Université de Pau et des Pays de l'Adour
Master technologie de l'internet
Pau, France

Réseaux émergents sans-fil et leurs applications

RAPPORT DE PRÉSENTATION DU PROJET

Projet

Bastien Delbouys
bastien.delbouys@etud.univ-pau.fr

Bagre Gnebehi
bagre.gnebehi@etud.univ-pau.fr

11 décembre 2019

Table des matières

1 Introduction

Dans le cadre de l'unité d'enseignement Réseaux émergents sans-fil et leurs applications, il nous a été demandé de réaliser un système de routage sans fil.

Pour cela, nous avions à disposition des Arduinos équipées de module XBee afin de permettre les communications sans fil entre elles.
Pour démontrer le fonctionnement de ce système, il a été décidé que chaque Arduino devrait envoyé sur le réseau sans-fil un message a destination d'une autre.
Cependant, cet autre Arduino n'est pas nécessairement atteignable directement, c'est pourquoi il est nécessaire de faire le routage de notre réseau avant l'envoi de la donnée.

Dans un premier temps, il nous a été demandé de définir le système global dans lequel les Arduinos communiqueraient.
Ce premier travail est détaillé dans la section Système Global du rapport.

La section Fonctionnalités, contient une description des différentes fonctionnalités implémentées dans notre projet.

Et pour finir, certaines composantes logicielle que nous jugeons importantes seront détaillées dans la dernière section du rapport.

2 Système Global

Afin de réaliser notre projet, il nous a fallut définir un système global. Ce système peut s'expliquer en trois points :
- La carte du réseau
- Les types de messages
- Le format des données

2.1 Carte du réseau

Travaillant dans un espace restreint, nous avons décider de simuler un réseau à l'aide du schéma suivant.

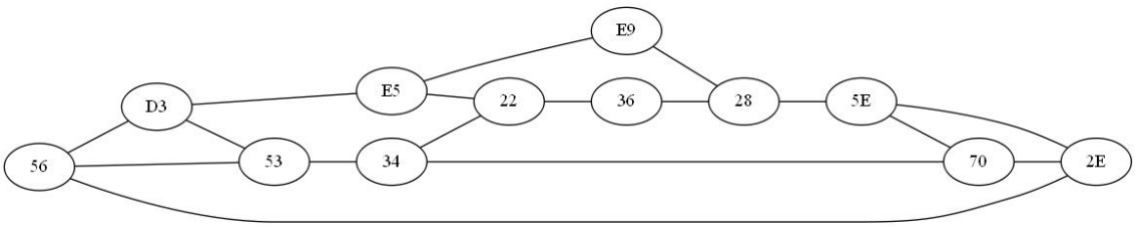

Figure 1 – Carte du réseau virtuel

La principale raison pour ce choix est pour la recherche de route.
En effet, dans un espace restreint et sans restriction de communication, il est très probable que notre demande de route atteigne directement notre cible sans devoir passer par des intermédiaires.
Pour la bonne démonstration et test de nos algorithmes, il sera donc nécessaire de travailler

avec ce schéma.

Chaque noeud représente une carte Arduino, la valeur présente dans le noeud correspond à l'identification de cette carte.
Chaque carte possède un identifiant unique et connaît seulement ces voisins directs.
De ce fait, les liens représente une route de communication sans-fil possible reliant deux cartes.

Notre groupe a travaillé sur l'Arduino identifiée par **22**, par conséquent, le réseau que nous connaissons est le suivant.

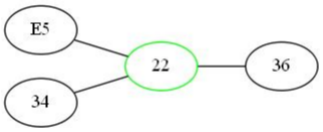

Figure 2 – Voisins de la carte 22

2.2 Les types de messages

Étant donné la nature du projet : le routage, il nous faut utiliser certains types de message pour faire parvenir la donnée d'une Arduino à une autre.
Trois types de messages vont être utilisés dans le cadre de ce projet.

Le premier est de type Route Request (**RREQ**). Ce message va être envoyé à tous avec comme but d'atteindre un destinataire précis.
Lors de sa réception, il est transféré en ajoutant son propre identifiant à la route afin que son destinataire sache comment il lui est parvenu.

Le deuxième type est le Route Reply (**RREP**). Lors de la réception d'un message RREQ nous étant destiné, l'information va être remontée jusqu'à la source dans le but de recevoir sa donnée.
C'est là que le message RREP intervient, il va remonter la route présente dans le message RREQ, et chaque Arduino sur sa route va donc le faire simplement suivre.

Le dernier type est le Data Delivery (**DATA**), lors de la réception d'un RREP nous étant destiné, le chemin est maintenant connu.
Le message est donc ajouté à la donnée à renvoyer par la route trouvée grâce aux précédents échanges.

2.3 Format des données

Afin de communiquer entre elles, les Arduinos se doivent de parler la même langue. Pour se faire, il a été décidé d'un format de données, comprit par toutes les Arduinos de notre réseau.

Cette donnée échangée est en fait un simple message. Ce message, une fois reçu, est analysé par les récepteurs et les données qu'il comprend sont exploitées.

Voici comment est décomposé ce message :

nom de la donnée	type	id	dst	src	route	data
position	0	1	2-3	4-5	6-25	26-95

Figure 3 – Format des données

— **TYPE** ; en position 0, le type du message :
  * Q pour un message de type RREQ
  * P pour un message de type RREP
  * M pour un message de type DATA
— **ID** ; la position 1 représente l'id du message, à chaque nouvel envoi de message cet id va changer.
— **DST** ; en position 2 et 3, l'identifiant de l'Arduino destinataire.
— **SRC** ; en position 4 et 5, l'identifiant de l'Arduino source du message.
— **ROUTE** ; de la position 6 à 25, la route prise par notre message, cette route est composé de la suite des identifiants des Arduinos traversées par le message.
Sa taille est limitée à 10 identifiants.
— **DATA** ; de la position 26 à 95, le message a envoyé sera stocké sur ces cases.

3 Fonctionnalités

Notre Arduino a pour rôle, l'**envoie du message "hello from 22" à l'Arduino identifiée par E9**.

Pour ce faire, certaines fonctionnalités sont nécessaires. Parmi elles, nous devons retrouver, l'envoi de messages, la réception de messages, le traitement des messages, l'information à l'utilisateur, ainsi que la configuration de l'Arduino.
Ces différents points seront présentés dans cette section du rapport.

3.1 Boucle d'exécution

La boucle d'exécution du programme comprends l'intégralité des fonctionnalités.
C'est pourquoi elle est présentée en premier, comme suit :

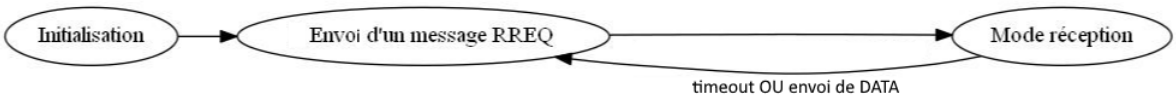

Figure 4 – Boucle d'exécution

Une fois la phase d'initialisation finie, le premier message RREQ est envoyé.
L'Arduino va ensuite passer en mode réception et n'en sortira que lors de l'envoi d'un message DATA ou bien lorsque un timeout est atteint, afin de recommencer son exécution.

NOTE : Ce timeout est une limite temporelle fixe qui va indiquer à l'Arduino que son message s'est peut être perdu.

Le mode de Réception représente la grosse partie du programme. Il y est effectué différentes opérations sur les messages, que ce soit filtre, modification, ou bien redirection et envoi.

3.2 Opérations sur les messages

3.2.1 Filtre

Pour gérer la simulation de notre réseau, l'idée de white-list est reprise. Seuls les messages des voisins directs sont traités.
Pour ce faire, un filtre est imposé lors de la réception de chaque message. Ce filtre va permettre de choisir quels messages seront ignorés et lesquels seront traités.

3.2.2 Traitement

Chaque message passant le filtre va être analyser afin de définir le traitement qui lui correspond.
Le traitement des messages va souvent correspondre à une modification de la données (par exemple : s'ajouter dans la route, changer le type, ou bien encore ajouter le message), mais pas seulement.

Chaque cas à un traitement qui lui est propre et qui est conditionné ;

Cas	Condition	Effet
-	Fin de l'initialisation OU Sortie du mode réception	Création du RREQ : "Q-*id*-E922" Broadcast le message
Réception RREQ	On est la cible	Création du RREP : "P-*id*-22-*src-route*-22" Envoi au dernier de la route
Réception RREQ	On n'est pas la cible ET On n'est pas présent dans la route	Modification du RREQ : "Q-*id-dst-src-route*-22" Broadcast le message
Réception RREP	On est la cible	Création du DATA : "M-*id*-E922-*route*-E9-*data*-" Envoi au premier de la route
Réception RREP	On n'est pas la cible	Envoi au précédent dans la route
Réception DATA	On est la cible	Affichage du message
Réception DATA	On n'est pas la cible	Envoi au suivant dans la route

3.3 Affichage

Pour informer l'utilisateur du travail de notre Arduino, deux moyens sont mis en place.
Le premier est un affichage dans le moniteur série utilisable depuis le l'IDE Arduino.
Le second est un affichage directement sur l'écran LCD branché sur la carte.

NOTE : Limité par la taille de l'écran LCD, le moniteur série va afficher des informations plus complètes.

Les informations affichées peuvent être catégorisées comme ceci :
- SEND ; création d'un nouveau message (RREQ, RREP, DATA).
- FORWARD ; renvoie d'un message reçu (RREQ, RREP, DATA).
- RECEIVE ; réception d'un message DATA, et affichage du contenu.

3.4 Changement de la configuration

Le changement de la configuration doit s'effectuer en modifiant le fichier source du code.

Les variables configurables ont été remontées en haut du fichier pour plus de facilitées. Ces variables sont : notre identifiant, notre cible, notre message, ainsi nos voisins et leurs adresses.

```
char      voisin[6] = "3436E5";
uint32_t  voisin_add[3] = {0x4086D834, 0x4086D836, 0x406FB3E5};
char      src[2] = "22";
char      dst[2] = "E9"; // cible de l'envoi
char      msg[MAX_DATA_SIZE - START_MSG] = "hello from 22";
```

Figure 5 – Variables de configuration

4 Composantes

Dans cette dernière partie du rapport seront présentées les composantes jugées comme étant les plus importantes de notre projet.

4.1 La réception

La partie réception du code est la partie la plus importante.
Elle commence avec une réception basique basée sur la fonction *Arduino_tutorial_Xbee_Receive*. Une fois un message reçu, il faut vérifier si il provient d'un de nos voisins directs à l'aide de la fonction *isWhitelisted* ;

```
// verifie si le message reÃ§u vient bien d'un voisin
boolean isWhitelisted() {
  int       i;
  char      remoteAdd[10];
  XBeeAddress64 &add_from = rx64.getRemoteAddress64();

  sprintf(remoteAdd, "0x%08lx", add_from.getLsb());
  i = 2;
  while(i < 10) {
    if(remoteAdd[i] >= 'a' && remoteAdd[i] <= 'z')
      remoteAdd[i] = remoteAdd[i] - 32;
    i++;
  }
  i = 0;
  while(i < 6) {
    if(remoteAdd[8] == voisin[i] && remoteAdd[9] == voisin[i+1])
      return true;
    i = i + 2;
  }
  return false;
}
```

Figure 6 – Fonction de filtre par rapport à une white-list

Une fois ce filtre passé, certaines opérations vont s'effectuer dans un ordre précis ;

1. Analyse du type de message.
2. Vérification de la présence dans la liste, afin d'éviter de se retrouver deux fois dans une route (pour les RREQ), ou bien d'éviter de faire suivre un message non désiré (réception d'un DATA ou RREP où notre identifiant n'est pas dans la route).
3. Vérification de la source et du destinataire, pour savoir quelle opération effectuer avec la donnée.
4. Traitement sur la donnée (voir *3.2.2 - Traitement*).
5. Affichage pour l'utilisateur.
6. Transfert ou envoi de la nouvelle donnée.

```
if((char)rpayload[0] == 'Q') {(1)
  // message de type RREQ
  // si on est deja dans la route on ne fait rien
  if(alreadyInList() == false) {(2)
    // si on est la cible, on renvoie un RREP
    if((char)rpayload[2] == src[0] && (char)rpayload[3] == src[1]) {(3)
      createRREP(); (4)
      Serial.print("Send RREP... ");
      displayData();
      printLCD(0, "Send RREP...", true);
      sendToPrevious();(6)
    }
    else if ((char)rpayload[4] != src[0] || (char)rpayload[5] != src[1]) {(3)
      // sinon on s'ajoute dans la liste et on fait suivre si on a pas atteind la taille max de la route
      completeRREQ();(4)
      if(data[START_MSG] == '\0') {
        Serial.print("Forward RREQ... ");
        displayData();
        printLCD(0, "Forward RREQ...", true);
        sendData(0x00000000, 0x0000FFFF);(6)
      }
    }
  }
}
```

(5)

Figure 7 – Code pour le traitement d'un message de type RREQ

4.2 Les fonctions send

Les fonctions *send* sont appelées à de nombreuses reprises au cours de l'exécution du programme, ce qui en fait une des composantes centrales.

La fonction *send* de base est appelée *sendData*, il s'agit de la fonction *Arduino_tutorial_Xbee_Send*. Son principe est simple, un envoi de message sur une adresse précisée en paramètre (broadcast ou ciblée).

Les autres fonctions *send* sont utilisées pour l'envoi de message à une cible. Elles sont au nombre de trois : *sendToPrevious*, *sendToNext*, et *directSend*.
La fonction *directSend* fait le lien entre les *sendTo* et *sendData*.
Les *sendTo* vont analyser la partie route des données reçues afin de choisir à qui envoyer le message suivant.

4.3 Les fonctions create et complete

Cette dernière composante est très simple mais est essentielle pour la création des messages.
Chaque fonction préfixée par *create* ou *complete* va suivre un déroulement souvent similaire, les changements entre les différentes fonctions sont adaptées à chaque cas.

En voici un exemple avec la fonction de création du RREQ ;

```
// creer un message de type RREQ
void  createRREQ() {
  emptyData();
  data[0] = 'Q';
  data[1] = '0' + msg_id;
  data[2] = dst[0];
  data[3] = dst[1];
  data[4] = src[0];
  data[5] = src[1];
}
```

Figure 8 – Fonction de création du message RREQ

5 Possibilités d'améliorations

Le projet même si complet peut se voir améliorer sur certains points.

La partie configuration est un de ces points, la modification de celle-ci directement dans le code n'est pas idéale d'un point de vue utilisateur. Et d'un point de vue pratique une configuration telle depuis l'Arduino peut être une lourde tâche.
Cependant dans le cas d'une application réelle de ce système de routage, la partie de configuration des voisins devraient disparaître et donc seul le destinataire et le message resterai à configurer ce qui allégerai la tâche de configuration.

Le second point restant possible à changer au besoin est la condition d'envoi du message.
Deux cas se présentent, le premier où il est nécessaire d'envoyer une donnée de façon continue, notre envoi actuel correspond à ce besoin.
Et le deuxième, où l'envoi de données se fait sous condition, par exemple lorsqu'un capteur reçoit une certaine valeur, ou à intervalle régulier. Dans ce dernier, certaines modifications mineures pourraient être apportées au programme afin de correspondre à ce besoin.