ESIREM Dijon

Projet 4A

Réalisation d'une API de configuration automatique de réseau WiFi maillé sous Linux

Thibaut Bugnot
Promo Neumann

Tuteur : M. Axel Moinet

Table des matières

Table des figures

Introduction

Chapitre 1

Objectifs, définitions, contraintes

1.1 Introduction aux réseaux wifi

Le wifi, abréviation de wireless fidelity, est un ensemble de protocoles permettant la communication sans fil entre deux appareils en utilisant des ondes radios. Ces protocoles se situent au niveau de la couche d'accés du modèle tcp/ip. La standardisation de cette norme a été initiée l'IEEE [1] en 1990. Cela a aboutit, en 1997, au standart IEEE 802.11 définissant les réseaux locaux sans fils [1]. La norme d'origine prévoyait l'utilisation d'ondes radios dans la bande de fréquences libre entre 2401 et 2495 MHz, couramment appelée bande à 2,4 GHz, ou d'infra rouges. Cependant, pour suivre l'évolution des technologies, le standart IEEE 802.11 s'est enrichi afin d'augmenter le débit et d'utiliser la bande de fréqences libre entre 5170 et 5710 MHz. Les standarts IEEE 802.11a et IEEE 802.11b ont donc été définis en 1999, le standart 802.11g et 2003 et le standart 802.11n en 2009.

Depuis sa création, la norme IEEE 802.11 définit 14 canaux dans la bande 2,4 GHz. Chaque canal à une largeur de 22 MHz et l'écart entre les centres de deux canaux successifs est de 5 MHz [2]. Il en résulte donc un fort recouvrement entre les différents canaux comme le montre la figure 1.1.

Un réseau wifi est un réseau local découpé en "cellules" appelées BSS [3]. Deux appareils doivent se trouver dans le même BSS pour communiquer entre eux.

1. Institute of Electrical and Electronics Engineers
2. Sauf les centres des cannaux 13 et 14 qui sont espacés de 12 MHz
3. Basic Service Set

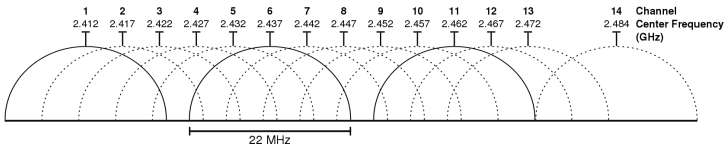

FIGURE 1.1 – Répartition des canaux dans la bande 2,4 GHz

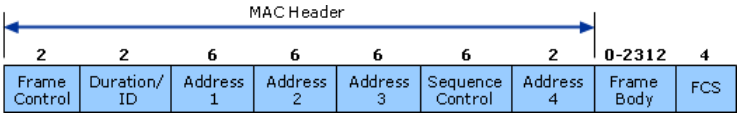

FIGURE 1.2 – Format des trames 802.11

Il existe deux modes de BSS : Le mode Infrastructure et le mode ad-hoc[2]. La plupart des réseaux wifi de particuliers ou d'entreprises sont des réseaux en mode Infrastructure.

Le mode infrastructure est une topologie centralisée. Il se caractérise par le fait que chaque BSS posséde une station de base, appelée aussi point d'accés, et que toutes les communications passent nécessairement par le point d'accés de la BSS, et ce même si l'émetteur et le récepteur du message se trouvent dans le même BSS. Un point d'accés peut être relié par un réseau cablé à un ou plusieurs autres points d'accés, étendant ainsi le LAN [4] [3],ou à un routeur pour accéder à un réseau WAN [5]. Le mode ad-hoc, au contraire, est un mode "d'égal à égal". Deux entités au sein du même BSS peuvent communiquer directement.

Comme le montre la figure 1.2[4], le premier champ de l'en-tête wifi est le FCF [6], permettant d'identifier les trames en fonction de leur rôle. Ainsi, les trames peuvent être de trois types, identifiées par les deux bits en position 3 et 4 du FCF : Management, Contrôle ou Donnée.[5]. Les 4 bits suivants identifient le sous type, et les 8 derniers bits sont des flags. Les trames de données sont utilisées pour transporter des données de plus haut niveau, les trames de contrôles sont utilisées pour les acquittements et les réservations, et les trames de management servent à organiser et maintenir le réseau[6].

Les Beacons frames sont des trames de management particulières qui permettent à un point d'accés de déclarer sa présence aux appareils à proximité. Ils transportent différentes informations comme le SSID [7] du réseau, qui est une chaîne de 2 à 32 caractères, un timestamp permettant de se synchroniser, le canal sur lequel il émet, et d'autres informations.[2].

1.2 La norme 802.11s

Comme dit précédemment, le mode infrastructure est actuellement le plus utilisé. Cependant, une de ces limites est que, dans certaines situations, il n'est pas toujours possible de connecter un point d'accés à un switch[7]. En effet, la longueur des cables ethernet est limitée, ce qui rend difficile le déploiement de points d'accés dans des environnements ouverts.

C'est ce qui fait la force du mode ad-hoc. Chaque appareil peut communiquer avec tous les autres appareils qui sont à portée. De plus, chaque appareil peut relayer le message si le destinataire final n'est pas à portée. Ainsi, dans l'exemple de la figure 1.3, chaque noeud peut communiquer avec nimporte quel autre, à la condition qu'un alogrithme de routage s'exécute sur le réseau et que chaque noeud sache quel est le suivant pour atteindre la destination. Ce genre de réseau

4. Local Area Network, ou Réseau local
5. Wide Area Network, ou Réseau étendu
6. Frame Control Field, ou Champ de contrôle de trame
7. Service Set IDentifier

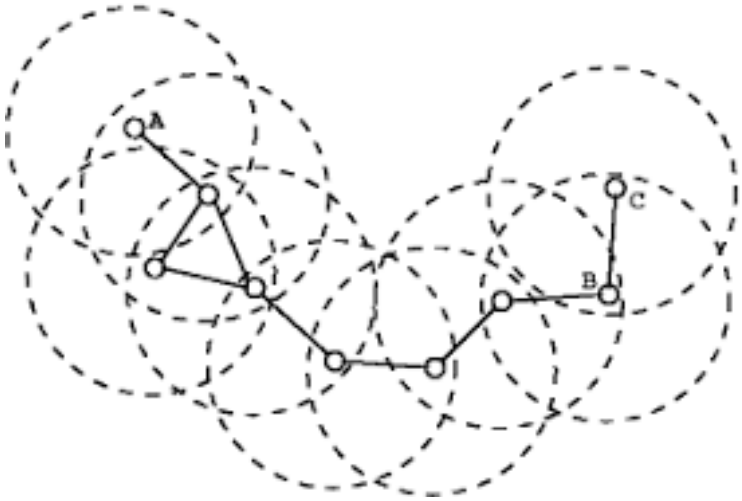

FIGURE 1.3 – Exemple de réseau ad-hoc

est appelé réseau maillé [8]. Le principal avantage de ces réseaux est qu'ils sont trés flexibles. On peut les étendre sans avoir à tirer de nouveaux cables ou à ajouter de nouveaux équipements intermédiaires[7]. De plus, le retrait d'un petit nombre de noeuds ne doit pas empêcher le réseau de fonctionner s'il est possible de trouver des routes alternatives pour les trames.

Le standart 802.11s est un amendement de la norme 802.11, définissant la manière dont les appareils disposant d'une carte réseau sans fils peuvent s'interconnecter pour former un réseau sans fil maillé. L'IEEE a commencé à travailler sur ce standart en 2003 et celui-ci a été adopté en 2006. Pour faciliter l'interopérabilité, un réseau 802.11s est vu de l'extérieur comme un unique segment ethernet. Pour permettre la retransmission des informations d'un noeud à l'autre, la norme 802.11s étend l'en-tête 802.11 classique avec un en-tête mesh comme montré dans la figure 1.4[6].

Les 4 champs d'adresses de l'en-tête 802.11 sont utilisés, puisqu'il faut à chaque transmission du message donner l'adresse du noeud qui a effectué la transmission, celle du prochain noeud, celle du destinataire final et celle de l'expéditeur originel. Dans certains cas plus complexes, il est nécessaire d'ajouter des adresses supplémentaires, ce qui explique que l'en-tête mesh comporte un champ optionel d'extention d'adresses. C'est le cas par exemple si l'émetteur et/ou le destinataire, ne se trouvent pas dans le réseau mesh, mais que la trame va traverser un réseau mesh. Parmi les autres valeurs ajoutées, le TTL [9] et le Mesh sequence number [10] permettent d'éviter les boucles infinies qui risqueraient de saturer le réseau.

8. Mesh Network en anglais
9. Nombre de fois maximal que peut être relayée une trame avant d'être abandonnée, cette valeur est décrémentée à chaque saut
10. nombre identifiant de manière unique une trame

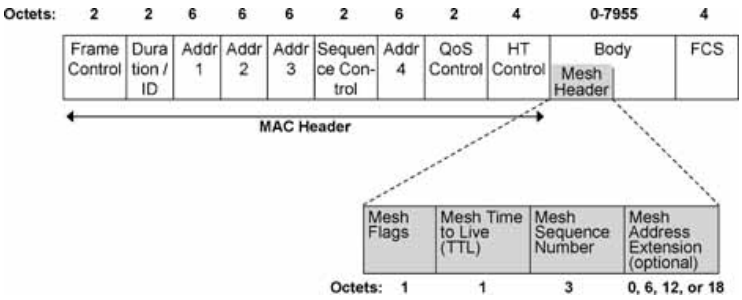

FIGURE 1.4 – Format des trames 802.11s

1.3 Adressage et routage

Dans un réseau TCP/IP, chaque noeud doit disposer de deux adresses. Chacune d'elles permet de l'identifier, en théorie, de manière unique. La première est l'adresse MAC, une adresse sur 48 bits, utilisée pour identifier les noeuds dans les protocoles de la couche d'accès du modèle TCP/IP. Cette adresse permet à une trame de voyager sur un LAN jusqu'à sa destination, mais sera changée à chaque fois que le paquet passe par un routeur. La deuxième est l'adresse IP, une adresse sur 32 bits qui est utlisée par le protocole IP, qui est un protocole de la couche réseau du modèle TCP/IP. Cette adresse est inchangée d'un bout de la transmission à l'autre [11].

L'adresse MAC est atribuée à une carte réseau par le constructeur. Ainsi, nous avons la garantie que chaque appareil possède une adresse MAC unique. L'adresse IP doit également être unique mais, contrairement à l'adresse MAC, elle n'est pas enregistrée dans la carte réseau par le constructeur car toutes les adresses IP identifiant les appareils d'un même LAN doivent avoir le même préfixe. Il existe des protocoles permettant d'affecter automatiquement des adresses IP à des appareils sans avoir besoin de recourir à une intervention humaine. Le protocole majoritairement utilisé est DHCP [12]. Ce protocole nécessite qu'un serveur dispose d'une liste d'adresses IP disponibles qu'il va affecter à chaque noeuds du réseau sur demande de ces derniers[8]. Néanmoins, le recours à un serveur central d'adresses IP amoindrit les avantages à l'utilisation d'une infrastructure décentralisée tel qu'un un réseau maillé.

Dans un réseau maillé, il est aussi nécessaire de prévoire le routage des trames. La norme 802.11s définit également le protocole HWMP [13] comme protocole de routage pour les réseaux wifi maillés. Contrairement à la majorité des protocoles de routages, HWMP ne se base pas sur les adresses IP, mais sur les adresses MAC, puisque le but est d'aiguiller les trames au sein d'un même LAN. Il s'agit d'un protocole de routage à vecteur distance puisque les noeuds n'ont pas conaissance de l'intégralité de la topologie du réseau mais uniquement des noeuds qui le constituent et de la "distance" de chacun d'eux [9].

11. en l'absence de mécanismes de traduction d'adresse (NAT)
12. Dynamic Host Configuration Protocol
13. Hybrid Wireless Mesh Protocol

Chapitre 2

Réseaux sans fils sous Linux

2.1 Gestion du réseau sous Linux

Une importante partie du projet consiste à pouvoir communiquer avec les interfaces sans fils dont disposent les appareils. En effet, il est nécessaire d'une part, de récupérer des informations à propos de ces dernières et d'autre part, de les configurer pour arriver à les utiliser de la manière que nous le souhaitons. Pour gérer les interfaces sans fils, il est important de bien distinguer deux niveaux : Les cartes réseaux [1] , qui existent physiquement, et les interfaces qui pourraient être qualifiées de "logiques", qui utilisent ces cartes réseaux et sont utilisées pour envoyer et recevoir des données. Un type (Point d'acces, client, noeud mesh, monitor, ...) est associé à chaque interface, determinant son mode de fonctionnement. Une seule carte réseau peut être utilisée par plusieurs interfaces.

Historiquement, sous linux, la communication avec les interfaces se faisait avec des appels systémes ioctl [2]. Des outils permettant de manipuler les interfaces en utilisant cette méthode sont depuis longtemps fournis avec les distributions linux. C'est le cas par exemple, du package net-tools, incluant le programme ifconfig, et permettant de manipuler les interfaces (état, informations d'adressages) ou de wireless-tools, incluant le programme iwconfig, permettant de manipuler plus précisément les interfaces sans fils.

Cependant, depuis 2007, il existe un autre moyen de manipuler les interfaces. En effet, se développe **netlink**, une famille de socket ayant pour but de faire communiquer les processus entre eux. Cela permet, entre autre, de faire communiquer un processus utilisateur avec un processus du noyeau linux. La librairie libnl implèmente les pré-requis fondamentaux pour utiliser le protocole netlink. Cependant, celle-ci se veut minimaliste. C'est pourquoi elle est completée par 3 API : libnl-route, libnl-genl et libnl-nf[10]. Des outils de configuration d'interfaces utilisent ces nouveaux moyens. C'est le cas de la suite d'outils iproute pour contrôler les interfaces et de iw pour contrôler plus précisément les interfaces sans fils. Nous utiliserons netlink dans notre code.

L'API que nous utiliserons principalement pour gérer les interfaces est libnl-

1. généralement abrégées **wiphy**, pour WIreless PHYsical device
2. Abréviation de Input Output ConTroL, il s'agit d'une fonction permettant de manipuler des fichiers spéciaux

genl. Celle-ci permet, grâce à nl80211, un en-tête netlink, d'envoyer des messages permettant de contrôler les interfaces wifi. Les commandes suceptibles d'être envoyées sont définies dans l'énumération nl80211_commands dans le fichier nl80211.h. En créant des messages netlink avec ces commandes, il nous est possible, entre autre, de récupérer les informations sur les cartes réseaux connectées (NL80211_CMD_GET_WIPHY), sur les différentes interfaces (NL80211_CMD_GET_INTERFACE), de choisir la fréquence d'émission et de réception de la carte réseau (NL80211_CMD_SET_WIPHY), de créer une interface sur une carte réseau (NL80211_CMD_NEW_INTERFACE) ou de changer le type d'une interface (NL80211_CMD_SET_INTERFACE)

Les commandes transmisent dans un message netlink peuvent avoir besoin d'un ou plusieurs attributs. La liste de tous les attributs existants dans le cadre de la gestion des interfaces sans fils est définit dans l'énumération nl80211_attrs dans le fichier nl80211.h.

L'API libnl-genl dispose déjà de fonction permettant de créer des socket netlink, des messages netlink, d'y ajouter des attributs et de l'envoyer sur le socket. Selon la commande donnée, le noyeau peut répondre en renvoyant un ou plusieurs messages netlink contenant des informations. C'est le cas, par exemple, lors de l'utilisation de la commande NL80211_CMD_GET_INTERFACE. Le message renvoyé contiendra alors les informations sur les interfaces. Il est alors nécessaire d'analyser le message pour en extraire les informations souhaitées. Dans tous les cas, le dernier message renvoyé par le noyeau sera un message d'acquitement ou, en cas de problème, un message d'erreur. Aprés avoir envoyé un message netlink, il est donc nécessaire d'écouter sur le socket jusqu'à la réception d'un de ces deux messages.

2.2 Création de réseaux 802.11s

Comme précisé précédemment, un réseau wifi mesh est un réseau ad hoc. En créer un ne nécessite aucune infrastructure particulière. Il suffit qu'au moins deux noeuds se détectent mutuellement comme faisant parti du même réseau. Le noyeau linux contient déjà les composantes logiciel nécessaires à la création et au maintient d'un tel réseau. Il faut que l'utilisateur paramétre deux choses : le nom du réseau (MeshID) et la fréquence[11]. Ainsi, en l'absence de sécurités, deux appareils suffisamment proches disposant chacun d'une interface réglée en mode mesh sur la même fréquence et avec la même meshID pourront communiquer via un réseau mesh qui se sera créé. Néamoins, ce genre de réseau mesh est "public" dans le sens ou n'importe quel appareil peut le rejoindre. De plus, les trames sont envoyées sans être chiffrées et tous les noeuds intermédiaires peuvent les voir et éventuellement les modifier. Il existe des moyens de sécuriser le réseau, mais ils ne seront pas abordés ici.

Du point de vue utilisateur, créer ou rejoindre un réseau mesh nécessite donc 3 actions : régler une interface sans fil en mode mesh point, régler sa fréquence de fonctionnement et lui donner un MeshID. Ces trois fonctionnalités doivent donc faire partie de celles que nous allons coder dans notre projet. L'API libnl-genl nous offre un moyen, via les messages netlink envoyés au noyeau, de les réaliser. En effet, un message netlink contenant la commande NL80211_CMD_SET_INTERFACE permet de changer le type d'une interface ou de lui donner un MeshID. L'utilisation de la commande NL80211_CMD_SET_WIPHY permet

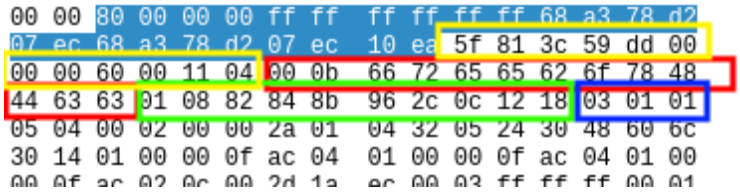

FIGURE 2.1 – Un exemple de beacon frame

quant à elle de changer la fréquence de fonctionnement de la carte réseau, et donc de choisir le canal d'émission et de réception.

2.3 Détection de réseaux existants et sélection de canal

Un des objectifs du projet consiste à détecter les réseaux wifi mesh déja existant et, s'il n'en existe aucun, à choisir le meilleur canal, c'est à dire le moins encombré, pour en créer un nouveau. Pour cela, il est nécessaire d'être en mesure de scanner les réseaux déjà existants. A cette fin, les beacons frames décrites chapitre 1, partie 1 sont particulièrement utiles. En effet, elles sont envoyées à intervales réguliers par les points d'accès, mais aussi par tous les noeuds d'un réseau mesh. Ces trames contiennent en effet de nombreuses informations à propos du réseau. Ces informations n'étant pas toujours les mêmes en fonction du contexte, elles se présentent, pour la plupart, sous la forme tag-longueur-valeurs. Effectivement, une fois les différents en-têtes enlevés et les informations fixes (dont la longueur est connue) écartées, le premier octet est un tag, c'est à dire un nombre identifant l'information qui suit, le deuxiéme octet nous donne la longueur de l'information, notons la n, et les n octets suivants sont l'information. Le prochain octet est à nouveau un tag, suivi d'une longueur, ect.

Prenons comme exemple la figure 2.1, qui est un exemple de beacon frame obtenu avec wireshark. Les octets surlignés et ceux qui les précédent constituent l'en-tête et ne nous intéressent pas. Les octets encadrés en jaune sont des informations "fixes", dans le sens où elles sont toujours présentes dans une beacon frame. Il s'agit d'un time stamp, du beacon interval [3] et quelques flags. Le reste du message illustre le fonctionnement décrit plus haut. Ainsi, le premier octet dans le cadre rouge est 00, ce qui indique que l'information qui va suivre est le SSID [4]. L'octet suivant est 0b [5] indiquant que le SSID est donné sur 11 octets. Les 11 octets suivants donnent le SSID au format ASCII. Le tag dans le cadre vert est 01, indiquant que l'information à suivre est la liste des débits supportés, et la longueur de ce champ est de 8 octets. Les 8 octets suivants indiquent donc chacun un débit. Dernier exemple, dans le cadre bleu, le tag est 03, indiquant que l'information à venir est le canal utilisé, la longueur de cette information est de 01 octet et l'information est 01. Ce point d'accés fonctionne donc sur le canal 1, à la fréquence 1412 [6].

3. L'intervale de temps entre l'envoi de deux beacon frames
4. nom du réseau wifi
5. Il s'agit d'une valeur en hexadécimal valant 11 en décimal
6. voir figure 1.1

Encore une fois, l'API libnl-genl nous fournit les outils pour demander au noyeau linux de faire un scan, via les commandes NL80211_CMD_GET_SCAN et NL80211_CMD_TRIGGER_SCAN à envoyer dans un message netlink. Cela nous permet de récupérer toutes les informations sur les différents réseaux contenues dans les beacon frames. Cependant, une phase de tests a montré empiriquement que ce scan ne détectait pas toujours tous les réseaux wifi mesh. Nous avons donc pris la décision, dans le but d'avoir les résultats les plus complets possibles, d'implémenter notre propre scanner pour exploiter nous-mêmes les beacon frames.

TCPdump est un programme généralement inclus dans les distributions linux permettant de "sniffer" des packets sur une interface réseau. Ce programme se base sur libpcap[12], une librairie permettant de capturer des paquets. C'est cette librairie que nous utiliserons. Elle permet de récupérer les messages captés par l'interface. Il est ensuite possible d'effectuer toutes opérations souhaités sur eux.

Pour être en mesure de capter les beacon frames, il est nécessaire de régler une interface en mode monitor. De plus, puisqu'il existe 14 canaux, il est essentiel de passer de l'un à l'autre pour tous les scanner. Ces deux opérations sont possibles avec la librairie libnl décrite chapitre 2, partie 1. La méthode que nous utiliserons est donc la suivante : nous paramétrons la carte réseau associée à une interface en mode monitor sur une fréquence, puis nous utilisons les fonctions de libpcap pour écouter sur l'interface pendant 3 secondes. Pendant ce temps, à chaque réception d'une trame, nous vérifions qu'il s'agisse d'une beacon frame en analysant son champ type. Dans ce cas, nous analysons chaque information qu'elle contient en recherchant les tags 0, 3, 113 et 114 indiquant respectivement le SSID, le canal de fonctionnement, la configuration mesh et le MeshId. Bien sûr, seuls les beacon frames indiquant un réseau mesh possédent les deux derniers tags. En outre, le tag 00 est toujours présent mais dans le cas d'une beacon frame indiquant un réseau mesh, sa longueur est de 0 bits, dû au fait qu'un réseau mesh n'est pas identifié par un SSID.

Une fois le scan répété sur chacun des 14 canaux, nous avons donc deux listes : Une liste des réseaux "normaux" et une liste des réseaux mesh. De plus, nous pouvons calculer simplement l'occupation de chaque fréquence, et si besoin trouver la moins encombrée. Cependant, comme évoqué précédemment, et comme le montre bien la figure 1.1, les canaux se superposent. Il n'est pas rare, en écoutant sur un canal, de recevoir un message envoyé sur un autre canal voisin. Ainsi, lors de la sélection de fréquence, si un canal est déjà trés encombré, il peut être judicieux d'exclure non seulement le dit canal, mais aussi ses voisins.

Chapitre 3

Adressage et routage

3.1 Adressage dans un réseau maillé

Dans la partie 3 du chapitre 1, nous avons évoqué l'utilité de donner aux interfaces des adresses IP. Sans elles, il est impossible de leur faire transmettre des paquets de données. Nous évoquions également le protocole DHCP, qui est le protocole le plus couramment utilisé pour affecter dynamiquement des adresses IP aux machines qui constituent le réseau. Puisque celui-ci nécessite un serveur dédié, il n'est pas adapté à une architecture décentralisée comme un réseau wifi mesh.

Cependant, il existe de nombreuses solutions plus adaptées à nos besoins. Ces solutions sont regroupées sous l'appelation zeroconf [1]. Elles permettent de créer un réseau IP sans avoir recours à un serveur. Les principales fonctionnalités de ces solutions sont l'affectation d'adresses IP sans serveur DHCP et la résolution de nom de domaine sans serveur DNS. Seule la premiére nous intéresse dans le cadre de ce projet.

La solution qui a été retenue est "IPv4 link-local autoconfiguration specification" [2] définie par l'IETF[13]. IPv4LL permet aux clients de choisir une adresse IP dans la plage d'adresse 169.254.0.0/16. Cette plage est réservée auprès de l'IANA [3] pour l'adressage local. Le principe de IPv4LL est de choisir une adresse IP aléatoirement dans la plage réservée, puis d'envoyer une requette ARP sur cette même adresse pour savoir si elle est déja utilisée par un autre noeud. En l'absence de réponse, l'adresse IP peut être utilisée. Sinon, il faut en choisir une autre et envoyer à nouveau une requette ARP.

Avahi est un programme fourni avec la plupart des distributions linux permettant, entre autre, d'affecter des adresses IP aux interfaces en utilisant IPv4LL. Il peut être lancé sous la forme d'un "deamon" [4], en lui précisant l'interface à laquelle il doit donner une adresse IP. Durant son execution, il affecte une adresse ip à l'interface en vérifiant que celle-ci est libre, puis répond aux requettes si une autre instance de avahi tournant sur une autre machine demande à utiliser la même adresse. Nous avons donc choisi, dans le cadre de ce projet, de lancer avahi, via un appel à la fonction system, afin d'affecter des adresses IP aux in-

1. Ou Zero-configuration networking
2. Abrégé en IPv4LL
3. Internet Assigned Numbers Authority
4. Un processus ou ensemble de processus s'executant en arrière plan

terfaces. Au besoin, l'option -k permet d'arrêter les instances d'avahi tournant sur une interface.

3.2 Routage

Dans le chapitre 1, partie 3, nous précisions aussi qu'il faut permettre de relayer les trames dans un réseau maillé. La norme 802.11s définit un protocole de routage, le protocole HWMP. Celui-ci est un protocole à vecteur distance, ce qui signifie que les noeuds n'ont pas conaissance de l'intégralité de la topologie. La "table des chemins [5]" contient, pour chaque destination, deux informations essentielles : Le prochain saut et la distance de la destination.

Plus précisément, le protocole HWMP dispose de deux modes de fonctionnement : "on demand" et "proactive tree building". Le second nécessite qu'un noeud soit désigné comme noeud racine[9].

Avec le mode "on demand", chaque fois qu'un noeud a besoin de connaître le chemin vers un autre noeud, il envoie un paquet "route request (RREQ)" en broadcast, en identifiant le noeud de destination. Le paquet RREQ contient aussi un champ métrique initialisé à 0. Chaque noeud intermédiaire va recevoir le paquet RREQ, éventuellemenent en plusieurs exemplaires. Si le paquet RREQ a une métrique plus faible que celle déjà connue, le noeud intermédiaire met à jour sa table de routage et le retransmet aprés avoir augmenté la métrique. Lorsque le paquet atteint le noeud de destination, ce dernier répond avec un paquet "Route Reply (RREP)" en unicast vers la source. Ainsi, tous les noeuds entre la destination et la source connaissent une route vers ces deux points.

Cet algorithme est contenu et executé par le noyeau linux. Cependant, il n'est pas le seul à exister pour trouver des chemins dans les réseaux maillés. D'autres algorithmes propriétaires lui ont précédé. C'est pourquoi nous avons programmé la possibilité d'ajouter ou de supprimer des entrées dans la table des chemins, en précisant la destination et le prochain saut. De plus, il est important de pouvoir récupérer cette table des chemins car elle contient les adresses MAC de toutes les machines connues du réseau. C'est donc un bon moyen d'avoir un aperçu de tous les noeuds qui le constituent.

5. Puisqu'on se situe au niveau de la couche d'accés du modèle TCP/IP, on parle plus souvent de chemins (path) que de routes (route)

Chapitre 4

Implémentation et architecture logicielle

4.1 Architecture globale

Le but de ce projet est d'écrire en C un framework permettant de créer ou rejoindre facilement des réseaux mesh. Nous avons donc décider de plusieurs régles facilitant l'exploitation du code.

Les fichiers .c contenant le code sont situés dans le dossier src. Pour chaque fichier .c, il existe un fichier .h de même nom dans le dossier include. Les prototypes des fonctions sont définis dans ce fichier .h. De plus, au dessus de chaque prototype, des commentaires au format doxygen renseignent sur l'utilité et le fonctionnement de chaque fonction, détaillant les paramétres à donner et la valeur retournée par celle-ci.

Les fichiers interface.c, interface.h, network.c, network.h, mpath.c et mpath.h ont pour but de définir des structures nécessaires ainsi que les fonctions permettant d'effectuer automatiquement les opérations de bases sur ces structures : allocation, initialisation, affichage, libération mémoire et, lorsque les structures peuvent faire partie d'une liste, destruction de la liste.

Les fichiers wifi.c, wifi.h, scan2.c, scan2.h, ip.c, ip.h, route.c et route.h définissent les fonctions répondant aux objectifs que nous nous sommes fixées. Ces fonctions utilisent les structures définies précédemment. Dans ces fichiers, la plupart des fonctions commencent par le préfixe wifi_. Celles qui n'ont pas ce préfixe ont un usage interne et ne sont pas destinées à être appelées depuis l'extérieur. Certaines fonctions servent à modifier des paramètres du systéme, d'autres à obtenir des informations. Dans ce second cas, le ou les premiers attributs sont des pointeurs indiquant une zone mémoire qui sera modifée pour y écrire les informations. Le type de retour est toujours un int, qui vaut 0 si l'exécution de la fonction s'est déroulée sans problème ou un nombre négatif si une erreur s'est produite . Ce nombre correspond à un code d'erreur et il est possible d'en obtenir une description avec la fonction wifi_geterror.

4.2 Contrôle des interfaces sans fils

Pour créer un réseau wifi mesh ou scanner les réseaux wifi déjà existant, il faut pouvoir contrôler les interfaces sans fils. Cependant, avant de contrôler les interfaces, il est nécessaire de récupérer les informations les concernant. Cela passe par l'utilisation de l'API libnl-genl et l'envoi de messages netlink au noyeau.

Comme précisé precédemment, il faut différencier les cartes réseaux physiques des interfaces "logiques". Nous avons donc décidé de les représenter par deux structures différentes, respectivement wifi_wiphy et wifi_interface, décritent dans le fichier interface.h. Les messages netlink envoyés par le noyeau nous donnent de nombreuses informations, mais nous avons choisi de ne retenir que celles qui seront utiles dans le cadre de ce projet. Ainsi, les cartes réseaux (wiphy) seront caractérisées par leur numéro, la liste des fréquences utilisables et la liste de types d'interfaces supportés. Les interfaces seront, quant à elles, représentées par leur nom, le numéro de la carte réseau associée, leur type, leur fréquence, la largeur du canal et leur adresse mac. On ajoute aussi à ces structures une variable list_head permettant de les lier dans une liste chaînée. Le fichier interface.c contient toutes les fonctions permettant de manipuler simplement ces structures : Création, affichage, libération de mémoire...

Les fonctions permettant l'envoi de messages netlink au noyeau et l'analyse des réponses sont écrites dans le fichier wifi.c. Celles-ci nécessitent l'utilisation d'un socket netlink. Pour éviter d'avoir à créer le socket à chaque appel d'une fonction, nous avons pris la décision de créer une structure, que nous avons appelée wifi_nlstate, contenant une référence sur le socket ainsi qu'un identifiant qui est également nécessaire pour l'envoi de messages netlink. Cette structure est définie dans le fichier wifi.h. Nous avons aussi créé une fonction permettant d'initialiser cette structure. Ainsi, l'utilisateur doit appeler une seule fois cette fonction d'initialisation, puis fournir un pointeur sur la structure à chaque appel d'une autre fonction nécessitant d'envoyer un message netlink.

La fonction send_recv_msg permet d'envoyer un message netlink et d'analyser la ou les réponses. Cette fonction prend en paramétre un pointeur sur la structure wifi_nlstate décrite dans le paragraphe précédent, la commande nl80211 à mettre dans le message netlink, les flags ainsi qu'une liste d'attributs à ajouter dans ce message, une réfèrence sur la fonction permettant d'analyser la réponse et un argument à fournir à cette fonction. Plusieurs fonctions définies dans le fichier wifi.c utilisent cette fonction send_recv_msg. Le but et fonctionnement de chacune d'elle sont donnés en commentaire dans le fichier wifi.h. Elles permettent, entre autre, de récupérer la liste des cartes réseaux (wifi_get_wiphy) et interfaces (wifi_get_interfaces), de limiter ces listes uniquement aux cartes réseaux supportant un type particulier (wifi_get_wiphy_supporting_type) ou aux interfaces utilisant ces cartes réseaux (wifi_get_if_supporting_type). Il est également possible de changer le type (wifi_change_type) ou la fréquence de fonctionnement (wifi_change_frequency) d'une interface, de lui associer un MeshID (wifi_set_meshid) ou d'en créer une nouvelle (wifi_create_interface). Seules les fonctions permettant d'allumer (wifi_up_interface) ou d'éteindre (wifi_down_interface) une interface utilisent un appel système à la place d'un message netlink.

4.3 Scan des réseaux

De même que pour les interfaces, nous avons du créer des structures pour représenter les réseaux. Ces structures, wifi_network et wifi_mesh_network, permettant de représenter respectivement des réseaux wifi centralisés et maillés, sont définies dans le fichier network.h. Les beacons frames contiennent plusieurs informations. Dans le cadre des réseaux centralisés, puisque nous ne cherchons à les détecter que pour savoir quelle fréquence est la moins occupée, nous nous contenterons de les représenter par leur SSID et leur canal. Par contre, un réseau mesh sera représenté par son MeshID, son canal, ainsi que tous les paramètres mesh fournis dans la beacon frame. Ces deux structures contiennent aussi une variable list_head leur permettant de faire partie d'une liste. Le fichier network.c contient les fonctions permettant de créer, d'initialiser, d'afficher de supprimer ces structures.

Le fichier scan2.c définit les fonctions permettant d'effectuer un scan et de lister les réseaux existants. La fonction wifi_scan_network effectue un scan pendant 3 secondes sur une interface en mode monitor. La fonction wifi_scan_all_frequencies effectue le même scan mais sur toutes les fréquences de la bande à 2.4GHz. Le scan se fait en utilisant les fonctions de la librairie pcap. Celle-ci n'utilise pas les codes d'erreur mais demande à ce que l'utilisateur fournisse un pointeur sur un tableau de caractères dans lequel sera écrit une description de l'erreur si besoin. Nous avons donc du reprendre ce principe pour les fonctions wifi_scan_network et wifi_scan_all_frequencies.

4.4 Adressage IP

Les fonctions relatives à l'adressage IP sont définies dans le fichier ip.c. La fonction wifi_start_avahi permet de lancer un deamon avahi sur une interface. Il est possible de spécifier en paramètre quelle adresse IP avahi doit tenter de réserver en priorité. La fonction wifi_stop_avahi permet au contraire d'arrêter le deamon avahi. La fonction wifi_get_ip_address permet de récupérer l'adresse IP associée à une interface. Elle utilise l'API libnl-route pour envoyer des messages netlink et en analyser le résultat. Elle le donne sous forme d'un tableau de 4 octets. La fonction print_ip_address permet d'afficher sous forme standart [1] une adresse donnée par 4 octets en prenant en paramètre un pointeur sur le premier de ces octets.

4.5 Routage

Nous avons créé une structure permettant de représenter une entrée dans la table des chemins. Celle-ci est définie dans le fichier mpath.h. Chaque entrée contient donc une destination et un prochain saut, sous la forme d'une adresse mac au format 6 octets, et une distance qui est un int. Pour représenter la table des chemins, il est nécessaire de pouvoir organiser ces entrées sous forme d'une liste, c'est pourquoi nous avons ajouté à la structure une variable list_head. Le fichier mpath.c contient donc les fonctions permettant de gérer cette structure.

1. quatre nombres entre 0 et 255 séparés par un point

Le fichier route.c contient les fonctions permettant de récupérer et modifier la table des chemins. La fonction wifi_add_mesh_path permet d'ajouter une entrée dans la table des chemins, wifi_del_mesh_path d'en supprimer une, et wifi_get_mesh_path permet de récupérer la liste de toutes les entrées, et donc d'avoir en même temps l'adresse MAC de tous les noeuds constituant le réseau. Ces oppérations utilisent l'API libnl-genl, c'est pourquoi il est indispensable de donner à ces fonctions une référence sur une structure wifi_nlstate initialisée, afin de pouvoir utiliser un socket netlink.

Bibliographie

[1] Michel Terré. Wifi, mars 2007.
http://easytp.cnam.fr/terre/images/WiFi.pdf.

[2] F.Nolot. Les réseaux sans-fil : Ieee 802.11.
http://www.nolot.eu/Download/Cours/reseaux/m1info/ProtoAv-Cours7-Wifi.pdf.

[3] James F. Kurose ; Keith W. Ross. Ieee 802.11 lans, 1999.
https://www.net.t-labs.tu-berlin.de/teaching/computer_networking/05.07.htm.

[4] How 802.11 wireless works, Mars 2003.
https://technet.microsoft.com/en-us/library/cc757419(v=ws.10).aspx.

[5] Nicolas Darchis. 802.11 frames : A starter guide to learn wireless sniffer traces, octobre 2010.
https://supportforums.cisco.com/t5/wireless-mobility-documents/802-11-frames-a-starter-guide-to-learn-wireless-sniffer-traces/ta-p/3110019.

[6] Guido Hiertz ; Dee Denteneer ; Sebastian Max ; Rakesh Taori ; Javier Cardona ; Lars Berlemann ; Bernhard Walke. Ieee 802.11s : The wlan mesh standard, fevrier 2010.
http://ieeexplore.ieee.org/document/5416357/.

[7] Jerome Henry. 802.11s mesh networking, novembre 2011.
https://www.cwnp.com/uploads/802-11s_mesh_networking_v1-0.pdf.

[8] R. Droms. Dynamic host configuration protocol, mars 1997.
https://tools.ietf.org/html/rfc2131.

[9] Avinash Joshi ; Hrishikesh Gossain ; Jorjeta Jetcheva ; Malik Audeh ; Michael Bahr ; Jan Kruys ; Azman-Osman Lim ; Shah Rahman ; Joseph Kim ; Steve Conner ; Guenael Strutt ; Hang Liu ; Susan Hares. Hwmp protocol specification, novembre 2006.
doc. : IEEE 802.11-06/1778r1.

[10] Netlink protocol library suite (libnl).
http://www.infradead.org/~tgr/libnl/.

[11] https://github.com/o11s/open80211s/wiki/HOWTO.

[12] Tcpdump et libpcap.
http://www.tcpdump.org/.

[13] M. Schukat ; M.P. Cullen ; D. O'Beirne. A redundant dynamic host configuration protocol for collaborating embedded systems.

https://pdfs.semanticscholar.org/da6f/
c3920e312ef74ce608bef8c94aae3962b8ce.pdf.