois la soudure du module radio terminé, on est passé au test de ce même; Tout paraissait marcher correctement mais le module ne démarrait pas lorsque le pin CS (chip select) était connecté.

Ensuite on avait essayé de trouver d'où venait le problème, en mesurant le niveau de tension à chaque pin tout en regardant la note d'application du composant at86rf231. On avait trouvé que sur les deux pins du quartz on devait avoir le même niveau de tension (0.9 Volts), pourtant on avait 0.7 Volts sur un seul pin et 0 volts sur l'autre; cela nous as emmené à une conclusion: le quartz n'oscille pas; Pour vérifier nous avons branché un oscilloscope et c'était vraiment le cas. Alors soit le problème viens du composant at86rf231, soit du quartz.

3.7 Mesure de la latence avec SNTP (one way latency)

Ayant des mauvais résultats lors du calcul du temps de latence, avec la méthode round trip delay... On a décidé d'implémenter le protocole SNTP afin de synchroniser l'horloge locale des cartes sur une référence d'heure. Nous avons mis en place la méthode serveur/client, cela marche de la manière suivant:

- un nœud « serveur » répond aux demandes d'heure émises par un nœud « client »
- les parents sont les serveurs, les enfants sont les clients
- en opérant dans le mode « serveur », un nœud annonce son désir de synchroniser
- en opérant dans le mode « client », un nœud annonce son désir d'être synchronisé
- le mode d'adressage « unicast » est utilisé pour transférer les messages de demande et de réponse

3.7.1 Description du modèle NTP « client/serveur » implémenté

Dans un premier temps le client fait une demande de synchronisation au serveur, une fois synchronisé le client calcul l'offset entre les deux horloges (offset), ensuite le serveur envoie un message avec l'heure actuelle (T1), enfin le client calcul le temps de réception du message (T2).

La latence est calculée de la manière suivante: T2 - T1 + offset.

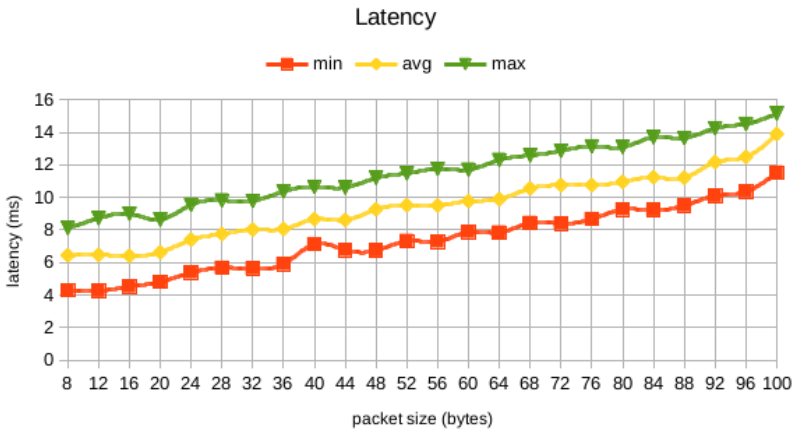


Figure 16: Temps de latence d'un réseau static

Comme nous pouvions le voir, la longueur du paquet a une influence sur le temps de transmission. Globalement, cette évolution est linéaire.

4 Application robot

L'application mise en place, est une application basée sur du temps réel grâce au système d'exploitation RIOT. En effet RIOT permet la programmation d'applications en langages C et C++ ainsi que le multithreading et le temps réel. La capacité en temps réel est possible grâce à une interruption de latence très faible (à peu près 50 cycles d'horloge) et à une programmation basée sur les priorités.

Le nœud robot est piloté par des commandes provenant du nœud racine; Afin de s'assurer que la requête est bien arrivée à sa destination, une fois que le nœud racine envoie la commandes, il attend (25 ms max) un retour de la part du robot. S'il ne reçoit aucun retour, un message est affiché disant: «mode auto». Côté robot, on attend 5 seconds, s'il ne reçoit rien on passe en mode manu, ce mode dure 15 seconds si aucun message n'est toujours pas arrivé, ensuite il s'arrête.

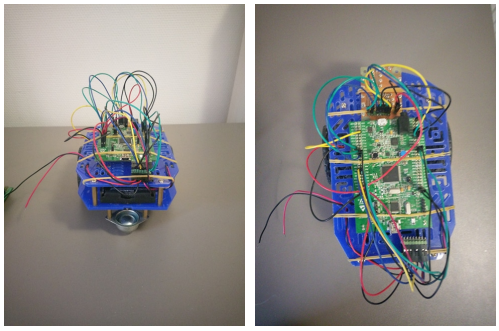


Figure 17: Robot utilisé pour l'application

5 Conclusion

Au terme de ce projet, le sujet est très ouvert et difficile. Cela m'a apporté beaucoup des connaissances dans le domaines de l'internet des objets et sur l'aspect temps réel sur un réseau. L'effet aussi de programmer dans système inconnu et très utilisé pour les réseaux à faible consommation m'a permis d'ouvrir une voie vers les IoT dans le cadre de mon projet professionnel. J'ai eu plusieurs difficultés différentes car ce projet est multidisciplinaire et plusieurs outils utilisés dans ce projet m'étaient inconnus.

Le système d'exploitation utilisé n'a pas une version stable, cela m'a empêchait de comprendre mieux le travail (code) qui a été fait avant par mes collègues de l'année dernière, et j'en ai perdu beaucoup de temps. Malgré toutes les difficultés j'ai pu respecter les objectifs fixés en début de projet. Le code de mon travail se trouve sur ce dépôt git: https://archives.plil.fr/elopes/PFE15.git