Université de Mons
Faculté des sciences
Département d'Informatique
Service de réseaux et télécommunications

Réseau Wi-Fi multi-sauts sur plateforme ESP

Rapport de projet

Directeur :
Bruno Quoitin

Auteur :
Arnaud Palgen

Année académique 2019-2020

Introduction

L'objectif du projet est de concevoir un réseau Wi-Fi multi-sauts composé de micro-contrôleurs Wi-Fi. L'ESP32 d'Espressif sera utilisé en raison de son très faible coût.
Le projet tiendra compte de la consommation énergétique des noeuds du réseau du fait qu'ils sont alimentés par batterie.
Le réseau ainsi créé sera testé pour en évaluer sa performance et ses fonctionnalités. Il sera également étudié pour notamment analyser la durée de vie du réseau sur batterie, analyser la longueur des chemins établis, etc.

Table des matières

Chapitre 1

Etat de l'Art

1.1 Présentation de l'ESP

L'ESP32 est un système embarqué développé par Espressif et dédié à l'internet des objets.
Il est compatible WiFi et Bluetooth 2.4 GHz. Il a une consommation faible en énergie et des mécanismes d'économies d'énergie.
Il est équipé de 34 pins GPIOs, d'un CPU Xtensa ® single-/dual-core 32-bit LX6

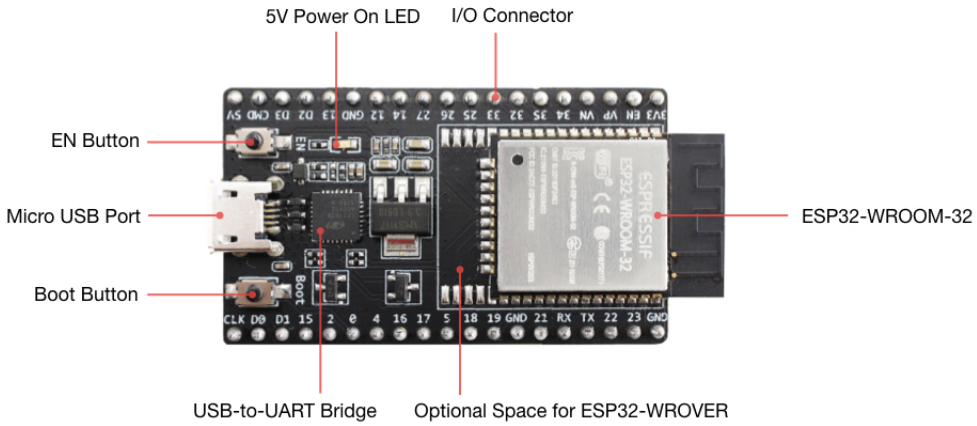

FIGURE 1.1 – ESP32-DevKitC V4 with ESP32-WROOM-32 module [4]

1.2 Choix de l'environnement

Trois environnements s'offrent à nous :

1. **MicroPython**
   Selon le site officiel de MicroPython [3], MicroPython est une implémentation simple et efficace de Python 3 incluant un petit sous-ensemble de la bibliothèque standard Python et est optimisé pour fonctionner sur des microcontrôleurs.
   MicroPython est open source et facile à utiliser. Cependant, il n'est pas assez bas niveau pour ce projet.
   Par exemple il serait impossible d'envoyer des paquets au niveau de la couche lisaison de données ou encore, avoir accès aux tables de routages IP.

2. **IoT Development Framework (IDF)**
   IDF est l'environnement du constucteur de l'ESP32 (Espressif). La documentation est complète mais le code source n'est pas entièrement disponible. Pour certaines parties du framework, nous n'avons accès qu'aux fichiers d'entête.
   Ce framework est natif et nous apportera donc une plus grande fidélité à l'ESP32.
   Cet environnement nous donne aussi accès à des fonctionnalités de FreeRTOS (free real-time operating system).
   FreeRTOS est un système d'exploitation temps réel open source pour microcontrôleurs. Ses fonctionnalités pourront nous être utiles pour le projet.

3. **Arduino**
   L'environnement Arduino se base sur IDF. Il est donc possible que certaines fonctionnalités d'IDF ne soient pas disponible. La documentation est moins complète qu'IDF mais tout le code source est disponible.
   Cet environnement semble assez bas niveau car nous avons accès au driver Ethernet et à la couche IP.

Il est évident que nous ne choisirons pas MicroPython car certaines fonctionnalités utiles pour ce projet ne sont pas accessibles.
Nous choisirons IDF pour sa documentation complète, sa nativité et l'accès aux fonctionnalités de FreeRTOS.

1.3 Protocoles de routage

Dans cette section nous discuterons des différents protocoles de routage envisageables.
Tout d'abord, nous établirons un classement des différents types de protocoles de routage MESH.
Ensuite, nous etudierons et comparerons différents protocoles de routage mesh : ESP-MESH [2], OLSR [9], AODV [5] et DSR [6]
Enfin nous pourrons choisir un protocole à implémenter sur ESP32.

1.3.1 Classification

Les protocoles de routages MESH peuvent être divisés en deux grandes catégories :

1. **Proactifs** : Les noeuds maintiennent une/des table(s) de routage qui stockent les routes vers tous les noeuds du réseau. Ils envoient régulièrement des paquets de contrôle à travers le réseau pour échanger et mettre à jour l'information de leurs voisins.
2. **Réactifs** : Ces protocoles établissent une route uniquement quand des paquets doivent être transférés.

Nous écarterons les protocoles proactifs pour ce projet car ils gardent beaucoup d'information en mémoire. Ils ne passent donc pas à l'échelle.
Les protocoles réactifs sont plus économes en ressources mais nécessitent parfois un délai plus long pour établir une route car elles sont établies à la demande.

Ci-dessous, une brève classifications des protocoles de routages MESH.

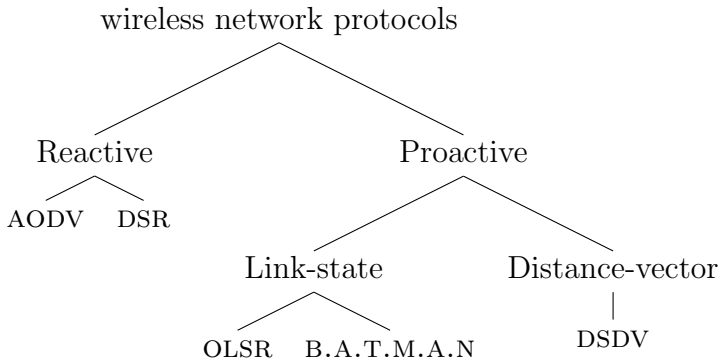

Diagram 1.1 – Classifications des protocols de routage

AODV Ad-hoc On-demand Distance Vector
DSR Dynamic Source Routing
OLSR Optimized Link State Routing
B.T.M.A.N Better Approach to Mobile Adhoc Networking
DSDV Destination Sequence Distance Vector

1.3.2 Etude des différents protocoles

ESP MESH

ESP-MESH est le protocole du constructeur Espressif permettant d'établir un réseau mesh avec des ESP32. Cette section explique le fonctionnement de ce protocole. La topologie utilisée ici est l'arbre. Il existe plusieurs types de noeuds :

1. **Racine** : Seule interface entre le réseau ESP-MESH et un réseau IP externe.
2. **Noeuds intermédiaires** : Noeuds qui ont un parent et au moins un enfant. Ils peuvent transmettre leurs paquets et ceux de leurs enfants.
3. **Feuilles** : noeuds qui n'ont pas d'enfants et ne transmettent que leur paquets.
4. **Noeuds idle** : noeuds qui n'ont pas encore rejoint un réseau ESP-MESH

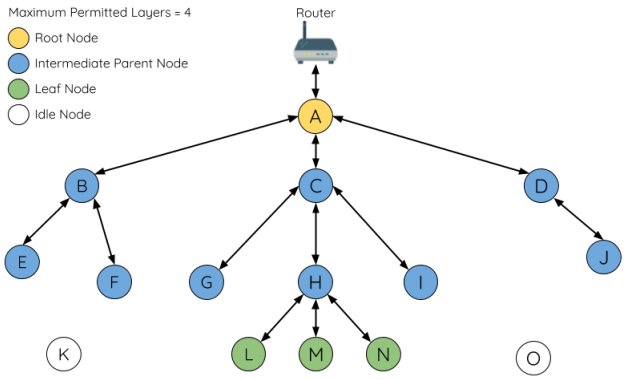

FIGURE 1.2 – Topologie d'un réseau ESP-MESH [2]

Routage

1. Table de routage
   Chaque noeud possède sa table de routage. Soit p un noeud, sa table

de routage contient les adresses MAC des noeuds du sous-arbre ayant p comme racine, et également celle de p.
Elle est partitionnée en sous-tables qui correspondent au sous-arbres des enfants de p.

2. Acheminement de paquets
Quand un paquet est reçu,
   — Si l'adresse MAC du paquet est dans la table de routage et si elle est différente de l'adresse du noeud l'ayant reçu, le paquet est envoyé à l'enfant correspondant à la sous-table contenant l'adresse.
   — Si l'adresse n'est pas dans la table de routage, le paquet est envoyé au parent.

Construction d'un réseau

1. Élection de la racine
   — **Sélection automatique**
   Chaque noeud idle va transmettre son adresse MAC et la valeur de son RSSI (Received Signal Strength Indication) avec le routeur via des beacons.
   Simultanément, chaque noeud scanne les beacons des autres noeuds. Si un noeud en détecte un autre avec un RSSI strictement plus fort, il va transmettre le contenu de ce beacon (càd voter pour ce noeud).
   Ce processus sera répété pendant un nombre minimum d'itérations.
   Après toutes les itérations, chaque noeud va calculer le ratio

$$\frac{\textit{nombre de votes}}{\textit{nombre de noeuds participants à l'élection}}$$

   Si ce ratio est au-dessus d'un certain seuil (par défaut 90%), ce noeud deviendra la racine. [1]
   — **Sélection par l'utilisateur**
   La racine se connecte au routeur et elle, ainsi que les autres noeuds, oublient le processus d'élection.

2. Formation de la deuxième couche
Une fois le processus d'élection d'une racine terminé, chaque noeud va émettre des beacons pour permettre aux autres noeuds de détecter sa présence et de connaître son statut. Ces beacons contiennent les informations suivantes :

1. Si plusieurs racines sont élues, deux réseaux ESP-MESH seront créés. Dans ce cas, ESP-MESH possède un mécanisme interne qui va fusionner les deux réseaux ssi les racines sont connectées au même routeur.

- Type du noeud (racine, intermédiaire, feuille, idle)
- Couche sur laquelle se trouve le noeud
- Nombre de couches maximum autorisées dans le réseau
- Nombre de noeuds enfants
- Nombre maximum d'enfants

Les noeuds idle à portée de la racine vont s'y connecter et devenir des noeuds intermédiaires.

3. Formation des autres couches
   Les noeuds idle à portée de noeuds intermédiaires vont s'y connecter. Si plusieurs parents sont possibles, un noeud choisira son parent selon deux critères connus par les beacons des noeuds intermédiaires.

   1. La couche sur laquelle se situe le candidat parent : le candidat se trouvant sur la couche la moins profonde sera choisi.

   2. Le nombre d'enfants du candidat parent : si plusieurs candidats se trouvent sur la couche la moins profonde, celui avec le moins d'enfants sera choisi.

   Un noeud peut aussi se connecter à un parent prédéfini.

   Une fois connectés, les noeuds deviennent des noeuds intermédiaires si le nombre maximal de couches n'est pas atteint. Sinon, les noeuds de la dernière couche deviennent automatiquement des feuilles, empêchant d'autres noeuds dans l'état idle de s'y connecter.

Pour éviter les boucles, un noeud ne va pas se connecter à un noeud dont l'adresse MAC se trouve dans sa table de routage.

Mise sous tension asynchrone

La structure du réseau peut être affectée par l'ordre dans lequel les noeuds sont mis sous tension. Les noeuds ayant une mise en tension retardée suivront les deux règles suivantes :

1. Si une racine existe déjà, le noeud ne vas pas essayer d'élire une nouvelle racine même si son RSSI avec le routeur est meilleur. Il va rejoindre le réseau comme un noeud idle.
   Si le noeud est la racine désignée, tous les autres noeuds vont rester idle jusqu'à ce que le noeud soit mis sous tension.
2. Si le noeud devient un noeud intermédiraire, il peut devenir le meilleur parent d'un autre noeud (cet autre noeud changera donc de parent).
3. Si un noeud idle a un parent prédéfini et que ce noeud n'est pas sous tension, il ne va pas essayer de se connecter à un autre parent.

Défaillance d'un noeud

— Défaillance de la racine
Si la racine tombe, les noeuds de la deuxième couche vont d'abord tenter de s'y reconnecter. Après plusieurs échecs, les noeuds de la deuxième couche vont entamer entre eux le processus d'élection d'une nouvelle racine.
Si la racine ainsi que plusieurs couches tombent, le processus d'élection sera initialisé sur la couche la plus haute.

— Défaillance d'un noeud intermédiaire
Si un noeud intermédiaire tombe, ses enfants vont d'abord tenter de s'y reconnecter. Après plusieurs échecs, ils se connecteront au meilleur parent disponible.
S'il n'y a aucun parent possible, ils se mettront dans l'état idle.

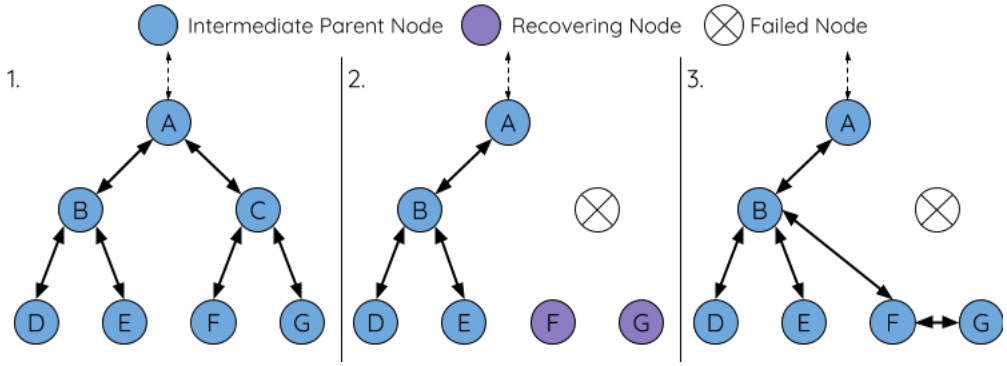

FIGURE 1.3 – Défaillance d'un noeud intermédiaire [2]

Changement de racine

Un changement de racine n'est possible que dans deux situations :

1. La racine tombe. (voir point précédent)
2. La racine le demande. Dans ce cas, un processus d'élection de racine sera initialisé. La nouvelle racine élue enverra alors une *switch request* à la racine actuelle qui répondra par un acquittement. Ensuite la nouvelle racine se déconnectera de son parent et se connectera au routeur. L'ancienne racine se déconnectera du routeur et deviendra un noeud idle pour enfin se connecter à un nouveau parent.

Paquets ESP-MESH

Les paquets ESP-MESH sont contenus dans une trame WiFi. Une transmission multi-sauts utilisera un paquet ESP-MESH transporté entre chaque noeud par

un paquet wifi différent.
La figure 1.4 montre la structure d'un paquet ESP-MESH :

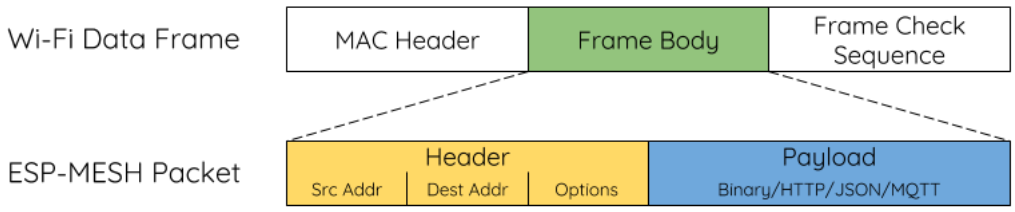

FIGURE 1.4 – Paquet ESP-MESH [2]

Le header d'un paquet ESP-MESH contient les adresses MAC source et destination ainsi que diverses options.
Le payload d'un paquet ESP-MESH contient les données de l'application.

Multicasting
Le multicasting permet d'envoyer simultanément un paquet ESP-MESH à plusieurs noeuds du réseau. Le multicasting peut être réalisé en spécifiant

— Soit un ensemble d'adresses MAC
Dans ce cas, l'adresse de destination doit être `01:00:5E:xx:xx:xx`
Cela signifie que le paquet est un paquet multicast et que la liste des adresses peut être obtenue dans les options du header.
— Soit un groupe préconfiguré de noeuds
Dans ce cas, l'adresse de destination du paquet doit être l'ID [2] du groupe et un flag MESH_DATA_GROUP doit être ajouté.

Broadcasting
Le broadcasting permet de transmettre un paquet ESP-MESH à tous les noeuds du réseau. Pour éviter de gaspiller de la bande passante, ESP-MESH utilise les règles suivantes :

1. Quand un noeud intermédiare reçoit un paquet broadcast de son parent, il va le transmettre à tous ses enfants et en stocker une copie
2. Quand un noeud intermédiaire est la source d'un paquet broadcast, il va le transmettre à son parent et à ses enfants
3. Quand un noeud intermédiaire reçoit un paquet d'un de ses enfants, il va le transmettre à ses autres enfants, son parent et en stocker une

2. Dans un réseauESP-MESH, chaque groupe a un ID unique

copie

4. Quand une feuille est la source d'un paquet broadcast, elle va le transmettre à son parent
5. Quand la racine est la source d'un paquet broadcast, elle va le transmettre à ses enfants
6. Quand la racine reçoit un paquet broadcast de l'un de ses enfants, elle va le transmettre à ses autres enfants et en stocker une copie
7. Quand un noeud reçoit un paquet broadcast avec son adresse MAC comme adresse source, il l'ignore
8. Quand un noeud intermédiaire reçoit un paquet broadcast de son parent, s'il possède une copie de ce paquet (càd que ce paquet a été à l'origine transmis par l'un de ses enfants), il va l'ignorer pour éviter les cycles (protocole d'inondation)

Contrôle de flux

Pour éviter que les parents soient submergés de flux venant de leurs enfants, chaque parent va assigner une fenêtre de réception à chaque enfant. Chaque noeud enfant doit demander une fenêtre de réception avant chaque transmission. La taille de la fenêtre peut être ajustée dynamiquement. Une transmission d'un enfant vers un parent se déroule en plusieurs étapes :

1. Le noeud enfant envoit à son parent une requête de fenêtre. Cette requête contient le numéro de séquence du paquet en attente d'envoi.
2. Le parent reçoit la requête et compare le numéro de séquence avec celui du précédent paquet envoyé par l'enfant. La comparaison est utilisée pour calculer la taille de la fenêtre qui est transmise à l'enfant.
3. L'enfant transmet le paquet en accord avec la taille de fenêtre. Une fois la fenêtre de réception utilisée, l'enfant doit renvoyer une demande de fenêtre.

Performances

Espressif fournit les performances d'ESP-MESH pour un réseau de 100 noeuds avec un nombre maximum de couches de 6 et un nombre d'enfants maximum par noeuds de 6.(voir table 1.1)

Temps de construction du réseau	$<$ 60 secondes
Latence par saut	10 à 30 millisecondes
Temps de réparation du réseau	Si la racine tombe : $<$ 10 secondes Si un noeud enfant tombe : $<$ 5 secondes

TABLE 1.1 – Performances d'ESP-MESH [2]

Discussion
A première vue, une topologie en arbre n'est pas robuste car si la racine tombe, tout le reste du réseau est déconnecté. Cependant le processus d'élection d'une nouvelle racine semble efficace selon les résulats fournis par Espressif. Un point négatif du protocole est que pour un noeud donné, sa table de routage contient tous les noeuds de son sous-arbre. On imagine donc difficilement utiliser ce protocole pour un nombre élevé de noeuds.

OLSR

OLSR est un protocole proactif à état de liens. Il fournit un support à certaines extensions comme sleep mode operation, multicast-routing etc.
Ce protocole utilise deux types de messages :

1. *hello*
   Ces paquets sont envoyés à un saut. Ils sont utilisés pour constituer le voisinage et l'ensemble des *"multipoint relays" (MPR)*. (concept expliqué ici)
   Ils contiennent le statut des liens d'un noeud avec tout son voisinage.
2. *topology control (TC)*
   Diffusés dans tous le réseau par les *MPR*. Ils contiennent la liste de tous les *MPRs* du noeud qui émet ce paquet.

MPR
Les multipoint relays sont des noeuds qui sont chargés d'effectuer le broadcasting des *TCs*.
L'ensemble des *MPRs* d'un réseau *OLSR* forme un arbre couvrant du réseau. Trouver ce sous-ensemble de sommets est un problème *NP-Complet*. *DSR* propose une heuristique simple pour calculer cet ensemble de *MPRs*.

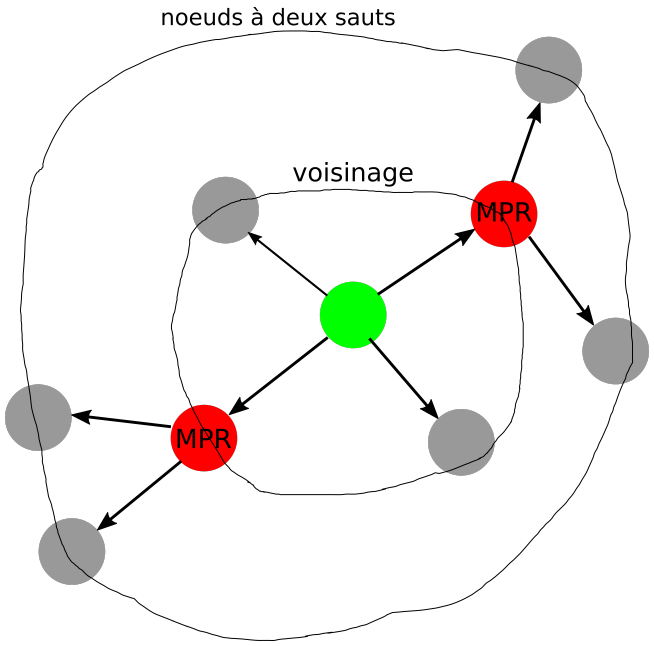

FIGURE 1.5 – Shéma d'un réseau OLSR

Découverte de voisins

La découverte de voisins se fait comme suit :

- Quand un noeud B reçoit un message *hello* d'un noeud A, il va rajouter A dans ses voisins avec le statut asymétrique
- B va ensuite envoyer un message *hello* avec A asymétrique dans la liste de voisins.
- Quand A reçoit ce message, il va pouvoir modifier le statut de B à symétrique.
- A renvoie un message *hello* avec B symétrique dans la liste des voisins.
- Quand B reçoit ce message il modifie le statut de A à symétrique

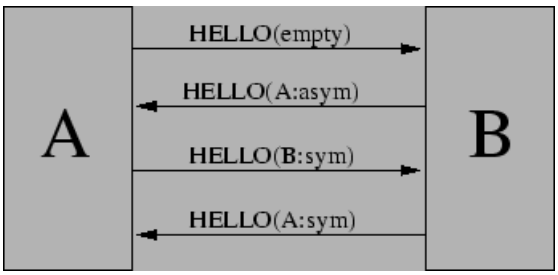

FIGURE 1.6 – découverte de voisins en utilisant les messages *hello* [1]

AODV

Ad-hoc On-demand Distance Vector (AODV) est un protocole réactif à vecteur de distance.
Ce protocole définit 3 types de messages :

1. Route request (*RREQs*)
2. Route Remplies (*RREPs*)
3. Route Errors (*RERRs*)

Format des paquets

RREQ

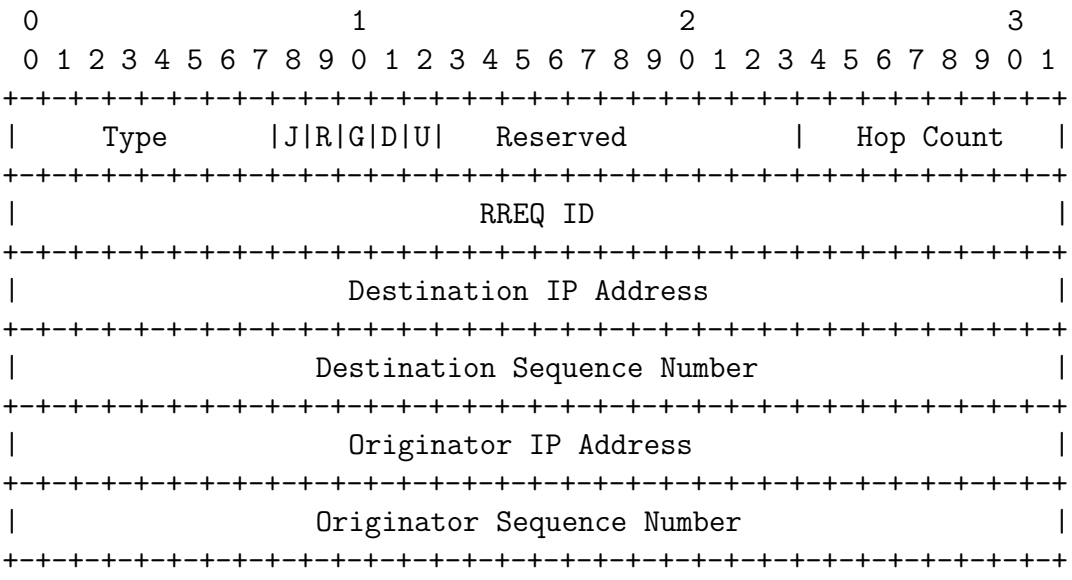

FIGURE 1.7 – format d'un paquet RREQ [5]

Où J,R,G,D,U sont des flags.

RREP

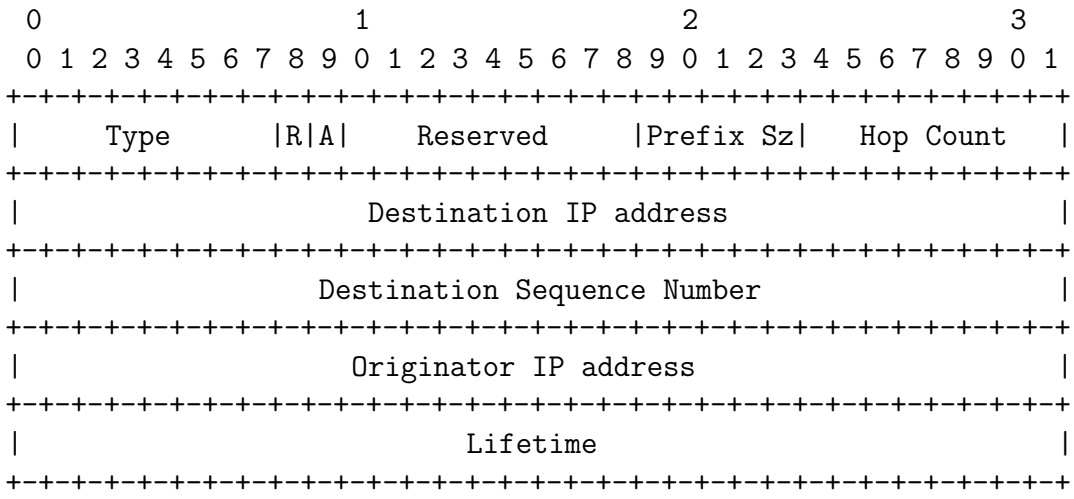

FIGURE 1.8 – format d'un paquet RREP [5]

Où R et A sont des flags.

Découverte d'un chemin

La découverte d'un chemin est intiée par un noeud voulant envoyer des paquets à une destination pour laquelle il n'a aucune information.
Chaque noeud possède deux compteurs : ***sequence_number*** et ***rreq_id***.

Génération du *RREQ*

Le noeud source incrémente ses compteurs ***sequence_number*** et ***rreq_id*** de 1. Il envoie ensuite un *RREQ* en broadcast à ses voisins.

Propagation du *RREQ*

- Noeud intermédiaire
  A la réception d'un *RREQ*, un noeud intermédiaire va pouvoir rajouter ou mettre à jour les routes vers le noeud source du *RREQ* et vers son prédécesseur.
  Ensuite deux situations sont possibles :
  1. Le noeud courant possède une route active vers la destination et le numéro de séquence de la route est plus grand ou égal au numéro de séquence de la destination dans le *RREQ*.
     Dans ce cas, il peut envoyer par unicast un *RREP* à la source du *RREQ*

2. La condition en 1. n'est pas satisfaite.
   Dans ce cas, le noeud va incrémenter le nombre de sauts du *RREQ* et le propager à ses voisins.

- Noeud destination
  A la réception d'un *RREQ* lui étant destiné, un noeud va, comme un noeud intermédiaire, rajouter ou mettre à jour les routes vers le noeud source du *RREQ* et vers son prédécesseur.
  Si le ***Destination Sequence Number*** du *RREQ* est égale à son ***sequence_number***, il va incrémenter ce dernier et envoyer un *RREP* en unicast vers la source du *RREQ*.

Propagation du *RREP*

A la réception d'un *RREP*, un noeud va pouvoir rajouter ou mettre à jour les routes vers le noeud source du *RREP* et vers son successeur.
Il va ensuite incrémenter le nombre de sauts du *RREP* et le propager en unicast vers la destination de ce *RREP*.

Table de routage

Chaque entrée d'une table de routage contient les informations suivantes :

dest	Adresse IP de destination
dest_SN	Numéro de séquence de destination
flag	Indicateur de numéro de séquence de destination valide
out	Interface réseau
hops	Comptage de sauts (nombre de sauts nécessaires pour atteindre la destination)
next-hop	Prochain saut
precursors	Liste des précurseurs
lifetime	temps d'expiration ou de suppression de l'itinéraire

TABLE 1.2 – entrée d'une table de routage AODV [5]

Mise à jour de la table de routage

Soit N une nouvelle route et O la route existante.
O est mise à jour si :

$$O.SN \leq N.SN$$
$$\textbf{ou } (O.SN = N.SN \textbf{ et } O.hop\_count > N.hop\_count)$$

Gestion du *lifetime*

Le temps de vie d'une route dans la table de routage est initialisé à *active_route_timeout* (3 millisecondes).
Quand ce timer expire, la route passe de active à inactive. Une route inactive ne pourra plus être utilisée pour transférer des données mais pourra fournir des informations pour de futurs *RREQ* et la réparation de routes.
Quand une route est utilisée, son temps de vie est actualisé à : $currenttime + active\_route\_timeout$

Evitement de boucles

A priori les numéros de séquences suffisent pour éviter les boucles. Cependant, d'après [8], il y a des ambiguités dans le RFC [5]. Dû à ces ambiguités, l'implémentation pourrait introduire des boucles. Nous approfondirons la lecture de cet article si nous choisissons ce protocole afin d'éviter les boucles dans notre implémentation.

Défaillance d'un lien

Un noeud faisant partie d'une route active broadcast des messages *hello* (RREP) régulièrement.
Si un noeud ne reçoit pas de message durant un certain temps pour un voisin, il va considérer que le lien avec ce voisin est perdu.
Dans ce cas, il va en informé ses voisins impactés par un RERR.

DSR

Comme *AODV*, *DSR* est un protocole réactif à vecteur de distance.
Deux mécanismes principaux caractérisent *DSR*

1. **Route Discovery**
   Similaire à la découverte de chemins dans AODV. La différence est que le *RREQ* va contenir l'id de tous les sauts de chemin.
   Quand la cible d'un *RREQ* le reçoit, elle va répondre avec un *RREP* contenant le chemin du *RREQ* inversé.
   Les paquets de données contiennent le chemin complet par lequel ils doivent transiter.
   On a donc des paquets d'une taille variant avec celle du chemin emprunté.
2. **Route Maintenance**
   Chaque noeud est responsable des liens entre lui et ses successeurs et possède une cache de routes. Pour cela, quand un noeud reçoit un paquet, il va transmettre à la source de ce paquet un *ack*. Si le *ack* n'est

pas reçu au bout d'un temps fixé, l'émetteur du paquet va émettre un paquet *route_error* à ses prédécesseurs pour leur signaler que le lien n'existe plus.
Si des routes sont interrompues, les noeuds peuvent soit utiliser d'autres routes de leur cache, soit initier une *route discovery*

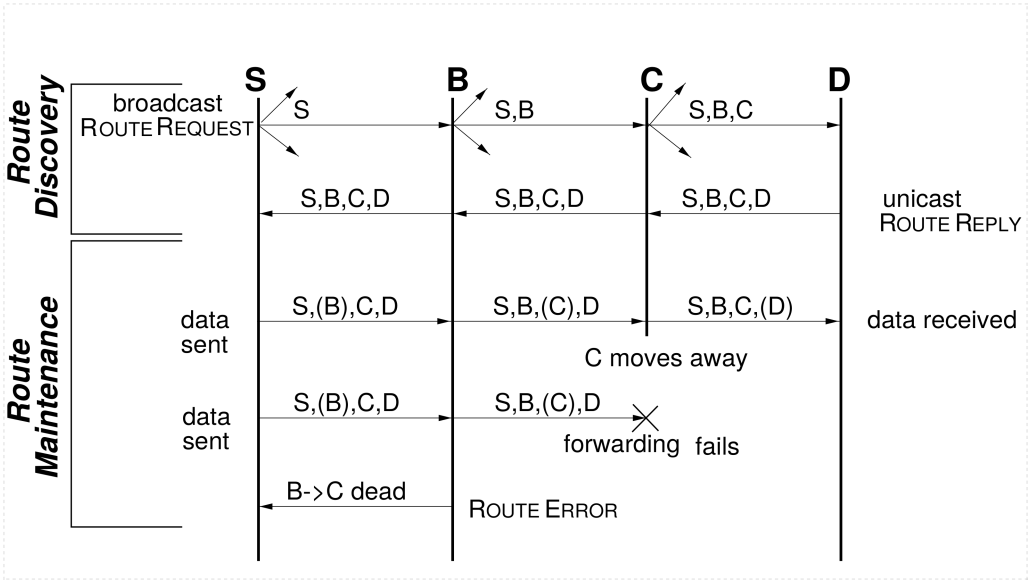

FIGURE 1.9 – Illustration de DSR [7]

Chapitre 2

Implémentation

2.1 Limitations

Le driver Wi-Fi d'*IDF* ne nous permet pas d'avoir une connexion avec plusieurs noeuds simultanément. ESP NOW pourraît être une solution pour palier à ce problème.

ESP NOW

ESP NOW est un protocole de communication Wi-Fi sans connexion défini par Espressif.
Nous n'avons trouvé aucune documentation décrivant le fonctionnement de ce protocole.
Avec la documentation disponible, nous savons qu'un noeud a une liste de *peers* (ses voisins) avec qui il peut échanger des données. Le nombre de voisins est limité à 20. Cette limite ne posera pas problème pour ce projet.

2.2 Prochaines étapes

1. Nous allons créer un réseau ESP-MESH. Ceci nous permettra de nous familiariser avec l'environnement *IDF*.
2. Nous évaluerons les performances et fonctionnalités de ce protocole.
3. Nous implémenterons un protocole de routage mesh choisis plus haut.
4. L'objectif est de l'implémenter au niveau de la couche liaison de données. Si nous rencontrons trop de difficulté ou que nous jugeons que ce choix n'est pas judicieux, nous implémenterons ce protocole au niveau de la couche réseau.
5. Nous étudierons les différentes possibilités d'économies d'énergie

6. Nous évaluerons les performances et fonctionnalités du prorotype créé.

Bibliographie

[1] OLSR neighbor discovery. http://www.olsr.org/docs/report_html/node34.html, 2004. [Accès en ligne le 24 décembre 2019].

[2] ESP-MESH api guide. https://docs.espressif.com/projects/esp-idf/en/latest/api-guides/mesh.html/, 2018. [Accès en ligne le 11 décembre 2019].

[3] MicroPython. https://micropython.org/, 2018. [Accès en ligne le 11 décembre 2019].

[4] ESP32-DevKitC V4 Getting Started Guide. https://docs.espressif.com/projects/esp-idf/en/latest/hw-reference/get-started-devkitc.html, 2019. [Accès en ligne le 24 décembre 2019].

[5] S. Das C. Perkins, E. Belding-Royer. RFC 3561. https://tools.ietf.org/html/rfc3561#section-6.2, 2003. [Accès en ligne le 22 décembre 2019].

[6] D. Maltz D. Johnson, Y. Hu. RFC4728. https://tools.ietf.org/html/rfc4728, 2007. [Accès en ligne le 22 décembre 2019].

[7] David Maltz. The dynamic source routing protocol for multi-hop ad hoc networks. 12 1999.

[8] Wee Lum Tan et Marius Portmann Rob van Glabbeek, Peter Höfner. Sequence Numbers Do Not Guarantee Loop Freedom —AODV Can Yield Routing Loops—. 2013. [Accès en ligne le 23 décembre 2019].

[9] P. Jacquet T. Clausen. RFC3626. https://tools.ietf.org/html/rfc3626, 2003. [Accès en ligne le 22 décembre 2019].