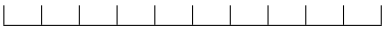
Réseaux sans fil de nouvelle génération : architectures spontanées et optimisations inter-couches

Yan Grunenberger

INSTITUT POLYTECHNIQUE DE GRENOBLE

N° attribué par la bibliothèque



THÈSE

pour obtenir le grade de

DOCTEUR DE Grenoble INP

Spécialité : « Informatique »

Préparée au Laboratoire d'Informatique de Grenoble

Dans le cadre de **l'École Doctorale « Mathématiques, Sciences et Technologies de l'Information, Informatique »**

préparée et soutenue publiquement par

Yan Grunenberger

Le 03.12.2008

TITRE :

Réseaux sans fil de nouvelle génération : architectures spontanées et optimisations inter-couches

Directeurs de thèse :
Andrzej Duda
Franck Rousseau

JURY

Pr Brigitte Plateau	Présidente
Pr Christian Bonnet	Rapporteur
Pr Thomas Noël	Rapporteur
Pr Andrzej Duda	Directeur de thèse
Dr Guillaume Chelius	Examinateur
Dr Franck Rousseau	Coencadrant

Ce travail est le fruit, comme bien souvent, d'un improbable cheminement. Au départ, c'est la curiosité pour les télécoms qui me conduit à orienter mes études dans ce domaine, et c'est ainsi que je finis par devenir élève-ingénieur au département Télécom à l'ENSIMAG en 2001. Au cours de ma scolarité, avec des camarades, nous décidons de suivre un programme optionnel de création d'entreprise, avec l'idée saugrenue que nous pourrions proposer un nouveau service pour la téléphonie sur les réseaux sans fil qui se développent énormément durant cette période.

C'est à l'occasion de l'étude technique de ce projet que je viens à poser un premier pas dans mon futur laboratoire, pour rencontrer mon futur encadrant, Franck. Je me souviendrais toujours de cette phrase : "L'étude des réseaux ad hoc sans fil est un domaine encore de l'ordre de la recherche. Il y a des années de travail à fournir". Et finalement, après avoir accepté un premier stage, puis avoir travaillé avec un doctorant de l'équipe, Hoang N'Guyen, j'ai accepté une thèse proposée par Andrzej sur ces réseaux sans fil de nouvelles générations qui se concrétise par le manuscrit ici présent.

C'est donc tout naturellement que je remercie chaleureusement Franck et Andrzej de m'avoir accordé leur confiance et de m'avoir accueilli chaleureusement au sein de l'équipe Drakkar. Nos échanges, à la fois techniques, sociaux et souvent philosophiques resteront pour moi inoubliables. Je remercie tout particulièrement Andrzej pour m'avoir permis de poursuivre ces échanges au-delà de nos frontières, au cours des nombreux projets et conférences, et lors d'un stage effectué au laboratoire d'Intel à Cambridge. Je remercie Franck d'avoir toujours encouragé ma créativité en écoutant patiemment mes idées de tous les jours et d'avoir su retenir et encourager celles qui étaient pertinentes.

Je remercie également mes deux rapporteurs, Christian Bonnet et Thomas Noel, d'avoir eu la patience de relire mon manuscrit et de m'avoir apporté leurs remarques. Leurs questions, ainsi que celles de Guillaume Chelius mon examinateur lors de ma soutenance, m'ont permis d'élargir ma réflexion et d'ouvrir de nouvelles pistes pour mes travaux futurs. J'ai été également honoré d'avoir Brigitte Plateau pour présidente dans mon jury.

Ce travail n'aurait pas été possible sans la présence et l'aide des membres du laboratoire. Dans un ordre quelconque, mes remerciements vont à Gilles et Jean-luc (pour nos débats techniques enflammés), à Carol et Pascale (pour leur efficacité et leur gentillesse à résoudre nos problèmes administratifs), à Christiane et François (pour avoir géré les tracas techniques avec douceur).

Je n'oublie pas mes collègues de bureau commun, Vianney, puis Fabrice et Benoit, pour avoir supporté mes excentricités diverses et d'avoir accepté toutes ces années un encombrement quotidien en câbles et machines. Merci à Kamila pour avoir le temps d'un stage distillé son humour dans notre bureau. Mais merci surtout à Aurélie pour m'avoir initié aux problématiques si intéressantes qui occupent les géographes et d'avoir maintenu une ambiance agréable au milieu de nous autres informaticiens.

Evidemment, je n'oublie pas ceux et celles qui sont désormais devenu des amis proches : la joyeuse bande, composée de Vincent, complice toujours présent et qui nous encourage à donner le meilleur de nous-mêmes ; Alph, mutant à l'énergie dense, qui fait décidemment trop de bruit sauf quand il dort ; Céline et Marius, mes chtits préférés qui n'ont de cesse de faire des mariages réussis ; Audrey et Fred, dont l'hospitalité légendaire et l'amitié se perpétuent malgré la distance et les déménagements ; Cathy et Nicolas dont l'appétit n'a d'égal que la gentillesse. Je n'oublie pas mes amis brésiliens Angelo et Manue pour m'avoir fait découvrir ce qu'est le coeur brésilien. Outre cette bande de joyeux drilles, j'ai eu le plaisir de retrouver cette même intensité avec de nouveaux venus, Noha, Maru et Michel, qui nous éclairent par leur culture et leur expériences.

C'est avec émotion que je remercie également Christine pour avoir partagé un temps ma vie et à qui je souhaite bon courage pour sa thèse. Je salue également mes colocataires successifs, Edson, pour m'avoir donné l'envie de partir au Brésil étancher ma soif, et Marie qui m'a fait découvrir le métier du Livre et la gastronomie avec sympathie et bonne humeur.

Enfin, je tiens à remercier tout particulièrement ma famille et surtout mes parents, Sylviane et Marc pour leur soutien de tous les jours, et d'avoir encourager toutes ces années cette curiosité envers la science et la technique qui m'anime encore et toujours aujourd'hui.

Merci à tous !

Yan

Le but de la science est de prévoir et non, comme on l'a dit souvent, de comprendre.
Pierre Lecomte de Noüy - L'Homme et sa destinée

Table des matières

Table des figures

Liste des tableaux

CHAPITRE 1

INTRODUCTION

Dans la société actuelle, nous sommes envahis par le terme "réseau". Internet, réseaux téléphoniques, réseaux sociaux, réseaux de chemins de fer... la simple évocation de ces entités nous ramène à des concepts simples : la mise en relation par des liens entre deux noeuds. L'origine du mot réseau provient du latin "rete" qui signifie "filet", et qui a donné l'adjectif "réticulé", caractérisant les objets ayant une structure formant un filet. Dès lors, l'utilisation du terme "la toile" pour décrire le World Wide Web d'Internet prend tout son sens.

Toutefois, il n'en a pas toujours été ainsi. Selon Pierre Musso, professeur, chercheur et philosophe de formation, c'est l'arrivée du chemin de fer qui a marqué le début d'une réflexion sur les réseaux. Claude-Henri de Rouvroy de Saint-Simon, économiste et philosophe, passionné par les scientifiques et en particulier Newton, a déjà développé à l'époque la notion de réseau et de capacité et l'a même étendu aux relations humaines, dans "Lettre d'un habitant de Genève à ses comtemporains", en 1803. Il y définit la notion de capacité du réseau à établir un lien.

Plus tard, le progrès aidant, de multiples réseaux technologiques vont se former, et notamment ceux dédiés au transport de l'information. Le téléphone, puis l'informatique vont contribuer à passer d'une construction désordonnée et propriétaire des réseaux à une volonté de normalisation et d'interconnexion. L'avènement du numérique a parachevé cette transformation, ouvrant définitivement la voie à la transmission de données.

Parallèlement, l'invention de la radio dès 1885 par Marconi révolutionne les moyens de communications. Tout d'abord limitée aux équivalents longue distance des liaisons télégraphiques, elle évoluera ensuite vers la diffusion radio, avec la TSF, puis permettra la retransmission de l'image, avec la télévision. Mais ce sont les progrès du numérique qui feront évoluer la radio vers la transmission de données, analogiques, puis numériques, ouvrant désormais la porte à des réseaux ambiants, à l'aide d'une technologie

telle que le WiFi (appelé souvent par sa norme, IEEE 802.11).

1.1 Historique

C'est tout naturellement que les réseaux informatiques ont donc atteint ainsi l'ère du tout radio : équivalent à leurs homologues filaires, offrant les mêmes fonctions, ils ouvrent la porte à de nouveaux usages, mais aussi à de nouvelles contraintes, dûes notamment au facteur historique. En effet, les réseaux informatiques, bien que récents, ont connu une histoire dense et complexe.

Tout commença avec les premières allusions aux interactions sociales liées à la mise en place d'un réseau, évoquées par J.C.R. Licklider du MIT en 1962 [4]. Un "réseau galactique" est proposé, où il envisage un réseau où les données et programmes peuvent être accessibles rapidement depuis n'importe où. Cependant, à cette époque, les transmissions de données s'effectuent par des circuits de données, établis grâce à l'invention du modem par les laboratoires Bell en 1958, ce qui réduit considérablement la possibilité d'un projet de réseau à grande échelle à l'aide des seuls circuits. Heureusement, en 1961, Leonard Kleinrock propose un sujet de thèse de doctorat [12] puis un livre [1] sur l'utilisation de réseaux à commutation de paquets, en lieu et place des circuits de communications. Cette idée va déboucher sur la création de réseaux par paquets [14] [8] et notamment ARPANET dès 1967 [6]. Le principe de base des réseaux modernes voit donc le jour.

Parallèlement, en 1968, l'université d'Hawaii et notamment Norman Abramson démarre des travaux de recherches sur l'utilisation des communications radio pour des liaisons informatiques : plusieurs machines doivent utiliser simultanément le même canal radio ; une première méthode d'accès multiple, ALOHA, voit le jour en 1970 [10]. L'utilisation des réseaux radio implique la prise en compte des pertes dans les transmissions. Les premiers logiciels conçus pour ARPANET, tels que NCP [7] sont dédiés à la communication entre les processus des différents hôtes : NCP introduit le concept de couches ("layers") : une couche inférieure, basé sur les hôtes/IMP destinés à la commutation, et une couche supérieure qui émule le comportement des circuits, en gérant les connexions, le concept de socket et l'association d'un flux de données à un processus ; cette couche supérieure, dite "second level layer" sera utilisée pour tous les programmes (transfert de fichier, terminal distant).

Cependant, ils ne prenaient pas en compte les éventuelles défaillances du mécanisme de transmission sous-jacent. Afin d'introduire de nouveaux médiums de transport et notamment les médiums sans fil ("Packet Radio") [11], R. Kahn introduit de nouveaux concepts (connectivité des différents réseaux par des passerelles, aspect distribué, correction d'erreur, conception standardisée). Avec Vinton Cerf, reprenant les idées de Louis Pouzin [3] et de son réseau Cyclades, ils proposent TCP [15]. Cependant, les recherches entamées sur le transport de la voix par paquet par Danny Cohen et son protocole Network Voice Protocol (NVP) [5] conduisent à une séparation de

TCP en deux protocoles en 1978 : un protocole IP dédié à l'adressage et au transfert des paquets, et un protocole TCP qui assure le contrôle de flots et la correction d'erreur. Pour les applicatifs ne nécessitant pas une correction de données, User Datagram Protocol (UDP) a été ajouté. L'Internet était né, et avec lui, l'architecture usuelle des réseaux informatiques actuels.

Parallèlement au développement de ces réseaux globaux, les réseaux locaux se sont fortement développés autour de la norme Ethernet, issue des recherches de R. Metcalfe [2], qui fût inspiré par le mécanisme d'accès d'ALOHA. Avec le développement des techniques de traitements de signal, les constructeurs ont pu proposer dès le milieu des années 1990 un remplacement de l'Ethernet sans fil, WaveLAN [9], qui allait donner naissance plus tard au standard IEEE 802.11 [13], ou WiFi.

Nous avons représenté figure 1.1 le cheminement qui a conduit à l'architecture des réseaux informatiques actuels .

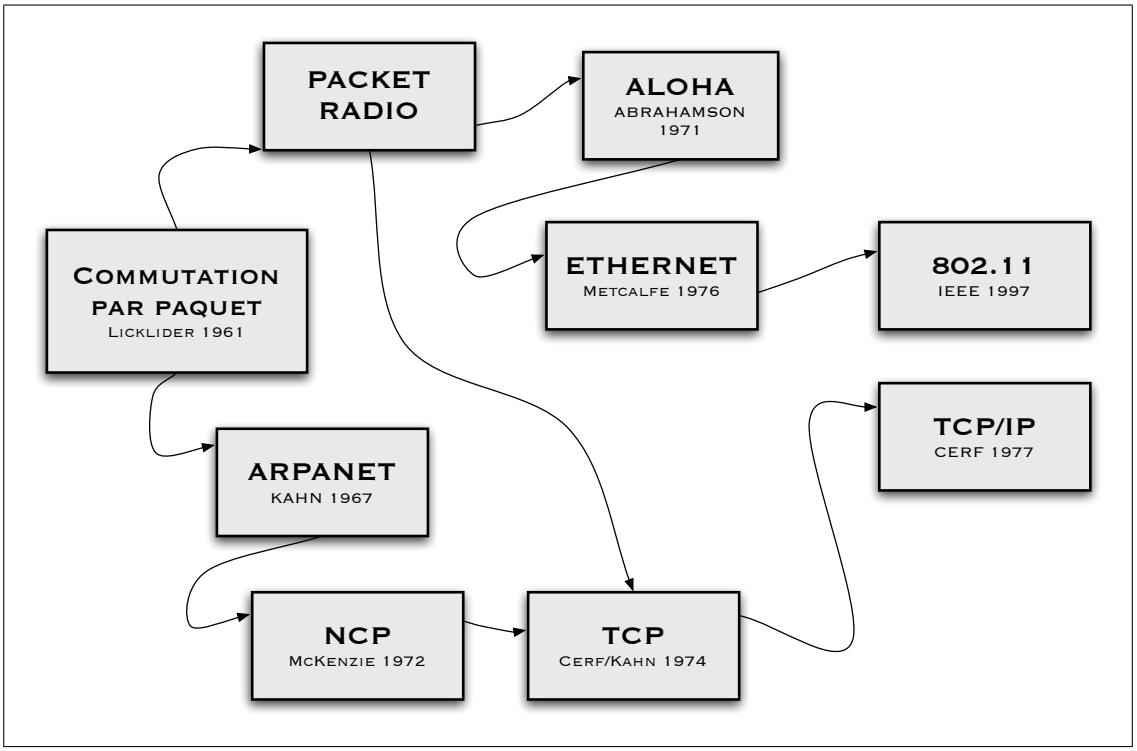


Figure 1.1 – Historique de l'architecture des réseaux actuels.

1.2 Motivations et problématique

L'architecture des réseaux actuels résultant de cette histoire complexe est principalement un modèle en couches, isolant d'une part le médium physique (câble, fibre optique, transmission radio) et les couches protocolaires supérieures. Ce modèle, permettant le développement indépendant des différentes composantes du réseau explique la réussite de l'Internet actuel : un constructeur peut proposer une modification tout en garantissant une comptabilité vis à vis des couches, ce qui a permis une transition perpétuelle des réseaux du tout filaire à un hypothétique tout sans fil, tout en conservant l'ensemble des principes applicatifs développés tout au long de ces années.

Ainsi, l'Ethernet puis son équivalent sans fil le WiFi ne sont pas les seules méthodes de transport physique du trafic IP de l'Internet. Parallèlement au développement des technologies de réseaux locaux (Local Access Network, puis Wireless-LAN), d'autres techniques d'accès sont apparues : les réseaux personnels, réduits en taille et en portée, appelés Personal Area Network ; les réseaux métropolitains, avec les technologies de boucle locale radio et désormais WiMAX (ou norme 802.16), appelée Metropolitan Access Network (MAN). Enfin, les réseaux étendus à l'échelle d'un pays, issus des technologies cellulaires des téléphones portables, comme le GPRS, l'EDGE, la 3G ou le HSDPA, constitue le Wide Wireless Access Network (WWAN).

La montée en puissance ainsi que la réduction de la taille des équipements ont permis l'intégration de technologies multiples dans nos équipements électroniques, agenda, téléphone, ordinateur. Il n'est donc plus rare aujourd'hui d'avoir dans notre environnement proche une connectivité quasi-permanente.

Cependant, cette connectivité n'est pas optimale.

D'une part, elle est orientée massivement vers l'Internet, donc fortement centralisée ; les contacts locaux et opportunistes ne sont pas du tout encouragés. Pourtant, l'évolution de nos usages montrent que sur les réseaux actuels liés à l'Internet se forment des réseaux sociaux, sous forme de sur-couches ("overlay").

Par exemple, il est étonnant de constater que , lors d'une réunion, la majeure partie des intervenants s'échangeront des données par le biais du mail, et donc d'Internet, sans profiter de l'abondance des technologies locales utilisables (Bluetooth, WiFi...). Autre exemple frappant, le développement des modems des opérateurs fournisseurs d'accès à Internet, appelés plus souvent sous le terme de "Box". Chacun de ces modems offre une connectivité sans fil ; il n'est pas rare, chez nous, de voir dans les réseaux WiFi disponibles le signal de la borne d'un ou plusieurs voisins. Pourtant, si nous désirons communiquer avec ceux qui nous sont géographiquement proches, ou partager la vision d'un contenu audiovisuel identique, nous nous reposons là encore une fois sur l'Internet centralisateur et sur le gaspillage de ressources qui en résulte. Nos réseaux manquent de pervasivité.

D'autre part, un autre problème sensible est l'inadaptation de l'architecture actuelle aux propriétés caractéristiques des liaisons sans fil. Par définition, une connexion sans

fil est synonyme de perte, de fluctuations, mais à contrario elle induit une liberté de mouvement. Or, l'architecture actuelle est largement inadaptée. Par exemple, le passage d'une technologie à une autre ne peut se faire de manière transparente (macro-mobilité entre deux réseaux différents) : systématiquement, une autre adresse IP de terminaison est utilisée par l'interface ce qui pose des problèmes pour les applications qui ne gèrent pas pour la plupart cet aspect. A une autre échelle, la micromobilité, ou déplacement d'un noeud au sein d'un même réseau, pose des problèmes des délais, qui peuvent perturber les connexions multimédias telles que la voix sur IP.

Mais un aspect plus problématique est la perte de performance : l'utilisation d'une technologie sans fil est trop souvent synonyme de perte de performance. Sur un réseau local WiFi, pourquoi le débit est-il limité, alors que le terminal est le seul à utiliser le réseau ? Pourquoi une communication VoIP est-elle hachée sur un téléphone lorsque je transfère des fichiers depuis mon ordinateur ?. Très souvent l'origine de ces problèmes se trouve dans l'indépendance entre les couches sous-jacente : elles ne permettent pas l'échange des données qui pourrait permettre une meilleure adaptation à ce médium imparfait qu'est le sans fil.

Dès lors, il est peut-être nécessaire d'examiner et de changer notre vision de l'architecture actuelle de l'Internet, et plus généralement, des réseaux IP pour les rendre plus adaptés à nos nouveaux usages ambiants : les réseaux de nouvelles générations doivent être à la fois des réseaux de proximité, et à la fois des réseaux longues distances ; ils doivent englober nos usages antérieurs et permettre de nouveaux usages liés à notre nouvelle connectivité permanente.

C'est dans ce contexte de bouleversement des usages et de connectivité ambiante que s'inscrivent les travaux effectués durant cette thèse, travaux centrés principalement sur les technologies d'accès à Internet, et plus particulièrement sur le WiFi.

1.3 Organisation du manuscrit

Le manuscript est organisé en 7 grands chapitres. Durant cette introduction nous avons évoqué l'inadaptation entre l'explosion de la connectivité sans fil et l'architecture historique en couches de l'Internet, et de fait, la probable remise en cause de celle-ci.

Dans une première partie, nous autopsierons les réseaux sans fil utilisant la technologie WiFi : à travers les examens des différentes composantes, nous en profiterons pour soulever quelques défaults et les solutions apportées par la communauté scientifique. La rationalisation de ces problèmes et de leur solution révélera un ensemble d'approches très spécifiques et peu adaptées pour faire des réseaux sans fil un équivalent sans fil de l'Internet actuel.

Puis, après avoir étudié l'architecture des systèmes WiFi et leur intégration dans les systèmes d'information, nous examinerons les propositions plus génériques d'optimisation, et les expérimentations effectuées dans ce sens, tout en évoquant les futures possibilités des interfaces radios.

Partant de ce constat, nous développerons ensuite notre proposition, basée sur une construction de l'architecture réseau respectant les exigences temporelles, qui nous permettra dans un premier temps, d'évaluer la pertinence des plates-formes actuelles pour l'implémentation de nouvelles méthodes d'accès, et dans un deuxième temps, de proposer un outil de traitement de paquets adapté à l'expérimentation de nouvelles architectures pour les réseaux sans fil.

Enfin, nous tirerons les conclusions de ce travail, tout en évoquant les perspectives futures en relation avec les développements des technologies sans fil.

Première partie

Etat de l'art

CHAPITRE 2

AUTOPSIE DES RÉSEAUX LOCAUX SANS FIL IEEE 802.11

Dans les spécifications publiées de IEEE 802.11 [49], il est indiqué que le but de ce standard est de "fournir une connectivité sans fil aux machines automatiques, aux équipements, ou aux stations qui requierent un déploiement rapide, pouvant être portables ou même dans la main, ou montés à bord de véhicules mobiles dans une zone délimitée". Cependant, il s'agit de plus que cela ; en fait, il s'agit de répondre à un besoin de connectivité sans fil, dans un environnement contraint, avec une réglementation souple (c'est à dire sans licence, mais sous réserve de respect des conditions de puissance/fréquence).

De ce fait les réseaux sans fil locaux sont très prometteurs. Ces réseaux présentent l'immense avantage par rapport aux autres types de réseaux sans fil d'être multi-rôles. En effet, de par les débits et les portées considérés, ils assument parfaitement une connectivité à tous les niveaux, que ce soit personnelle (liaison radio d'ordinateur à ordinateur, ou d'ordinateur à périphérique), locale (remplacement des réseaux Ethernet traditionnels), métropolitaines (accès à internet par un réseau de bornes métropolitaines formant un unique réseau d'opérateur ou d'une ville) ou globales (accès à un hotspot).

Néanmoins, le problème principal réside dans leur conception et leur intégration dans les systèmes d'exploitation, profondément ancrés dans une vision réseau locale. Les réseaux sans fils locaux, de type IEEE 802.11 ou WiFi, sont intrinsèquement un remplacement ad hoc des réseaux filaires Ethernet. A ce titre, ils reprennent de nombreux concepts de ceux-ci : méthode d'accès CSMA/CA basée sur CSMA/CD, reprise d'une architecture en couche physique (PHY) et MAC, utilisation d'une machine à état permettant de reproduire l'état connecté/déconnecté d'une connexion par câble, et bien évidemment, support de la couche protocolaire TCP/IP.

2.1 La couche physique

IEEE 802.11 repose principalement sur les ondes radio pour la transmission d'information. Les suites de bits à transmettre (par exemple, des paquets suivant le protocole TCP/IP) ne sont pas transmis tels quel dans les airs ; en effet, le signal binaire transmis directement (transmission dite en bande de base) ne serait pas adapté au canal de communication hertzien. Afin de faciliter la propagation, on module le signal autour d'une fréquence porteuse. Le réseau IEEE 802.11 étant conçu pour une utilisation sans licence, les concepteurs s'orientent rapidement vers des bandes de fréquences libres, dont les bandes ISM (Industrial, Scientific and Medical) à 2.4GHz puis les bandes N-II à 5GHz. Ces fréquences appartiennent au domaine fréquentiel des micro-ondes dont la longueur d'onde est réduite, ce qui facilite l'utilisation d'antennes de tailles réduites et adaptées à l'intégration dans des équipements mobiles. Enfin, à cette fréquence, et pour des puissances faibles (entre 10 et 100mW), la portée du signal radio atteint les 500 mètres, ce qui couvre le spectre des usages prévus pour IEEE 802.11.

Afin d'améliorer l'efficacité de la transmission, les bits sont encodés sous forme de symboles, grâce à une opération de modulation. IEEE 802.11 propose plusieurs techniques de modulations.

2.1.1 Différentes techniques de modulations

La modulation est réalisée généralement dans le domaine électromagnétique en modifiant l'amplitude, la fréquence, ou la phase du signal transmis. IEEE 802.11 utilise quasi-exclusivement des modulations de phase et d'amplitude. L'ensemble des modulations utilisées par IEEE 802.11 repose sur un moduleur I (In phase)/ Q (Quadrature), illustré sur la figure 2.1, composé de deux multiplieurs, un signal de porteuse (ici, soit 2.4GHz, soit 5GHz), et un sommateur. L'intérêt de ce modulateur est de transmettre deux informations (I et Q) sur une seule porteuse et utilisant la bande passante d'un seul composant, I ou Q.

Le principe est de représenter dans le plan I/Q les bits à transmettre qui constituent un symbole, puis à l'aide de convertisseur analogique numérique, de constituer un signal I et un signal Q à transmettre.

Les principales modulations utilisées par IEEE 802.11 sont les suivantes :

- BPSK : 1 bit est transmis par symbole ;
- QPSK : 2 bits sont transmis par symbole ;
- QAM : 4 ou 6 bits sont transmis par symbole ;

La figure 2.2 illustre le codage de 4 bits formant un symbole unique à 16 états dans la plan I/Q.

Cependant, la transmission de ces symboles sur la bande publique peut-être altérée par d'autres signaux d'interférences ; afin d'améliorer la résistance, IEEE 802.11 utilise deux techniques : l'élargissement de spectre, et l'OFDM.

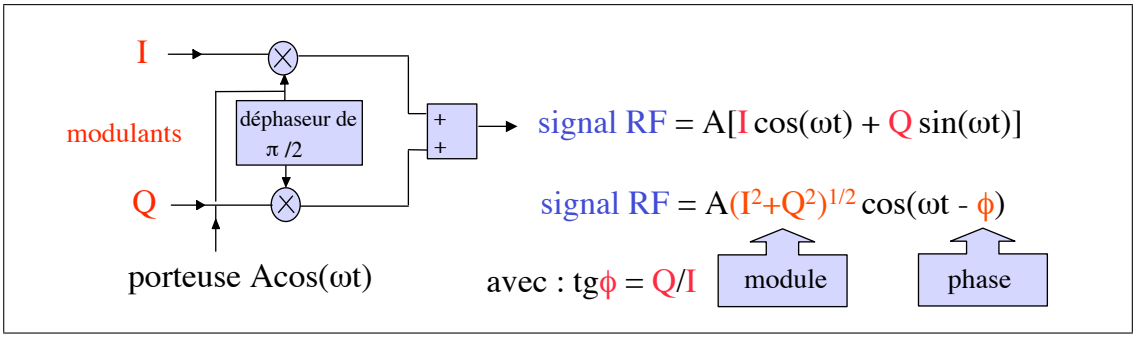
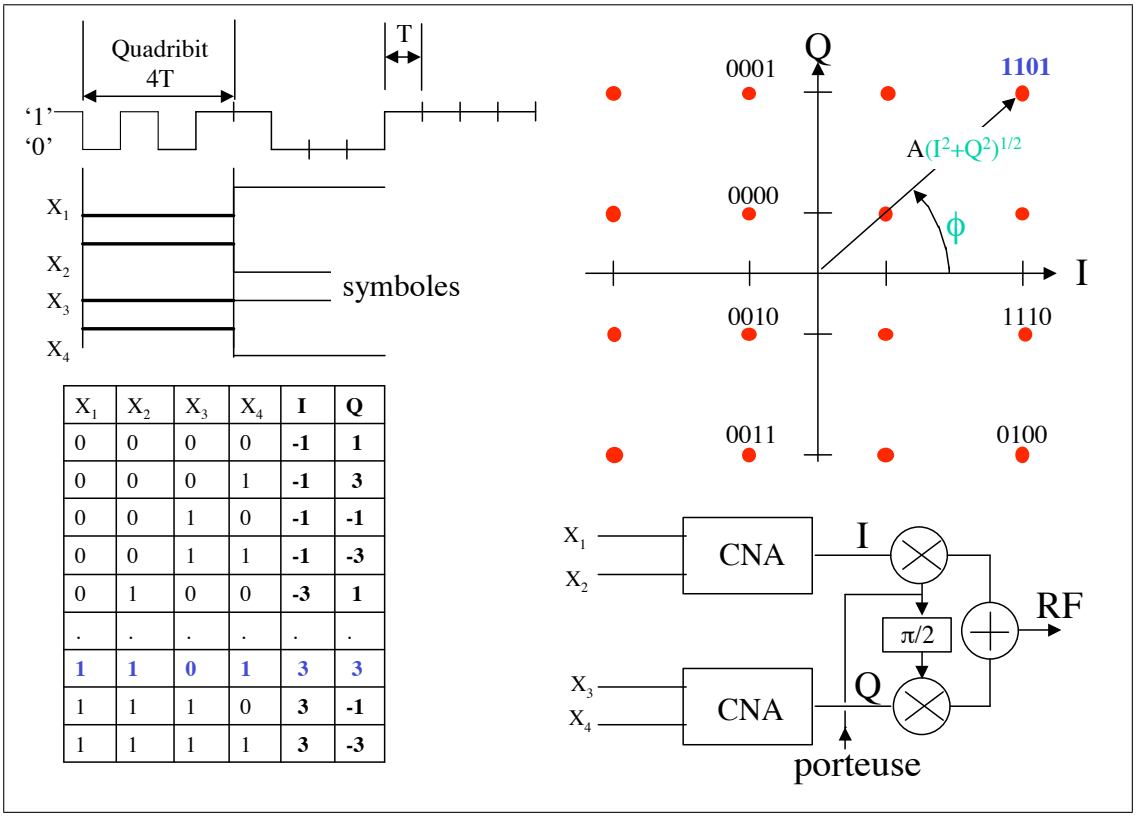


Figure 2.1 – Modulateur I/Q : utilisé pour les modulations dans 802.11. (Extrait du cours de Gérard Couturier, IUT de Bordeaux)



X_1	X_2	X_3	X_4	I	Q
0	0	0	0	**-1**	**1**
0	0	0	1	**-1**	**3**
0	0	1	0	**-1**	**-1**
0	0	1	1	**-1**	**-3**
0	1	0	0	**-3**	**1**
.	.	.	.	.	.
1	**1**	**0**	**1**	**3**	**3**
1	1	1	0	**3**	**-1**
1	1	1	1	**3**	**-3**

Figure 2.2 – Principe de codage des symboles pour la modulation QAM16. (Extrait du cours de Gérard Couturier, IUT de Bordeaux)

Famille à l'élargissement de spectre DSSS

Cette technique repose sur l'élargissement de la bande utilisée par le signal utile de manière à le rendre plus résistant. On multiplie à cet effet les signaux I/Q issus de

la mise en symbole par une séquence pseudo-aléatoire (Pseudo Random, P/R) dite de chipping, qui, si elle est connue à l'émetteur et au récepteur, permet la réception du signal d'origine. La séquence doit disposer de propriété d'équilibre (il y a autant de 0 et que de 1) et sa corrélation croisée par elle-même doit permettre la détection immédiate des symboles.

Dans le cas de 802.11, la séquence de chipping est dite de barker (voir figure 2.3), et est composée de 11 bits (appelés chips). Pour obtenir le débit nominal de 802.11, à savoir 1Mbits/s, il faut donc disposer d'un taux d'encodage d'1MSymboles/s. Afin de maintenir ce taux, la séquence est donc encodée à 11Mhz. Pour recevoir tous les éléments de la séquence, il faut donc avoir une fréquence d'échantillonnage suffisante : le théorème de Shannon nous garantit que celle-ci doit être le double de la fréquence maximale du signal à échantillonner, soit ici 22Mhz. Les signaux de 802.11 occupent donc une fréquence de 22Mhz sur la bande de fréquence.

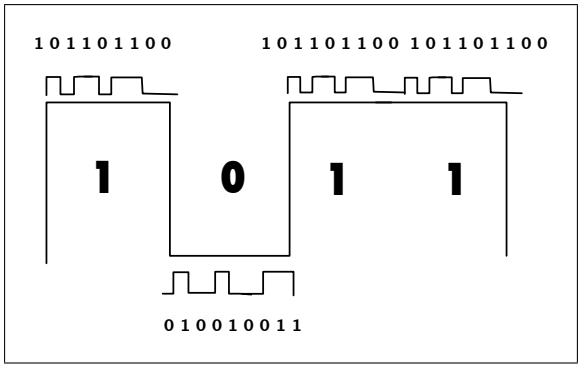


Figure 2.3 – Séquence de Barker : chips utilisés dans 802.11 pour les débits 1 et 2Mbits/s.

Afin de permettre l'utilisation simultanée de plusieurs réseaux 802.11, le standard initial [49] prévoit que les bandes de fréquence sont décomposées en canaux sur les bandes de fréquence à 2.4GHz. La bande ISM 2.4GHz est décomposée en canaux espacés de 5Mhz, et répartis entre 2.400-2.4835GHz, présentés dans les tableaux situés en annexe A.1. L'espacement de 5Mhz pour une largeur de canaux de 22Mhz implique que les canaux adjacents sont brouillés entre deux : il n'y a donc que 3 canaux indépendants sur la bande ISM 2.4GHz. Néanmoins, l'utilisation de l'étalement de spectre par chipping permet la cohabitation de réseaux sur des canaux adjacents, mais l'efficacité de cette cohabitation est à relativiser compte tenu de la méthode d'accès retenue pour 802.11, comme nous le verrons dans la section suivante, dédiée à l'étude de la couche MAC.

Famille CCK

Les concepteurs désirent améliorer la méthode d'étalement pour augmenter les débits. Lors de l'amendement 802.11b [57], le standard suggère une augmentation des débits en modifiant la modulation initiale : la modulation utilise des codes à phases multiples complémentaires (Polyphase Complementary Code). Le code possède les mêmes propriétés d'autocorrélation que la séquence de barker, sauf qu'il s'agit cette fois de symboles complexes en lieu et place d'une séquence binaire. Le modulateur encode par blocs de bits successifs : 2 bits sont utilisés pour décrire la rotation du code dans le plan complexe, puis les bits restants sont utilisés pour le code polynomial, soit sur 2 bits (4 codes disponibles) soit sur 6 bits (64 codes disponibles).

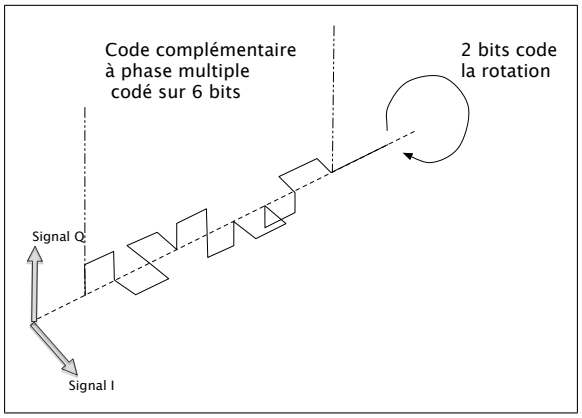


Figure 2.4 – Séquence CCK : chips complexes utilisés pour les débits 5.5 et 11 Mbits/s.

Les débits théoriques prévus par l'amendement 802.11b sont alors résumés sur la figure 2.1.

Modulation	BPSK	QPSK	DQPSK	DQPSK
Bits par symboles	1	2	4	8
Type d'étalement	Barker	Barker	CCK	CCK
Taille du chip(bits)	11	11	8	8
Vitesse de chipping	11	11	11	11
Débits (Mbits/s)	1	2	5.5	11

Table 2.1 – Modulations utilisées pour le standard IEEE 802.11a et débits nominaux correspondants.

Ces débits constituèrent un premier palier dans les débits théoriques. Cependant, la recherche de plus hauts débits va conduire le groupe IEEE 802.11 à s'inspirer de nouvelles techniques de modulation issues des transmissions par modem : l'Orthogonal Frequency Division Multiplexing (OFDM).

Famille OFDM

Dans les transmissions par modem, le but recherché était l'augmentation significative de la bande passante ; les premiers modems utilisaient des modulations qui maximisaient le nombre de bits transmis dans la bande passante de la voix (soit 4 Khz). Les débits étaient donc rapidement limités (les modems standards ont atteints des débits de 28,8 puis 56 Kbits/s). L'idée était donc d'utiliser des fréquences différentes de celle de la voix mais le canal utilisé (fil de cuivre) était très sélectif en fréquence : l'étalement des retards est grand devant le temps de transmission d'un symbole.

Le principe de l'OFDM est donc de transmettre les symboles sous plusieurs porteuses (appelées sous-porteuses) ; il est donc possible de transmettre des symboles plus longs, insensibles aux retards, mais en plus grand nombre, ce qui conduit à conserver le même débit qu'une modulation traditionnelle mono-porteuse. L'OFDM est donc une version duale du multiplexage temporel utilisé sur les modulations monofréquences : le multiplexage est ici fréquentiel. L'efficacité spectrale va caractériser les modulations par sous-porteuse : il s'agit du débit binaire par unité de fréquence. Le choix des sous-porteuses et leur écartement va donc influencer l'efficacité spectrale.

Afin de maximiser l'efficacité spectrale, les sous-porteuses doivent être les plus proches les unes des autres, tout en garantissant que le récepteur soit capable de distinguer celles-ci et soit capable de récupérer les symboles transmis. Cela est vrai si le spectre d'une sous porteuse est nul pour les fréquences d'une autre sous-porteuse : on parle alors de sous-porteuse orthogonale, d'où le terme OFDM.

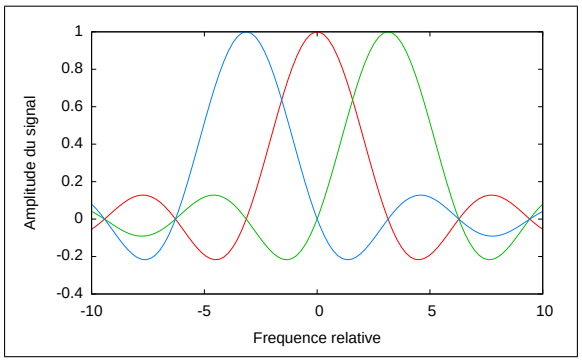


Figure 2.5 – Superposition du spectre de 3 sous-porteuses orthogonales.

Le signal modulé par une porteuse avec l'utilisation d'une forme d'onde rectangulaire (par exemple, un signal binaire composé d'une suite de 0 et de 1) a un spectre défini par un sinus cardinal ; Celui-ci s'annule à des intervalles donnés. Si Ts est la durée d'un symbole, et que l'on écarte les sous-porteuses d'un intervalle d'un 1/Ts, on peut obtenir une superposition de spectres de sous-porteuses garantissant la récupération de chaque sous-porteuse, comme l'illustre la figure 2.5.

La réalisation d'un modulateur OFDM peut paraître complexe : en effet, il faut disposer d'un ensemble de sous-porteuses orthogonales. Cependant, un symbole OFDM se décompose comme une somme de composantes, qui s'identifie à la transformée de Fourier inverse. A partir du formalisme lié à la transformée de Fourier, on peut donc tout à fait procéder à la transformation par Fourier inverse d'un signal continu I ou Q (issu d'un modulateur I/Q par exemple) en une somme de composante (Figure 2.6) ; à la transmission de celle-ci, puis à la réception des composantes qui permettent par transformée de Fourier de retrouver le signal continu initial (Figure 2.7).

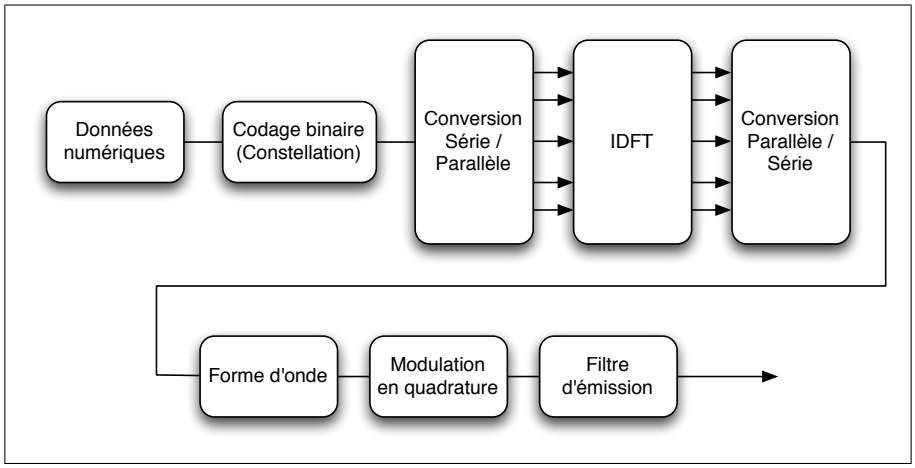


Figure 2.6 – Transmetteur OFDM par transformée inverse discrète.

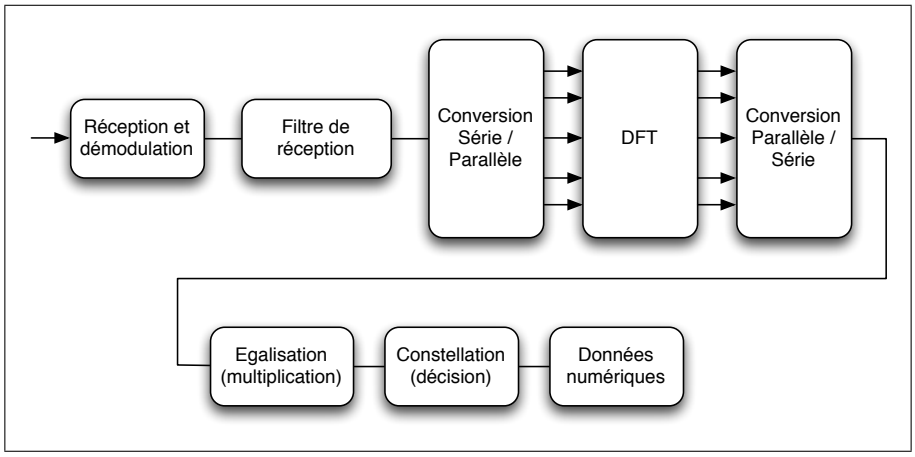


Figure 2.7 – Récepteur OFDM par transformée discrète.

Dans le cadre de IEEE 802.11, l'OFDM a été introduit par l'amendement 802.11a [51], sur la bande de fréquence de 5GHz, notamment sur la bande publique appelée N-II, et introduite à cet usage par Apple par une pétition en 1995. Nous détaillons les canaux offerts sur cette gamme de fréquences dans les tableaux situés en annexe A.2, A.3, A.4. L'écartement des canaux garantit un fonctionnement sans interférence entre les réseaux 802.11 utilisant des canaux adjacents.

IEEE 802.11a reprend donc les techniques de codage par constellation dans le plan I/Q introduites par les amendements précédents. Etant donné les hauts débits visés, le standard introduit des codes de correction d'erreurs FEC (Forward Error Correction). Ces codes consistent en une redondance d'informations permettant de retrouver les données corrompues. Cette redondance s'exprime par des taux d'encodage, qui indiquent la proportion d'informations uniques sur la somme d'informations totales transmises : 1/2 signifie que 2 bits sont transmis pour 1 bit d'information unique.

Ainsi IEEE 802.11a utilise 52 sous-porteuses OFDM, dont 48 utilisées pour le transport de données, et 4 pour l'étalonnage du canal (appelées pilotes). Chaque sous-porteuse utilise le même type de modulation que les anciens amendements de 802.11, à savoir BPSK, QPSK, 16- Quadrature Amplitude Modulation (16-QAM) ou 64-QAM. La durée d'un symbole est de 4 μs.

Ainsi, pour 48 porteuses, les débits de la figure 2.2 sont obtenus.

Modulation	BPSK	BPSK	QPSK	QPSK
Bits par sous-porteuse	1	1	2	2
code FEC	1/2	3/4	1/2	3/4
Nombre de bits utiles par symboles	24	36	48	72
Vitesses	6	9	12	18
Modulation	QAM16	QAM16	QAM64	QAM64
Bits par sous-porteuse	4	4	6	6
code FEC	1/2	3/4	2/3	3/4
Nombre de bits utiles par symboles	96	144	192	216
Vitesses Mbits/s	24	36	48	54

Table 2.2 – Modulations utilisées pour le standard IEEE 802.11a et débits nominaux correspondants.

Ces modulations ont été ultérieurement reportées sur la bande ISM 2.4GHz lors de l'amendement 802.11g [33]. L'amélioration graduelle des techniques de modulation a permis de disposer d'un éventail complet de débits.

Dès lors, ces multiples modulations permettent le maintien de la connectivité sur un large périmètre, au détriment du débit qui va donc diminuer avec la distance. 802.11 ne propose pas d'algorithme particulier pour la gestion de ces différentes modulations. Lors de l'amendement 802.11b, les constructeurs ont principalement adopté l'algorithme d'Auto Rate Fallback ARF implementé sur les cartes WaveLan-II [9] : si trop de pertes sont enregistrées avec une modulation, la carte utilise une modulation

plus résistante (donc, à débit moindre) dans les modulations qui sont autorisées dans le réseau 802.11 considéré.

Cependant, l'utilisation de ces multiples débits sur une même bande implique l'usage de techniques permettant une compatibilité, même minimale. 802.11 propose donc deux entités pour résoudre ce problème : le PLCP et le PMD.

2.1.2 Un effort pour la compatibilité : le PLCP et le PMD

Dès sa conception, 802.11 a inclus la nécessité de faire cohabiter plusieurs technologies sur une même base. Pour la cohabitation avec les technologies 802.11, le standard offre donc une abstraction entre la transmission des données et les différents standards de modulations : il s'agit du Physical Layer Convergence Protocol (PLCP) et du Physical Medium Dependent (PMD).

Le PLCP offre des primitives à la couche de contrôle d'accès au médium (couche MAC - Medium Access Control). Il s'agit principalement d'offrir une abstraction à la couche MAC vis à vis des modulations physiques qui sont gérées par le PMD. Le PLCP fournit une en-tête radio PLCP permettant d'assurer des fonctions.

Le PLCP propose donc les primitives suivantes :

- détermination de l'état du canal : cela est réalisé par deux sous-fonctions : le Carrier Sense (CS) et le Clear Channel Assessment (CCA). Le CS est utilisé pour la réception : une interface radio 802.11 scrute le médium en permanence pour décoder l'en-tête PLCP, indiquant qu'une transmission a lieu. Le CCA est utilisé lorsque la couche d'accès au médium souhaite transmettre un paquet : si le canal est libre, une disponibilité est indiquée à la couche MAC ; sinon, une occupation est transmisse à celle-ci.

- réception : lors d'un CS réussi, l'en-tête PLCP décodée permet de récupérer le PMD, qui sera décodé par démodulation physique spécifique. Le résultat de cette opération est transmis aux couches supérieures.

- émission : si le CCA a indiqué un canal libre, alors les données à transmettre sont encodées par le PMD et sont précédées d'une en-tête PLCP. L'interface transmet une en-tête PLCP constitué d'un préambule.

L'en-tête introduit par le protocole PLCP est composé de plusieurs parties : il est constitué d'abord d'un préambule permettant la synchronisation des stations réceptrices et d'identifier le début d'une transmission. Celui-ci est distinct en fonction de la famille de modulation utilisée.

Les différentes formats sont explicités dans l'annexe B.

Dès lors, avec PLCP, les différents hôtes 802.11 peuvent recevoir des paquets issus de technologies différentes et néanmoins en tenir compte : grâce au champ longueur introduit par PLCP, ils sont en mesure d'estimer la durée d'occupation du canal.

2.1.3 Réflexions sur la gestion des débits multiples

L'adaptation du débit (ou rate control) a fait l'objet de nombreuses publications, mentionnées par Lacage dans [39]. Le premier algorithme proposé et adopté par la majorité des constructeurs, ARF, a été introduit dans les cartes WaveLAN, ancêtres des cartes IEEE 802.11 [9]. Avec ARF, les émetteurs choisissent une modulation à débit supérieur à partir d'un certain nombre de transmissions consécutives réussies (10 par exemple), et basculent sur des modulations plus faibles en débits lors de deux échecs consécutifs dans une fenêtre de temps donnée. La transmission qui suit la transmission après augmentation ou diminution est appelée sonde (ou Probing) et doit réussir immédiatement sinon le débit revient à la valeur initiale et le compteur des tentatives est remis à zéro.

Le problème d'une telle méthode est l'inadaptation à des changements rapides du canal, par exemple lors d'un hôte mobile ou lors de connexion ad hoc ; d'un paquet à l'autre, la situation peut changer ; dès lors, le débit optimal change de paquet en paquet. L'intervalle de paquet qui engendre une réaction (2 paquets pour un débit faible, 10 pour un débit plus élevé) est trop grand comparé à la variabilité du système.

Paradoxalement, ARF est aussi trop réactif : dans le cadre d'une utilisation de bureau du réseau sans fil, l'utilisateur ne se déplace pas ; il y a donc une relative stabilité de ses transmissions. Cependant, les transmissions sont souvent altérées par groupes consécutifs dans un tel environnement. ARF est donc trop réactif dans ce cas, puisque 2 transmissions consécutives suffisent à basculer sur une modulation plus faible. Cette instabilité est la conséquence du fait que le même mécanisme de régulation est utilisé pour gérer des perturbations à court terme et à long terme.

Afin d'améliorer la réactivité d'ARF, plusieurs solutions ont été proposées. AARF [39] proposent de modifier dynamiquement les seuils de réactivité d'ARF. Dans AARF, l'évolution est contrôlée par un algorithme de Binary Exponential Backoff, similaire à celui utilisé dans l'algorithme de gestion de la fenêtre de contention dans 802.11, que nous étudierons dans la section consacrée à la couche MAC. Ainsi, lorsque le paquet de sonde n'est pas acquitté, la modulation est rétrogradée à une valeur plus faible, mais l'intervalle de transmissions réussies est doublé avant de pouvoir passer à une modulation supérieure. L'effet immédiat est que AARF stabilise la connexion en créant moins d'échec de transmission, puisque les paquets de sonde sont plus espacés. Les auteurs ont étudié les possibilités d'implémentations, et ont identifié des cartes 802.11 à basse et haute latence ; AARF fait l'objet d'une implémentation nommée AMRR sur des cartes à haute latence, au prix d'une moins bonne réactivité.

ONOE d'Atsushi Onoe est un algorithme implémenté librement dans le code source des cartes 802.11 à base de chipset Atheros. Son principe est de conserver pour chaque hôte un crédit qui va être incrémenté s' il y a peu de pertes. A partir d'un certain seuil le débit est augmenté ; si il y a des erreurs, le crédit est réinitialisé et le débit est diminué. L'algorithme repose sur les critères suivants : si aucun paquet n'est transmis,

le débit est diminué ; si 10 paquets ont été transmis et que le taux de retransmission est supérieur à 1, on repasse au débit immédiatement plus faible. Si plus de 10% des paquets ont fait l'objet d'une perte, on diminue le crédit ; si moins de 10%, on augmente le crédit ; si aucune des situations précédentes ne se présente, alors on continue d'utiliser le même débit.

RBAR [19] est un algorithme de gestion des débits qui vise à augmenter la bande passante disponible pour les applications. Elle est cependant toutefois incompatible avec la norme 802.11, car elle change la sémantique des messages du protocole 802.11. Le protocole est conçu pour les réseaux ad hoc, il utilise donc les paquets RTS/CTS de la détection de porteuse virtuelle de la couche MAC. Le principe repose sur la mesure de la valeur énergétique du signal radio émis (rapport signal/bruit, SNR) du paquet RTS. A partir d'un modèle de canal préexistant, une modulation est choisie en fonction du SNR et le débit choisi est utilisé dans le paquet CTS de retour. Cette méthode présente plusieurs défauts : elle présuppose un modèle de canal donné et un modèle symétrique pour la transmission et finalement, impose l'usage du RTS/CTS inducteur d'une surcharge importante.

OAR [52] propose une exploitation plus intéressante des canaux sans fil de bonne qualité. Il exploite le principe que le temps caractéristique du canal (Channel Coherence Time), ou temps où la qualité d'un canal est constante, est plus grand qu'un temps de transmission de plusieurs paquets. Le principe repose sur une équité temporelle : chaque station dispose du même temps de transmission sur le canal, et donc les stations utilisant un débit plus élevé peuvent donc envoyer plus de données. Le protocole utilise l'échange RTS/CTS de manière similaire à RBAR, mais utilise ensuite la fragmentation de paquets 802.11 pour transmettre plus de paquets si la station dispose d'un débit rapide. OAR observe les conditions du canal pour adapter son régime de fragmentation. Cependant, le principal problème d'OAR réside dans son incapacité à s'adapter à une variation du canal qui pourrait surgir pendant une transmission fragmentée ; dans ce cas, l'ensemble des fragments seront probablement perdus par la station 802.11 et un mécanisme de type RBAR sera alors utilisé comme gestion du débit.

Toutefois, l'étude de ces différentes gestions des débits multiples révèle une caractéristique commune : la coopération étroite avec les caractéristiques de la couche MAC. En effet, bien que les concepteurs aient réussi à isoler la plupart des problématiques de la transmission physique dans la couche physique, les caractéristiques temporelles de la transmission sont partie entière de la gestion de l'accès au médium. Dès lors, l'élaboration de solutions de gestion des débits ne peut passer que par une connaissance approfondie des mécanismes de gestion de l'accès multiple introduit dans la couche MAC de IEEE 802.11.

2.2 La couche MAC

La couche Medium Access Control (MAC) gère l'accès au canal. Dans le cadre de 802.11, comme dans le cadre des réseaux packet radio vu plus en amont dans ce manuscript, plusieurs hôtes sont en concurrence pour utiliser la bande passante offerte par le canal radio utilisé.

Dans un médium quasi parfait, tel la paire de cuivre d'Ethernet, il est facile d'identifier si une transmission a lieu ou pas ; en effet, les signaux électriques sont aisément détectables. Dans le cadre d'une transmission radio, cet exercice n'est pas des plus aisés. En effet, la complexité des transmissions radio font qu'une transmission n'est pas forcément acquise ; des perturbations électromagnétiques peuvent affecter la qualité du signal reçu, le rapport signal bruit, ce qui contribue à des erreurs d'interprétation ; un signal plus puissant qu'un autre peut totalement masquer ce dernier, c'est l'effet de capture. En Ethernet, une collision entre deux émissions est détectable ; sur un lien radio, la collision n'est pas détectable, car la situation au récepteur est inconnue, les conditions n'étant pas forcément les mêmes.

Nous l'avons vu dans la section précédente, le standard 802.11 utilise un ensemble de primitives (CS,CCA) fournies par PLCP pour évaluer la disponibilité du canal lors d'une transmission ou pour détecter une transmission dans le cadre d'une réception. Mais tout 802.11 met en oeuvre tout un ensemble de techniques pour essayer de réduire l'impact des contraintes de la transmission radio. Pour cela, plusieurs modes d'accès ont été élaborés : d'une part le modèle distribué, DCF, le plus utilisé actuellement dans les équipements supportant la norme 802.11,d 'autre part le modèle centralisé, ou Point-Controlled Function, qui offre un contrôle centralisé de l'accès au canal.

Nous nous concentrons sur DCF, car le mode centralisé reste optionnel, et à ce jour ne fait toujours pas partie des tests de conformité de l'association des constructeurs 802.11, la WiFi Alliance. Il est donc de ce fait peu implanté et quasiment inusité.

2.2.1 Modèle distribué, DCF

La couche d'accès au médium utilisée principalement par IEEE 802.11 est la Distributed Coordination Function ; elle permet le partage automatique du médium entre plusieurs hôtes grâce à l'utilisation du CSMA/CA et d'un backoff aléatoire suivant un état occupé du canal. Tout trafic direct utilise des acquittements (ACK) auquel cas toute retransmission est initiée si l'ACK n'est pas reçu par la station émettrice.

Le protocole CSMA/CA est conçu pour réduire la probabilité de collisions lorsque de multiples stations accède au médium. CSMA/CA est un dérivé de la méthode d'accès utilisé dans les réseaux ALOHA [10]. ALOHA repose sur le principe de fonctionnement suivant : un hôte désirant transmettre des données les envoient immédiatement, et si il y a collision, alors le paquet est tout simplement renvoyé.

L'analyse de l'efficacité de cette méthode montre qu'elle ne permet pas d'utiliser une majeure partie du canal disponible. Si l'on considère l'arrivée des paquets selon

un processus de poisson de taux λ durant l'intervalle $[0, t]$, alors le nombre de paquets transmis X(t) est donné par la probabilité suivante :

$$p(X(t) = k) = \frac{(\lambda t)^k e^{-\lambda t}}{k!}$$

Dans le cas d'un réseau ALOHA original, les paquets qui précédent une émission ou qui sont émis pendant une émission constituent des collisions. Dans ce cas, la bande passante est définie par

$$S = G * p(aucunecollision)$$

où G est la charge, c'est à dire $G = \lambda * \tau = \lambda * N/R$ où N est le nombre de bits par paquets et R le débit du canal. La probabilité de n'avoir aucune collision durant l'émission d'un utilisateur pendant $[0, \tau]$ est donc la probabilité qu'aucune transmission n'intervienne durant $[-\tau, \tau]$, c'est à dire :

$$p(X(t = 2\tau) = 0) = e^{-2\lambda\tau}$$

soit une bande passante effective de :

$$S = Ge^{-2G}$$

La bande passante maximale dans ce cas est atteinte pour $G = 0.5$, soit $S = 0.18$. En d'autres termes le débit est seulement 18% de celui qu'on aurait un utilisateur seul, ce qui est très faible. L'inefficacité s'explique par le fait que dans ALOHA, l'absence de synchronisation conduit les utilisateurs à émettre les uns sur les autres. En synchronisant les utilisateurs de telle sorte que les transmissions soient alignées temporellement, les recouvrements partiels de transmission peuvent être évités.

C'est ce que propose Slotted ALOHA [30] en divisant le temps en slot temporel de taille τ. Ainsi, il n'y a pas de recouvrement partiel, et donc un paquet transmis durant la période $[0, \tau]$ est reçu de manière correcte si aucun autre paquet n'est transmis, ce qui correspond au calcul de probabilité avec $t = \tau$:

$$p(X(t = 2\tau) = 0) = e^{-\lambda\tau}$$

soit une bande passante effective :

$$S = Ge^{-G}$$

L'efficacité atteint ici 36% de la bande passante pour $G = 1$. Ainsi, Slotted ALOHA a une bande passante maximale double de celle d'ALOHA, au prix d'une synchronisation nécessaire. Cependant, des collisions peuvent encore avoir lieu, si deux hôtes émettent durant le même slot. Pour réduire l'inpact des collisions, la méthode d'accès peut utiliser le Carrier-Sense [48]. Le principe est d'écouter le canal et d'attendre avant de transmettre si le canal est occupé. Le Carrier Sense Medium Access peut être p-persistant ou non persistant :

- en mode p-persistant, quand le canal est vide, l'hôte transmet avec la probabilité p ; il diffère son émission avec la probabilité 1-p et réitère la même procédure. Si le canal est occupé, l'hôte attend la disponibilité pour reprendre la même procédure.

- en mode non-persistant, quand le canal est vide l'hôte transmet les données ; si le canal est occupé, il choisit un intervalle de temps aléatoire avant de retenter une émission. Si le canal est libre à ce moment, il transmet, sinon il choisit une nouvelle valeur aléatoire d'attente. Le choix d'un temps aléatoire élimine la plupart des collisions qui résultent d'une émission simultanée à la transition occupé/libre.

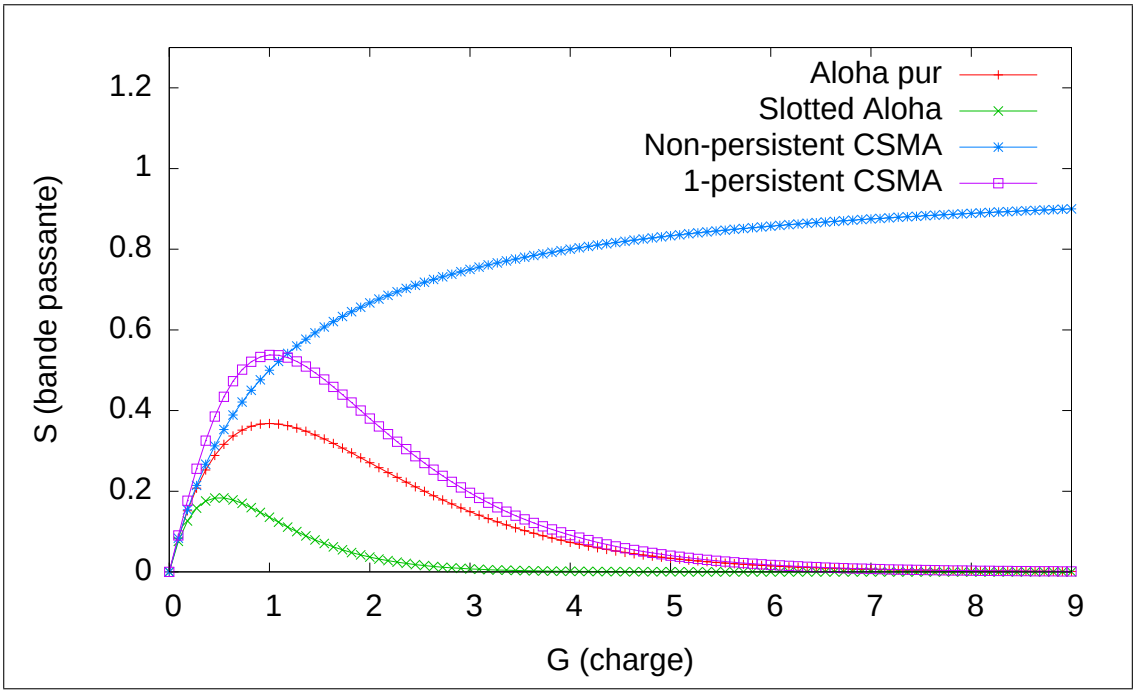


Figure 2.8 – Efficacité comparée de ALOHA, slotted ALOHA et CSMA.

L'efficacité des différentes méthodes est comparée sur la figure 2.8. En résumé, les méthodes basées sur ALOHA ou slotted ALOHA ne permettent pas une utilisation maximale du canal. Pour de faibles niveaux de trafic, CSMA p-persistant offre la meilleure bande passante ; pour de fort volumes, les CSMA non persistant offre les plus haut débits. La recherche d'un p optimum peut conduire à des performances intéressantes dans le cas de CSMA p-persistant ; cependant, les performances de Slotted non persistent CSMA sont intéressantes dans la mesure où la méthode n'exige pas une recherche de l'optimum : c'est donc cette méthode qui a été retenu pour 802.11 DCF.

802.11 utilise donc un partage du temps sous forme de slot. Le slot a une durée de 20 μs dans la version initiale du standard ; pour l'amendement 802.11a et 802.11g, cette durée est réduite à 9 μs.

Dans 802.11, l'étape de Carrier-Sense (CS) requise par CSMA est effectuée à travers des mécanismes physiques et virtuels. Le mécanisme physique est effectué grâce au PLCP, détaillé dans la section précédente.

Le mécanisme virtuel est effectué quant à lui à travers une opération de réservation du canal annonçant l'usage du médium aux autres stations. Un échange de trame Ready-To-Send (RTS) et Clear-To-Send (CTS) est effectué avant l'envoi des données. Les trames RTS/CTS contiennent une en-tête 802.11 indiquant la Durée (Duration) et l' Identification (ID) qui définit la période de temps pour laquelle le médium doit être réservé, ce qui correspond à l'envoi des données et à la réception de l'acquittement. Toutes les stations qui sont à portée de réception de la station émettrice (qui envoie le RTS) ou de la station réceptrice (qui envoie le CTS) peuvent ainsi obtenir l'information sur l'usage du canal. Dans ce cas, les stations mettent à jour un vecteur d'allocation réseau (Network Allocation Vector - NAV) qui maintient en mémoire la prédiction du trafic futur sur le réseau. L'échange RTS/CTS est détaillé sur la figure 2.9. Cet échange est toutefois faculatif, et le choix de son utilisation est laissé à la discrétion des utilisateurs.

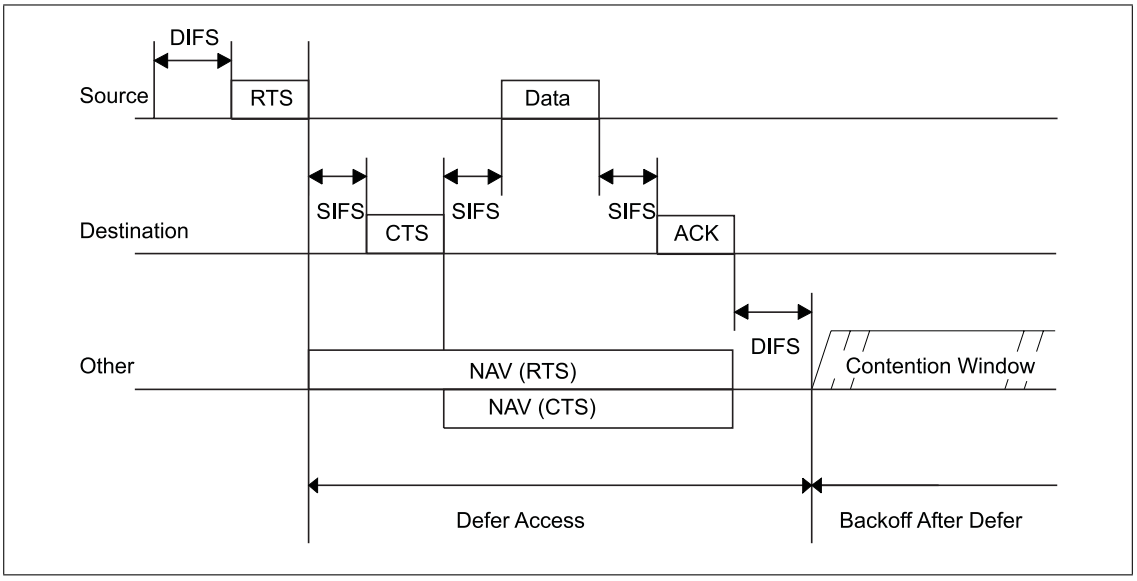


Figure 2.9 – Mécanisme de Virtual Carrier Sense : RTS/CTS. (Extrait des spécifications IEEE)

Les intervalles entre les trames 802.11 sont appelés InterFrame Space (IFS). Une station utilise donc l'état du canal grâce au Carrier Sense durant un intervalle dédié. Quatre intervalles différents sont utilisés : SIFS, short interframe space, PIFS, PCF interframe space, DIFS, DCF interframe space, et EIFS, extended interframe space. La figure 2.10 montre les relations entre les différents types d'intervalle.

L'intervalle entre une transmission et son acquittement est fixé à SIFS, dont la

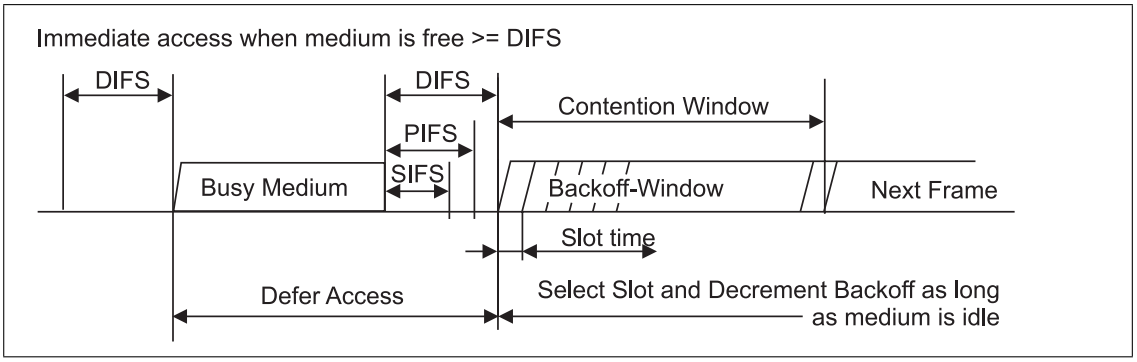


Figure 2.10 – Relation entre les différentes unités temporelles IFS. (Extrait des spécifications IEEE)

valeur est fixée à 10 μs pour 802.11-802.11b, et 16 μs pour 802.11a. Dans le cas d'un mode de compatibilité pour 802.11g, SIFS est fixé à 10 μs.

EIFS est l'intervalle correspondant à une expiration indiquant une erreur de transmission suite à la non réception d'un acquittement ; cette valeur est calculée à partir de la formule suivante :

$$EIFS = SIFS + (8 * ACKSize) + Preambule + PLCPHeader + DIFS$$

L'intervalle de silence à respecter avant une tentative de transmission est fixé à DIFS. DIFS se calcule à partir de la durée d'une slot et de SIFS. Ainsi pour 802.11/802.11b, $DIFS = 2 * slot + SIFS = 2 * 20 + 10 = 50\mu s$; Pour 802.11a, $DIFS = 2 * 9 + 16 = 34\mu s$.

Ainsi, après un intervalle DIFS (ou EIFS si la dernière trame était erronée) la station doit générer un backoff aléatoire pour effectuer une attente supplémentaire, en utilisant un décompte du backoff. Celui ci est composé d'un nombre aléatoire de slots, calculé suivant :

$$Backoff = Random() * SlotTime$$

où Random() est une fonction de tirage uniforme d'un entier entre l'intervalle $[0, CW[$. CW étant un entier représentant la fenêtre de contention (ou Contention Window) variant entre [CWmin, CWmax], deux paramètres fonction du nombre de machines théoriques admises en contention. Le mécanisme de backoff est détaillé sur la figure 2.11.

L'évolution de la fenêtre de contention (CW) suit un algorithme de Binary Exponential Backoff (BEB), c'est à dire que sa valeur est doublée à chaque tentative d'envoi avortée suite à un canal occupé. Lorsque CW atteint CWmax, la valeur reste maximale jusqu'à ce que le processus soit remis à zéro par l'envoi réussi d'une trame. En 802.11/802.11b, CWmin est fixé à 31 slots, en 802.11a/g CWmin est fixé à 15 slots. La valeur de backoff maximale est fixé à $CWmax = 1024\ slots$. L'évolution des valeurs de CW est visible sur la figure 2.12.

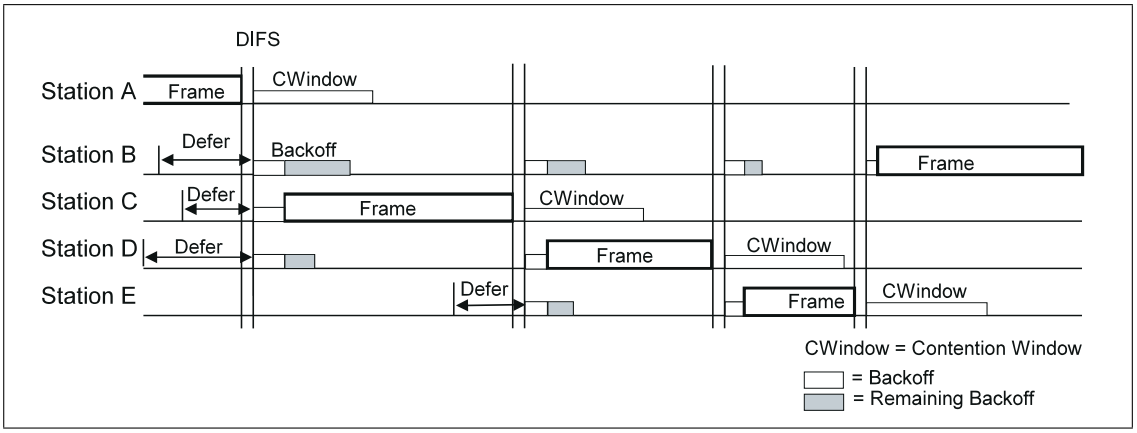


Figure 2.11 – Mécanisme de backoff. (Extrait des spécifications IEEE)

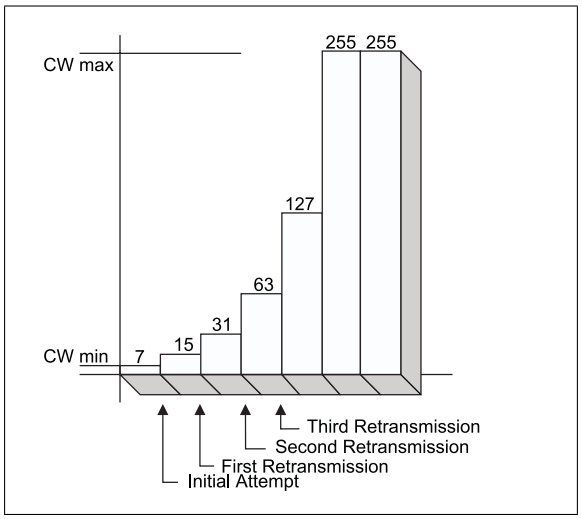


Figure 2.12 – Evolution de la Contention Window CW selon le Binary Exponentiel Backoff. (Extrait des spécifications IEEE)

Le fonctionnement minimal de DCF est résumé sur la figure 2.13. Il offre donc un mécanisme distribué et automatique de partage du médium grâce à la méthode d'accès CSMA, et aux techniques utilisées pour la réduction des collisions (Collision Avoidance CA). Le Carrier Sense, à la fois physique (fourni par le PLCP) et virtuel (fourni par les en-têtes de l'échange RTS/CTS) permet de déterminer l'occupation du canal (positionnement du NAV).

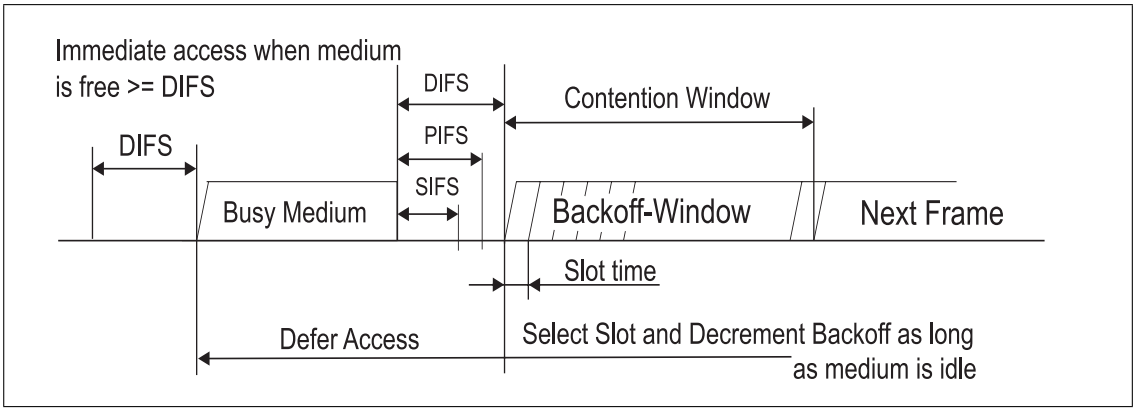


Figure 2.13 – Fonctionnement de DCF. (Extrait des spécifications IEEE)

2.2.2 Réflexions sur la capacité et l'équité des réseaux 802.11

Dès 1998, Cali et Conti [45] étudient la capacité des réseaux à la norme 802.11. Ils identifient un écart substantiel entre la capacité théorique et réelle des réseaux, principalement dû à l'utilisation de la fenêtre de contention. En effet, la valeur moyenne obtenue en usage réel ne permet pas d'atteindre les bornes de capacité du réseau 802.11. Cette valeur moyenne de contention, CW, correspond à un certain nombre fixe de stations actives en concurrence sur le canal ; or, par définition, les stations n'émettent pas toutes simultanément, et certaines stations peuvent garder le silence pour une durée variable. Les auteurs proposent donc de rendre la fenêtre de contention variable en fonction du nombre de stations actives sur le réseau, et démontrent ainsi que les bornes de capacité peuvent être atteintes.

Cependant, au-delà de la simple évaluation de la capacité du canal, l'étude des performances des réseaux 802.11 a révélé une anomalie de performance [63] dans le cas de l'utilisation de modulations différentes pour la transmission sur le canal. Si la méthode d'accès à base de CSMA/CA de 802.11 garantit un accès au canal équitable à long terme pour l'ensemble des stations, elle ne garantit aucunement une équité à court terme : les machines ayant utilisé le canal pour une plus longue durée à cause de leur plus faible débit pénalisent les machines disposant d'un plus haut débit. Dès lors, la bande passante est donc la même quel que soit le débit utilisé sur le canal, ce qui constitue l'anomalie de performances.

Plusieurs solutions ont été proposées pour résoudre de manière conjointe l'anomalie de performances et augmenter la capacité du canal. Dans [53], les auteurs évoquent 3 familles de solutions pour résoudre l'anomalie de performance : la fragmentation de paquets, l'aggrégation de paquets et l'adaptation de la fenêtre de contention.

La fragmentation de paquet consiste à réduire la taille individuelle des paquets émis en utilisant diverses solutions : soit une réduction du MTU (fragmentation au

niveau IP), soit au niveau 802.11 (qui offre un mécanisme de fragmentation de niveau 2). Iannone explore cette dernière solution dans sa méthode d'accès SDT.11b [41]. A chaque débit correspond une taille de donnée maximale pour une modulation donnée. La durée d'occupation du canal est donc réduite, ce qui tend à réduire l'anomalie de performance. Cependant ce type de solution offre un rendement faible pour un nombre conséquent de stations à débit réduit : l'overhead des en-têtes et des acquittements supplémentaires est trop important dans ce cas précis.

Une autre solution a été identifiée et correspond à des techniques d'agrégation de paquets. Le principe est d'augmenter le temps d'utilisation du canal par les stations utilisant un haut débit en concaténant un ou plusieurs paquets à la suite de l'autre. Cette solution permet d'augmenter sensiblement les débits en l'absence de concurrence d'accès, et permet de résoudre l'anomalie en présence de clients à faible débit. Cette solution a été mise en oeuvre dans plusieurs publications [31] [53].

Paradoxalement, la course aux débits des constructeurs de chipset WiFi a conduit au même type de solution. L'ensemble des constructeurs a adopté le Frame Bursting, ou mode rafale. Par rapport au régime de fonctionnement usuel de 802.11 (figure 2.14), cette solution repose sur la conservation de l'usage du canal par les stations pour émettre une suite de paquets de données, qui seront acquittés au fur et à mesure (figure 2.15) . Cette solution au problème de l'anomalie de performance constitue même l'un des apports de l'amendement 802.11e, qui suggère l'introduction du paramètre TXOP, correspondant à la durée temporelle maximale d'une séquence de burst consécutifs. Atheros a également proposé le Fast Frame, une technique visant à réduire l'overhead de l'encapsulation radio, par l'utilisation d'une seul préambule et d'un seul PLCP pour une trame de taille double, comme indiqué sur la figure 2.16. Les techniques de bursting font partie d'ailleurs de la dernière norme en cours de ratification, 802.11n. Ce type de solution augmente le débit de manière efficace et significative, par contre, lors de la mise en concurrence de stations haut débit, le problème de l'iniquité à court terme n'est toujours pas résolu.

C'est pourquoi les solutions qui privilégient une modification de la fenêtre de contention apparaissent comme les plus intéressantes. Outre la possibilité d'approcher les bornes de débit offerte par le canal, elles permettent aussi d'éliminer le problème de l'anomalie de performance et de l'iniquité à court terme.

Dès 1996, Bianchi et al. [62] proposent un algorithme dynamique pour la fenêtre de contention basé sur une estimation des stations actives par la mesure des slots d'activité par les stations qui désirent émettre sur le canal. D'autres pistes sont explorées, comme l'adoption d'un mode p-persistant CSMA pour la méthode d'accès. Ici aussi, Cali et al. [26] proposent une adaptation dynamique de la probabilité d'accès p au canal en fonction du nombre de machines actives, nombre obtenu par mesure des temps de silence et mesure des temps de collisions. L'adaptation de la probabilité d'accès p permet d'atteindre les capacités théoriques du canal considéré.

Cependant, le problème principal reste l'évaluation en temps réel du nombre de

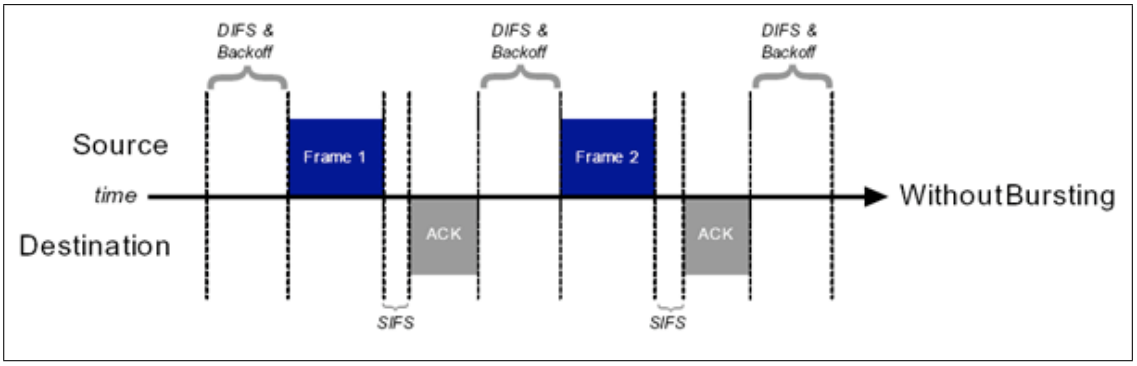


Figure 2.14 – Régime de fonctionnement usuel pour l'envoi de trame dans 802.11. (Extrait de la documentation commerciale Atheros)

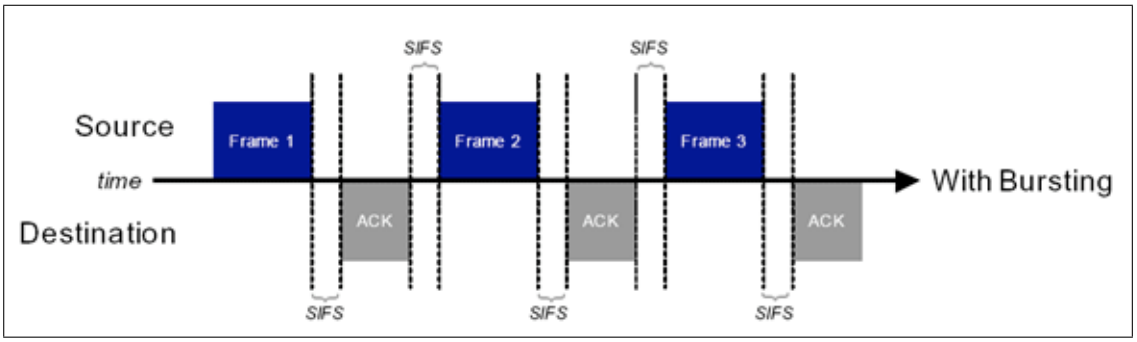


Figure 2.15 – Régime de fonctionnement en Frame Bursting. (Extrait de la documentation commerciale Atheros)

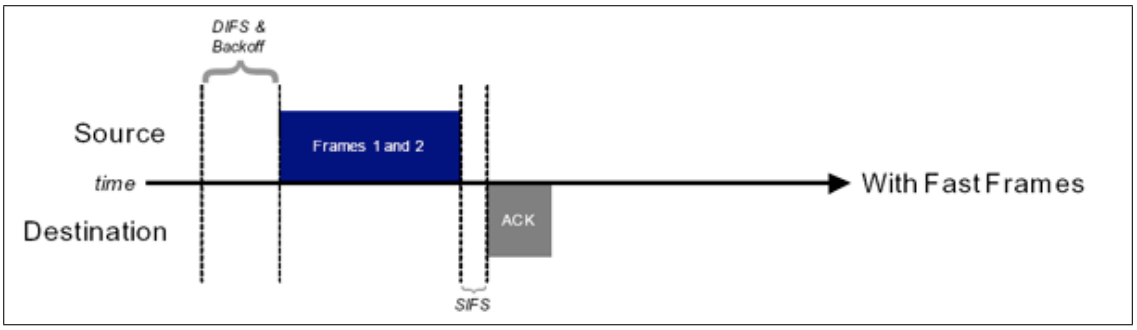


Figure 2.16 – Régime de fonctionnement en Fast Frame. (Extrait de la documentation commerciale Atheros)

stations actives sur le canal pour l'optimisation de la fenêtre de contention. Cali et al

proposent une formulation qui permet l'évaluation du nombre de stations actives à partir du nombre de slot vide détectés durant une durée de transmission virtuelle, caractérisée par le temps entre deux acquittements consécutifs. Bianchi et al. ont proposé dans [28] l'utilisation des filtres de Kalman pour déterminer le nombre de stations en concurrence sur le réseau. Ma et al [27] proposent d'utiliser le point d'accès dans le mode d'infrastructure pour collecter les informations sur les stations mises en concurrence, pour ensuite les rediffuser. Dans [34] Bonomi et Al. proposent une amélioration de DCF appelée Asymptotically Optimal Backoff, qui utilise le taux de slots utilisés ainsi que la taille moyenne des tailles transmises pour adapter la fenêtre de contention.

D'autres algorithmes évitent la phase critique d'évaluation du nombre d'hôtes en adoptant des politiques plus simples pour l'adaptation de la fenêtre de contention. Aad et Al. ont proposé la méthode Slow Decrease dans [50] : le principe est de diviser la fenêtre de contention par 2 au lieu de la réinitialiser à sa valeur d'origine après une transmission réussie, ce qui améliore l'efficacité par rapport à DCF mais agrave grandement l'inéquité de par l'usage de fenêtres de contention différentes pour chaque client. Kwon et Al. définissent Fast Collision Resolution [65] : la fenêtre de contention est doublée pour toute station qui détecte une collision ou perd une contention ; elle tire alors une nouvelle valeur pour son compteur de backoff. Pour décrémenter le temps passer en backoff, les stations peuvent le faire de manière exponentielle lors de l'observation de slots vides. Seulement, les stations qui ont réussi à transmettre reviennent toujours à la valeur initiale de CW minimale. Il en résulte là encore une iniquité à très court terme entre les stations qui ont réussi à accéder au canal et celle qui sont en attente, et dont l'attente peut être importante. La variante Fairly Scheduled Fast Collision Resolution (FS-FCR) des mêmes auteurs limite la fenêtre de contention au bout d'un certain nombre de tentatives de retransmission.

L'essentiel de ces propositions utilise la détection des collisions (à savoir, la non-réception d'un acquittement) pour effectuer un contrôle dynamique de la fenêtre de contention, et donc indirectement de la charge ; en effet, aucune de ces méthodes ne permet de distinguer les collisions des trames perdues ni ne peut gérer l'effet de capture [23]. De plus, fondamentalement, le fait d'attendre les collisions pour agir sur la fenêtre de contention constitue en soit un manque d'optimalité, surtout dans le cadre d'une méthode d'accès qui vise à réduire les collisions.

L'équipe DRAKKAR a proposé en 2005 une méthode d'accès essayant de répondre à l'anomalie de performance, Idle Sense [36]. Le principe de cette méthode repose sur une gestion de la fenêtre de contention dynamique permettant de garantir une équité temporelle ; cependant, la technique de mesure du nombre d'hôtes est basée cette fois sur les slots de silence observés sur le canal entre deux transmissions réelles. Ainsi, à la différence de [26], la méthode n'exige pas un profond changement de la méthode d'accès actuelle puisqu'elle prend en compte des transmissions réelles. Enfin, l'algorithme de gestion de l'évolution de la fenêtre de contention repose sur le principe connu de l'Additive Increase Multiple Decrease (AIMD) largement étudiée dans la

littérature [25].

Au travers de ces différentes solutions aux problèmes de capacité des réseaux sans fil, nous pouvons mesurer tout l'impact des choix techniques de la couche MAC sur les performances effectives de ceux-ci : de simples modifications de la gestion de la fenêtre de contention ont permis de montrer un réel gain dans la capacité offerte par les réseaux. Ces modifications sont d'autant plus efficaces quand elles tiennent compte de l'état physique du canal : une meilleure coopération entre les couches physiques et la couche MAC conduisent donc à une amélioration des performances.

Cependant, à ce stade les réseaux sans fil ne sont encore qu'un médium de communication brut : seul l'apport du protocole IEEE 802.11 permet d'obtenir des modes de fonctionnement distincts.

2.3 Fonctionnalités de 802.11

La méthode d'accès DCF est mise en oeuvre durant les différents modes de fonctionnement de IEEE 802.11. Ceux-ci regroupent différents usages des réseaux locaux : équivalent sans fil d'un switch (mode infrastructure), connexion point à point ponctuelle (mode ad hoc), et interconnexion de plusieurs réseaux (WDS). L'ensemble de ces modes de fonctionnement est orienté autour du protocole IEEE 802.11.

2.3.1 Protocole 802.11 et modes de fonctionnement

Les couches PHY et MAC assurant la transmission des données de manière brute, un protocole est utilisé dans IEEE 802.11 pour gérer les différents types de messages (données, signalisation), ainsi que les fonctions habituelles des couches protocolaires (gestion des erreurs, gestion de la fragmentation, identification de l'émetteur, du récepteur).

Le protocole en lui-même est une encapsulation dotée d'une en-tête au format défini par IEEE 802.11. Les trames sont constituées d'une en-tête d'une longueur de 30 octets (cette longueur pouvant varier avec certaines versions du protocole) , d'un corps, et d'un FCS (Frame Sequence Check) permettant d'évaluer la validité de la trame et d'effectuer une correction d'erreur par retransmission de la trame impactée.

L'ensemble des paramètres de l'en-tête sont explicités en annexe C.

Mode Infrastructure

Le mode infrastructure repose sur l'utilisation d'un équipement au rôle particulier dans le réseau 802.11, le point d'accès (Access Point ou AP). L'AP émet des balises (ou beacons), trames de type gestion contenant les informations minimales permettant aux clients (appelés stations, STA) de se connecter au réseau : nom du réseau, type de modulation supportée, information temporelle.

Une fois ces informations recueillies, les stations entrent dans un cycle d'authentification (par l'envoi de trame d'authentification) et d'association (trame d'association). Durant le cycle, représenté sur la figure 2.17, les trames envoyées sont principalement les trames de trafic ainsi que des trames de réassociation parfois nécessaire. La désassociation apparait lorsque la station client émet une trame, ou lorsque la station est inactive.

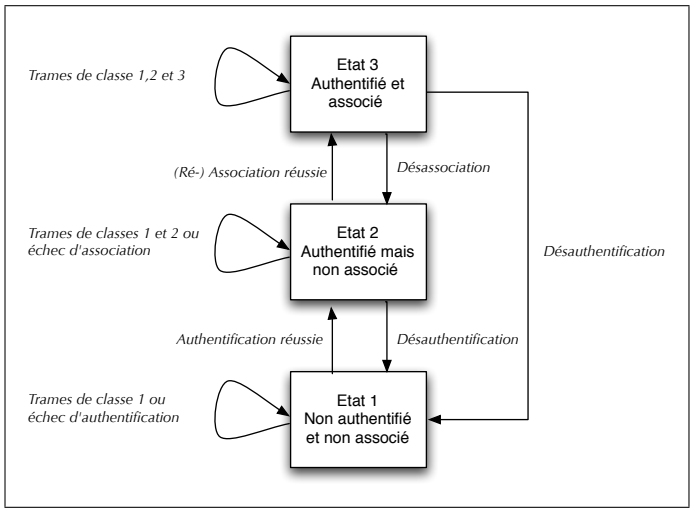


Figure 2.17 – Machine à état 802.11.

Les points d'accès forment individuellement avec les stations associées une cellule, ou BSS. Ils peuvent également être interconnectés entre eux, et dans ce cas, ils forment un système étendu ou Extended Service Set.

Un utilisateur nomade peut donc passer d'un BSS à un autre, tout en restant dans le même réseau. La norme prévoit en théorie que les AP peuvent être reliés entre eux par un système de distribution, mais aucun détail d'implémentation n'est donné.

Les champs adresses sont utilisés pour décrire : la station émettrice, la station réceptrice, et enfin, le BSSID, c'est à dire l'identifiant du BSS dans lequel les stations sont associées. En général, cet identifiant est en vérité l'adresse sur 48 bits du point d'accès, qui est donc réutilisé dans les champs adresses.

Mode ad hoc

Le mode ad hoc fait disparaître le point d'accès, chaque station dispose donc des mêmes règles de fonctionnement. Pour gérer la visibilité du réseau et l'arrivée de nouveaux participants, la norme IEEE 802.11 indique que les stations doivent recourir à un mode d'émission des balises distribué entre tous les participants. Les modalités de cette distribution ne sont pas explicitées, les constructeurs procèdent donc par expi-

ration d'un compteur de non réception de balise, qui incite la station à émettre à son tour une balise.

Dans ce mode, les champs "adresse" sont utilisés de manière similaire au mode infrastructure, si ce n'est que le champ adresse 3, utilisé auparavant par l'adresse du BSSID, est remplacé par une adresse générée aléatoirement par la première station qui a initié le réseau ad hoc.

Mode WDS

Ce mode est utilisé pour relier des points d'accès entre eux. Il met en oeuvre un quatrième champ d'adresse dans l'en-tête 802.11. Dès lors, un champ est utilisé pour l'émetteur, un champ pour le récepteur, puis un champ pour le ré-émetteur sans fil, et un dernier pour le récepteur sans fil. De cette façon, WDS peut être utilisé de manière transparente en tant que pont au sens Ethernet (bridge) entre deux réseaux filaires.

Ce mode est généralement utilisable sur des équipements identiques, car il n'y a pas de cycle d'association ou de désassociation. Les constructeurs utilisent donc une initialisation spécifique, ce qui introduit généralement des incompatibilités entre les différents équipements. De ce fait, ce mode est peu utilisé.

En outre, utilisé pour relier deux réseaux sans fil, il introduit une perte de bande passante, puisque la bande passante est effectivement utilisée pour ré-émettre le trafic reçu vers une autre station.

2.3.2 Calcul des débits théoriques

Connaissant les caractéristiques physiques des transmissions, ainsi que la méthode d'accès retenue pour les réseaux IEEE 802.11, et les caractéristiques du protocole 802.11, il nous est facile d'évaluer théoriquement les débits moyens offerts par réseaux 802.11 en régime saturé pour une seule station.

La méthode consiste à établir un bilan temporel sur une transmission donnée : chaque transmission sur le canal est caractérisée par un temps inter-trame (DIFS, SIFS...), par une contention éventuelle (tirage aléatoire d'un nombre de slots parmi $CWmin$), d'une en-tête PLCP transmis à une vitesse constante, puis d'une paquet 802.11 doté d'une en-tête d'une certaine taille (fonction de sa nature : paquet de données, acquittement...).

Un flot UDP transmis par une station peut se définir de la manière suivante :

$T_{UDP} = T_{DIFS} + T_{Contention} + T_{DATA} + T_{SIFS} + T_{ACK}$

où T_{DIFS} et T_{SIFS} sont constants pour une technologie 802.11 donnée. Les autres temps se calculent suivant :

$T_{DATA} = T_{PLCP} + T_{802.11} + T_{Ethernet}$

$T_{ACK} = T_{PLCP} + T_{802.11-ACK}$

Pour la contention, nous considérons la valeur moyenne d'un tirage, sachant qu'il n'y a pas pas de collision, la station étant seule. Dès lors :

$T_{Contention} = \frac{CWmin}{2} * T_{slot}$
où T_{slot} est la durée d'un slot.

Les durées $T_{802.11}$, $T_{802.11-ACK}$, $T_{Ethernet}$ correspondent à la quantité d'octets transmis à la vitesse considérée. Donc, en général : $T = \frac{Data}{Vitesse}$. Les durées T_{PLCP} sont fonction de la technologie 802.11, mais restent constantes, de manière à ce que toutes les stations à portée puissent décoder l'en-tête et puissent positionner le vecteur d'allocation NAV pour bloquer leur émission concurrente.

Après calcul, on obtient les tableaux de débits situés en annexe D dont nous résumons les résultats numériques pour 802.11b (2.3), 802.11a (2.4), 802.11g (2.5) et en mode de compatibilité 802.11b/g (2.6). Nous y avons également reporté le pourcentage du temps d'émission passé en contention ($T_{Contention}$).

$T_{Contention}$	2.4%	4.6%	11%	18,1%
Modulation	1Mbits/s	2Mbits/s	5.5Mbits/s	11Mbits/s
Débit 802.11	0.92Mbits/s	1.77Mbits/s	4.24Mbits/s	7.02Mbits/s
Efficacité protocolaire	92%	88%	77%	63%

Table 2.3 – Comparaison entre modulation utilisée et débit théorique attendu, pour 802.11b

$T_{Contention}$	3%	4.4%	5.6%	7.9%
Modulation	6Mbits/s	9Mbits/s	12Mbits/s	18Mbits/s
Débit 802.11	5.42Mbits/s	7.82Mbits/s	10.04Mbits/s	14.03Mbits/s
Efficacité protocolaire	90%	86%	83%	77%
$T_{Contention}$	9.8%	13.4%	16.2%	17.4%
Modulation	24Mbits/s	36Mbits/s	48Mbits/s	54Mbits/s
Débit 802.11	17.50Mbits/s	23.91Mbits/s	28.78Mbits/s	30.88Mbits/s
Efficacité protocolaire	72%	66%	59%	57%

Table 2.4 – Comparaison entre modulation utilisée et débit théorique attendu, pour 802.11a

$T_{Contention}$	3%	4.4%	5.7%	8%
Modulation	6Mbits/s	9Mbits/s	12Mbits/s	18Mbits/s
Débit 802.11	5.45Mbits/s	7.88Mbits/s	10.14Mbits/s	14.23Mbits/s
Efficacité protocolaire	90%	87%	84%	79%
$T_{Contention}$	10%	13.7%	16.6%	17.9%
Modulation	24Mbits/s	36Mbits/s	48Mbits/s	54Mbits/s
Débit 802.11	17.81Mbits/s	24.50Mbits/s	29.64Mbits/s	31.86Mbits/s
Efficacité protocolaire	74%	68%	61%	59%

Table 2.5 – Comparaison entre modulation utilisée et débit théorique attendu, pour 802.11g

L'analyse de ces résultats apporte quelques éclaicissements sur les performances réelles des réseaux sans fil à la norme 802.11. Dans le cadre de ces calculs, nos hypothèses de travail sont optimistes : un seul hôte sur le réseau, émettant un trafic UDP

$T_{Contention}$	2.8%	3.9%	4.9%	6.5%
Modulation	6Mbits/s	9Mbits/s	12Mbits/s	18Mbits/s
Débit 802.11	5.02Mbits/s	7.01Mbits/s	8.75Mbits/s	11.63Mbits/s
Efficacité protocolaire	83%	77%	72%	64%
$T_{Contention}$	7.9%	9.9%	11.3%	11.9%
Modulation	24Mbits/s	36Mbits/s	48Mbits/s	54Mbits/s
Débit 802.11	14.15Mbits/s	17.70Mbits/s	20.24Mbits/s	21.25Mbits/s
Efficacité protocolaire	59%	49%	42%	39%

Table 2.6 – Comparaison entre modulation utilisée et débit théorique attendu, pour le mode de compatibilité 802.11b/g

à destination d'un autre hôte. De plus, la contention retenue est une valeur moyenne, qui ne tient pas compte des problèmes éventuels d'aléatoirité qui augmenteraient artificiellement le tirage de certains valeurs.

Pourtant déjà, nous observons que le coût protocolaire de l'ensemble PLCP et 802.11 constitue un frein à l'efficacité. Mais plus encore, c'est l'héritage historique des normes successives, ainsi que le maintien de la rétro-compabilité (capacité d'un point d'accès 802.11g à autoriser la connexion d'hôtes 802.11b) qui constitue une part importante de l'overhead protocolaire (particulièrement visible sur le mode de compatibilité, où l'utilisation d'une modulation de 54Mbits/s se traduit par 39% du débit auquel l'utilisateur pourrait s'attendre).

Cependant, cette courte étude montre également le poids du choix de la méthode d'accès dans les débits obtenus, notamment l'influence du temps passé en contention ($T_{Contention}$). La capacité d'amélioration de la méthode d'accès est donc primordiale pour les futurs réseaux sans fil.

2.3.3 Réflexions sur les usages de 802.11

Les différents modes de fonctionnement offrent des fonctionnalités séduisantes. Par exemple, le mode infrastructure, lorsqu'il est utilisé en présence de multiples points d'accès, offre la perspective d'une utilisation nomade de l'accès réseau. Le mode ad hoc, lui, ouvre la porte à des problèmes de routage, notamment sous la vision MANET.

Mobilité dans les réseaux 802.11

La mobilité des stations a été un problème largement étudié. En effet, débarrassé des connectiques filaires, les utilisateurs aspirent à la même fonctionnalité qu'avec un téléphone portable : à savoir le maintien de la connectivité au cours du déplacement. Dans le cadre de 802.11, il s'agit principalement de maintenir la connectivité de l'interface 802.11 d'un terminal mobile.

Il existe plusieurs types de mobilité, étudiés notamment dans plusieurs thèses, dont [67]. Du point de vue de l'utilisateur, le critère est la connexion à un nouveau

réseau ou sous-réseau, ce qui implique d'effectuer des opérations au niveau de 802.11, ainsi qu'au niveau supérieur (demande d'une adresse par DHCP, obtention d'une nouvelle adresse IP etc...), ou la connexion à un même sous-réseau (conservation des paramètres IP notamment). Dans le cas d'un changement de réseau, on parle de macro-mobilité ; dans le cas d'une mobilité dans un même sous-réseau, on parle de micro-mobilité.

Nous nous intéresserons principalement à la micro-mobilité dans le cadre du protocole IEEE 802.11, ou plus exactement ce que les auteurs de [61] ont appelé le handover de niveau 2. En effet, le fait de maintenir une connexion entre deux points d'accès d'un même sous-réseau IEEE 802.11 va impliquer des exigences temporelles afin de maintenir la continuité des opérations supérieurs à la couche 802.11, typiquement IP, UDP, TCP. Les techniques de buffering permettent d'éviter les pertes de données, mais certains types de trafic, notamment la voix sur IP (VoIP) ne peuvent tolérer la présence de délai supplémentaire lors d'un déplacement.

Lors de la commutation (handoff) d'un point d'accès à un autre dans le mode infrastructure, les points d'accès forment un Extended Service Set (ESS) composés de Basic Service Set (BSS) indépendants. Cependant, la norme 802.11 n'ayant pas défini de manière générique le système de distribution, le handoff 802.11 consiste dans les implémentations actuelles des stations en les opérations suivantes, comme décrit par Mishra [16] :

- évaluation du rapport signal bruit des balises du réseau actuel, notification de diminution ;

- lancement d'un scan des autres balises indiquant la présence d'autres points d'accès ; cette opération peut être soit active soit passive ; si elle est active, des paquets de sondes sont envoyés par la station pour interroger les différentes canaux ; dans ce cas, l'opération peut entraîner une perte de paquet, car le changement de canal n'est pas instantané, comme l'a montré Bahl et Al dans [37]. Certains cartes implémentent un mode passif qui assure un scan durant des phases de blocage(NAV par exemple).

- ré-authentification : la station s'authentifie à nouveau en fonction d'une liste de priorités des AP. La ré-authentification peut être accomplie grâce au transfert des informations d'accréditation par un protocole dédié, tel Inter Access Point Protocol (IAPP) [68].

La figure 2.18 illustre le processus de Hand-off utilisé dans les réseaux 802.11.

Mishra et Al. ont étudié de manière empirique le handover dans les réseaux 802.11 [16] et ont évalué les durées temporelles de l'ensemble de ces opérations. Les auteurs ont relevé une grand disparité dans les délais de transition - entre 200 et 1000 ms. Sachant qu'un flux temps réel transportant une conversation téléphonique ne peut supporter un décalage de plus de 200ms, plusieurs auteurs tirent la sonnette d'alarme, notamment l'équipe de Henning Schulzrinne [22], inventeur du protocole SIP dédiée à la gestion de sessions de flux multimédia temps réels.

Les délais identifiés par Mishra sont principalement le délai de sonde (Probe De-

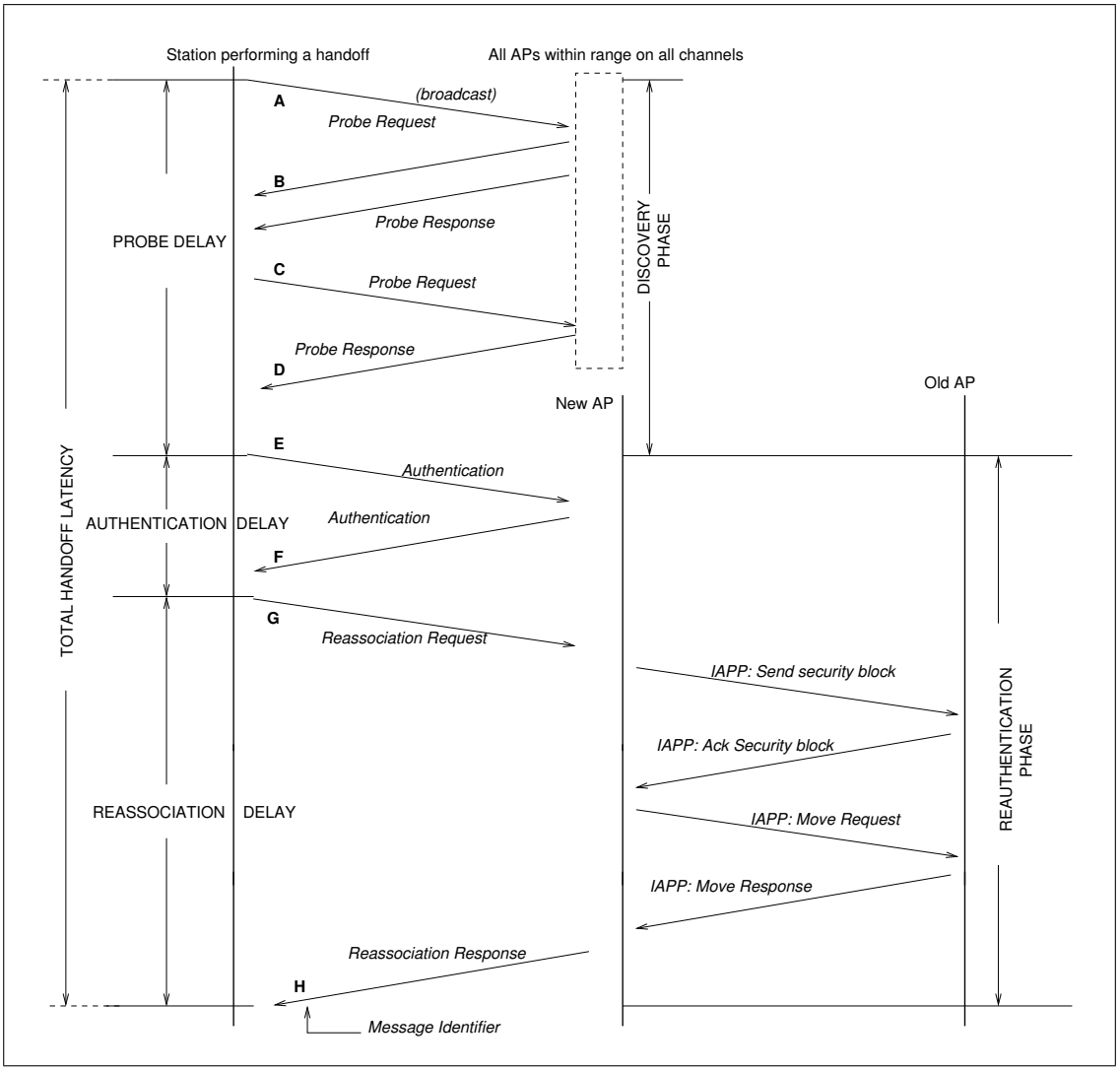


Figure 2.18 – Procédure de Handoff utilisée dans les implémentations 802.11 (extrait de la proposition IAPP).

lay), qui occasionne un échange de 3 à 7 trames ; le délai d'authentification consiste en un échange de 3 à 4 trames ; enfin, le délai de réassociation est aussi composé de 3 à 4 trames 802.11. Les auteurs mentionnent aussi le délai dit de Bridging, correspondant au rafraîchissement des tables ARP des points d'accès utilisés dans la procédure de Handoff. Pour les auteurs, le délai de sonde (Probe request) constitue l'essentiel (90%) de la perte de temps passé par la carte pour effectuer son handoff. D'autres expériences [32] confirment la même origine du délai lors du handoff : les phases de détection et de recherche prennent trop de temps.

Le protocole 802.11 exerce donc un overhead trop important lors du handoff entre point d'accès, notamment à cause des phases de détection. Plusieurs auteurs ont proposé des solutions.

Mishra et Al. [16] proposent une réduction du délai en réduisant le délai d'attente d'une réponse à une trame de sonde et des heuristiques visant à utiliser un minimum de phases de détection active pour augmenter le temps de réponse. Les auteurs proposent ensuite d'utiliser des graphes de voisinage (Neighbor Graphs) dans [35] et ont proposé une modification d'IAPP en conséquence. Cette modification inclut un système de cache proactif : il utilise l'information sur le voisinage pour envoyer de manière proactive le contexte au voisin concerné. Ainsi, il dissocie le contexte du processus de réassociation, ce qui permet une réduction drastique des délais de handoff. Par la suite, cette modification suggérée a été retenue dans la version finale d'IAPP. Plus tard, les auteurs ont étendu le concept de graphe à l'authentification en proposant un modèle d'authentification distribué sur les différents points d'accès du réseau dans [54].

Un mécanisme de cache a été également proposé par Shin et al. dans [22] : il consiste en la réutilisation des informations obtenues lors d'un scan actif précédent, comme proposé sur la figure 2.19. Les auteurs proposent également de profiter du caractère adjacent des canaux et d'effectuer un scan total des canaux pour construire un masque qui sera utilisé lors des handoffs suivants (figure 2.20). Ces propositions ne sont efficaces que lorsque la densité du réseau est suffisante, et réduise la réactivité des clients à l'apparition d'un nouveau point d'accès.

Velayos et Al. [32] ont une approche différente : ils proposent une modification du comportement des cartes 802.11 en déclenchant le phase de sondage dès que la perte de paquets en l'absence de collision est identifiée. Les auteurs démontrent ainsi que la phase de scan peut être déclenchée dès que 3 paquets consécutifs sont perdus en l'absence de collision, ce qui réduit pour les auteurs le délai de handoff de 900 ms à 3 ms (en l'absence d'authentification), en présence de trafic par la station ; en l'absence de trafic, les auteurs préconisent un raccourcissement du délai de réémission des balises par le point d'accès à 60ms (alors qu'il est d'ordinaire fixé à 100 ms). Ainsi les délais de détection sont évidemment réduits en présence et en l'absence de trafic de la part de la station mobile.

Mhatre et Al. [40] approfondissent la thématique des déclencheurs (Triggers) permettant d'identifier un besoin d'effectuer un handoff. Ils proposent ainsi un ensemble de modifications des drivers de la carte cliente pour améliorer le handoff tout en s'affranchissant de modifications au niveau du point d'accès. Tout d'abord, ils utilisent la nature diffusante des ondes radio pour mesurer en permanence le niveau du rapport signal/bruit de tous les points d'accès présents sur le canal et des canaux adjacents (valable uniquement en 802.11b/g, où les canaux ne sont pas strictement indépendants). Le scan actif par sondage ne doit être réalisé que lorsque aucune information n'a pu être récupérée de manière passive. Enfin, le sens même du terme handoff est modi-

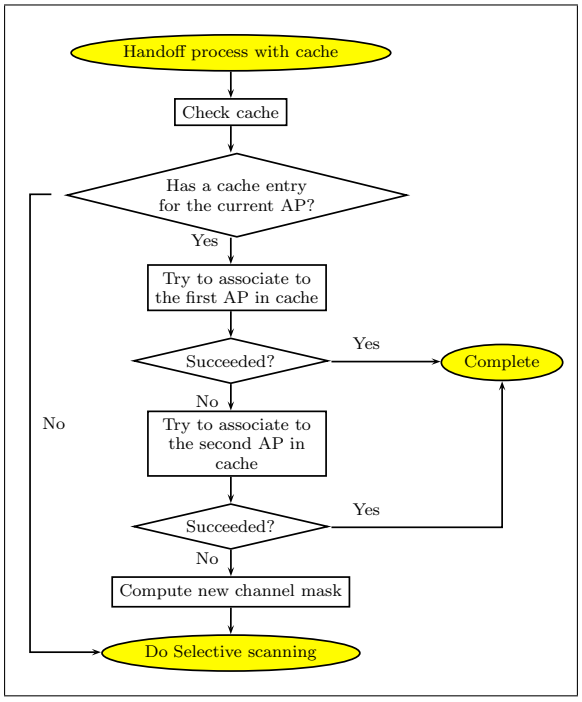


Figure 2.19 – Proposition de handoff basé sur un cache des sondages précédents. (Extrait de "Context Caching using Neighbor Graphs for Fast Handoffs in a Wireless Network", Mishra et al.)

fié car ici il s'agit plutôt d'améliorer la performance du client en présence de points d'accès multiples d'un même sous réseau. Les auteurs proposent ensuite et évaluent une série de déclencheurs sur les valeurs de signal reçu pour déclencher un changement de point d'accès. Ainsi, ils évaluent un algorithme à base de détection de beacon, un algorithme à seuil, puis à base d'hystérésis, et enfin un algorithme analysant les tendances. Au final, les délais de handoff obtenus sur leur plate-forme expérimentale indiquent un fort délai (entre 530 et 860 ms) pour les algorithmes à base de seuil et de détection de beacon. Il s'agit d'algorithmes réactifs, puisqu'ils initient un handoff lors d'un changement d'état. Pour les autres algorithmes, les délais sont nettement plus courts (140 à 450 ms), car il s'agit en fait d'algorithmes proactifs, initiant un handoff même lorsque la connexion est quand même existante.

Cependant, les méthodes précédentes reposent sur un handoff déclenché par le client. D'autres approches privilégient donc une évaluation de la mobilité du client pour préparer de manière proactive le handoff.

Dans la littérature, les chercheurs s'intéressent principalement à des modèles de mobilité, notamment dans le domaine des réseaux ad hoc pour véhicules. Plusieurs travaux dont [20] proposent des modèles de mobilité basés sur les propriétés de dé-

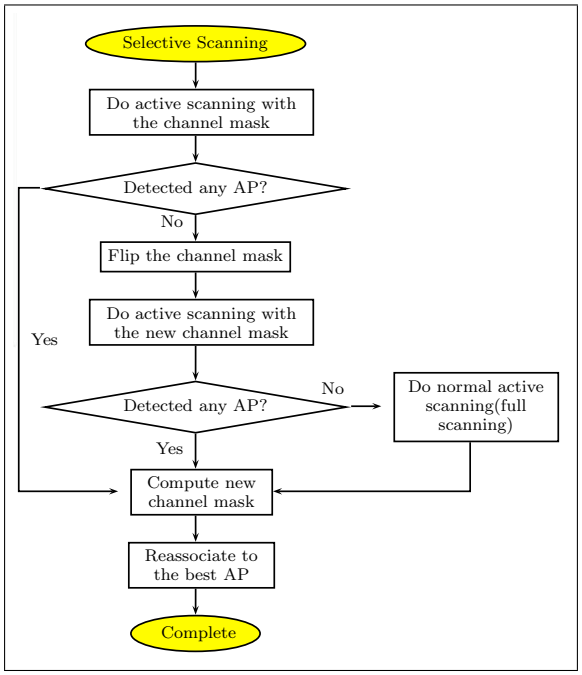


Figure 2.20 – Proposition de handoff basé sur un masque des canaux à scanner.(Extrait de "Context Caching using Neighbor Graphs for Fast Handoffs in a Wireless Network", Mishra et al.)

placement du mobile pour fournir des traces réseaux exploitables par des simulateurs. Mais ces modèles de mobilités peuvent être aussi utilisés pour anticiper les déplacements de l'utilisateur hors de la couverture d'un point d'accès ou des autres noeuds mobiles.

Cependant, même si des modèles de mobilité sont proposés, il n'existe pas à notre connaissance de proposition d'amélioration du handoff 802.11 bénéficiant d'une décision prise par le réseau lui-même, comme dans les réseaux cellulaires par exemple. Cependant, les réseaux 802.11 représentent donc une opportunité intéressante pour l'étude des mécanismes de micro-mobilité : en effet, la prise en compte des caractères physiques, comme le niveau du signal, ou encore la présence de balises permettent des améliorations substantielles de la micro-mobilité. Nous reviendrons plus tard sur le problème de la mobilité et sur l'apport de la prise en compte des couches inférieures, notamment au travers de nos contributions.

Les réseaux ad hoc dans la vision MANET

MANET, Mobile Ad hoc NETworks est un groupe de travail de l'IETF né en marge des travaux de recherche sur le problème de la mobilité. MANET vise en effet à résoudre

un autre problème lié à l'extension de la connectivité IP à des réseaux sans fil de machines autonomes, mobiles, où l'infrastructure de routage est constituée par les mobiles eux-mêmes. Dans les premières publications liées à MANET [64], et dans la RFC 2501, le groupe précise le contexte des recherches et notamment les applications envisagées : situation de crise, scénario militaire, ou encore réseaux de capteurs.

MANET a donné naissance à plusieurs groupes de protocoles de routages. Bien que 802.11 soit l'une des technologies sans fil évoquées lors de la mise en place des MANET, celle-ci est devenue au fil des années l'une des technologies les plus utilisées en expérimentation des réseaux MANET.

En effet, au cours des années de recherches, plusieurs types de protocoles ont été évoqués : proactifs, avec des routes entretenues périodiquement (FSR [29], OLSR [60], DREAM [66], DSDV [38]) ; réactifs avec construction des routes à la demande (AODV [43], DSR [21], RDMAR [47]) ; hybrides, avec une part locale de proactivité et de réactivité vis à vis de l'extérieur (ZRP [18], ToRA [55]) ; mais aussi hiérarchique, où certaines unités ont un rôle particulier (HSR [58], VSR [59]). Enfin, des algorithmes géographiques, utilisant l'information sur la position des noeuds mobiles ont été aussi développés (LAR [24], GPSR [46]).

La plupart de ces protocoles ont fait l'objet de simulation et d'expérimentation avec les réseaux sans fils 802.11.

Fish Eye Routing - FSR utilise la librairie d'émulation GloMoSim pour effectuer les simulations sur un réseau 802.11 dans [29] ; OLSR a fait l'objet de nombreuses simulations et implémentations, dont [56] qui présente à la fois une expérimentation et des travaux de simulations.

Les vérifications expérimentales, notamment sur OLSR, ou encore AODV ont conduit les chercheurs à s'interroger sur la prise en compte du médium. Dans [42], Lundgren et Al. s'interrogent sur l'existence de zones de diffusion grises, introduite par les protocoles de routage. Les auteurs identifient le problème comme provenant de la manière dont la norme effectue la transmission des paquets diffusés : ceux-ci, non acquittés, sont envoyés à un taux marginal plus faible permettant une transmission avec une probabilité d'erreur des plus faibles. Cependant, ce paramètre étant ignoré par les couches de routage supérieures, la zone de couverture réelle est nettement plus réduite et le routage théorique prévu n'est donc pas atteint dans la pratique. Les auteurs proposent de valider par exemple la bidirectionnalité des communications par un échange de message de découverte (HELLO).

Mais les recherches ne s'arrêtent pas là : en effet, outre la découverte des différents participants au réseau, l'étude du routage en environnement sans fil nécessite de repenser les métriques jusqu'à alors utilisées : en effet, les liens pouvant utiliser des modulations très variables, il peut être judicieux de limiter le routage multi-saut sur des liens à haut débit par quelques sauts à bas débits. Ainsi, plusieurs auteurs proposent un ensemble de nouvelles métriques (ETX [44], ETT [17] par exemple) qui exploitent les informations spécifiques aux transmissions sans fil (taux de perte de

paquets, modulation utilisée, ou encore occupation du canal).

Les travaux du groupe MANET sont fort révélateurs de la tendance globale concernant la recherche dans le domaine des réseaux mesh utilisant IEEE 802.11 : les éléments constitutifs de ces réseaux (méthode d'accès, routage) ont été conçus dans une totale indépendance. Cependant, lorsque les chercheurs associent des indices physiques au fonctionnement usuel des différents éléments, il en émerge systématiquement une amélioration des performances. Il est donc primordial de permettre des interactions entre les éléments constitutifs des réseaux sans fil.

2.4 Conclusions

Les réseaux sans fils à la norme 802.11 ou WiFi sont le fruit d'une construction complexe et progressive, issue du succès des structures en couches que sont les réseaux globaux de type Internet ou locaux comme Ethernet.

En effet, la conception en couches indépendantes mais réglementées par la norme 802.11 a permis le développement de nouvelles couches physiques tout en conservant sa couche MAC originelle ainsi que le protocole 802.11. Ainsi, les modulations ont progressé durant les amendements successifs 802.11b, 802.11a puis 802.11g : les constructeurs n'ayant qu'à modifier les composants dédiés à la partie radiofréquences et en conservant les logiques de la couche MAC et, le cas échéant, les pilotes du périphérique.

Cependant, la technique a atteint ses limites. Comme nous l'avons vu, chaque couche présente intrinsèquement des limitations : la couche physique doit être capable de gérer de mulitples modulations ; la couche mac doit être capable de maximiser la capacité utilisable ; enfin, les réseaux mesh changent radicalement les usages de 802.11.

Or, l'architecture en couche a ici atteint ses limites. La plupart des travaux de la communauté scientifique sur ces problèmes impliquent intrinsèquement une coopération plus étroite des couches : par exemple, la gestion des débits multiples est directement liée à l'anomalie de performances. Dès lors, il est inconcevable de dissocier couche physique et couche mac.

Dès lors, toute amélioration des performances ne peut passer que par des solutions impliquant plusieurs couches.

Les dernières évolutions de 802.11 en sont un exemple frappant : le standard 802.11n, toujours en cours de ratification, est l'un des premiers amendements à proposer des modifications simultanées des couches physiques, MAC, ou du protocole 802.11. Tout d'abord, le standard exploite les nouvelles capacités des technologies MIMO qui utilisent un nombre multiple d'antenne et donc de flux physiques pour augmenter le débit ; ajoutons à cela une modification de la couche MAC, visant à améliorer la capacité du canal, en privilégiant les techniques d'agrégation, proposées notamment pour résoudre les problèmes de capacité des réseaux 802.11. En outre, des mécanismes d'acquittement retardé vont permettre d'améliorer significativement l'uti-

lisation du canal en réduisant la surcharge protocolaire de 802.11, comme nos calculs l'ont montré.

Les constructeurs de matériels 802.11 ont donc migré, progressivement, d'un modèle en couche strict à un modèle plus souple intégrant des modifications sur l'ensemble des couches. Cependant, ces modifications ne visent qu'à améliorer les performances des modes de fonctionnement actuel, surtout dans un scénario de point d'accès. L'amendement 802.11e, visant à apporter de la qualité de service, a surtout était conçu dans une approche classique d'une connexion à un point d'accès : il met en oeuvre 4 classes de trafic, best-effort, background, voice et video, donc les paramètres MAC sont sensiblement différents (CWmin distinct, possibilité d'envoyer des paquets en rafale). Le but avoué est clairement une amélioration de la qualité en cas d'utilisation simultanée d'un point d'accès par plusieurs clients.

Pourtant, un amendement en cours de ratification essaye d'introduire le concept de mesh : il s'agit de 802.11s. Il essaie de standardiser un protocole de routage hybride, permettant à l'ensemble des vendeurs d'utiliser au choix un protocole de type AODV ou OLSR. Les premières implémentations (OLPC puis sa généralisation open802.11s[1].) reprennent les couches existantes pour simplement ajouter de nouveaux messages 802.11 et rajouter le routage, traditionnellement au niveau 3 du modèle TCP/IP, dans les couches 802.11.

C'est donc tout naturellement que les réseaux sans fil se dirigent vers une architecture inter-couches : cependant, à ce jour, l'architecture traditionnelle des systèmes d'exploitation semble limiter cette évolution. Celle-ci doit donc dépasser son modèle théorique pour offrir de nouvelles perspectives.

[1] http ://www.open80211s.org/

CHAPITRE 3

IMPLÉMENTATION DES RÉSEAUX SANS FIL 802.11 : AU-DELÀ DU MODÈLE THÉORIQUE

Lors de notre présente autopsie, nous avons étudié et identifié les principales composantes des réseaux sans fil locaux et nous avons présenté quelques uns des travaux de la communauté scientifique sur ces mêmes réseaux. Néanmoins, et la malgré l'ambition du groupe MANET, force est de constater que l'ensemble des contributions sur les réseaux sans fil est focalisé sur la gestion des couches physiques, les méthodes d'accès, la mobilité, le routage. Ceci est en partie dû à la complexité inhérente à la construction en couche de ces réseaux. Il n'y a pas eu de bouleversement des usages liés à la multiplication des réseaux sans fil : au contraire, ces réseaux sont aujourd'hui principalement utilisés comme réseau local chez les particuliers pour distribuer l'accès à Internet.

Pourtant, les exemples de nouveaux usages évoqués dans MANET, comme les réseaux meshs, sont quasiment inexistants en utilisation courante ; les rares initiatives sont l'oeuvre de groupes de recherche, désirant expérimenter le comportement de protocoles sur des déploiements localisés (Orbit [82], RoofNet [76], Netbed [75], UnWiReD [71]) ou bien de groupes d'utilisateurs bénévoles, qui profitent des possibilités de certains matériels embarqués, tels que certains routeurs utilisant Linux comme système d'exploitation pour intégrer à un point d'accès 802.11 fonctionnant en mode ad hoc des fonctions de routage, et créant ainsi un réseau communautaire, comme le réseau Citoyen en Belgique[1]. Cependant, de l'aveux même des utilisateurs, ces réseaux sont assez peu performants, et posent des problèmes allant de la mise en place de nouveaux noeuds, à l'adressage, au routage, au nommage, aux temps de réactions, à la sécurité des données transférées.

Il y a donc un problème sur l'utilisation des réseaux sans fil : pourquoi nous limiter

[1] http ://reseaucitoyen.be/

au seul mode traditionnel du client d'un point d'accès ? Nous pensons que la réponse à cette question trouve son origine dans l'implémentation actuelle des réseaux sans fil 802.11 dans nos systèmes d'information actuels. En effet, la construction par héritage des couches réseaux, et la comptabilité avec IP a conduit à une complexité dans la mise en place de ces réseaux.

Afin de mieux comprendre ce phénomène, nous étudierons dans un premier temps l'approche retenue dans les systèmes d'information actuels et le fonctionnement relatif des différents éléments intervenants dans les diverses couches dans les réseaux sans fil. Puis, nous évoquerons de nouvelles approches, basées sur des concepts remettant en cause la structuration en couche ; enfin, par l'intermédiaire de travaux connexes relatifs à la sécurité des réseaux sans fil, nous évoquons une autre vision de l'architecture réseau.

3.1 Approche traditionnelle dans les systèmes d'exploitation

Les systèmes d'exploitation suivent plus ou moins le modèle en couche issu de l'Internet. Afin de permettre de suivre les évolutions matérielles tout en garantissant un jeu stable d'interface programmatique (API), les systèmes d'exploitation (ou Operating System, OS) utilisent des abstractions diverses.

La plus connue de ces abstractions est celle concernant la communication en réseau : il s'agit de la couche TCP/IP.

3.1.1 Fonctionnement en couche

La pile TCP/IP est articulée autour de 5 couches principales (voir figure 3.1) :

- une couche application, géré par le programmeur d'application, considéré comme l'utilisateur final du point de vue du système d'exploitation ;

- une couche de liaison hôte à hôte : c'est là que s'effectue les connexions proprement dite et le contrôle de flux, par des protocoles comme TCP.

- une couche internet ou interconnexion : qui définit les adresses IP, et où le routage des paquets IP s'effectue ;

- une couche d'accès réseau, qui fournit un moyen physique, de même qu'une technique d'accès au médium pour la diffusion des paquets issu de la couche internet, et enfin une couche de liaison. C'est une abstraction vis à vis des principes physiques et de la méthode d'accès à celui-ci.

Cette pile constitue de facto l'organisation actuelle des systèmes de communication basé sur IP, les modèles théoriques de type OSI n'ayant pas reçu d'implémentations, si ce n'est dans le cadre de protocoles mis en oeuvre dans les télécoms. L'implémentation de TCP/IP la plus connue, celle de BSD, se retrouve dans divers systèmes d'exploitation actuel : Windows, OSX, FreeBSD, celle de Linux étant sur un principe très proche.

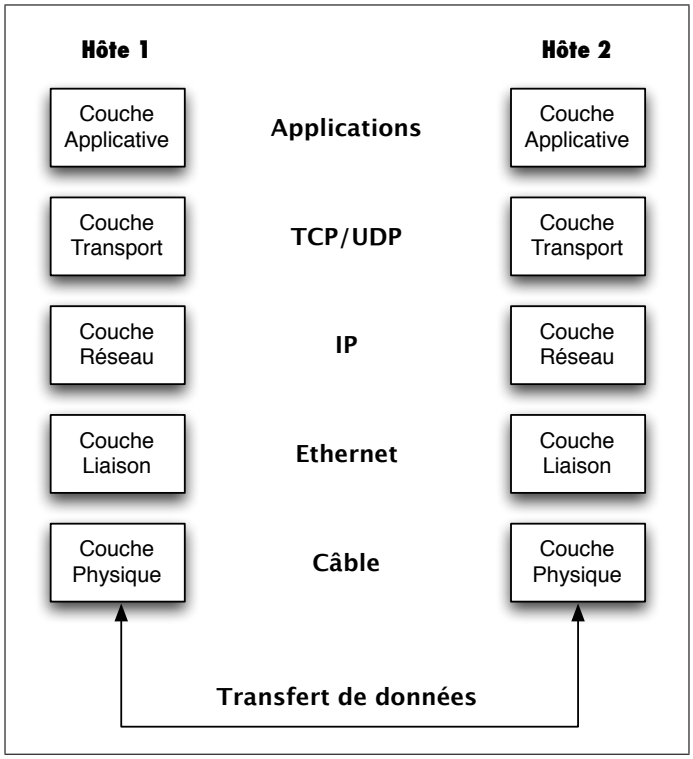


Figure 3.1 – Modèle TCP/IP simplifié.

Les systèmes d'exploitation sont le plus souvent organisés en fonction de l'architecture matérielle sur laquelle ils fonctionnent ; ces architectures présentent, pour une grande partie, une segmentation en domaine (des anneaux, ou rings), qui permet d'introduire de la sécurité au sein du système d'exploitation. Ce concept, introduit par Multics, est toujours d'actualité dans les systèmes actuels.

Dès lors, la pile protocolaire TCP/IP est intégrée dans ce type d'architecture sécuritaire : alors que la couche applicative est accessible par les utilisateurs, les autres couches, à savoir l'accès réseau, l'interconnexion et la couche de liaison sont exécutées en espace noyau privilégié. Cette sécurisation entraîne des conséquences : tout échange de données entre le noyau et l'espace utilisateur ne peut être effectué qu'en l'espace d'un certain nombre de cycles, entre 500 et 1000 cycles en moyenne.

Cette architecture particulière explique la faible modification des couches protocolaires à travers les années : en effet, la couche applicative étant isolée en espace utilisateur, de nouvelles applications peuvent s'exécuter sans remettre en cause la stabilité du système d'exploitation ; en contre-partie, sur la plupart des systèmes propriétaires, il est impossible de programmer en espace noyau ; dès lors, les modifications architecturales sont donc isolées à l'espace applicatif.

Cependant, l'avancée des systèmes d'exploitation open source tels que FreeBSD ou

Linux a contribué à l'amélioration des couches intermédiaires au fur et à mesure des années : de nombreux travaux offrent des perspectives d'évolution intéressantes.

3.1.2 Couche physique et de la couche MAC

De l'aveu même des concepteurs de FreeBSD (voir les publications techniques d'André Oppermann), les concepteurs ont très peu d'influence sur les couches physiques. Comme nous l'avons vu, les débits physiques s'améliorent par de nouvelles modulations, ou de nouvelles techniques de transmission physique (MIMO par exemple) : l'essentiel de ces opérations est confiée à des processeurs de traitement du signal qui assurent les fonctions de décodage. Les flux binaires obtenus sont stockés dans les tampons de la carte réseau.

La couche MAC, est pour les mêmes raisons, non accessible depuis le système d'exploitation. Les implémentations des couches MAC sont soit réalisées sous forme hardware, avec un comportement non reprogrammable, par exemple les cartes n'utilisant pas de micrologiciel, comme les cartes Atheros qui reposent sur une EEPROM ; soit sous forme logicielle, dans un micrologiciel (ou firmware) chargé par le système d'exploitation à l'initialisation de la carte, comme avec les cartes Prism ou les cartes Intel de la plate-forme Centrino.

3.1.3 Protocole 802.11 et couche de liaison de données

Par contre, le protocole 802.11 a quant à lui, fait l'objet d'une arrivée progressive dans l'espace noyau du système d'exploitation ; au départ, les premières cartes sans fil étant prévues comme un remplacement ad hoc des cartes Ethernet, comme les premières cartes WaveLAN : le micrologiciel de ces premières cartes effectuait lui-même une grande partie des opérations requises et fournissait au système la première couche consolidée : la couche de liaison (ou data link layer).

L'ensemble des systèmes s'est plus ou moins aligné sur le format d'une trame Ethernet. Quand une trame est reçue par la carte sans fil, elle est transférée dans la mémoire principale du système. A ce moment-là, le CPU est capable de traiter ces données. Ce procédé est appelé DMA (direct memory access) où la carte réseau écrit la trame reçue dans un endroit prédéterminé de la mémoire centrale. Ces informations transitent par le bus PCI, qui à l'heure actuelle, n'est pas encore un goulot d'étranglement vu les modulations offertes par le standard 802.11.

Le paquet va être examiné à plusieurs reprises, notamment par chaque couche du modèle TCP/IP. C'est pourquoi les systèmes modernes d'exploitation utilisent des techniques de traitement en cache avancé (Cache Prefetch) pour placer dès la réception le paquet dans la mémoire cache.

Afin de réduire les coûts et la mémoire nécessaire sur les cartes sans fil, les constructeurs ont migré la gestion du protocole 802.11 dans le noyau, ajoutant ainsi une couche

supplémentaire au traitement des paquets. Si la majeure partie des drivers et des systèmes d'exploitation exploite ce décodage en zone noyau, certains drivers transfèrent le décodage en partie utilisateur, multipliant de ce fait les échanges mémoires entre les zones utilisateurs et noyau et ralentissant de ce fait le traitement des données.

Ainsi, le marché des cartes présente une grande diversité mais des fonctionnements similaires :

- les cartes intel et broadcom utilisent des firmwares programmables et un driver opensource ; la machine à état 802.11 étant minimale et placée dans un module noyau ;

- les cartes atheros utilisent une eeprom avec une couche MAC au comportement fixe. Seuls certains aspects relatifs à des paramètres de certains amendements (comme 802.11e) sont réglables. L'essentiel du traitement s'effectue par un module noyau, à la fois pour le traitement des paquets et la machine à état 802.11 ; notamment, la gestion des débits (Rate control) est effectuée par le système hôte.

- les clé USB utilisent un firmware pour la gestion de la couche MAC, puis un driver avec une machine à état 802.11.

Une fois le décodage 802.11 effectué, les abstractions utilisées pour les cartes Ethernet sont donc réutilisées : le travail du noyau peut commencer.

3.1.4 Couche IP

Le noyau effectue des vérifications sur l'intégrité du paquet, puis ensuite soumet le paquet à des régles de pare-feu ; puis, il détermine si le paquet est à destination de l'hôte, ou s'il doit être retransmis selon une table de routage ; si le routage est inactif, un paquet ICMP est alors envoyé signalant l'échec.

Le parcours de la table de routage utilise des techniques de hâchage. Encore une fois, les développeurs tentent d'optimiser les performances en plaçant ces tables dans la mémoire cache. La gestion du routage, c'est à dire, l'ajout, la suppression la modification des règles s'effectue par un démon de routage placé en espace utilisateur. La réactivité de celui-ci est donc forcément réduite.

Les paquets non locaux sont donc retransmis suivant une table de hâchage, puis renvoyés sur une interface réseau associée à la route, qui peut être la carte sans fil ou une carte. Dès lors, le paquet doit être transféré sur la carte, occasionnant un accès à la mémoire du système par la carte réseau.

Les paquets locaux, quand à eux, sont alors associés à leur couche transport respective.

3.1.5 Couche transport

Les paquets locaux sont affectés à la socket dédiée à leur protocole - TCP ou UDP. La couche transport vérifie l'intégrité du message et détermine si la socket réceptrice existe dans le système. Si ce n'est pas le cas, un message d'erreur ICMP est renvoyé à la source. Pour les paquets TCP, appelés segments à ce stade, le noyau doit examiner

l'ensemble des sessions TCP actives et les sockets en écoute. Encore une fois, une table de hâchage est employée pour retrouver la socket destinataire.

TCP gère l'arrivée ordonnée des paquets et la gestion des erreurs. Il maintient donc une file d'attente de réassemblage et utilise des paquets d'ACK TCP pour obtenir la retransmission ou non des paquets manquants. Des listes chaînées sont utilisées, mais de nouvelles techniques apparaissent et préfèrent l'utilisation de blocs permettant de simplifier la manipulation des segments.

Les cartes Ethernet ont introduit des dispositifs physiques pour le calcul et le réassemblage des segments (techniques dite de "offload"), mais, d'un point de vue système, ces techniques sont trop peu génériques et remettent trop en question l'architecture réseau des noyaux actuels. A notre connaissance, seules les cartes Ethernet filaires exploitent ces techniques physiques.

A partir de ce moment là, les paquets sont disponibles pour les applications en espace utilisateur.

3.1.6 Couche application

L'essentiel des tâches consiste à transférer les données de l'espace noyau à l'espace utilisateur. Les applications utilisent donc, en fonction de la couche transport, des appels systèmes de type socket, puis listen, accept pour les connexions basées sur TCP. En fonction du degré d'optimisation, l'application peut utiliser des fonctions select ou poll pour vérifier la présence de données sur une socket, de même, d'autres mécanismes peuvent être intéressants lors d'une application massivement multi-processus (comme un serveur web).

3.2 Remise en question de l'architecture en couches

Le système en couche a donc permis un formidable déploiement d'applications en réseau, de même qu'un fort développement concurrentiel d'adaptateurs. En effet, constructeurs et développeurs d'applications n'ont en aucun cas besoin de connaître leur métier réciproque. Mais, comme nous l'avons vu, dans une optique de réseaux sans fil, ces couches sont considérablement multipliées : apparition du protocole 802.11, diverses solutions matérielles d'intégration...

Dès lors, pour les nouveaux usages des réseaux sans fil, cette architecture apparaît inadéquate pour 3 raisons essentielles : d'abord, les liens sans fil posent des problèmes spécifiques à cette technologie ; ensuite, la création de liens est opportuniste par communication radio ; et enfin, la communication sur ce médium est par définition très différente.

Tout d'abord, le lien sans fil crée intrinsèquement des nouveaux problèmes pour les concepteurs de protocoles, comme nous l'avons étudié dans notre autopsie de 802.11 :

l'ensemble logiciel formé par la pile de protocoles ne permet pas la résolution simplifiée de ces problèmes. Le cas classique de TCP en est le parfait exemple : un erreur au niveau physique peut conduire la couche transport à réagir à un problème de congestion qui n'en est pas un.

Par contre, les réseaux sans fil offrent des possibilités de communication opportuniste qui ne peuvent être utilisées dans la conception traditionnelle en couche. Notamment, les liens radio varient dans le temps, et cette adaptation temporelle peut faire l'objet d'une adaptation des paramètres de transmission, qui nous l'avons vu, est peu utilisée dans les implémentations : les codes correcteurs utilisés dans les modulations OFDM sont par exemple inaccessibles à la couche MAC : les mécanismes d'adaptation que nous avons évoqués reposent donc sur le taux d'erreur de paquet, et non sur des indices physiques qui permettraient une adaptation rapide au médium.

De même, les couches physiques peuvent offrir une réception simultanée (au sens statistique du terme) de plusieurs paquets en même temps, et la nature même de la diffusion radio n'est pas employée par la couche protocolaire (la première étape d'une carte étant d'identifier si le paquet lui est destiné ou pas). Dès lors, les mécanismes de routage proposés par exemple dans MANET, même s'ils tentent d'approcher la réalité du médium par l'utilisation de métriques adaptées par exemple, ne tiennent finalement pas compte de la topologie radio ambiante : les mécanismes de retransmission ne tiennent pas compte des variations temporelles du canal, et donc, entre la réception et la réémission d'un paquet, trop d'éléments ont pu fluctuer.

Les optimisations proposées par les chercheurs, étudiées au cours de notre précédente autospie, représentent de manière localisée des violations de l'architectures en couche. Dès lors, une partie de la communauté a exploré la piste d'architecture dite inter-couches, visant à dépasser le modèle en couche face aux challenges posés par les réseaux sans fil.

3.3 Approches inter-couches : architectures et frameworks

Plusieurs auteurs ont proposé une vision inter-couches de l'architecture réseau. Deux contributions complémentaires apportent une analyse complète sur les possibilités et les connaissances d'une architecture inter-couches, aspects positifs et négatifs. Dans la première [88], les auteurs Kawadia et Kumar proposent une approche précautionneuse des architectures inter-couches.

Une de leurs principales inquiétudes est la conception en code spaghetti : chaque optimisation peut être isolée et conduire à une non réutilisabilité. Dès lors, si les réseaux sans fil doivent devenir des architectures innovantes, ils doivent conserver des principes fondateurs forts, comme les architectures des ordinateurs, héritière de l'architecture de Von Neumann, qui par exemple distingue bien la mémoire, l'unité de contrôle, les unités arithmétiques et logiques. De même, les auteurs citent le modèle TCP/IP en exemple : nous l'avons vu, la séparation en couche a été autrefois pri-

mordiale pour le développement à grande échelle de l'interconnexion des réseaux. Cependant, comme nous l'avons précédemment souligné, le modèle en couche utilisé de facto pour les réseaux sans fil n'est pas forcément le plus adapté, notamment à cause des nouveaux paradigmes de la communication sans fil.

Les auteurs reprennent donc les bases de la communication numérique sans fil. Ainsi, la construction de cette communication s'appuie sur la théorie développée par Shannon, selon laquelle les sources peuvent être décorrélées du canal de transmission sans perte d'optimalité. Il n'y a donc aucun intérêt à fabriquer une source de flux binaire adaptée à un canal donné. Dès lors, un premier élément d'architecture apparaît : la carte sans fil peut donc effectuer un codage canal optimal pour un canal de transmission donné pour une grande variété de sources (donc de données numériques).

Les auteurs affirment donc que le problème de positonnement des couches est ailleurs, et plus précisement se situe au niveau du transfert des données. Shannon explique que la capacité d'un canal est donnée par son rapport signal à bruit. En sans fil, plusieurs stratégies de retransmission des données sont possibles : la notion de lien n'existe pas : chaque noeud émet de l'énergie qui se superpose aux autres et qui est donc simultanément reçue par les destinataires. Plusieurs types de coopération sont donc possibles : un groupe de noeuds peut annuler les interférences d'un groupe donné au profit d'un tiers ; ou plutôt, il peut relayer le signal selon différentes stratégies : amplifier et retransmettre, ou décoder et réémettre. Pour les auteurs, la nouvelle vision n'est pas un déplacement mécanique de l'information, mais plutôt un transfert électronique de celle-ci.

Les auteurs justifient donc dans cette publication deux éléments fondamentaux des réseaux sans fil :

- le multihopping est optimal pour un ordre de grandeur donné : dans le cadre du traditionnel principe de décodage et de retransmission, cette opération est optimale lorsqu'elle est effectuée de noeud en noeud tant que la charge des noeuds peut être équilibrée par un routage multichemins.

- d'autres stratégies sont optimales si les conditions d'atténuation sont faibles : le relaying avec suppression des interférences peut donc être plus intéressant.

Dès lors, ces résultats sont intéressants. Ils justifient au niveau de l'architecture la présence de l'opération de décodage/réémission dans le cas le plus courant. De plus, ils justifient aussi la nécessité d'un routage par chemin, et donc l'existence de cette notion dans les architectures sans fil. Enfin, l'existence de protocole de transport, permettant l'établissement d'un tuyau point à point est de ce fait justifiée. Les auteurs retrouvent une grande partie des principes actuels des réseaux et donc par extension des réseaux sans fil actuels. Cependant, leur démarche montre que tout approche d'optimisation inter couches ne peut résulter que dans une optimisation de la bande passante, et pas une amélioration infinie même dans les réseaux à grand nombre de noeuds.

Les conséquences des premières conclusions de cette publication sont intéressantes :

il ne faut pas attendre des réseaux sans fil un incrément fondamental de la capacité réseau par la présence de multiples noeuds ; au mieux, on peut, grâce à l'utilisation de techniques inter-couches, voir une optimisation de la capacité réseau utilisée et approcher la capacité maximale théorique. Il nous faut donc quitter le paradigme des réseaux filaires introduit par Ethernet : l'ajout d'un switch augmentant la capacité des réseaux doit donc disparaître dans l'esprit des concepteurs des réseaux sans fil. L'ajout de nouveaux noeuds ne résulte pas en une augmentation de la bande passante, mais plutôt en une augmentation de la connectivité et l'efficacité de cette connectivité ne peut-être améliorée que par l'utilisation de système inter-couches.

3.3.1 Architectures inter-couches théoriques

Dans la continuité de cette première réflexion, les auteurs de [70] proposent une étude des architectures cross-layer permettant d'atteindre ces nouveaux objectifs de performance, complétant ainsi la contribution étudiée précédemment. Srivastava et Motani énumèrent les différents types de propositions, illustrées dans le schéma de la figure 3.2 .

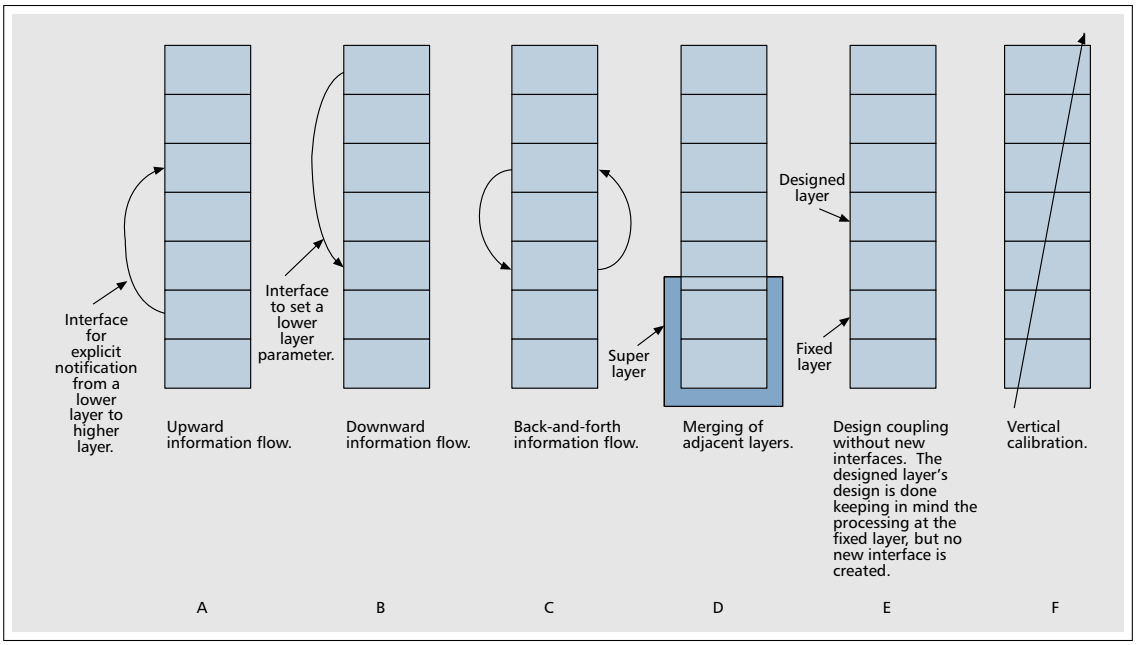


Figure 3.2 – Types d'architectures crosslayer observées dans la littérature. (Extrait de la publication de Srivastava et al.)

La première architecture présentée consiste à remonter des informations des couches inférieures vers les couches supérieures. Ce type d'architecture est utilisé dans plusieurs cas réels : notamment par les mécanismes d'adaptation des débits [9] [39] [52]

[84] [19], où les informations de la couche physique, comme le taux d'erreur par paquet est utilisé pour déterminer si la modulation physique doit être adaptée. Mais ce type d'architecture concerne aussi les couches plus hautes : un routeur peut notifier la couche transport d'une congestion explicite par l'utilisation du bit ECN, rajouté dans les spécifications de TCP [78].

Une deuxième architecture consiste à utiliser une communication inverse, des couches supérieures aux couches inférieures. Typiquement, cette architecture est mise en oeuvre indirectement dans l'amendement 802.11e du standard WiFi : les quatres files d'attentes offertes (Best Effort, Background, Video, Voice) sont remplies en fonction du champ Type Of Service (TOS) du paquet IP, comme proposé par Park et Choi [98]. Ainsi, par une simple indication des couches supérieures, on peut modifier l'action des couches inférieures.

Ces deux architectures peuvent être combinées, par exemple dans les techniques avancées de scheduling qui prennent en compte le trafic à transmettre (couche MAC supérieure) et les contraintes d'émission de puissance (couche inférieure physique), comme ce qui est proposé dans [87] : grâce à des mécanismes d'intercommunication entre les deux couches, les auteurs parviennent à réduire des interférences non réductibles par le contrôle de puissance par un scheduling plus efficace des utilisateurs, qui tient compte lui-même des contraintes de puissance de chacun.

D'autres architectures sont plus simplificatrices : par exemple, la création d'une super-couche par fusion de deux couches. Bien que les travaux ne citent pas cette supercouche en tant que telle, la plupart des travaux sur des optimisations cohérentes entre couche physique et couche MAC conduit indirectement à ce type d'architecture.

Une autre solution consiste aussi à développer une couche en anticipant le fonctionnement d'une couche inférieure. C'est notamment ce genre de solution architecturale qui est utilisé pour les couches applicatives : l'absence d'accès aux couches inférieures nécessite donc des mécanismes adaptatifs au niveau applicatif. L'utilisation du RTCP comme boucle de rétroaction dans les réseaux multimédias utilisant le protocole RTP en est la concrétisation pratique [73].

La dernière solution évoquée par les auteurs est une forme de calibrage vertical : par exemple, les auteurs de [92] présentent une architecture où l'adaptation au délai requis modifie le mécanisme de correction des erreurs (ARQ) et également les codes de correction d'erreur physique. Ainsi, l'architecture permet de conserver les choix en terme de délai en agissant sur l'ensemble des couches.

La multiplicité de ces architectures conduit à une multiplicité de solutions pour les implémenter dans les systèmes. Srivastava et Motani établissent trois types de réalisations (voir figure 3.3) :

- une communication directe entre les couches : les variables de fonctionnement d'une couche sont accessibles par une autre couche, au lieu d'être interne et restreintes à une couche donnée. Cependant, d'autres types de communication peuvent être envisagées : par exemple, l'utilisation des en-têtes de chaque protocole peut permettre une

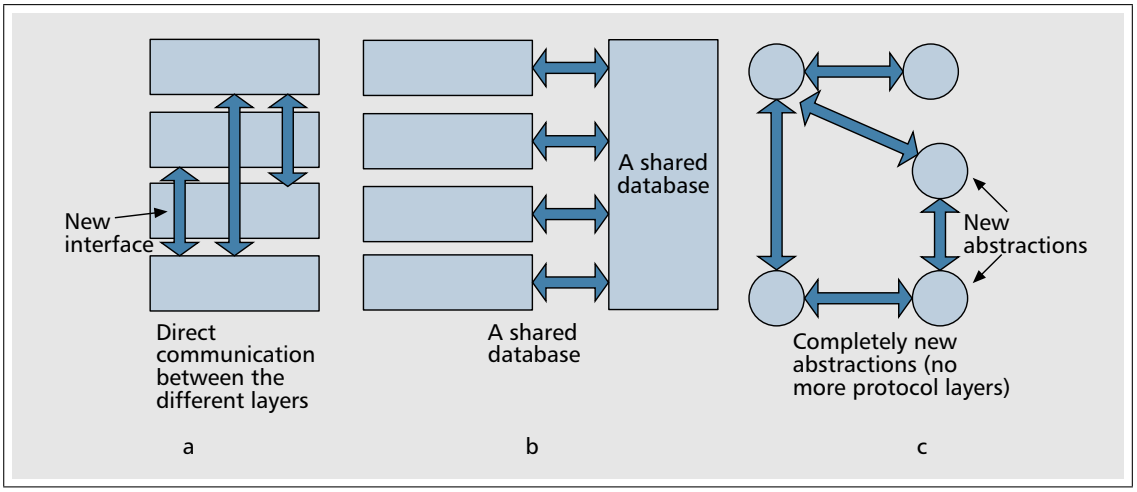


Figure 3.3 – Types de propositions d'implémentation Crosslayer. (Extrait de la publication de Srivastava et al.)

communication entre les couches. CLASS fournit par exemple un ensemble restreint de raccourcis entre les couches. Clairement, ce type de communication est implémentable dans les noyaux actuels, puisque l'implémentation de TCP/IP utilise directement un même tampon mémoire pour l'ensemble des couches d'un paquet. On peut donc imaginer un mécanisme intercouche donnant accès à l'ensemble des couches dans le noyau des systèmes d'exploitation. Cependant, cette solution est exposée au problème de gestion des zones mémoires entre mode superviseur et mode utilisateur, et dès lors les mécanismes inter-couches peuvent être sous performants ou poser des problèmes de sécurité.

- d'autre propositions évoquent la possibilité d'une base de données commune et renseignée par l'ensemble des acteurs des couches. Evidemment, cette vision transversale s'applique très bien aux interactions impliquant l'ensemble des couches. Cependant, la lourdeur d'un tel mécanisme peut poser des problèmes dans un environnement embarqué et contraint comme le noyau d'un système d'exploitation.

- enfin, il reste la possibilité de créer de nouvelles abstractions : c'est à dire, dépasser le modèle en couches, les concepts d'encapsulation et de proposer de nouvelles visions.

Ces 3 types d'architecture se retrouvent dans les frameworks inter-couches issus de l'expérimentation.

3.3.2 Frameworks issus de l'expérimentation

Les expérimentations en matière de réseaux sans fil ont donné lieu à un nombre conséquent de frameworks dédiés aux réseaux sans fil.

L'un des premiers frameworks d'expérimentation sur les réseaux est proposé par Morris en 2000 [69] et son extension sans fil par Bicket en 2005 [72]. Ce framework a été utilisé pour développer Roofnet, un réseau mesh non planifié localisé à Cambridge dans le Massachusetts. Chaque noeud de Roofnet est un pc standard silencieux, doté d'une carte 802.11b et d'une antenne omnidirectionnelle. Les cartes utilisent un dérivé du mode ad hoc de 802.11b, et n'utilisent pas le virtual carrier sensing (pas de RTS/CTS). Ce mode dérivé n'envoie pas de Beacon, ce qui élimine le phénomène de partionnement observé dans les réseaux utilisant le mode ad hoc (soulevé par les développeurs de drivers des chipsets Atheros, MadWiFi).

Le logiciel est organisé comme suit : Click fournit une architecture de routage en mode utilisateur, reposant sur la juxtaposition d'éléments élémentaires. Le modèle est orienté objet : chaque élément d'une classe élément propose une entrée, plusieurs sorties, et une entrée de configuration. Deux modes de communication sont implémentés : push et pull. En mode push, les paquets envoyés à un élément sont traités ; en pull, c'est l'élément qui sollicite l'entrée de paquets. Le traitement des paquets peut s'appuyer sur des files d'attentes qui sont des éléments comme les autres. Ce formalisme est très puisssant, car il permet de s'affranchir du mécanismes des couches en isolant la manipulation de celle-ci au niveau de chaque élément et réduit l'étude à la construction d'un schéma de routage. L'interaction avec l'espace noyau est réalisée par l'intermédiaire d'interfaces virtuelles.

Dans le cadre de Roofnet, chaque carte 802.11 est utilisée comme entrée et sortie finale des paquets. L'adressage est déterminée par l'adresse Ethernet unique de chaque carte, dont dérive les derniers bits de l'adresse IP. Chaque noeud est équipé d'une interface Ethernet pour la connexion filaire à Internet. Si une telle connexion existe, le noeud offre alors une passerelle vers Internet. Lors de la transmission de paquet vers un noeud Roofnet, la configuration de Click utilise une métrique basée sur la bande passante la plus importante entre les noeuds émetteur et destinataire. Dans [72], les auteurs ont évalué l'influence des métriques basée sur la qualité du lien, recueillie lors de la demande de route, le protocole utilisé étant un dérivé de DSR. La métrique utilisée est le temps estimé de transmission (ETT), dérivée d'ETX. ETT prédit le temps total de transmission d'un paquet le long d'une route. Cette métrique est construite autour des beacons courts et longs transmis régulièrement. Cette métrique s'appuyant fortement sur la modulation utilisée sur chaque lien, les auteurs ont modifié l'algorithme de choix du débit en remplaçant ARF par Sample Rate, qui ajuste le débit en fonction de paquets sondes envoyés à intervalle régulier.

Les performances observées sont similaires à celle d'autres réseaux mesh, pourtant la construction de celui-ci est grandement facilitée par l'approche originale du routeur modulaire Click.

Biswas et Morris [97] ont également utilisé Click pour implémenter ExOR, un protocole de couche MAC et de routage qui utilise les conditions opportunistes à chaque saut pour améliorer la bande passante totale. Le principe de fonctionnement est oppor-

tuniste : chaque noeud à portée radio détermine s'il est le plus proche de la destination, et renvoie le paquet à son tour. ExOR repose sur la métrique ETX, et est construit sur Click pour son implémentation.

D'autres auteurs ont utilisé Click avec succès pour leurs expériences.

Stoica [94] a utilisé Click pour construire une couche MAC virtuelle se superposant aux couches MAC habituelles. Cette couche d'abstraction permet de faciliter les tests de nouveaux protocoles pour la couche MAC. La couche d'abstraction repose sur les capacités des cartes : transmission, réception, activation éventuelle du RTS/CTS, mais plus important, la gestion fine des files d'attente. En effet, cet overlay repose sur la capacité à modifier le contenu des files d'attente des cartes. Cette approche est intéressante, cependant on peut s'interroger sur la granularité et le réalisme d'une telle couche MAC, sachant que les mécanismes intrinsèques d'accès au canal ne sont pas manipulables aisément. Par exemple, l'utilisation du Binary Exponential Backoff n'est pas débrayable sur ce prototype. Quoiqu'il en soit, les auteurs obtiennent des bons résultats en terme d'équité notamment sur les flots TCP.

Un autre exemple d'utilisation est WildNET de Brewer [83], qui utilise Click dans le cadre des réseaux WiFi longue distance. Dans ce domaine, les fonctionnalités habituelles de 802.11 sont remises en question : premièrement, le mécanisme de transmission des paquets basé sur un acquittement est soumis aux phénomènes de propagation : à longue distance, l'émetteur doit attendre plus longtemps l'arrivée de l'acquittement. L'utilisation du canal s'en trouve réduite. De plus, si le délai de réception de l'acquittement est supérieur à l'expiration standard, le paquet est considéré comme perdu par l'émetteur qui va alors le réémettre. Deuxièmement, le mécanisme d'accès au canal CSMA/CA est également perturbé. Pour le DIFS standard de 802.11, soit 50 μs, la distance pour laquelle l'évitement de collision est valide est de 15km. Troisième et dernier point, les interférences et plus particulièrement les autres réseaux 802.11 réduisent l'efficacité des transmissions.

Pour palier ces problèmes, les auteurs ont donc proposé un remplacement du CSMA/CA par un mécanisme de type TDMA, avec l'adjonction d'acquittement en rafale, permettant l'acquittement d'un certain nombre de paquets avec un seul acquittement. Pour implémenter ces propositions, les auteurs ont utilisé Click pour construire la nouvelle couche MAC, en modifiant le driver pour désactiver les acquittements, ainsi que le CCA (ce qui permet d'effectuer une transmission immédiate sur le canal sans délai). Les auteurs de WildNET n'ont pu utiliser les mécanismes de gestion du temps de Click car la granularité des slots temporels est insuffisante pour le noyau considéré. De plus, entre le moment où l'interface de Click envoie un paquet et son envoi réel, une différence est constatée. Cependant, cette différence étant bornée, les auteurs ont pu éliminer cet écueil en prenant en compte ces différences dans le calcul de l'allocation TDMA.

Dans le même esprit, XORP [89] permet l'implémentation de nouveaux protocoles de routage. XORP est composé d'un moteur de transmission des paquets (qui peut être

réalisé sous forme de module Click) et d'applicatifs haut niveau en mode utilisateur, permettant l'implémentation des nouveaux protocoles. XORP est principalement dédié au problème du routage.

D'autres frameworks d'expérimentation ont été développés dans le cadre des recherches sur le cross-layer dans les réseaux sans fil. Aïache et al. ont proposé XIAN [79], un framework permettant de rassembler sous la forme d'une API l'ensemble des statistiques d'une carte sans fil, en l'occurence les cartes Atheros munies du driver MadWiFi. Cette API est ensuite mise en oeuvre pour effectuer un routage tenant compte de contraintes de qualité de service. Xian est construit autour d'un module noyau récupérant les données statistiques auprès du driver hardware , puis les exportent pour une bibliothèque en mode utilisateur. Xian par contre ne permet pas de modifier les paramètres d'accès au canal de la carte, que ce soit physique ou MAC. Il s'agit plus d'une instrumentation des couches dans ce cas, mais l'architecture retenue laisse la porte ouverte à de futures modifications.

Ben Adesslem et al. ont proposé Prawn [90], un outil pour l'implémentation rapide de protocole de communication au-dessus des réseaux 802.11. Le but des auteurs n'est pas ici de proposer une architecture performante, mais de fournir un moyen de tester un protocole, contrairement à Click ou XORP qui offre la possibilité d'une implémentation performante d'un protocole. L'environnement se compose de deux éléments, le moteur, ou Prawn Engine, constitué de blocs élémentaires par dessus lesquels les protocoles sont construits. Ensuite, la bibliothèque Prawn propose une API permettant un accès transparent aux briques élémentaires sous-jascentes. Le but ici est de se concentrer sur les couches supérieures, c'est à dire le niveau IP.

Ces frameworks constituent un premier pas vers une nouvelle vision du modèle de l'architecture réseau. Au-delà du modèle traditionnel en couche de la pile TCP/IP, ils offrent des facilités pour l'expérimentation de nouvelles méthodes d'optimisation, soit par construction de briques élémentaires (Click ou Prawn), soit par l'exposition de données internes à l'ensemble du système (XIAN). Dès lors, une question se pose : peut-on encore aller plus loin dans une nouvelle vision ?

3.4 Vers une nouvelle vision des interfaces radio

Les frameworks que nous venons d'étudier dans la section précédente répondent fondamentalement à des besoins en terme de vision orientée "réseau", c'est à dire que le modèle en couche est peu revisité. Pourtant, dans le domaine des réseaux sans fil, des chercheurs d'autres domaines se sont appliqués à élaborer des techniques permettant l'analyse pointue et le dépistage de défauts : il s'agit d'une part de la recherche en sécurité, et d'autres part des chercheurs explorant les radio logicielles.

3.4.1 Une autre vision : contributions issues de la sécurité

Les chercheurs se sont très vite intéressés aux challenges posés par les réseaux sans fil. En effet, contrairement aux réseaux filaires peu sensibles à la fuite d'information par le médium, les réseaux sans fil, mal configurés, peuvent constituer un point d'entrée non autorisé dans un réseau. Le confinement physique par la gestion de la puissance d'émission par exemple est une fausse solution car un attaquant peut toujours utiliser des aériens (antennes, amplificateur) à haut gain permettant l'écoute et l'émission de données sur la fréquence de fonctionnement du réseau.

Très rapidement les chercheurs en sécurité ont cherché à évaluer la sécurité des réseaux sans fil. Le domaine s'est d'abord orienté sur la sécurité du protocole de sécurisation fourni par 802.11, le WEP (Wireless Equivalent Privacy) puis le WPA, censé fournir un moyen de sécurisation au niveau 802.11 des données par chiffrement.

L'étude théorique poussée de ce protocole dès 2001 par Borisov et al. [93] révèle des défaillances importantes des fonctions cryptographiques dans le protocole 802.11. En effet, l'écoute passive du trafic d'un réseau 802.11 permet la récupération de la clé de session commune utilisée entre 2 équipements 802.11 utilisant des messages 802.11 chiffrés en utilisant WEP. Dès lors, les annonces de découverte de faille autour de 802.11 se succèdent ([91], [86]) et la communauté recherche des techniques permettant une démonstration (Proof-of-concept, PoC) de récupération rapide des clés de session 802.11.

C'est alors qu'un nombre conséquent d'outils dédiés à l'examen de la sécurité des réseaux apparaissent. Tout d'abord, un certain nombre de scanners, qui énumère les beacons reçus par l'équipement 802.11 embarqué, soit de manière passive (par écoute des paquets de type management) soit de manière active (par l'envoi de paquet 802.11 de type Probe Request). La recherche sur le WEP encourage l'apparition de nouvelles techniques d'attaque, basées sur des concepts courants en sécurité, tels que le rejeu [85] de paquet ARP afin d'augmenter le nombre de paquets chiffrés transmis et ainsi découvrir plus rapidement la clé de session. Les chercheurs cherchent donc à fabriquer leurs propres paquets et à les injecter sur une fréquence donnée, en utilisant les possibilités de certains chipsets sans fil (Atheros, Zydas, Prism permettent d'envoyer arbitrairement un paquet depuis une interface configurée en mode moniteur par exemple).

Parallèlement, d'autres problèmes affectent les réseaux sans fil : en effet, l'envoi de certains types de paquets conduit certaines cartes à faire planter leur driver en espace noyau, et dès lors, à entraîner également le noyau du système d'exploitation. Maynor a par exemple démontré avec son attaque "Hijacking a MacBook in 60 seconds or less" la vulnérabilité de certains drivers à certains types de paquets 802.11 corrompus, ce qui entraînaient une possibilité de prise de contrôle à distance de la machine. Pour mettre en évidence ces failles de sécurité, certains chercheurs ont proposé des techniques de Fuzzing [77], [81]. Ces techniques consistent à soumettre à un ensemble (carte sans fil, driver, système d'exploitation) un ensemble de paquets malformés pour tester la

robustesse de l'ensemble.

L'ensemble de ces deux besoins (génération de paquet d'un type donné, ou génération de paquet malformé) a rendu nécessaire l'emploi de frameworks dédiés à la création et à l'injection de paquets. A ce jour, il existe plusieurs types d'abstraction relativement génériques :

- plusieurs bibliothèques permettent la création de paquet 802.11 de tout type : Airbase[2], Lorcon[3]. Ces bibliothèques permettent la réutilisation d'un même code sur plusieurs types de cartes sans fil.

- d'autres frameworks sont plus génériques : Scapy est un ensemble de lignes de langage Python permettant la création arbitraire de tout type de paquet, 802.11, ARP, IP, TCP, de même que la création de nouveau protocole. Packet Forge est un ensemble similaire mais en C et est plutôt dédié aux systèmes BSD. Ils permettent tous deux l'enregistrement de nouveaux protocoles.

L'étude de Scapy, outil développé par Philippe Biondi est intéressante. En effet, il s'appuie sur le langage Python pour faciliter le prototypage de manipulation de paquets. Scapy fonctionne donc en espace utilisateur, et utilise des socket ouvertes sur les interfaces pour la récupération et l'injection de trafic. Le traitement s'effectue donc intégralement en espace utilisateur, ce qui évite les échanges de données entre l'espace utilisateur et le noyau, gourmand en terme de gestion pour le système d'exploitation. Scapy utilise un formalisme très simple pour la définition d'un nouveau type de paquet. Les auteurs de [74] l'ont utilisé avec succès pour implémenter de nouveaux outils dédiés à la sécurité des Enterprise Service Bus. La déclaration des structures de données utilisées par le protocole se fait comme suit :

```
1  class Modbus(Packet):
2      name = "Modbus"
3      fields_desc = [ ShortField("transaction_id",0),
4      ShortField("protocol_id", 0),
5      ShortField("data_length",None),
6      ByteField("unit_id", 0),
7      ByteField("function_code", 0),
8      DataField("data",{}) ]
9
10     def post_build(self, p, pay):
11         if self.data_length is None:
12         length = len(p[8:])+2
13         p = p[:4] + struct.pack('>H',length)+ p[6:]
14         return p
```

La classe définit le type de paquet, le format des différents champs, les actions à effectuer (calcul de checksum) après construction d'un paquet. Scapy intègre déjà l'ensemble des protocoles 802.11, IP, TCP et UDP. Il est donc largement utilisé pour

[2]http ://www.802.11mercenary.net/
[3]http ://www.802.11mercenary.net/lorcon/

l'implémentation d'outils dédiés à l'étude de la sécurité. Certaines applications dérivées de Scapy, telle que WiFiTap, offre en quelque lignes de code la possibilité de communiquer par un canal sans fil sans utiliser la lourdeur protocolaire de 802.11.

Ces outils sont intéressants à plus d'un titre : non seulement ils offrent des abstractions assez différentes des frameworks issus de la recherche, notamment par leur vision boîte à outil du réseau traditionnel, mais en plus, ces outils sont utilisés de manière opérationnelle, de telle sorte qu'ils sont couramment utilisés en terme de preuve de concept. Indirectement, ces outils remettent en cause le modèle en couches en permettant l'accès direct à certaines couches (802.11, IP etc...) directement depuis l'espace utilisateur, où d'habitude seuls les démons de routage sont tolérés. Ces outils contribuent donc à dissocier la couche réseau de son intégration habituelle dans le système d'exploitation, et en soit, ils rejoignent certaines visions cross-layer proposées par les chercheurs en réseaux. Cependant, comme nous l'avons vu, ces outils même s'ils approchent les couches basses du réseau se limitent à la manipulation du protocole 802.11. Est-il possible de manipuler aussi aisément la couche physique ?

3.4.2 Les nouvelles possibilités des radios logicielles

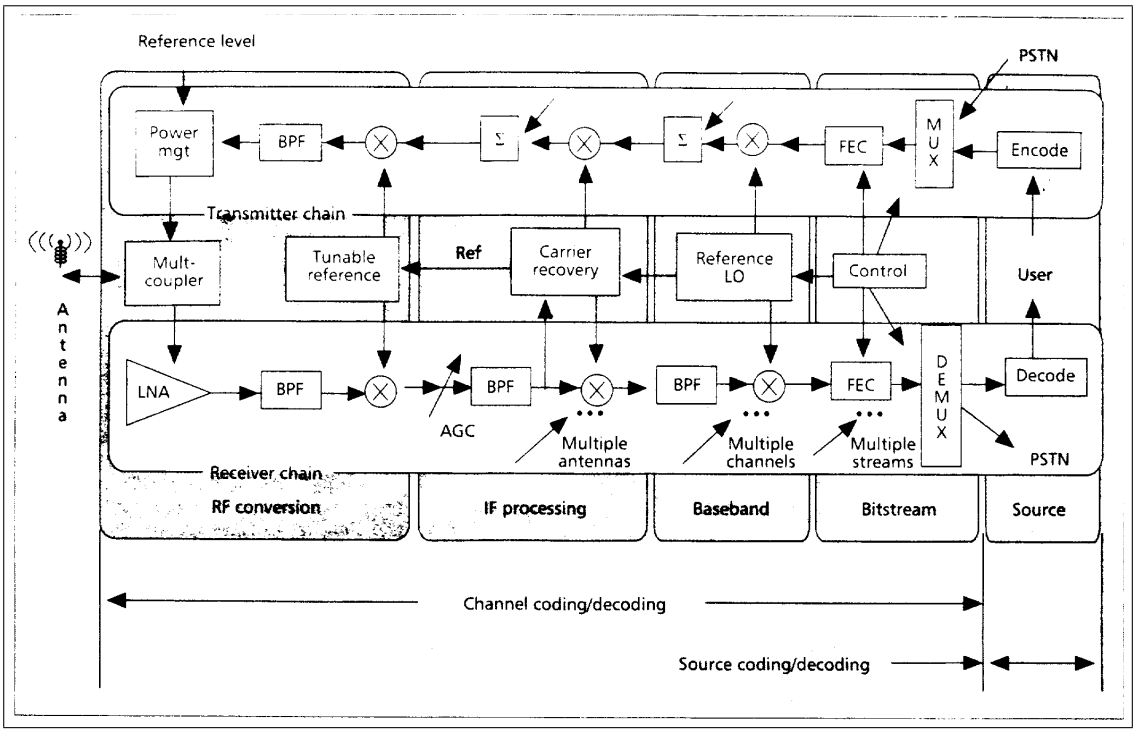


Figure 3.4 – Architecture d'une radio logicielle (extrait de "The software radio architecture" par Mitola et al).

Pour répondre à la dernière question soulevée dans notre précédente section, il convient de replacer notre discours compte tenu des derniers progrès en matière de transmission numérique. Comme nous l'avons vu dans notre chapitre dédié à l'étude du fonctionnement du protocole 802.11, les parties physiques des équipements 802.11 sont constituées de composants dédiés au traitement numérique du signal (mélangeurs, filtres, amplificateurs, modulateurs, démodulateurs, FFT...). L'idée des radios logicielles (ou Software Defined Radio, SDR) n'est pas récente : dès les années 1990, Mitola propose une architecture radio logicielle dans [80]. Mitola y définit les éléments principaux des radios logicielles (voir figure 3.4) : une alimentation, une antenne, un convertisseur Radio fréquence multibandes, et des convertisseurs analogique/numérique dotés d'un processeur et de mémoire assurant les fonctions auparavant réalisées par des composants dédiés. Mitola précise également le placement idéal de chaque élément, et évalue le coût en terme d'opération et les timings requis. De son analyse, nous retenons la nécessité d'un fonctionnement temps réel (voir figure 3.5) et les requis en termes de millions d'opérations par seconde (voir figure 3.6).

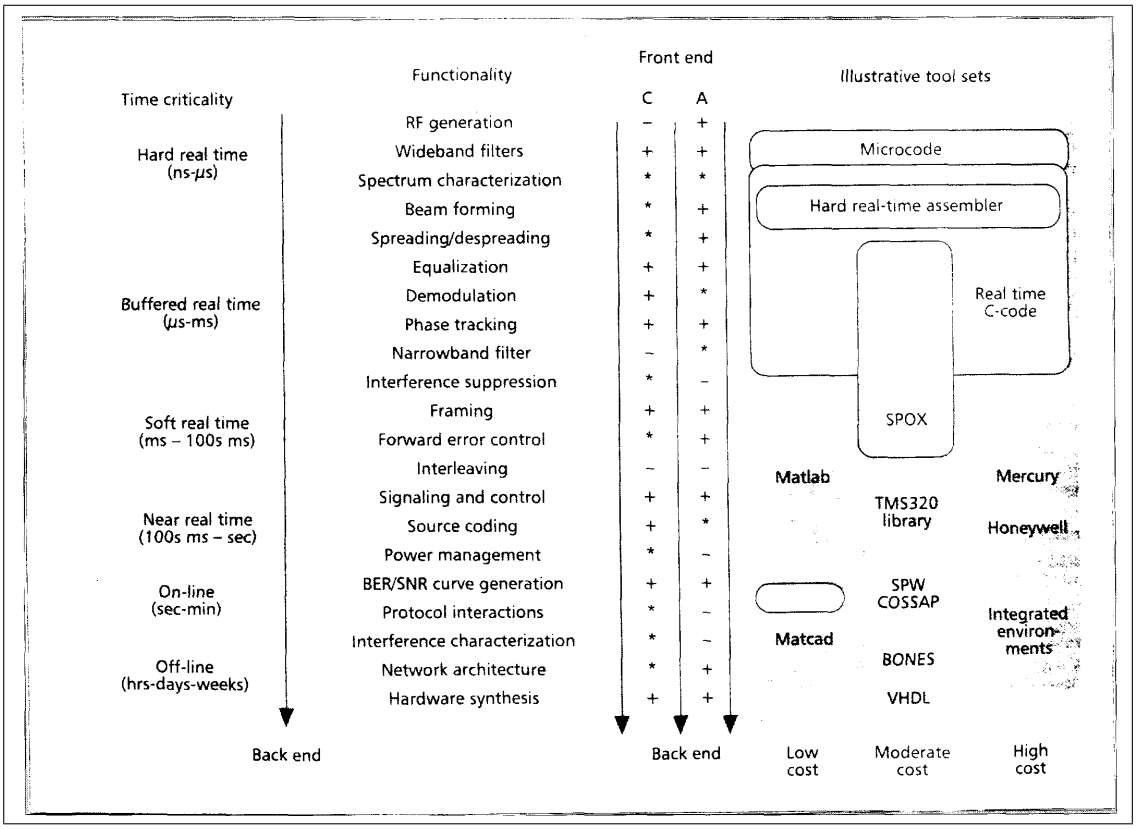


Figure 3.5 – Contraintes temporelles d'intégration d'une radio logicielle (extrait de "The software radio architecture" par Mitola et al).

Segment	Parameter	Illustrative value	Demand estimate
IF	W_s	10 MHz (2.5 oversampling)	
	IF Filter	100 OPS/ Hz	D_{if} = 2500 MOPS*
Users	N	30/ cell site	
Baseband	W_c	30 kHz	
	Demodulator	50 OPS/Hz	D_{bb}= 1.5 MOPS
Bitstream	R_b	32 kb/s	
	FEC, Signaling	100 OPS/b/s	D_{bs} = 3.2 MOPS
Source	CELP Codec	1.6 MOPS/user	D_s = 1.6 MOPS/user
Signaling	SS7	2 MOPS/site	D_o = 2 MOPS
Aggregate	DSP MOPS		D = 142.6 MOPS per cell site

* D_{if} is off-loaded to dedicated digital hardware and is therefore not included in D.

Figure 3.6 – Contraintes en terme de nombres d'opération par seconde pour une radio logicielle (extrait de "The software radio architecture" par Mitola et al).

De cette étude, nous retenons les interrogations soulevées par la figure 3.5 : les contraintes temporelles sont différentes pour l'ensemble des éléments de la chaîne constitutive d'une radio logicielle. Nous distinguons au niveau physique des opérations critiques au niveau de la génération du signal RF, puis des contraintes de l'ordre de la nanosecondes/microsecondes au niveau du traitement du signal, puis des contraintes de l'ordre de la millisecondes au niveau de la correction d'erreur, puis de la seconde au niveau du protocole. Cependant, ces contraintes ont été établies pour des fréquences plus faibles que nos réseaux sans fil : de l'ordre du Gigahertz. Pour les réseaux 802.11, les symboles transmis sont de l'ordre de la microseconde (4 microsecondes pour un symbole OFDM). quant au messages protocolaires, les temps IFS sont de l'ordre du multiple du slot, soit 9 microsecondes en 802.11a ou 802.11g. Dès lors, ces considérations temporelles sont encore plus exacerbées par les communications hyperfréquences mises en jeu dans 802.11.

Cette problématique des contraintes temporelles est également présente dans d'autres projets concernant les communications radio. L'initiative OpenAirInterface [96] qui

vise à fournir une plate-forme ouverte pour l'expérimentation dans les communications radio numériques est composée de plusieurs sous-ensembles, dont certains sont dédiés notamment à la résolution des problèmes liés au temps réel (à l'aide de radio logicielles et systèmes embarqués utilisant des systèmes d'exploitation temps réel). Parallèlement, le projet inclut aussi des travaux sur les méthodes d'accès (topologies cellulaires et mesh, scheduling intercouche, contrôle de ressource distribué) et sur la mise en réseau sans fil (Gestion de la mobilité, protocoles de routage adaptés en mode mesh et cellulaire). L'intégration de ces différents travaux dès la conception, notamment par l'utilisation de couches physiques simulées par des éléments fonctionnant en temps réel, marque un changement certain dans la conception de nouvelles architectures.

Les radio logicielles mettant en jeu le protocole 802.11 avec des fonctionnalités complètes sont donc pour l'heure assez difficiles à mettre en oeuvre. Cependant, certains projets sont à l'heure de cette rédaction bien avancés. Le projet GNU Software Radio offre un framework assez complet pour la programmation de radio logicielle. Pour la partie matérielle, il repose sur un ensemble de composants nommé USRP, dont plusieurs chercheurs ont évalué les performances, dont Dhar et al. [95]. Pour l'heure, il semble que cette architecture ne permette pas d'atteindre les débits équivalents aux cartes intégrées, notamment à cause de l'architecture matérielle qui ne permet pas de supporter un fonctionnement en temps réel. Cependant, en se limitant à des débits raisonnables et à des fréquences de fonctionnement réduites, les chercheurs ont réussi à intégrer cette radio logicielle et à l'utiliser en temps que couche physique et couche MAC pour le frameworks Click de routage que nous avons étudié précédemment. En une seule approche, les chercheurs ont donc réussi à considérer les éléments de la couche physique comme des éléments Click usuels, permettant une grande flexibilité pour la conception de ces nouvelles radios.

Dhar et al. évoquent aussi dans leur article les possibilités en terme d'approche intercouche : l'espace mémoire étant le même entre les composants radio physique et les composants des couches supérieures, il devient très facile d'envisager des échanges d'informations entre les différentes couches, comme le suggère la figure 3.7. La nature complexe des opérations à effectuer pour assurer les fonctions usuelles en radio fréquence fait qu'elles sont réalisées en espace utilisateur, comme n'importe quelle autre application.

3.5 Conclusions

De cette étude de l'architecture actuelle des systèmes d'exploitation, nous avons pu aisément identifier les raisons du succès de l'architecture en couches. En effet, celle-ci a permis d'isoler chaque problème (accès physique, gestion de l'accès multiple, routage de paquet, établissement d'un canal de transmission) afin de fournir des fonctions de connectivité aux applications. Cependant, comme notre précédente autopsie des

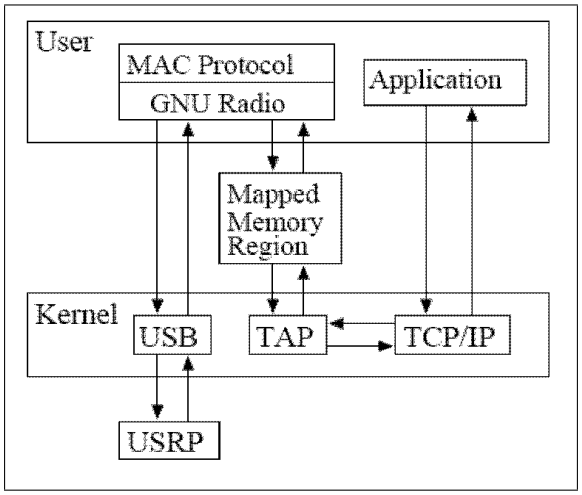


Figure 3.7 – Architecture d'une radio logicielle complète intégrant partie physique (GNURadio) et couches supérieures (Click). (Extrait de la publication de Dhar et al.)

réseaux sans fil 802.11 l'a clairement montré, ce modèle en couche est remis en question lors de son utilisation avec les réseaux sans fil, notamment pour des raisons de performances. Si les chercheurs ont pu avec succès fournir des solutions à plusieurs problèmes de performances (gestion des débits multiples, utilisation raisonnée du canal par des utilisateurs multiples, nouvelles méthodes de communications spontanées dans les réseaux mesh), ces solutions restent trop spécifiques et surtout tendent à diminuer l'intérêt du modèle en couches qui a été un succès architectural comme le réseau Internet le prouve.

La nécessité de mettre en communication l'ensemble des couches du modèle TCP/IP est donc cependant justifiée, notamment au regard de la nature des solutions trouvées. Cependant, ces solutions doivent s'inscrire dans un nouveau modèle architectural, les architectures inter-couches. A ce jour, plusieurs travaux de recherches ont essayé de décrire de manière générique ce modèle, produisant des recommandations sans toutefois atteindre un consensus permettant le développement de ces nouvelles architectures.

L'apport de l'expérimentation a permis de mettre en lumière l'intérêt de travaux visant à transformer le réseau en une suite de blocs fonctionnels, comme le framework Click. Cependant, dans le cadre des réseaux sans fil, les couches basses telles que la couche MAC et surtout la couche physique résistent à cette transformation, notamment à cause de considérations d'ordre temporel. En effet, les exigences temporelles sont croissantes au fur et à mesure que nous plongeons dans les couches basses du modèle TCP/IP.

L'arrivée des radios logicielles et des premiers frameworks expérimentaux (comme le projet GNU Radio) a mis en exergue ces problèmes, et à ce jour il est encore impossible d'atteindre les niveaux de performances en terme de débits physiques sur

des radios logicielles embarquées sur du matériel usuel. Cependant, ces limitations peuvent être levées par une adaptation de l'architecture de communication entre les différents composants.

Un des points de départ de ces nouvelles architectures peut être la vision issue de la recherche en sécurité : dans ce domaine, le réseau est considéré au niveau de son entité la plus élémentaire, le paquet. Les outils développés dans cet discipline offrent des possibilités très intéressantes en termes de manipulation de ces paquets et du coup, offre des possibilités intéressantes pour la création rapide d'architectures sans fil spontanées. De plus, déjà en espace utilisateur, ils peuvent s'intégrer plus facilement à des architectures radio logicielles futures.

Il est donc possible de repenser l'architecture des réseaux actuels en raisonnant sur la réaction du système à la réception des paquets. En effet, que ce soit au niveau routage, transport, session ou applicatif, les mécanismes de machine à état sont principalement dictés par des événements liés à la réception de paquets : découverte de nouvelle routes par envoi de paquet, acquittement au niveau TCP par envoi d'un paquet donné, et bien évidemment, changement de l'application à la réception de certains paquets.

Ceci isole de facto les couches à partir de la couche IP : en effet, la granularité temporelle de ces couches est identique. Au niveau MAC, les choses sont déjà différentes : l'envoi d'un paquet est soumis au mécanisme de multiplexage statistique, qui dépend de paramètres liés à l'occupation physique du canal, ce qui, indirectement, lie une partie de la couche physique et de la couche MAC au sein d'un même espace temporel. La couche physique peut-être isolée temporellement à cause des phénomènes mis en jeu (durée d'un symbole sur le canal physique très courte).

Avec ces nouvelles bases, nous pouvons élaborer une nouvelle architecture tenant compte des contraintes irréductibles tout en s'ouvrant aux perspectives d'une vision intercouche.

Deuxième partie

Contributions

CHAPITRE 4

CADRE DES TRAVAUX

4.1 Objectif : vers des réseaux natifs sans fil

Les réseaux sans fil de nouvelles générations peuvent revêtir plusieurs aspects, de par leur exceptionnelle souplesse à assurer les fonctions de réseaux autrefois disjoints (WPAN, WLAN, WMAN, WWAN). Il est donc nécessaire de préciser le cadre de nos travaux, qui se placent dans une optique de changement de l'architecture historique. Le but avoué est de s'affranchir des contraintes des systèmes en couches, permettant ainsi le développement plus générique d'optimisations liées à la vision inter-couches.

Mais nous l'avons vu, les contraintes sont fortes : d'une part, il faut maintenir un support stable similaire à celui fourni par les architectures en couches, encourageant la réutilisation d'un maximum d'éléments tout en éliminant le problème du code spaghetti, c'est à dire l'usage régulier de saut, exception et autre gestion événementielle déviante, qui indiquerait plus une architecture conçue pour la gestion des exceptions, et non d'une vision générique du réseau.

Notre étude précédente a mis aussi en lumière le problème des différentes exigences temporelles liées à la nature même des phénomènes mis en jeu dans les réseaux sans fil. Il nous apparait nécessaire de délimiter très rapidement les domaines temporels communs, c'est à dire, associer certaines des couches traditionnelles en une supercouche temporellement cohérente.

Enfin, et c'est le but originel de notre réflexion sur les réseaux sans fil, nous cherchons à diversifier l'utilisation actuelle des réseaux sans fil. Cependant, l'évolution des usages doit tenir compte des aspects de compatibilité d'une part avec l'ensemble des matériels, et d'autre part avec l'ensemble des protocoles réseaux en interconnexion.

Dans ce chapitre, nous commençons donc par énumérer les usages visés. Ensuite, à partir de là, nous établissons les besoins intrinsèques en compatibilité, puis les besoins en fonctionnalités fort de ce que nous avons observé dans les travaux de la commu-

nauté scientifique. Puis, à partir de cela, nous formulerons notre proposition.

4.2 Usages envisagés

La nouvelle architecture doit permettre aux réseaux sans fil de nouvelle génération d'effectuer l'ensemble des rôles prévus dans les réseaux WPAN, WLAN, WMAN, WWAN.

Ces rôles, même s'ils convergent maintenant autour du même protocole IP, mettent en oeuvre des approches radicalement différentes. Les réseaux WPAN sont plutôt dédiés à la communication interpériphérique : les mécanismes sont donc limités, tant en terme de capacité d'adressage que de routage ; bien souvent, la communication privilégie uniquement les noeuds à portée radio, comme dans les réseaux Bluetooth (ou 802.15.1). Ces réseaux présentent également la particularité de viser une consommation d'énergie réduite, comme les réseaux ZigBee (ou 802.15.4). Ces réseaux présentent l'avantage de ne requérir aucun point central d'interconnexion pour leur usage.

A l'inverse, les réseaux métropolitains et globaux, comme le Wimax et la famille GSM tendent à une vision cellule/client et donc à une architecture plutôt centralisée, avec une gestion fine des fréquences de fonctionnement et de l'accès au canal facilitée par la centralisation. Ce type d'usage est intéressant pour l'interconnexion de deux noeuds distants, car les mécanismes sont adaptés à l'échelle de ce type de communication, que ce soit en terme d'adressage et de routage. De plus, ces réseaux utilisent des méthodes d'accès déterministes, basées sur de l'allocation temporelle TDMA, ce qui contribue à garantir une équité du trafic pour l'ensemble des clients d'une même borne, au détriment de l'utilisation maximale de la capacité radio de la borne.

Et donc, à mi-chemin de l'ensemble de ces usages, les réseaux sans fil de nouvelle génération, bâtis sur les caractéristiques de réseaux WLAN (usage libre de la bande radio notamment), doivent réussir à réunir l'ensemble des qualités mentionnées ci-dessous. Nous avons bon espoir d'y parvenir, car ces réseaux présentent une souplesse intéressante, que ce soit en terme de diversité de modulations physiques (possibilité donc d'adapter la portée et donc la taille des réseaux), d'accès au canal (usage d'un accès probabiliste ou déterministe), et plus généralement, en terme d'adressage et de routage (apport d'IP).

Nous pouvons désormais évoquer les contraintes qui en résultent.

4.3 Contraintes historiques

Les contraintes intrinsèques des réseaux sans fil de nouvelle génération sont avant tout celles des réseaux ad hocs, et donc avant tout chose, d'assurer la communication entre deux noeuds quelconques. Ces considérations sont développées dans Ananas [99], où les auteurs définissent plusieurs contraintes.

4.3.1 Fonctionnement en réseau local

Le premier challenge est d'arriver à obtenir un fonctionnement identique fonctionnellement aux réseaux locaux. Dans ces réseaux, la communication entre deux nodes est facilitée par le fait que les changements de topologie sont propagés très rapidement. Cette hypothèse n'est plus valide dans le cadre des réseaux ad hoc, composés de multiples sous-réseaux dynamiques et de technologies et débits variables. Dès lors, la recherche en routage dans ces réseaux deviennent spécifiques, ce qui constituent le gros des résultats du groupe de recherche MANET, dont nous avons évoqué brièvement les travaux précédemment. Les résultats de ces recherches aboutissent en une famille nombreuse de protocoles de routage, de types variés (proactif et réactifs, voire hybrides) qui permettent l'interconnexion de deux noeuds.

Un autre aspect du fonctionnement est la problématique de la diffusion, communément représentée par les notions de broadcast et de multicast. La nature broadcast du médium radio modifie considérablement le mode de diffusion, puisque naturelle aux abord du noeud diffusant, mais très couteuse en matière de diffusion aux autres noeuds. Les techniques d'innondation (flooding) sont inefficaces en matière de gestion de la bande passante. Pourtant, la diffusion reste un élément essentiel, d'une part car elle est utilisée par nombre d'algorithme de routage, et d'autres par certains protocoles, comme le protocole IPv6. La clé d'une compatibilité des réseaux de nouvelle génération avec les réseaux existants réside dans le maintien de cette fonctionnalité.

4.3.2 Pile TCP/IP

Il est nécessaire de maintenir la pile TCP/IP une fois la connectivité assurée, afin de conserver les applications et protocoles développés sur cette pile. Plus encore, derrière la pile TCP/IP se cache toute la problématique de l'adressage, qui doit permettre une interconnexion avec les réseaux existants, tout en maintenant ses qualités propres, à savoir assurer l'identification des différents noeuds et la possibilité de déterminer une route vers un noeud connaissant l'adresse de celui-ci. Enfin, les techniques d'autoconfiguration sont communément utilisées dans le domaine IP : IPv4 repose principalement sur le protocole DHCP pour obtenir une adresse, tandis qu'IPv6 repose sur des mécanismes d'annonce de préfixe pour assurer son autoconfiguration.

4.3.3 Connectivité à Internet

Les réseaux de nouvelles générations, s'ils constituent une nouvelle vision de la communication, en permettant les échanges de proximité comme les échanges longues distances, devront néanmoins permettre et assurer les transports des données vers l'Internet. D'une part, le contenu des réseaux, même s'il est constitué pour une part d'échange entre individu, repose aussi sur l'accès à des ressources centralisées fourni par les différents acteurs d'Internet, comme les moteurs de recherche, ou les sites de

contenus.

Les mécanismes d'adressage, de routage devront donc assurer l'interconnexion avec l'extérieur. Cependant, avec le fort développement des accès aux débits personnels, on peut tout à fait imaginer que les accès à Internet seront multiples mais pas forcément uniformément distribué sur l'ensemble des réseaux sans fil. Cette problématique a été évoqué dans MANET lorsque le problème des passerelles a été soulevé. De nombreux protocoles de routages, comme OLSR, offre cependant une gestion intégrée des passerelles pour l'accès à Internet. Ces travaux pourront donc être exploités dans les réseaux de nouvelles générations.

4.4 Proposition

Il est temps de formuler notre proposition concernant la réalisation de ces réseaux sans fil de nouvelle génération. La conception d'une nouvelle architecture étant un but ambitieux, nous cherchons à limiter la portée de notre travail à la réalisation d'une architecture générique, permettant d'une part la réutillisation de l'ensemble des travaux actuels de recherche sur les réseaux sans fil, notamment 802.11, que nous avons étudié dans notre autopsie, et d'autre part permettant la conception rapide de nouvelles visions du réseau, utilisant du matériel générique disponible sur le marché, sans souffrir des limites actuelles du modèle en couches, comme les récents travaux en matière d'architecture inter-couches et en matière de radio logicielle le prouvent.

L'étude précédente de ces nouvelles approches a soulevé le délicat problème des exigences temporelles des différents éléments constitutifs du réseau, de même que l'actuelle impossibilité d'utiliser une architecture basée sur les radios logicielles. Nous devons donc en tenir compte lors de la conception de l'architecture.

4.4.1 Reconsidérer la vision en couches

La contrainte temporelle étant la plus forte, nous proposons de diviser l'architecture en deux composants aux temporalités différentes.

Notre idée est de séparer le modèle en couche en deux sous-ensembles, d'une part un sous-ensemble lié à la notion des paquets, dotés d'exigences temporelles accessibles par nos ordinateurs, et d'autre part, un sous-ensemble lié à l'échelle du slot, voir des symboles transmis sur le médium physique, dont le traitement n'est accessible que par des composants dédiés. Nous rejoignons en cela les propositions de chercheurs évoquées dans les architectures inter-couches.

L'intérêt de cette dissociation permet de résoudre temporairement les problèmes liés aux contraintes des couches basses non-aisément modifiables : en les isolant et en définissant une méthode d'interaction avec un sous-ensemble de gestion de paquet, nous pensons éviter un écueil fréquent dans les travaux que nous avons étudiés : la forte spécialisation des solutions aux problèmes soulevés. Si cette démarche est perti-

nente pour l'analyse d'un problème, lors de sa résolution, il est nécessaire d'essayer d'adopter une solution générique, du moins, fournir le substrat rendant possible la résolution commune des problèmes. Avec cette segmentation, nous espérons à la fois pouvoir fournir une architecture permettant la mise en place d'optimisation inter-couches au niveau du traitement des paquets, tout en permettant la réalisation des solutions proposées au niveau des méthodes d'accès.

Pour évaluer la pertinence de cette idée, nous avons décidé de procéder par deux approches complémentaires :

- d'une part, nous avons essayé d'évaluer les possibilités du hardware actuel en matière de modification du comportement de la couche d'accès. Pour ce faire, nous avons cherché à implémenter une méthode d'accès proche de celle de 802.11, Idle Sense, développée par l'équipe DRAKKAR et jusque là, évaluée sous forme de simulations. Nous consacrons donc un des chapitres qui suit à l'utilisation de cette plate-forme pour implémenter la méthode d'accès IdleSense et qui servira de prétexte à l'évaluation des possibilités d'une plate-forme réelle en tant que plate-forme reprogrammable. Le chapitre relate les problématiques liées au passage de la simulation à l'implémentation, les problèmes posés par un environnement contraint, puis évoque les perspectives d'une telle plate-forme, et de sa capacité à interagir avec le système hôte.

- d'autre part, nous nous concentrons ensuite sur une architecture de gestion des paquets, non pas conçue dans l'esprit habituel des systèmes d'exploitation, mais d'un point de vue plus réseau. Pour ce faire, nous proposons dans le chapitre dédié à cette étude cette architecture en nous basant sur les travaux et outils développés par le domaine de la sécurité : l'approche dans cette optique est basée sur un traitement et une analyse de chaque paquet en espace utilisateur, à l'aide d'outils permettant l'implémentation très rapide des protocoles existants, tel le formalisme utilisé par Scapy ou Click. Ainsi, nous espérons obtenir un système de gestion des paquets, similaire en ce sens aux premières propositions dans le domaine des réseaux qui cherchaient à concevoir des systèmes de commutation de paquets. Sur ce système, nous reconstruirons la gestion habituelle des réseaux : les protocoles traditionnels comme 802.11 ou IP ne seront vus simplement comme des applications pour ce système de gestion. Ces applications fonctionnant dans le même espace mémoire, nous pouvons montrer la capacité du système à fournir des solutions inter-couches. Enfin, à l'aide d'études de cas liés à des problèmes concrets, comme la gestion de la comptabilité avec les équipements existants ainsi que la gestion de mobilité, nous nous efforçons de montrer la pertinence de ce système sur des cas réels en proposant des approches différentes à la résolution de ces problèmes.

4.5 Conclusion

Par ce chapitre court, nous nous sommes efforcés de préciser le cadre dans lequel s'inscrivent nos travaux de thèse. Ces travaux, orientés initialement vers la mise en place de nouvelles architectures pour les réseaux sans fil de nouvelle génération, ont débouché suite à l'état de l'art à la possibilité d'utiliser le matériel actuel comme plate-forme pour ces nouvelles architectures. Cette approche s'explique par la richesse des travaux accomplis par la recherche : dans notre autopsie, nous avons eu l'occasion d'observer que les réseaux 802.11 avaient permis une large exploration des problèmes et avaient proposé, grâce à des groupes comme MANET, un ensemble de solutions nouvelles, profitant souvent d'une meilleure communication entre les couches. A partir de là, lorsque nous avons étudié l'architecture actuelle des réseaux. Il est apparu que certains chercheurs visaient à offrir des architectures génériques à la résolution de ces problèmes, grâce à l'apport de la vision inter-couches. Certains architectures existent en terme de prototype, comme Click, mais cependant, ces architectures ne s'intégrent pas facilement à la résolution des problèmes soulevés par les couches basses, à cause de leur dépendance dans le modèle en couches dans les systèmes d'exploitations actuels.

C'est pourquoi le renouveau apporté par le domaine de la sécurité (fabrication de paquets, fuzzing...) est important. En traitant le réseau comme un gestionnaire de paquets en mode utilisateur, les outils développés par cette branche de la recherche permettent d'aborder différemment le problème architectural. Notre proposition s'appuie donc sur cette hypothèse de travail pour fournir une architecture générique, qui permettra d'une part l'utilisation des solutions proposées précédemment à chacun des problèmes soulevés, et d'autre part pour fournir de nouvelles approches nativement inter-couches, puisque fonctionnement dans le même espace.

Pour ce faire, nous avons pris la décision de dissocier les couches actuelles en fonction de leurs exigences temporelles, et de formuler le problème comme étant la recherche de la meilleure interaction possible entre les deux sous-ensembles. Nous pouvons donc maintenant traiter chaque problème indépendamment.

CHAPITRE 5

PLATES-FORMES 802.11 : VERS DES MÉTHODES D'ACCÈS FLEXIBLES

Dans ce chapitre, nous cherchons à évaluer la capacité des plates-formes 802.11 actuellement sur le marché à assurer le rôle de plates-formes reprogrammables pour la réalisation des réseaux sans fil de nouvelle génération. Pour ce faire, nous proposons d'implémenter une méthode d'accès, Idle Sense, développée en 2003 au sein de l'équipe Drakkar, et visant à améliorer l'équité des réseaux sans fil 802.11. Cette méthode d'accès étant basée sur une modification dynamique de la fenêtre de contention en fonction de l'état du canal radio, elle nécessite donc d'intervenir à bas niveau dans le fonctionnement usuel des couches. Elle apparaît comme une excellente application pour évaluer la pertinence de l'utilisation du matériel usuel dans les réseaux sans fil de nouvelle génération.

Nous présentons tout d'abord la méthode d'accès telle qu'elle a été proposée lors des publications par ses auteurs ; puis, en isolant les spécificités requises, nous isolons un ensemble de plates-formes candidates. Nous présentons ensuite les limitations de la plate-forme retenue et donc les incidences sur les algorithmes initiaux de la méthode d'accès, qui orienteront l'implémentation finale. Nous procédons à l'évaluation des performances de la plate-forme en réutilisant au maximum les indices de performances utilisés initialement par les auteurs, pour enfin tirer un ensemble de remarques relatives à cette expérience.

5.1 La méthode d'accès Idle Sense

Cette méthode d'accès s'apparente aux méthodes d'accès déjà présentées lors de notre autopsie des réseaux 802.11 comme solutions à la sous-utilisation du canal par la méthode d'accès d'origine, DCF. Dans une première publication [36], les auteurs énoncent

le principe de base du fonctionnement de leur méthode d'accès : Idle Sense utilise une information commune sur l'utilisation du réseau, obtenue à partir de l'écoute de l'état du canal, pour déterminer la fenêtre de contention optimale. Idle Sense utilise ensuite un algorithme AIMD (Additive Increase/Multiple Decrease) pour faire évoluer la fenêtre de contention. Plus tard, dans une seconde publication, les auteurs ont proposé des améliorations de l'algorithme d'origine. Nous présentons ces travaux ci-dessous.

5.1.1 Principes

Idle Sense optimise la méthode de DCF de 802.11 pour obtenir une plus grande bande passante et une meilleure équité. Les stations qui concourent pour l'accès au réseau n'utilisent donc plus l'algorithme de Binary Exponential Backoff après des collisions ou des transmissions ratées, mais utilisent au contraire une fenêtre dynamique (CW) qui va converger de manière distribuée vers une valeur commune pour l'ensemble des stations en n'effectuant qu'une surveillance des slots vacants entre deux transmissions consécutives.

La méthode fonctionne de la manière suivante : chaque station mesure n_i, le nombre de slots consécutifs vacants entre deux tentatives de transmission. Lorsque le nombre de transmissions *maxtrans* est atteint, l'algorithme procède à une évaluation de $\hat{n_i}$, le nombre moyen de n_i observés. Puis l'algorithme utilise cette valeur pour ajuster la fenêtre de contention pour atteindre une valeur objectif ${n_i}^{target}$ calculée préalablement pour une norme de 802.11 donnée, en fonction des paramètres temporels de la couche PHY et MAC.

L'algorithme original d'Idle Sense fait tendre n_i vers ${n_i}^{target}$ en appliquant AIMD (*Additive Increase Multiplicative Decrease*) à la probabilité d'émission P_{e}. Quand les stations n'effectuent plus l'algorithme original de backoff exponentiel de 802.11, la probabilité de transmission peut s'exprimer suivant :

$$P_{\mathrm{e}} = \frac{2}{CW + 1}. \tag{5.1}$$

Si une station observe trop de slots vides vis à vis de la valeur à atteindre, elle doit donc augmenter la probabilité d'émission P_{e}, qui à son tour diminuera le nombre de slots n_{i} ; à l'inverse, si trop peu de slots de silence sont observés, la station augmente de manière multiplicative la probabilité P_{e}, qui augmentera à son tour le nombre de slots de silence observés n_{i}, ce qui conduit à l'algorithme suivant :

- If $n_i \geq {n_i}^{target}$, $P_{\mathrm{e}} \leftarrow P_{\mathrm{e}} + \epsilon$
- If $n_i < {n_i}^{target}$, $P_{\mathrm{e}} \leftarrow \alpha P_{\mathrm{e}}$

où ϵ et α sont des paramètres d'adaptation. En pratique, les stations modifient directement leur *CW*s en combinant ces règles de mise à jour avec l'équation 5.1.

Idle Sense propose une méthode distribuée pour le contrôle de la contention mais offre également la possibilité de distinguer les pertes liées aux collisions des pertes liées à des transmissions ratées sur le médium physique. La méthode permet donc de découpler le problème du contrôle de charge de l'adaptation aux conditions du canal, ce qui permet la résolution des problèmes induits par l'utilisation d'algorithme tels que ARF sur l'équité entre les différents hôtes.

5.1.2 Evolution de l'algorithme initial

Les auteurs ont fait évoluer l'algorithme théorique d'Idle Sense. Celui-ci repose en effet sur l'application d'AIMD sur la probabilité d'émission P_e. Cependant, la manipulation de cette variable n'est pas aisée, car elle requiert une bornage afin que celle-ci ne deviennent pas plus grande que 1, ce qui révèle un problème dans l'approche originale. Les auteurs de la méthode d'accès ont donc proposé une substitution de la probabilité d'accès par la fenêtre de contention CW. Ainsi, l'application d'AIMD permet le contrôle d'une entité entière positive, adaptée à AIMD :

- If $n_i \geq n_i^{target}, CW \leftarrow \alpha \cdot CW$
- If $n_i < n_i^{target}, CW \leftarrow CW + \epsilon$

Ce nouvel algorithme limite également les modifications à apporter aux systèmes existants, car il ne nécessite plus de déterminer la probabilité P_e pour déterminer la fenêtre de contention CW. Suite aux travaux d'Elena Lopez-Aguilera [101], les auteurs ont également remarqué que la précision de l'algorithme augmente si l'on ajuste le paramètre *maxtrans* pour qu'il devienne proportionnel au nombre de stations (chaque station calculant sa CW toutes les *maxtrans* transmissions). Avec l'algorithme original, les stations mettent à jour leur CW plusieurs fois entre deux transmissions propres, qui est le moment où le CW est effectivement utilisé. Cela signifie que pour un jeu de paramètre AIMD donné, le comportement de l'algorithme change en fonction du nombre de stations, ce qui n'est pas satisfaisant. Les auteurs ont donc suggéré d'utiliser le fait qu'Idle Sense fait converger la fenêtre de contention en fonction du nombre de stations dans une cellule. Puis, pour accélérer la convergence quand $\hat{n}_i$ est trop éloigné de l'objectif, ils utilisent une petite valeur de maxtrans quand n_i est vraiment différent de la valeur désirée. Le mécanisme d'adaptation raffiné est donc le suivant :

- If $\left| n_i - n_i^{target} \right| < \beta \rightarrow maxtrans = \frac{CW}{\gamma}$,
- If $\left| n_i - n_i^{target} \right| \geq \beta \rightarrow maxtrans = 5$,

où γprend la valeur $\gamma = 4$. En utilisant les Eq. 5 and 9 de , les auteurs obtiennent les relations suivantes

- Pour IEEE 802.11b, $maxtrans = \frac{CW}{\gamma} \approx 3N$
- Pour IEEE 802.11a/g, $maxtrans = \frac{CW}{\gamma} \approx 2N$

L'algorithme 1 présente la spécification formelle d'Idle Sense.

Algorithm 1 *Idle Sense*

$maxtrans \leftarrow 5$; $sum \leftarrow 0$; $ntrans \leftarrow 0$
After each transmission {
/* Station observes n_i idle slots before a transmission */
$sum \leftarrow sum + n_i$
$ntrans \leftarrow ntrans + 1$
if $(ntrans >= maxtrans)$ **then**
  /* Compute the estimator */
  $\hat{n_i} \leftarrow sum/ntrans$
  /* Reset variables */
  $sum \leftarrow 0$
  $ntrans \leftarrow 0$
  if $(\hat{n_i} < n_i^{\text{target}})$ **then**
    /* Increase CW */
    $CW \leftarrow CW + \epsilon$
  else
    /* Decrease CW */
    $CW \leftarrow \alpha \cdot CW$
  end if
  if $(\left| n_i^{\text{target}} - \hat{n_i} \right| < \beta)$ **then**
    $maxtrans \leftarrow \frac{CW}{\gamma}$
  else
    $maxtrans \leftarrow 5$
  end if
end if
}

5.2 Implémentation

L'objectif est d'implémenter la méthode d'accès Idle Sense sur un nombre significatif de cartes 802.11 afin de pouvoir évaluer l'équité entre plusieurs stations en concurrence et reproduire une partie des résultats théoriques des publications précédentes. Il est donc exclu de développer sur un matériel dédié au développement : nous préférons une méthode visant à modifier le firmware d'une carte disponible sur le marché.

Le nombre de cartes et de chipsets disponibles sont plutôt conséquents, mais elles doivent ici respecter un certain nombre de prérequis : Idle Sense repose en effet sur la capacité de la carte à observer le nombre de slots vides et sur sa capacité à modifier dynamiquement la fenêtre de contention.

5.2.1 Prérequis sur le matériel

Le matériel choisi doit fournir un accès à des fonctions de bas niveau : comptage de slots et gestion de la fenêtre de contention. Comme nous l'avons vu à maintes reprises dans notre état de l'art, la communauté des chercheurs utilise largement les chipset Atheros, grâce au driver libre MadWIFI, qui donne accès au code source de l'ensemble des commandes envoyées au contrôleur de la carte sans fil. Cependant, MadWIFI n'offre aucune fonction de bibliothèque (API) pour accéder à l'envoi ou plus précisément au mécanisme d'envoi des paquets. Au mieux, nous pouvons obtenir un compte rendu sur l'envoi précédent d'un paquet, à savoir si l'envoi a été effectué directement ou si des tentatives successives ont été nécessaires, ou si l'envoi s'est soldé par un échec. De plus, l'existence de files d'attentes multiples sur les cartes Atheros suggèrent que ces cartes gèrent de manière autonome l'envoi de trames, ce qui semble raisonnable en regard du système hôte qui ne peut garantir les contraintes temps-réel de l'envoi de trames.

Le chipset Atheros est donc inutilisable pour tout modification concernant le niveau temps réel. Cependant, plusieurs autres candidats existent. Les chipsets Prism54 et Broadcom exigent qu'au chargement de leur pilote un micro-logiciel soit envoyé sur la carte enfin d'initier la transmission et la réception de paquets, de telle sorte que la carte peut se comporter différemment en fonction du micro-logiciel utilisé : cela facilite l'ajout de nouvelles fonctionnalités, comme l'ajout du protocole de chiffrement WPA dans la cadre de la révision 802.11i du protocole.

Le projet Prism54 fournit des informations intéressantes sur le fonctionnement interne du chipset : il s'agit d'un processeur ARM classique connecté à un composant radio-fréquence RF. Le projet propose deux types de firmware : FullMAC et SoftMAC. FullMAC est un firmware pour les premiers adaptateurs fabriqués par Intersil. Ces adaptateurs requierent un firmware monolithique qui implémente l'intégralité du protocole 802.11 pour le faire fonctionner sur la carte. Pour réduire les coûts de production, le firmware SoftMAC a été développé en séparant le firmware original FullMAC en deux morceaux : la couche LMAC (Lower MAC) executée en temps réel sur le processeur ARM et la couche UMAC (Upper MAC) exécutée par le logiciel sur le système hôte par un jeu d'API privées. Du reverse engineering sur cette composante binaire a montré que cette partie du firmware implémentait l'envoi de trames de manière logicielle : cette famille de composant est donc théoriquement utilisable pour implémenter Idle Sense. Cependant le principal obstacle est la nécessité d'étudier le code binaire pour modifier le comportement d'envoi des paquets. Le chipset Broadcom

présente malheureusement les mêmes restrictions quant à l'étude du fonctionnement de son micro-logiciel.

Les cartes Intel basées sur le chipset IPW2200 ou IPW2915 ont le même type d'architecture logicielle : le firmware contrôle le comportement de la carte. L'étude des drivers d'Intel nous indique que le firmware est spécifique à chaque mode (infrastructure, ad hoc, ou point d'accès) et est fourni sous forme de 3 composants : bootcode, microcode, et firmware proprement dit. Le bootcode initialise les fonctions basiques du matériel ; le microcode est responsable des opérations temps réel (ie, le comportement de la carte vis à vis de l'activité radio, c'est à dire la couche MAC) et le firmware, qui gère la machine à état 802.11. L'ensemble de ces éléments logiciels est envoyé sur la carte par le driver à l'initialisation avant toute connexion. Après, le driver pilote le firmware par un ensemble de commandes (envoi/réception de paquets, changement de canal, association, désassociation...).

En 2006, à l'occasion d'un stage au sein du laboratoire Intel Labs à Cambridge (UK), nous avons pu avoir accès aux sources du micro-logiciel grâce à Dina Papagiannaki : nous avons pu implémenter Idle Sense à cette occasion sur la plate-forme intel Centrino sous la forme d'un nouveau microcode chargeable par les pilotes existants sous Linux et FreeBSD.

Le microcode est programmé en langage assembleur et le firmware en langage C. Afin d'implémenter Idle Sense, nous devons donc modifier le microcode, responsable des opérations de la couche MAC.

5.2.2 Simplification des calculs

Jusqu'ici, Idle Sense a été évalué en utilisant des simulations pour optimiser les différents paramètres tel que n_i^{target}et les coefficients AIMD. Deux simulateurs à évènements discrets différents ont été utilisés : un qui modélise le comportement PHY et MAC [101], et un autre dédié à la couche MAC [36]. Les paramètres sont représentés comme des nombres flottants, les calculs sont donc d'une grande précision.

n_i^{target}est calculé à partir de la valeur optimale de la CW donnée par la formule indiquée ici. $\frac{1}{\alpha}$ et ϵ sont des paramètres qui impactent la stabilité de la boucle de rétroaction. Les valeurs de ses paramètres sont ajustées en simulation pour obtenir un bon compromis entre stabilité et vitesse de convergence.

Les auteurs ont obtenu par simulation finalement les valeurs suivantes :

- $n_i^{\text{target}} = 3.91$
- $\frac{1}{\alpha} = 1.0666$
- $\epsilon = 6.0$
- $\beta = 0.75$
- $\gamma = 4$

Cependant, la plate-forme Intel s'appuie pour la programmation de la partie temps réel sur du code écrit en assembleur ; dès lors, les fonctions arithmétiques et les différents formats de données offerts sont forcément restreints. La plate-forme Intel ne propose que l'utilisation des entiers et ne supporte pas directement la division ou la multiplication. Nous devons donc choisir convenablement les bonnes valeurs et les techniques de programmation.

Nous avons donc décidé d'utiliser les valeurs les plus proches de la puissance de 2 immédiatement proche de la valeur de chaque paramètre. En effet, le langage assembleur offre un ensemble d'opérande utilisés pour la modification et le décalage de registre, ce qui correspond à des multiplications ou à des divisions par 2. Par exemple, nous pouvons approximer la valeur du paramètre α, qui est égal à 0.93755, avec le ratio suivant : $15/16 = 0.9375$. Ce ratio peut être calculé par $1 - 1/16$ qui est une opération facile à exécuter en langage assembleur. Nous avons donc adopté l'ensemble de ces valeurs comme paramètres :

- $n_i^{target} = 4$
- $\alpha = 1 - 1/16$
- $\epsilon = 6$
- $\beta = 1$
- $\gamma = 4$

Nous pouvons réécrire notre boucle de contrôle en langage pseudo-assembleur avec les valeurs choisies en paramètres :

```
1  ; after each transmission:
2      mov ntrans AX
3      addi    1 AX
4      mov AX ntrans
5      mov maxtrans BX
6      cmp
7      jmp     lt noupdate
8  ; we have reached maximal transmission
9      swap    AX BX
10     shl ; *2
11     shl ; *2
12 ; we got maxtrans* target (=4)
13     mov sum BX
14     cmp
15     jmp gt increase
16 decrease:
17     mov CW AX
18     mov AX BX
19     shr ; /2
```

```
20      shr ; /4
21      shr ; /8
22      shr ; /16
23      swap    AX BX
24      sub
25  ; now we have 15/16 of CW in AX
26      jmp update
27  increase:
28      mov CW AX
29      mov 6 BX
30      add
31  update:
32      mov AX CW
33  noupdate:
```

5.3 Contraintes particulières

5.3.1 Générateur aléatoire

Dans le standard 802.11, la taille de la fenêtre de contention prend exclusivement des valeurs de puissance de deux, par exemple la valeur de $CW_{\text{min}} = 31$ implique que la station va choisir une valeur de backoff comprise entre 0 et 31. Le mécanisme de backoff exponentiel double la taille de la CW à chaque tentative de transmission échouée jusqu'à la limite de CW égale à 1023 for 802.11b. L'implémentation de la génération aléatoire du backoff est donc très simple : chaque nouvelle incrémentation du backoff est obtenu par décalage d'un registre, où les bits inutiles sont ensuite masqués.

La figure 5.1 montre un exemple de mesure de la distribution des valeurs aléatoires sur les cartes intel pour la fenêtre de contention en 802.11a ($CW_{\text{min}} = 15$). Dans cette expérience, une station essaie d'émettre des trames UDP le plus vite possible. Nous utilisons une méthode de mesures précise des temps d'interarrivées, qui permet des mesures de l'ordre de 1 μs sur du matériel standard. Le principe est d'enregistrer le traffic entrant sur le point d'accès destinataire et de mesurer le tampon d'arrivée du paquet avec une précision correcte et de mesurer le délai entre deux paquets. Les résultats se présentent sous la forme d'un histogramme, qui nous permet d'identifier les valeurs les plus fréquentes dans les durées de transmission et d'en déduire le comportement sous-jacent de la carte émettrice. Dans la figure présentée, nous observons 16 pics qui ont la même amplitude et qui correspondent aux différents valeurs prises par le tirage de la valeur aléatoire de la fenêtre de contention.

Dans Idle Sense, la valeur de la contention CW peut prendre n'importe quelle valeur, pas nécessairement une valeur issue d'une puissance de deux. Cela signifie que nous devons trouver un autre moyen de générer des valeurs aléatoires dans l'intervalle $[0, CW]$. Notre idée pour résoudre ce problème est de multiplier CW en utilisant une boucle d'addition successive (opérande ADD disponible) par un nombre pseudo-

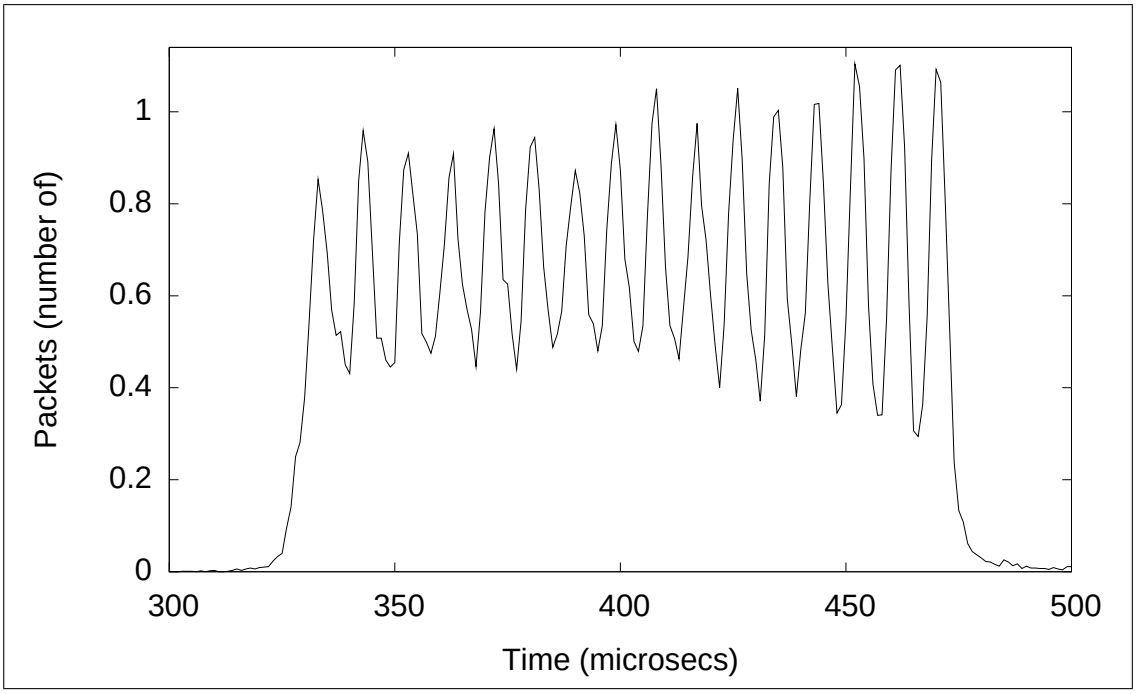


Figure 5.1 – Histogramme des interarrivées sur le firmware standard d'une carte intel, en 802.11a, $CW_{\min} = 15$.

aléatoire sur k bits et de la diviser par 2^k (obtenu par décalage de registre). L'algorithme 2 présente l'implémentation de l'algorithme de génération de nombre aléatoire dans l'intervalle $[0, CW]$ (LFSR signifiant Linear Feedback Shift Register, registre présent sur la carte et utilisé pour la génération de séquence pseudo-aléatoire). Comme l'unité arithmétique et logique fonctionne sur des mots d'une longueur de 16 bits utilise des opérandes sur 8 bits, la méthode est limitée sur le matériel utilisé à une valeur maximale de $CW = 255$.

La valeur maximale nous permet de gérer jusqu'à 28 stations actives avec des performances qui se dégraderont en cas d'activité supplémentaire. Un générateur plus

Variable	Value	Comment
Random register	**10010011** 10101110	generated
Random register	00000000 **10010011**	register shift
CW	00000000 **00001101**	CW=13
Temp	**00000111** 01110111	Random * CW
Backoff	00000000 **00000111**	Backoff = 7

Figure 5.2 – Exemple de génération d'un backoff pour $CW = 13$.

Algorithm 2 *Generation of Random Backoff*

/* take upper 8 bits */
$random \leftarrow LFSR$ & $0x00FF$
$random >> 8$
$temp \leftarrow random * CW$
/* take upper 8 bits */
$backoff \leftarrow temp$ & $0xFF00$
$backoff >> 8$
}

élaboré permettrait d'éliminer cet écueil. Un exemple de génération de backoff est présenté sur la figure 5.2.

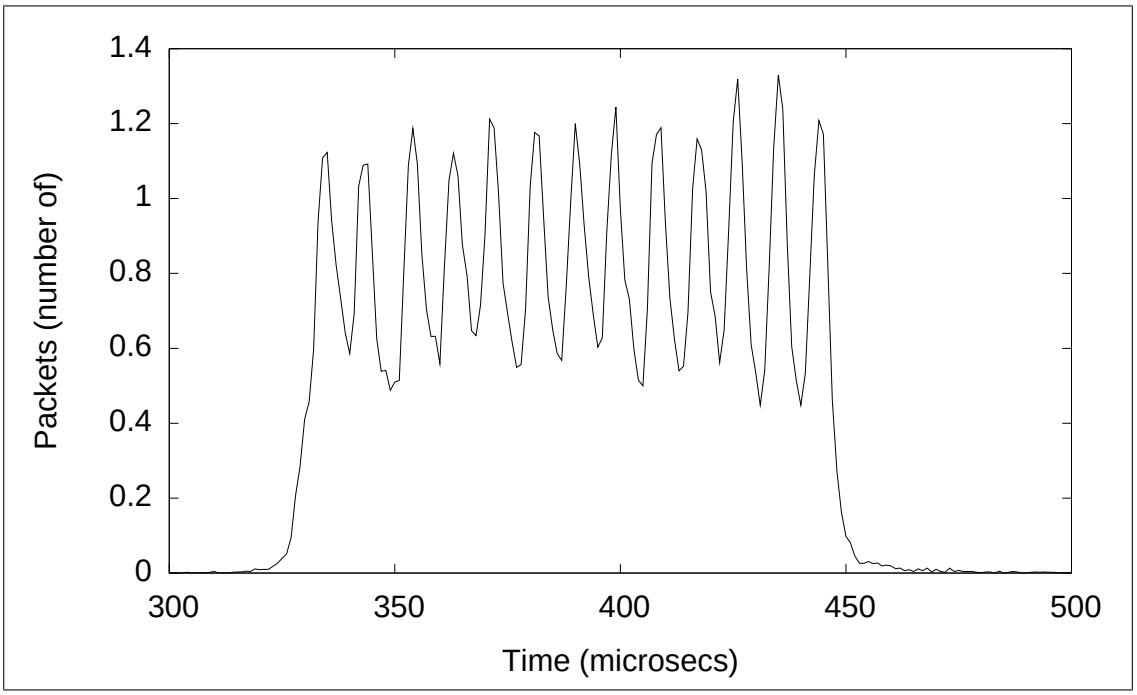


Figure 5.3 – Histogrammes d'interarrivées pour le firmware *Idle Sense* avec $CW = 12$.

Les figures 5.3 et 5.4 présentent les histogrammes obtenus dans une expérience similaire, mais avec une fenêtre de contention fixée à 12 et 13, respectivement. Nous observons de bonnes propriétés statistiques, puisque les pics disposent d'une amplitude similaire, ce qui indique une génération sur un intervalle relativement uniforme.

Le code binaire résultant de l'implémentation d'Idle Sense et du générateur aléatoire est sensiblement plus important que le code original (un peu plus de 64 octets

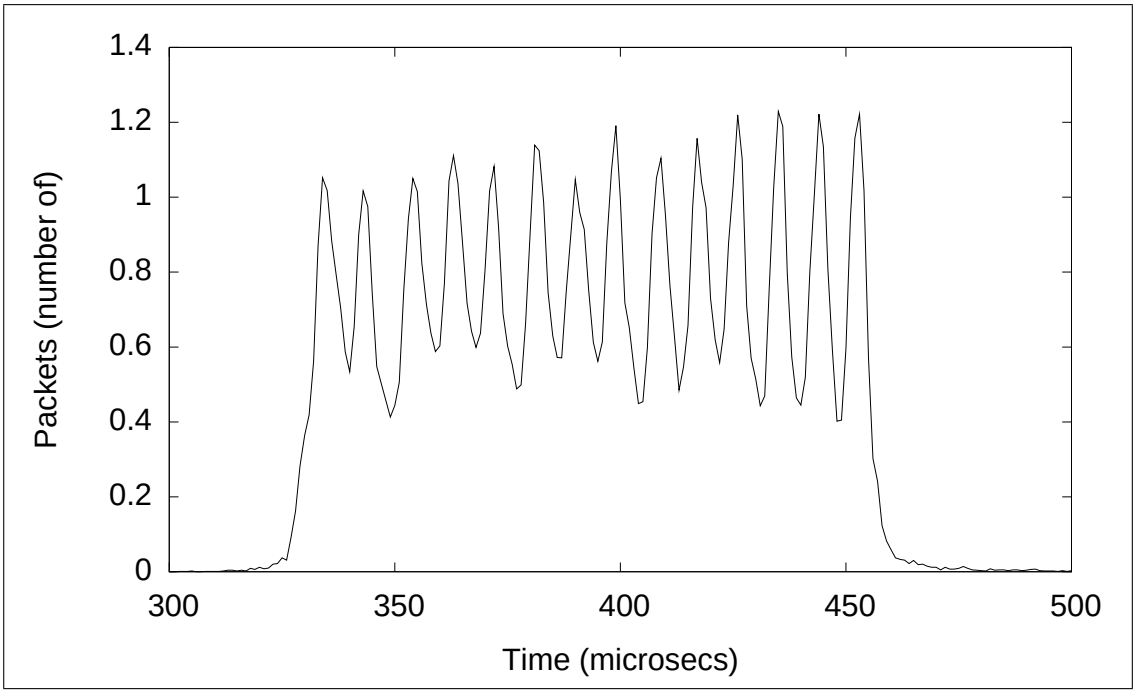


Figure 5.4 – Histogrammes d'interarrivées pour le firmware *Idle Sense* avec $CW = 13$.

supplémentaires sur un code de 16Ko).

5.3.2 Techniques de déboguage

Le déboguage est l'un des problèmes principaux lors de l'implémentation d'une nouvelle méthode d'accès sur du matériel standard. En effet, les limitations du processeur et de la mémoire nous ont empêché d'effectuer une exécution contrôlée pas à pas. Dans ces conditions, comment vérifier le comportement de notre algorithme de gestion de la fenêtre de contention ? Ce dont nous avions besoin, c'était une méthode pour observer la valeur actuelle de la fenêtre de contention CW. Comme nous avons utilisé du matériel standard, nous ne pouvions mesurer la valeur de CW durant l'exécution, parce qu'il n'y avait aucun port de déboguage JTAG ou de sortie verbeuse.

Pour résoudre ce problème, nous avons décidé d'utiliser les capacités de transmission de la carte à notre avantage, c'est à dire d'utiliser un des champs de l'en-tête 802.11 pour stocker la valeur de la fenêtre de contention et de capturer toutes les trames envoyées par la carte grâce à une station placée dans un mode de surveillance (mode moniteur). Nous avons utilisé le champ de contrôle de séquence usuellement dédié aux réordonnancement des paquets pour afficher la fenêtre de contention. La figure 5.5 présente la nouvelle utilisation de l'en-tête 802.11.

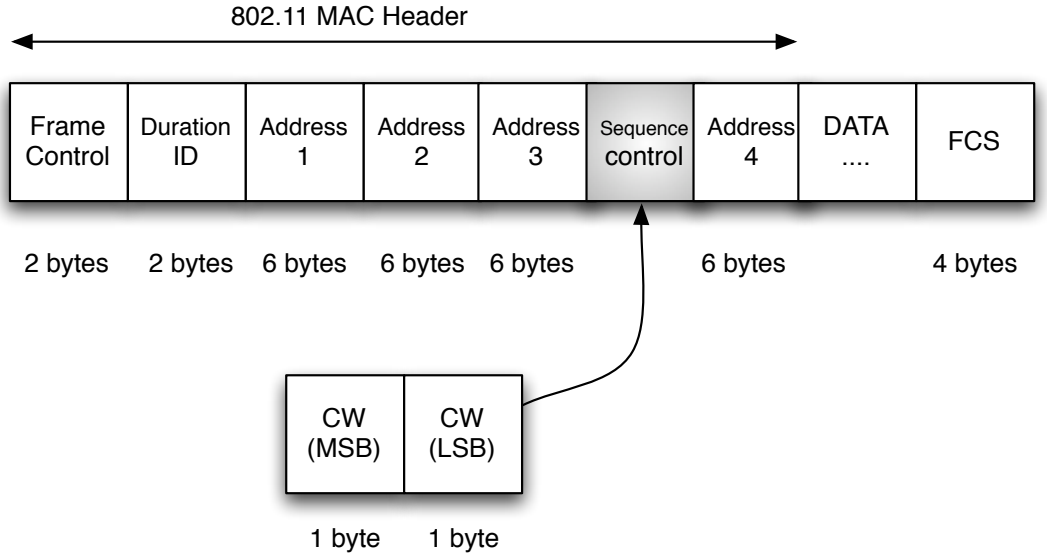


Figure 5.5 – Utilisation de l'en-tête 802.11 pour le déboguage.

5.4 Tests et validations

Dans cette section, nous avons évalué les performance du firmware Idle Sense implémenté sur les cartes Intel IPW2915. Nous comparons les performances obtenues avec celles du firmware standard DCF sur les questions de la vitesse de convergence, la bande passante obtenue, la latence, l'équité et le taux de collisions.

Pour illustrer les différents comportements, nous commencons par présenter l'évolution de la fenêtre de contention *CW* en fonction du nombre de stations actives dans les deux méthodes d'accès. Puis, nous analysons les mesures de débits : nous ne nous attendons pas à une grande amélioration, puisque les débits obtenus par DCF sont presque optimaux pour le nombre de stations utilisées dans nos mesures. Même si les résultats sont similaires, ils illustrent cependant deux politiques différentes. Nous observons donc une plus grande équité, des délais réduits, et moins de collisions avec *Idle Sense* comme les simulations l'ont prédit. Quand Idle Sense est en concurrence avec des stations en DCF, il tend à être moins aggressif, et donc sa bande passante est réduite.

5.4.1 Plate-forme

Nous avons mis en place une plate-forme d'évaluation à partir d'ordinateurs portables utilisant la plate-forme Centrino d'intel. Les cartes Intel IPW2915 opèrent dans une bande de fréquences relativement dégagée d'interférences dans la bande des 5 Ghz.

La plupart des noyaux Linux et FreeBSD peut utiliser sans aucune modification

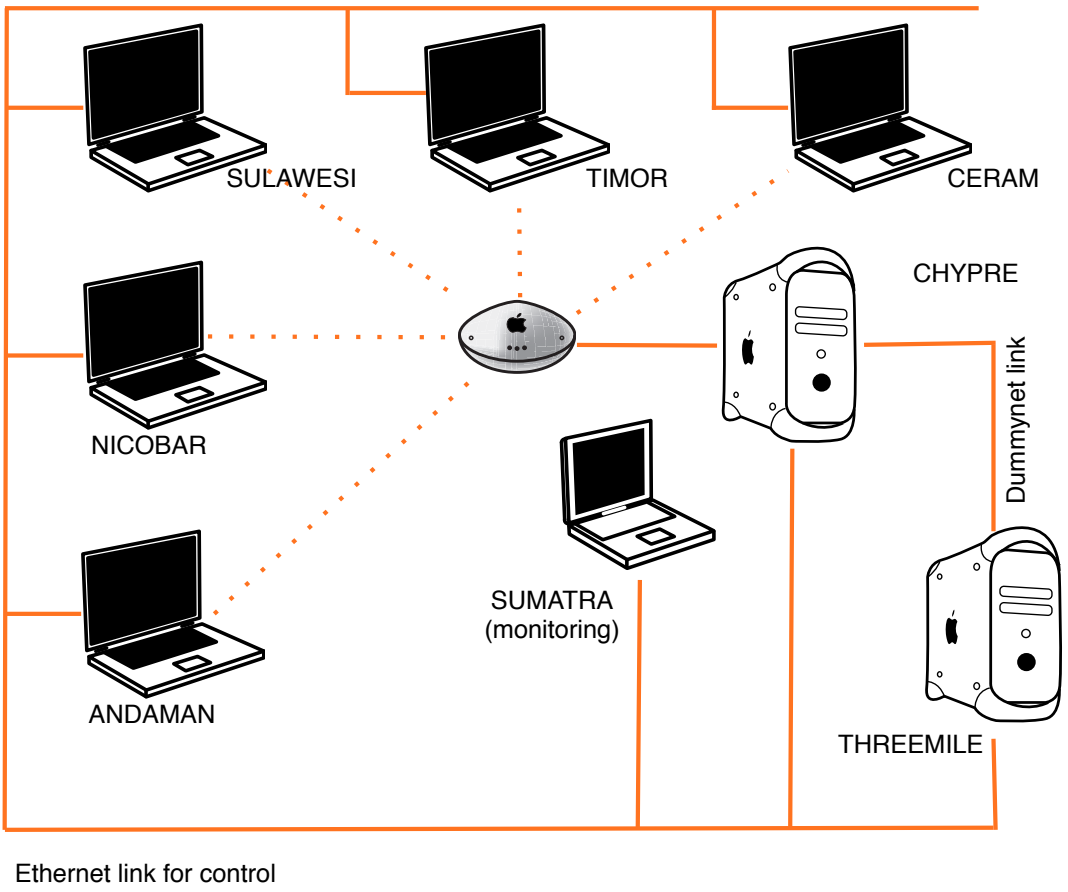


Figure 5.6 – Plate-forme expérimentale

notre microcode binaire à condition que la version de celui-ci soit identique à celle réclamée par le driver (nous utilisons la version 2.4). Nous avons utilisé FreeBSD comme système d'exploitation principal pour nos mesures. Un ordinateur supplémentaire doté d'une carte Atheros en mode moniteur capture les informations de déboguage situées dans les en-têtes. La figure 5.6 présente la plate-forme mise en place.

Nous avons inclus un lien Dummynet pour émuler la latence de longues liaisons pour simuler un traffic internet réaliste. Le point d'accès est utilisé seulement pour la réception des trames qui sont réexpédiées par une liaison filaire. De cette manière nous pouvons isoler la mesure de la contention entre les cartes disposant d'Idle Sense sans que d'autres perturbations puissent être introduites par le trafic bidirectionnel au niveau du point d'accès.

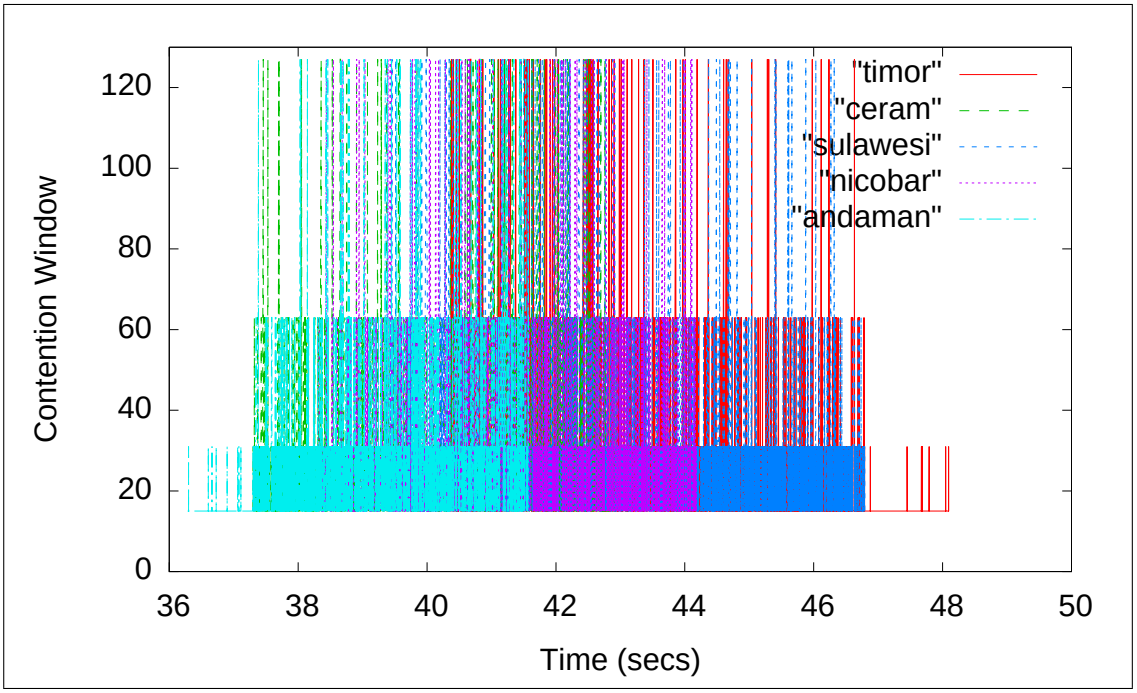


Figure 5.7 – Evolution de la fenêtre de contention pour 802.11 DCF.

5.4.2 Convergence de l'algorithme

Notre première expérience concerne l'évolution de la fenêtre de contention dans un scénario de mise en place dynamique des stations actives : chaque station s'active au fur et à mesure et commence à générer un traffic UDP.

La figure 5.7 montre l'évolution de la fenêtre de contention pour 802.11 DCF. Au début, CW est égal à la valeur de $CW_{\min} = 15$, donc la valeur aléatoire est tirée entre

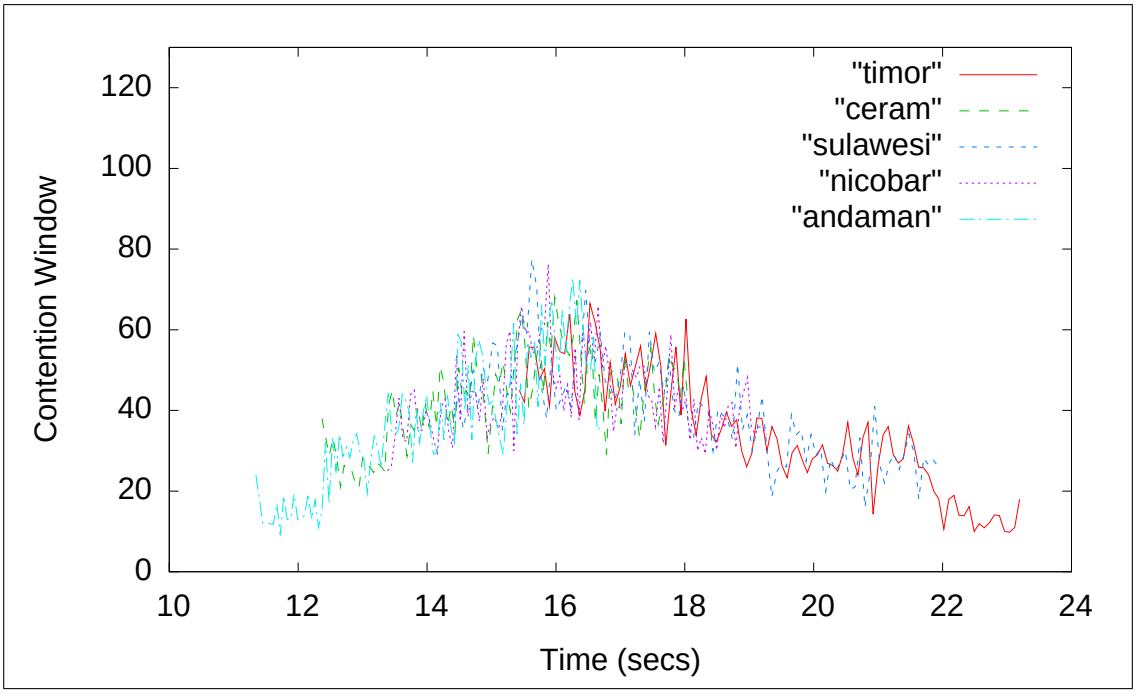


Figure 5.8 – Evolution de la fenêtre de contention pour *Idle Sense*.

0 et 15. Puis, 1s plus tard, une deuxième station entre en concurrence pour l'accès au canal. Nous pouvons voir un saut à chaque arrivée d'une nouvelle station entrant en activité. La valeur de *CW* sautre de 15 à 31, 63 puis 127. Après 5 stations, la valeur de *CW* reste aux alentours de 127.

La figure 5.8 présente le comportement d'Idle Sense dans le même genre de scénario. Nous pouvons identifier également plusieurs phases de convergence de l'algorithme après la mise en activité successive des différentes stations. A chaque nouvelle arrivée, toutes les stations convergent vers une nouvelle valeur de la fenêtre de contention, qui est égale pour les 5 stations. La valeur obtenue pour 5 stations en compétition est très inférieure à celle établie par 802.11 DCF.

Enfin, nous souhaitons examiner la vitesse de convergence d'Idle Sense de manière plus précise. Au démarrage, une station génère du traffic et à l'instant 2s, une seconde station commence à émettre du traffic.

La figure 5.9 montre que l'adaptation AIMD permet au firmware Idle Sense de s'adapter rapidement au changement des conditions du médium, comme prédit par les simulations.

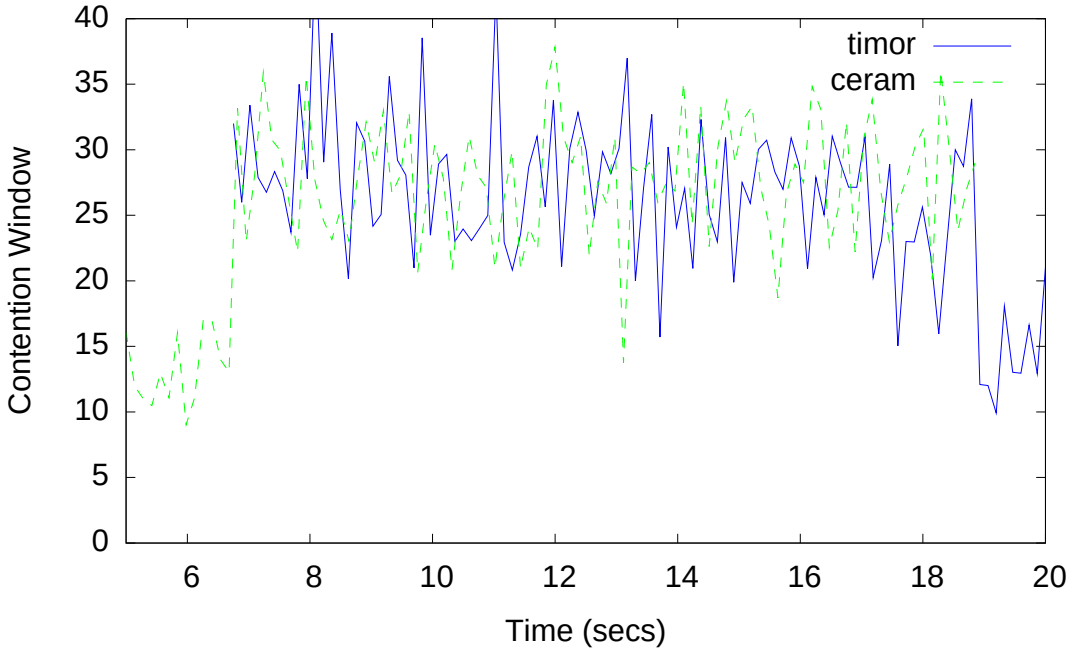


Figure 5.9 – Vitesse de convergence de Idle Sense

5.4.3 Bande passante

Nous avons mesuré les débits obtenus par les firmwares 802.11 DCF et Idle Sense lorsque 5 stations émettent du trafic UDP en concurrence vers un point d'accès à la modulation de 54 Mbits/s et sur la bande des 5Ghz. En utilisant la simulation, nous obtenons les mesures suivantes : 5.848 Mbits/s pour DCF et 5.985 Mbits/s pour Idle Sense.

Les figures 5.10 et 5.11 présentent les débits mesurés par les stations en concurrence pour les différentes méthodes d'accès. Nous pouvons voir que le débit moyen est similaire pour les deux méthodes d'accès, mais 802.11 DCF présente une variance supérieure dans les débits des différentes stations. Par exemple, la station TIMOR obtient un débit plus faible (5.46 Mbits/s) comparé à SULAWESI (5.9 Mbits/s), à cause de l'inéquité à court terme.

Pour le firmware Idle Sense, les stations obtiennent un débit avec une variance plus faible (5.9 Mbits/s à 6.1 Mbits/s) qui résulte d'une meilleure équité à court terme. Comme nous pouvons le voir, les résultats expérimentaux et théoriques par simulation sont équivalents (DCF : 5.46<5.848<5.9 et *Idle Sense* : 5.9 <5.985<6.1).

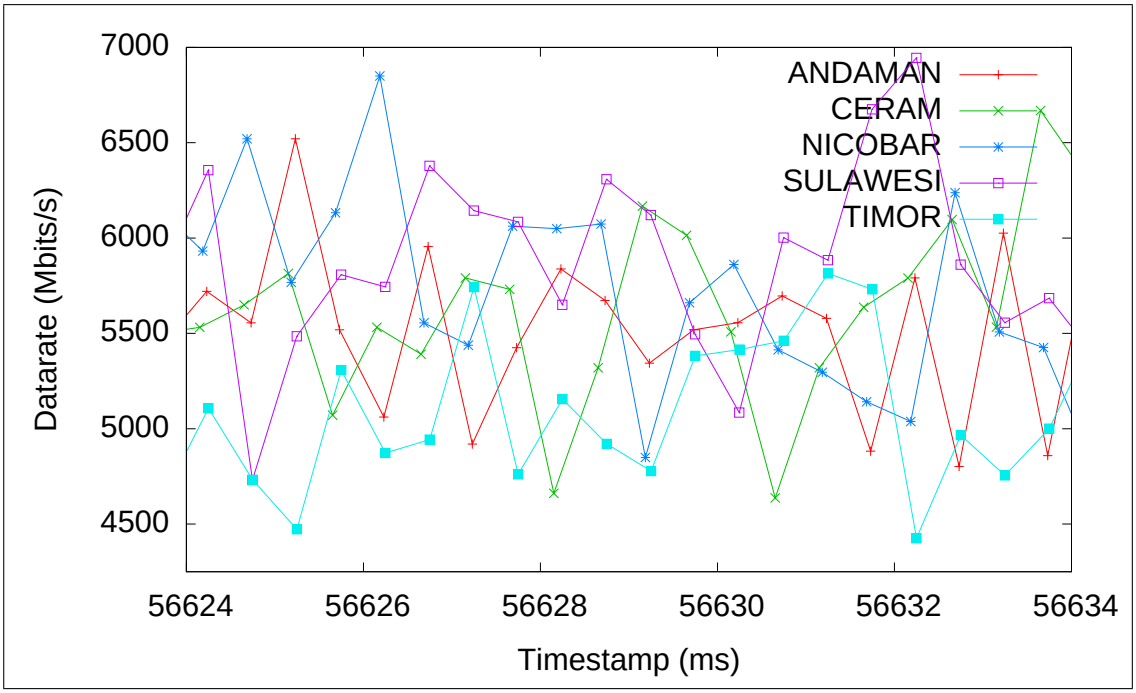


Figure 5.10 – Débits de 5 stations concurrentes avec 802.11 DCF.

5.4.4 Equité

Pour évaluer les améliorations attendues par l'utilisation d'Idle Sense au niveau de l'équité à court terme, nous avons réalisé l'expérience suivante : nous avons mis en concurrence 5 stations et nous avons surveillé les délais d'arrivée entre chaque paquet. Ensuite, en utilisant une méthode par fenêtre glissante, nous avons calculé l'index de Jain moyen pour une fenêtre de taille croissante. Nous normalisons la taille de la fenêtre pour qu'elle soit un multiple du nombre des stations en concurrence. Un index de 1 représente une équité parfaite, ce qui est le cas pour une méthode d'accès de type TDMA. La figure 5.12 compare l'index de Jain calculé pour des mesures sur les deux méthodes d'accès. Nous pouvons voir que la valeur seuil de 0.95 est obtenu pour une fenêtre bien plus réduite avec le firmware Idle Sense, ce qui contribue à une meilleure équité à court terme.

5.4.5 Statistiques de transmission

Nous avons essayé d'évaluer le comportement d'Idle Sense vis à vis des retransmissions. Dans ce but, nous avons étudié les en-têtes 802.11 des traces obtenues par une carte moniteur en présence de 5 stations émettrices. En utilisant le numéro de séquence 802.11 ainsi qu'un numéro de séquence dans l'en-tête UDP, nous avons pu

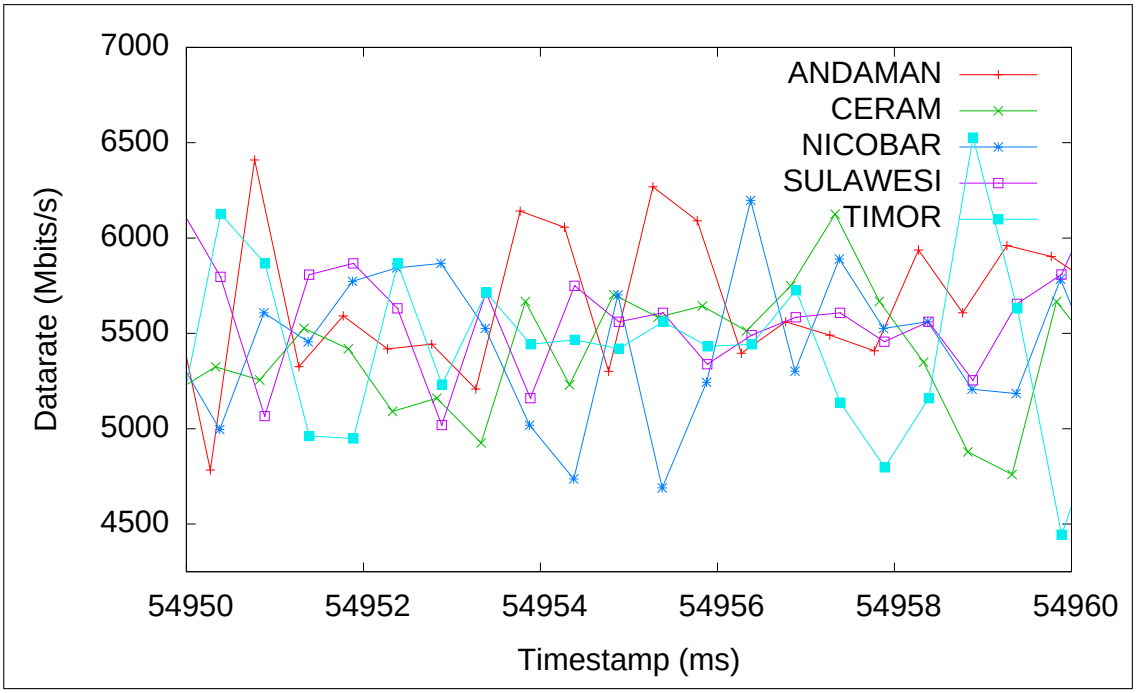


Figure 5.11 – Débits de 5 stations concurrentes avec *Idle Sense*.

isolé les trames correctement transmises à la première tentative et les trames retransmises (doté d'un flag indiquant une retransmission dans l'en-tête 802.11). Le firmware Idle Sense tend à faire prendre à l'ensemble des stations un *CW* optimal de telle sorte qu'il y ait moins de collisions et moins de temps perdu en retransmissions. Toutes les stations attendent pour le canal un peu plus longtemps (étant donné que la fenêtre de contention *CW* est fixée par les stations de manière à ce que le nombre moyen de slots vides soit le même). Dans 802.11 DCF, les stations attendent moins, mais après une collision les stations en compétition doivent attendre plus longtemps à cause du mécanisme d'incrémentation de la taille de la fenêtre de contention. La table 5.13 présente les résultats pour 100000 trames capturées, alors que la table 5.14 montre les statistiques pour des paquets capturés durant 10 secondes.

Les résultats confirment les propriétés déduites du fonctionnement d'Idle Sense et de DCF. DCF présente plus de trames retransmises qu'Idle Sense. Nous avons également observé qu'Idle Sense transmettait plus de trames à la première tentative que DCF. Ces statistiques expliquent également les résultats obtenus sur les débits : avec Idle Sense, une station attend plus longtemps pour envoyer sa trame, mais la probabilité d'une transmission réussie est plus importante à la première tentative. Ce compromis permet aux stations équipées du firmware Idle Sense de gagner en équité à court terme, comme nous l'avons vu précédemment.

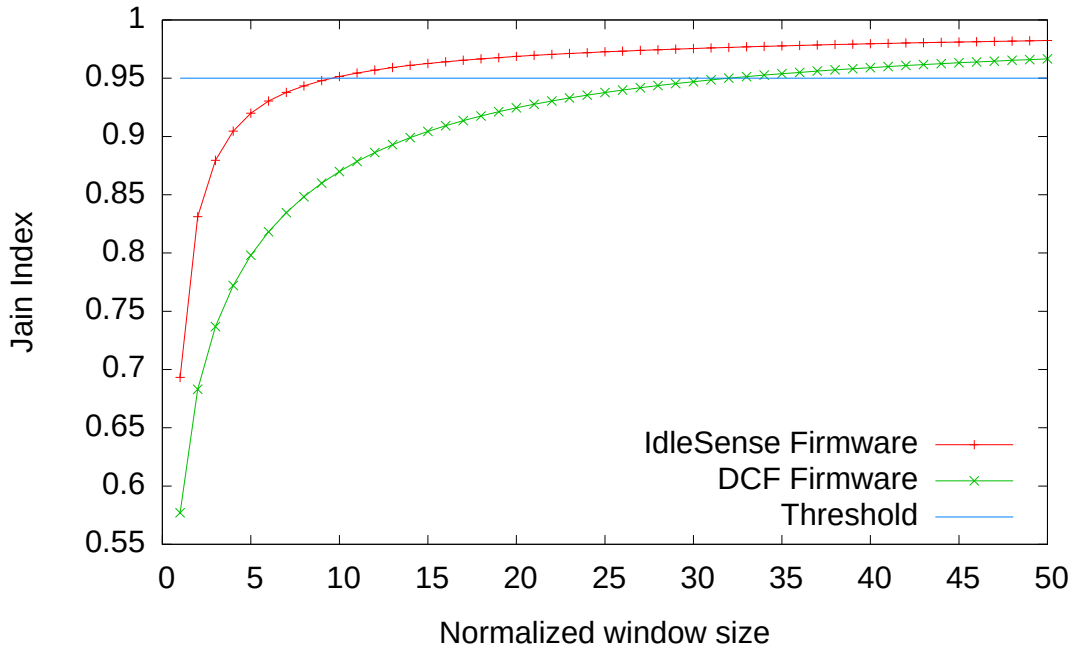


Figure 5.12 – Index de Jain pour 802.11 DCF et le firmware *Idle Sense* pour 5 stations en compétition.

Number of	802.11 DCF	*Idle Sense*
data frames	34086	40704
retransmissions	14253	7860

Figure 5.13 – Statistiques de transmission pour 100000 paquets.

Number of	802.11 DCF	*Idle Sense*
data frames	17533	19505
retransmissions	7208	3707

Figure 5.14 – Statistiques de transmission pour une session de 10 secondes.

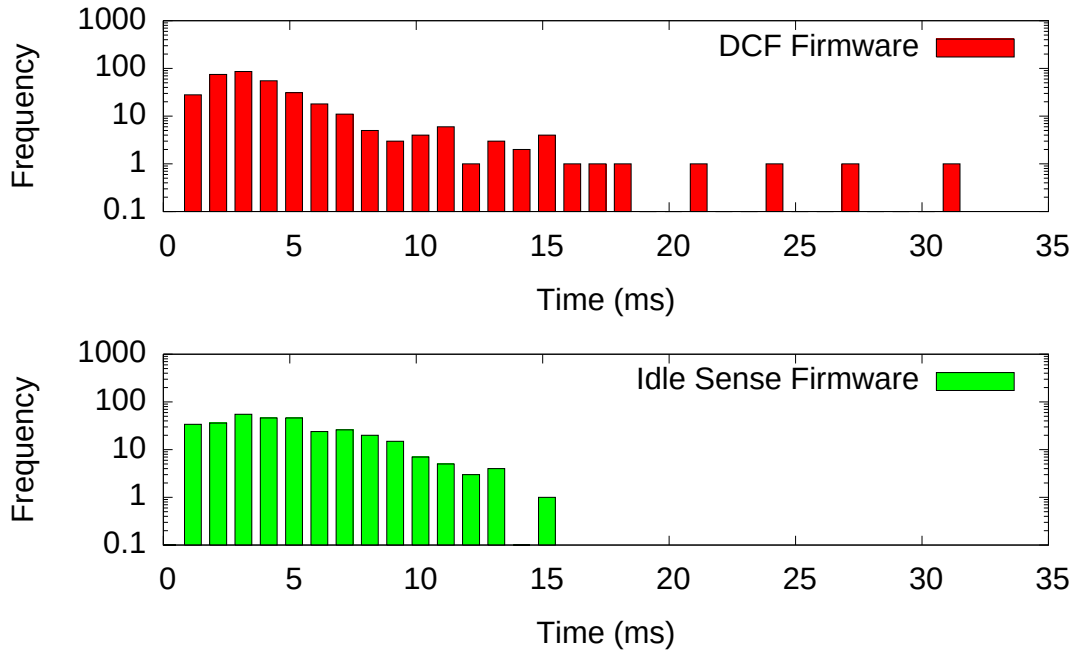


Figure 5.15 – Histogramme de la latence des paquets ICMP pour 4 transmissions UDP concurrentes.

5.4.6 Latence

En réduisant la probabilité de collision, Idle Sense réduit également la grande variabilité entre les intervalles inter-paquets, qui est une cause de latence incrémentée dans 802.11 DCF. Avec Idle Sense, nous espérons une latence plus déterministe. Nous utilisons 4 stations en concurrence sur des trafics UDP et une cinquième station qui envoie des paquets ICMP de 1472 octets toutes les 0.2 sec pendant 1 minute. La figure 5.15 présente l'histogramme de la latence des paquets ICMP. Comme nous pouvons le voir, il n'y a pas de paquet ICMP avec une latence supérieure à 15 ms avec Idle Sense. L'histogramme est beaucoup plus équilibré entre 0 et 10 ms avec Idle Sense qu'avec 802.11 DCF.

5.4.7 Interopérabilité avec DCF

Il n'y a pas d'incompatibilité fondamentale entre DCF et Idle Sense. Cependant, Idle Sense est basé sur l'hypothèse que chaque station cherche à atteindre la même valeur de la fenêtre de contention CW, et donc cela implique que l'ensemble des stations utilise la même méthode d'accès. Quand un réseau mixte est composé par des stations DCF et des stations Idle Sense, ces dernières tendent à être moins aggressives en regard de l'accès au canal. Pour évaluer cet effet, nous avons mesuré les débits de deux stations en compétition pour l'accès au canal, dotées soit de 802.11 DCF ou Idle

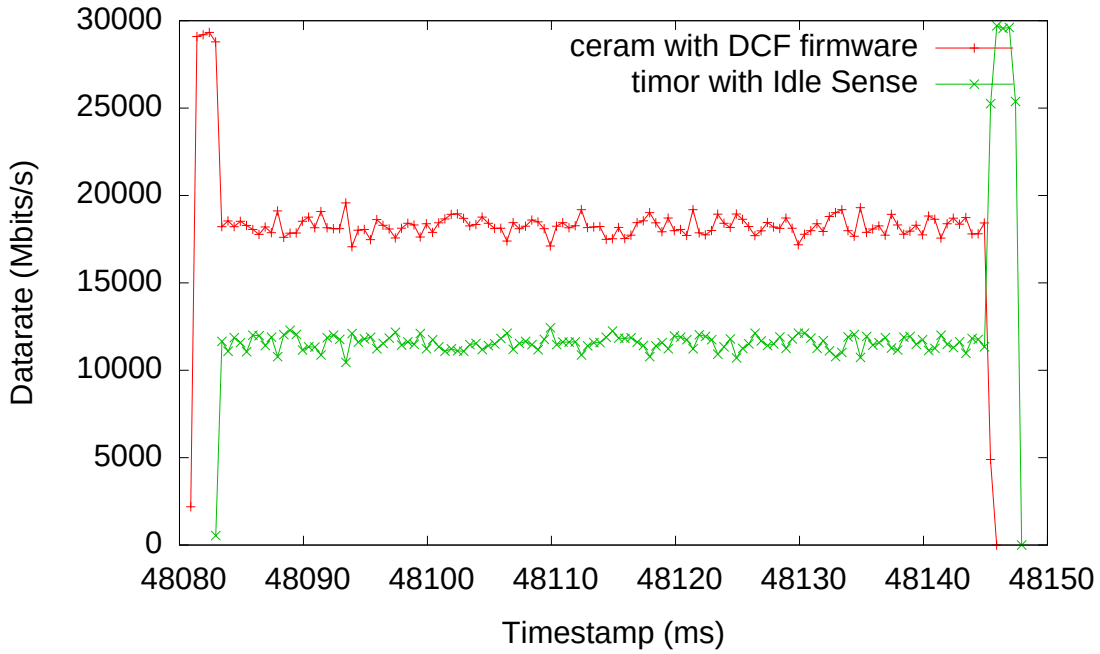


Figure 5.16 – Débits de stations en concurrence équipées de DCF et d'*Idle Sense*

Sense (cf. Figure 5.16).

Nous voyons sur ces résultats qu'une station équipée d'IdleSense tend à avoir un débit plus faible qu'une station utilisant DCF, ce qui correspond au comportement théorique attendu.

5.4.8 Conclusion sur l'implémentation d'IdleSense

Les résultats d'expérimentations obtenus sont très satisfaisants : nous retrouvons dans nos résultats pratiques l'essentiel des propriétés prédites par les simulations effectuées par les auteurs d'IdleSense, malgré les concessions effectuées concernant les paramètres numériques de l'algorithme original.

Cependant, les expérimentations ont mis en valeur les limites de cette méthode d'accès : d'une part, elle exige que l'ensemble des stations voient le même état radio, c'est à dire que le même nombre de slots vides doit être perçu par l'ensemble des stations d'une cellule pour que les propriétés d'équité soient retrouvées. IdleSense est donc dans cette première implémentation plus adapté à un fonctionnement en point d'accès, mais la souplesse de la reprogrammation de la couche d'accès offre la possibilité de prendre en considération des scénarios plus complexe, comme une hétérogénéité de l'état radio observé par les stations. D'autres part, la cohabitation avec les stations utilisant la méthode d'accès DCF tend à réduire les performances d'IdleSense. Cependant, dans un contexte de rétrocompabilité, nous pouvons aisément

détecter cette situation (en mesurant la convergence de l'algorithme par exemple pour voir si le nombre de slots vides évolue dans le bon sens) et basculer en mode DCF dynamiquement.

5.5 Réflexions sur l'implémentation des méthodes d'accès

Au travers de l'implémentation précédente, nous avons pu tirer des conclusions intéressantes concernant les capacités du matériel actuel à permettre la concrétisation des réseaux sans fil de nouvelles générations. Si bien entendu ces réseaux vont évoluer vers des nouvelles capacités et méthodes de transmission physique, l'accès au canal restera un point crucial dans le bon fonctionnement des réseaux.

5.5.1 Architecture suivant les contraintes temporelles

L'amélioration des méthodes d'accès est une part importante du travail fourni par la recherche sur les réseaux sans fil : en effet, les problèmes liés à la sous-utilisation du canal, ainsi que les problèmes d'accès multiples, de terminaux cachés en utilisation mesh sont des problèmes qui ne peuvent être réellement traités que si la méthode d'accès est accessible et reprogrammable.

Au cours de nos investigations concernant l'implémentation d'IdleSense, si certaines plates-formes comme le chipset Atheros et son pilote MADWiFi se sont révélées inadaptées à l'implémentation d'une nouvelle méthode d'accès, la plate-forme d'intel s'est révélée très intéressante. D'une part, elle sépare en éléments distincts la méthode d'accès, la gestion du protocole 802.11 et enfin la communication avec le système hôte. Ainsi, la méthode d'accès est programmée dans le microcode, en assembleur, au plus près du fonctionnement temporel du protocole, puisque la résolution temporelle gérée est le slot. Ensuite, le firmware de la carte s'occupe des aspects 802.11 programmés dans un langage plus haut niveau en C, compilé pour l'architecture de la carte Intel. Enfin, ce dernier communique avec le pilote du système d'exploitation par un bus de commande.

Cet aspect du fonctionnement de la carte est révélé dans le code source du pilote sous licence GPL de la carte :

```
1  static int __ipw_send_cmd(struct ipw_priv *priv, struct host_cmd *cmd)
2  {
3  /* .. */
4      rc = ipw_queue_tx_hcmd(priv, cmd->cmd, cmd->param, cmd->len, 0);
5  /* .. */
6  }
7
8  static int ipw_queue_tx_hcmd(struct ipw_priv *priv, int hcmd, void *buf,
9                               int len, int sync)
10 {
```

```
11 /* */
12
13     ipw_write32(priv, q->reg_w, q->first_empty);
14
15 /* */
16 }
```

Le fonctionnement décrit est en fait très simple : ici le pilote établit des commandes (par la fonction send cmd), qu'il envoie ensuite par l'intermédiaire de registres (write32) accessibles de la carte Intel. Pour la réception de messages issus de la carte, le pilote exploite les interruptions, comme l'indique cet autre aspect du code :

```
1  static void ipw_irq_tasklet(struct ipw_priv *priv)
2  {
3  /* .. */
4      inta = ipw_read32(priv, IPW_INTA_RW);
5
6      if (inta & IPW_INTA_BIT_RX_TRANSFER) {
7          ipw_rx(priv);
8  /* .. */
9      if (inta & IPW_INTA_BIT_TX_CMD_QUEUE) {
10 /* .. */
11     if (inta & IPW_INTA_BIT_TX_QUEUE_1) {
12 /* .. */
13     if (inta & IPW_INTA_BIT_TX_QUEUE_2) {
14 /* .. */
15     if (inta & IPW_INTA_BIT_TX_QUEUE_3) {
16 /* .. */
17     if (inta & IPW_INTA_BIT_TX_QUEUE_4) {
18 /* .. */
19     if (inta & IPW_INTA_BIT_STATUS_CHANGE) {
20 /* .. */
21
22     if (inta & IPW_INTA_BIT_BEACON_PERIOD_EXPIRED) {
23 /* .. */
24
25 }
```

Ici, en fonction de l'interruption levée, le traitement est distinct, et les évènements spécifiques de la carte (CMD QUEUE) sont des interruptions similaires aux paquets effectivement reçus (RX TRANSFER).

Notre proposition de séparer traitement temps réel et traitement à l'échelle des paquets est donc parfaitement justifiable en cas d'utilisation d'architectures matérielles similaires à celle de la carte Intel : en effet, non seulement la partie temps réel peut être programmée de manière indépendante du système hôte, mais elle peut communiquer avec lui de manière similaire aux interruptions générées par la réception des paquets.

La réussite de l'implémentation d'IdleSense sur la carte Intel montre qu'une telle approche est viable pour résoudre les problèmes architecturaux des nouveaux réseaux

sans fil. Cependant, cette implémentation a soulevé des problèmes intéressants, que nous résumons dans la section suivante.

5.5.2 Problèmes liés aux architectures programmables

Comme nous l'avons vu, l'implémentation d'IdleSense a révélé des problèmes intéressants.

Tout d'abord, l'utilisation de simulateurs, si elle a été très utile pour la mise au point de l'algorithme par les auteurs d'IdleSense, a conduit à des impératifs techniques sur l'implémentation, comme la nécessité initiale d'utiliser des nombres flottants, fonctionnalité souvent absente des systèmes embarqués.

Cela relance le problème de la crédibilité des simulateurs. Ici, les simulateurs utilisés étaient à évènements discrets ; cependant, ils n'intégraient aucune contrainte réelle liée aux matériels embarqués. Dans le cadre du développement de nouvelles méthodes d'accès, il est donc primordial à l'avenir de rapprocher les simulateurs des systèmes réels. Plusieurs projets prennent cette direction : NS-3, la nouvelle révision de Network Simulator, va s'interfacer avec le noyau Linux pour utiliser des interfaces réelles sur des systèmes distribués [103], permettant ainsi l'usage de simulation sur des réseaux quasi-réels. Au niveau des méthodes d'accès, les recherches visent à intégrer des émulateurs dans les simulateurs de réseau [104]. Ainsi, l'utilisation de FPGA émulant les interfaces physiques permettant d'approcher le fonctionnement réel des réseaux sans fil est désormais utilisé dans les travaux expérimentaux sur les méthodes d'accès [100] [105] , se rapprochant ainsi des expérimentations menées dans le cadre des radio-logicielles, comme suggéré dès 2004 par [102].

Un autre problème suggéré par l'implémentation a mis en exergue la difficulté à obtenir des informations de débuggage lors du développement. Si sur une plate-forme expérimentale, l'accès aux informations est facile, dans une plate-forme réelle utilisée en production, ces informations sont inexistantes, et il faut procéder par déduction liée aux observations du fonctionnement réel. Ici, l'utilisation des paquets 802.11 comme transport de l'information de debug a permis d'éliminer ce problème ; cependant cela consiste un obstacle non négligeable, mais pas insurmontable : la présence d'une communication avec l'hôte suggère que l'on pourrait aisément ajouter ces informations, à condition de disposer de l'ensemble du code source de la plate-forme.

5.6 Conclusions

En implémentant la méthode d'accès IdleSense, nous avons pu évaluer les possibilités des plates-formes radio actuellement disponibles sur le marché. Si l'architecture de celles-ci est plutôt dictée par des considérations économiques, ce qui a conduit de nombreuses cartes à être inadaptes au changement de leur méthode d'accès, il n'en est pas de même pour toutes : certaines cartes offrent même une architecture distinguant

aspect temps réel et gestion de paquets, et qui tendent à être de parfaites plates-formes pour les futures réseaux sans fil.

Au cours de cette implémentation, nous nous sommes rendus compte de l'importance du travail en amont nécessaire pour la mise au point de nouvelles méthodes d'accès adaptées aux réseaux sans fil de nouvelles générations. De l'idée initiale sur l'adaptation de la fenêtre de contention, il aura fallu plusieurs simulateurs et améliorations conceptuelles pour atteindre un modèle satisfaisant en terme de performance ; puis, au travers du prisme de l'implémentation, ce modèle a été légèrement adapté pour tenir compte des contraintes matérielles d'une plate-forme réelle. Cependant, les résultats expérimentaux obtenus sont très satisfaisants, et rejoignent pour la plupart les résultats théoriques obtenus par simulateur.

Ce travail d'implémentation permet donc de justifier par l'exemple la pertinence des approches actuelles de la recherche, proposant tout d'abord un modèle, puis une simulation, puis enfin une expérimentation. Cependant, ce dernier effort doit être dans le futur rendu systématique, sous peine de concevoir des réseaux sans fil de nouvelles générations figés et peu adaptés à l'amélioration et à l'optimisation.

Suite à ce travail, nous ne pouvons que souscrire au développement d'architecture réseaux au moins similaires à celle utilisée dans cette implémentation, c'est à dire basée sur une approche distincte entre temps réel et paquets. L'accès au code source dans son intégralité permettra en outre une communication parfaite entre le système embarqué responsable des aspects temps réel et du système hôte, responsable du traitement des paquets, auquel nous allons désormais nous intéresser.

CHAPITRE 6

VERS UNE ARCHITECTURE DE GESTION DE PAQUETS

Au travers de notre travail précédent, nous avons pu évaluer la pertinence des plates-formes actuelles pour l'accueil des nouveaux développements sur les réseaux sans fil. Cependant, dans ce travail nous nous sommes uniquement attachés à traiter le problème des méthodes d'accès et la faisabilité de la mise en place de celles-ci. Cependant, comme notre autopsie des réseaux 802.11 l'a montrée, la résolution des défis posés par les réseaux de nouvelles génération passent par des architectures mettant en oeuvre des techniques inter-couches, notamment les couches supérieures aux couches d'accès.

Afin d'éviter une dispersion et une spécialisation des architectures données, nous avons étudié des solutions génériques dans notre état de l'art, et notamment des frameworks développés par la communauté scientifique, comme les frameworks Click ou Xian, qui utilisent du matériel générique. Cependant, ces frameworks sont relativement spécialisés : Xian propose de rendre accessible un ensemble de variables pour établir de nouvelles métriques, quand à Click, il propose surtout une infrastructure dédiée à la gestion du routage. Ces solutions manquent de généricité et d'intégration, qui sont pourtant vitaux pour la création d'architectures réellement inter-couches.

Afin de proposer une solution réalisable et utilisable pour répondre à des problèmes concrets, nous visons à utiliser les résultats issus de la recherche en sécurité : les frameworks qui y sont développés visent à faciliter le traitement et la génération de trafic pour évaluer la sécurité des systèmes existants. Ces architectures présentent deux caractéristiques intéressantes pour nous : d'une part, elles fonctionnement à l'échelle temporelle du paquet, ce qui est le produit obtenu après traitement de la couche d'accès dans notre infrastructure précédente. D'autre part, l'ajout et la manipulation des ces paquets y sont grandement facilités, ce qui est constitue le coeur des architecture inter-couches : l'ajout de protocole et l'altération du fonctionnement des protocoles existants.

Dès lors, nous nous proposons de développer dans ce chapitre une architecture de

traitement de paquets fonctionnant sur du matériel usuel pour finalement essayer de revenir sur les bases historiques du réseau : la commutation de paquets. Notre méthodologie sera la suivante : dans un premier temps, nous évaluons plus précisément l'architecture et les performances des frameworks proposés. Puis à partir de là, nous formulerons notre architecture.

6.1 Analyse technique des propositions existantes

Nous analysons d'un point de vue plus technique les propositions existantes, que ce soit dans le domaine des architectures inter-couches, ou dans le domaine des architectures pour l'analyse de la sécurité.

6.1.1 Architectures inter-couches

Précédemment, nous avons énuméré plusieurs architectures inter-couches : Click, XIAN et Prawn. A notre connaissance, ce sont les principales architectures agissant sur le réseau sans fil dont les propriétés de manipulation de paquet rejoignent nos considérations architecturales : le traitement à l'échelle temporelle du paquet.

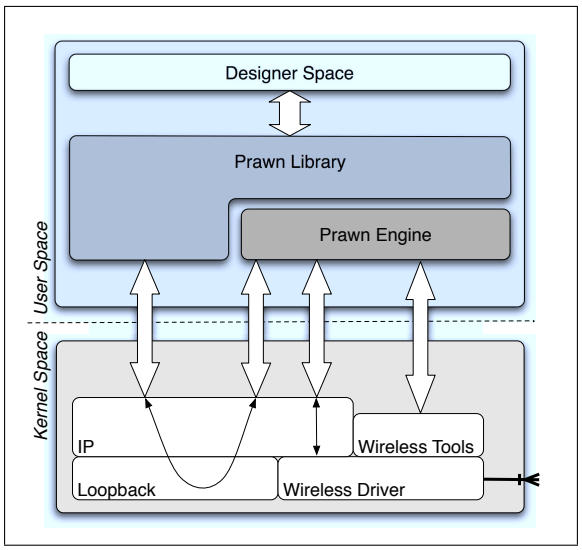


Figure 6.1 – Architecture de Prawn. (Extrait de la publication de Abdesslem et al.)

Prawn [90] est conçu sur une architecture utilisant les modèles en couches du système d'exploitation. Dès lors, le moteur Prawn et la bibliothèque utilise un wrapper permettant d'accéder aux fonctions du noyau, notamment la manipulation de la couche IP, ainsi que les paramètres des cartes sans fil utilisés, comme l'indique la

figure 6.1 . Cet interfaçage est à discuter, car cela signifie que les modifications appliquées aux paquets doivent rester dans les possibilités implémentées dans le noyau du système hôte. De même, l'accès aux statistiques sans fil fournies par les API du noyau du système est considéré comme peu fiable, compte tenu de la manière dont les drivers enrichissent les statistiques du système d'exploitation.

L'examen de plusieurs codes sources [1] de pilotes de cartes sans fil ainsi que l'étude du manuel de l'API wireless-tools[2] révèlent que les valeurs renvoyées pour certaines statistiques sont dépendantes des choix d'implémentation. En particulier, concernant la qualité du signal, l'aide en ligne nous indique que "celle-ci *peut* être fonction du niveau de contention, d'interférence, du taux d'erreur par bit, ou de toute autre métrique matérielle" (texte original : *"Overall quality of the link. May be based on the level of contention or interference, the bit or frame error rate, how good the received signal is, some timing synchronisation, or other hardware metric. This is an aggregate value, and depends totally on the driver and hardware."*). Par exemple, nous avons noté que les cartes Atheros utilisaient un niveau de bruit constant par défaut et une valeur de signal mise à jour à chaque paquet reçu, contrairement aux cartes Intel qui utilisent une valeur dynamique mise à jour périodiquement par un message spécifique. Dans ces conditions, l'utilisation d'une API générique qui ne tient pas compte des spécificités matérielles ne peut qu'induire des anomalies au niveau des résultats obtenus dans l'architecture inter-couche, puisque les statistiques utilisées varient d'une carte à l'autre.

Intéressons-nous maintenant à l'architecture de XIAN, présentée sur la figure 6.2. L'architecture reprend certains éléments communs avec Prawn : l'interfaçage se fait au niveau du noyau, par un module dédié qui communique ensuite avec une bibliothèque en espace utilisateur. Cependant, XIAN étant dédié à l'utilisation des statistiques de la carte sans fil, l'interfaçage avec celle-ci est spécifique : XIAN épouse les caractéristiques de l'adaptateur utilisé, ici une carte Atheros. Dès lors, cette architecture est nettement plus pertinente. XIAN intègre de plus un mécanisme d'envoi/réception de paquet, appelé Xian Nano-handler, qui permet l'échange de messages dédiés aux métriques entre systèmes dotés de Xian ou à l'intérieur même d'un système. XIAN cependant ne permet pas d'altérer le fonctionnement original de 802.11 : il s'agit juste d'un complément. Dès lors, si les fonctionnalités à modifier dans un système dépendent de 802.11, XIAN ne permet pas d'atteindre ce niveau de modification.

Le framework Click de John Bicket et Robert Morris [69] [72] est utilisé principalement pour l'animation du réseau Roofnet. Son développement ayant démarré en 2000, il est bien plus riche en terme d'architecture que les 2 frameworks présentés précédemment. Click hérite d'une architecture d'abord conçue pour le traitement de paquets Ethernet, puis étendue après aux paquets 802.11, par l'intermédiaire d'un driver pour carte Atheros, madWiFi-stripped. Comme son nom l'indique, ce driver élimine les fonctions 802.11 du constructeur pour transformer la carte en un simple

[1] http ://http ://linuxwireless.org/en/users/Download
[2] http ://www.hpl.hp.com/personal/Jean_Tourrilhes/Linux/Tools.html

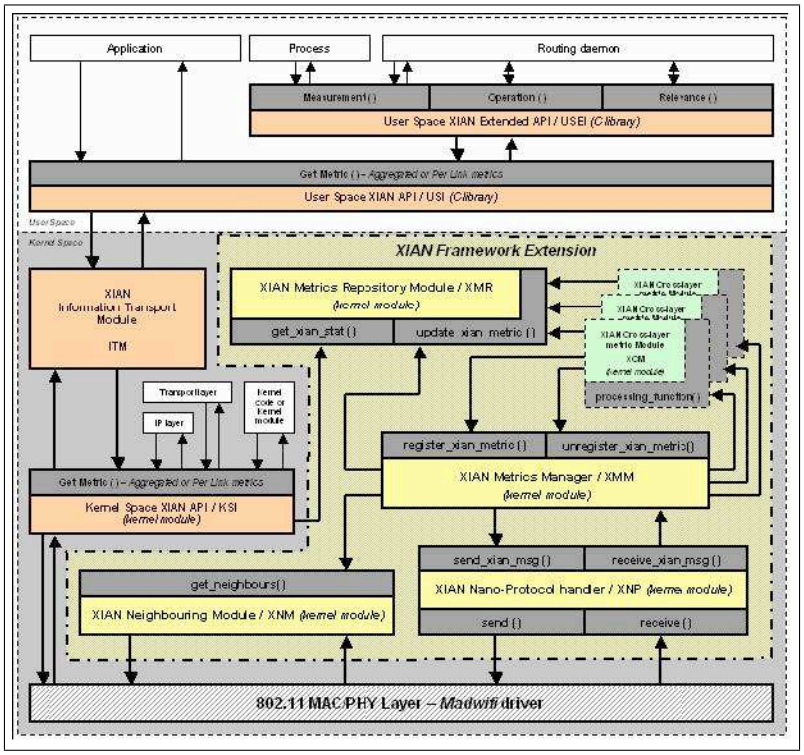


Figure 6.2 – Architecture de XIAN. (Extrait de la publication de Aiache et al.)

émetteur récepteur, sur lequel les éléments traditionnels de Click peuvent fonctionner. L'architecture originale de Click est décrite plus précisément dans [108]. Click utilise un interfaçage profond avec le noyau Linux. Le principe est de transférer tout paquet reçu au module Click, qui va exécuter la chaîne des éléments prédéfinie, éléments qui peuvent être des files d'attente, des modules de manipulation de protocole, ou même des protocoles (par exemple DHCP, ARP...). Les auteurs de la publication se sont efforcés d'étudier les performances de routage de Click en détaillant finement les mécanismes utilisés pour le traitement des paquets.

Dès la réception d'un paquet par la carte réseau, celle-ci lève une interruption pour notifier le noyau qu'un paquet doit être traité. Ensuite, ce paquet est recopié dans un buffer noyau (sous Linux appelé Socket Kernel Buffer), puis Linux crée un nouveau buffer pour un éventuel nouveau paquet, puis active les fonctions impliquées dans le traitement des paquets, tels que Click. Click court-circuite ensuite le fonctionnement de pile réseau de Linux, pour transmettre le paquet dans la fonction d'entrée du trafic dans Click. A partir de là, les éléments de Click s'exécutent suivant le scénario décrit dans le langage haut-niveau associé à Click.

A partir d'un certain moment, le périphérique peut envoyer des paquets, et notifie le noyau, qui récupère la file d'attente d'envoi de Click. La programmation du routeur

Click s'effectue dans un fichier de configuration puis est transféré au noyau grâce au mécanisme de communication de l'arborescence /proc. Ce système de fichier est particulier car il est partagé entre espace noyau et espace utilisateur. Dès lors, les programmes en espace utilisateur peuvent interagir sur Click en écrivant dans un simple fichier qui impactera le fonctionnement de Click.

Les auteurs ont évalué les performances de l'architecture Click de manière très précise, en évaluant des paramètres comme le nombre de paquets traités par interruption, le coût temporel du changement de contexte du processeur, celui de la répartition du traitement de l'interruption, puis le traitement de l'interruption en lui-même. Selon leurs résultats, le noyau transfère 1.5 paquet par interruption, puis le processeur effectue le changement de contexte en 1 μs, puis Linux répartit les traitants de l'interruption en 6.7 μs, puis le traitant dédié au trafic réseau s'exécute en 8.3 μs. En prenant un cas générique de traitement de paquets IP, Click réalise celui-ci en 2.4 μs. Afin d'évaluer le coût protocolaire de Click, les auteurs ont réduit le code à un simple traitement IP sans passer par les éléments Click. Cette fois, le traitement s'exécute en 1.9 μs. Même si ces résultats sont à réévaluer avec les architectures des processus actuels, plus rapides, les conclusions des auteurs restent toutefois valides : le traitement par élément de Click induit des défauts de cache mémoire, ce qui oblige le processeur à aller chercher les instructions des éléments de Click, ce qui impose un coût plus important. Enfin, les auteurs suggèrent qu'un mécanisme de polling [109] peut réduire les coûts liés aux changements de contexte et à la répartition des traitants d'interruption.

Click est donc très enrichissant pour notre architecture nouvelle génération. Certes, son architecture a été conçue pour le traitement de paquets Ethernet, et ne tient pas assez compte des caractérisques particulières des cartes sans fil actuelles, notamment du surcoût temporel lié au traitement du protocole 802.11. Click ne permet pas une réelle manipulation du protocole 802.11, qui dans les dernières extensions de Click est utilisé uniquement pour le conditionnement des trames Ethernet. Il manque intrinsèquement une capacité de manipuler le protocole 802.11 afin de l'adapter aux nouveaux usages des réseaux sans fil de nouvelle génération, comme pour le mesh par exemple. Cependant, l'architecture de Click permet de fixer les premières conditions d'une architecture performante : privilégier un traitement en espace noyau, piloté depuis l'espace utilisateur ; enfin, l'architecture doit tenir compte du mécanisme de réception et d'émission bas niveau des trames dans le noyau pour éviter toute perte de temps lié au traitement du paquet dans le noyau.

Afin d'apporter la brique de manipulation du protocole 802.11 qui semble manquer à l'ensemble de ces prototypes, nous nous tournons maintenant vers les architectures issus du monde de la sécurité.

6.1.2 Architectures issues de recherche en sécurité

La recherche en sécurité, nous l'avons vu, a mis en oeuvre une quantité très importante d'outils dédiés à la manipulation du protocole 802.11. Ces outils fonctionnent en espace utilisateur, l'accent étant mis non pas sur la performance mais sur la souplesse de ces outils à manipuler les paquets. L'architecture de ces outils est principalement basée sur un modèle unique : ils utilisent une bibliothèque de capture, qui fournit dans l'espace utilisateur des primitives nécessaires à la capture ou l'injection de trafic sur des interfaces réelles du système. Ces bibliothèques sont conçues pour transférer de l'espace noyau à l'espace utilisateur une copie des paquets bruts.

Les principales bibliothèques de capture sont libdnet[3] et libpcap[4]. Le positionnement des deux bibliothèques est sensiblement différent : libdnet est plutôt destiné à une manipulation haut niveau (tout trafic véhiculé par Ethernet) car il offre des fonctions de manipulation d'adresse, du cache ARP et des tables de routage, ainsi que la possibilité d'injecter des paquets IP de toute nature. libpcap est une bibliothèque de plus bas niveau : les paquets sont extraits de manière brute du noyau sans traitement. libpcap dispose d'un moteur de filtrage compatible avec le filtrage de paquets BPF (Berkeley Packet Filter) ce qui permet l'utilisation de règles simples pour l'élimination des paquets indésirables. En terme de fonction programmatique, libpcap permet, après initialisation et configuration des filtres, de réceptionner les paquets de manière continue (à l'aide d'une boucle et d'un traitant) ou à la demande.

Les aspects sans fil sont gérés de manière très simple dans libpcap. Lorsqu'un paquet est reçu par la carte sans fil, celle-ci va enrichir le buffer noyau associé en rajoutant une en-tête correspondant à des informations radio. Dès lors, les cartes peuvent y ajouter des informations sur le niveau de signal reçu, sur l'antenne utilisée, sur le niveau de bruit etc... .Ce mécanisme est relativement générique : il existe plusieurs types d'en-têtes qui diffèrent très légèrement (en-tête de type Prism, RadioTap ou Atheros notamment) mais libpcap permet la reconnaissance automatique du type d'en-tête. Pour l'injection de trafic, une fonction dédiée est offerte pour injecter tout type de trame sur une carte donnée, y compris au niveau couche de liaison. Pour un paquet possible, il est donc possible de transmettre une en-tête 802.11 complet.

L'outil de manipulation scapy dont nous avons précédemment évoqué l'existence utilise la bibliothèque pcap par l'intermédiaire de liaisons en Python sur cette même bibiliothèque. Les auteurs ont donc un outil uniquement en langage interprété Python, ce qui rend extrêmement facile la manipulation de protocoles et l'ajout de nouveaux de protocoles.

Certains programmes n'utilisent pas libpcap et utilisent le mécanisme de socket raw ouverte sur l'interface sans fil pour récupérer les paquets. C'est le cas de la suite d'outils aircrack-ng[5] dédiés à la recherche de clé WEP et WPA sur les points d'accès. La

[3] http ://libdnet.sourceforge.net/
[4] http ://www.tcpdump.org/
[5] http ://www.aircrack-ng.org

réception de paquets s'effectue manuellement sur la présence de données sur la socket ouverte. Pour l'injection, le procédé est identique à une socket standard : l'écriture de données sur la socket raw va conduire à l'envoi d'un paquet sur l'interface noyau de la carte sans fil. Aircrack-ng est une suite d'applications qui s'enrichit au fur et à mesure de la découverte de nouvelles failles, cependant cela reste un outil d'exploitation et pas un outil de prototypage efficace.

L'interfaçage de ces outils en sécurité est donc minimaliste et ne bénéficie pas de performances avancées comme les architectures proposées par Click par exemple, qui proposent un interfaçage plus fin avec le noyau et le système d'exploitation. De plus, les outils en langage interprétés comme Scapy sont victimes d'une lenteur supplémentaire liée à l'exécution du code interprété. Cependant, ils restent idéaux pour effectuer rapidement un prototype d'une nouvelle attaque des protocoles de sécurité ou la manipulation du protocole 802.11.

6.1.3 Réflexions sur une nouvelle architecture

Notre choix initial de séparer l'architecture en deux sous-ensembles aux temporalités différentes nous a permis d'isoler d'une part les protocoles de bas niveau dans la partie temps réel (modulation physique et couche d'accès) et d'autre part les protocoles de haut niveau (IP, TCP, Applicatifs) à l'échelle temporelle du paquet. Pour ce deuxième ensemble que nous traitons dans ce chapitre, nous avons vu précédemment que les architectures inter-couches prévues par les chercheurs mettaient en oeuvre des mécanismes de partage d'informations, soit au moyen de message entre les couches, soit au sein d'un espace partagé.

Pour la réalisation d'une architecture avec de telles propriétés, il nous faut donc disposer d'une souplesse au niveau de l'accès et de la manipulation des données. Lors de l'examen précédent des architectures inter-couches, nous savons qu'une des premières décisions à prendre consiste à choisir l'espace mémoire qui accueillera la manipulation des paquets proprement dit.

Dans un système d'exploitation actuel, le traitement du trafic réseau s'effectue selon deux espaces distincts. D'une part, l'espace noyau reçoit les paquets bruts, effectue le filtrage éventuel des paquets, gère le routage du trafic IP selon une table de routage, puis gère l'état des connexions TCP/UDP de la couche transport. Ensuite, les données issues de connexion TCP/UDP sont recopiées en espace utilisateur pour être accessible par les applications, qui sont donc les protocoles de haut niveau (DHCP, HTTP, DNS etc...). De plus, la plupart des applicatifs dédiés à la gestion du routage est exécuté en espace utilisateur, puis interagit avec les tables de routage du noyau par l'intermédiaire d'API variable en fonction du système d'exploitation utilisé (iptables pour Linux, bpf pour les systèmes BSD...). Il est donc naturellement impossible de réaliser une architecture inter-couches efficace entre les protocoles et les couches de transport avec l'espace applicatif dans les architectures réseaux actuelles.

Les propositions examinées précédemment mettaient en oeuvre des mécanismes de communication entre l'espace noyau et l'espace utilisateur, notamment par l'utilisation de protocole de messages, comme celui mis en oeuvre dans XIAN. Cependant, ces mécanismes ne permettent pas la réalisation d'architectures dotées d'espace mémoire partagé, et de plus, requiert des modifications du noyau, ce qui peut être impossible sur des systèmes d'exploitation propriétaires. Il faut donc associer au sein d'un même espace l'ensemble des éléments relatifs aux traitements des paquets, soit en espace noyau, soit en espace utilisateur.

Loger l'intégralité du traitement des paquets en espace noyau est très séduisant : en évitant les échanges entre mode utilisateur et mode noyau, on économise des cycles processeurs dédiés à ces transferts. Cependant, si l'intégration de la pile TCP/IP permet un traitement des paquets, il n'en est pas de même pour les applications : généralement les fonctions utilisées dans les applications sont de haut niveau, ce qui implique de disposer de ces mêmes fonctions en espace noyau. Or, pour assurer la stabilité de celui-ci, les architectures tendent à restreindre les fonctions à un sous-ensemble très réduit et minimaliste. Il n'est donc pas envisageable d'importer dans cet espace les applicatifs : d'une part pour des raisons de stabilité, et d'autre part pour des raisons de comptabilité avec les applications existantes.

Nous nous orientons donc vers une solution visant à effectuer les traitements en espace utilisateur : c'est pourquoi les approches liées à la recherche à la sécurité sont si intéressantes. A travers l'étude des architectures des outils de sécurité, nous disposons des travaux déjà réalisés en matière de récupération de données en espace utilisateur, notamment par les applicatifs utilisant les différentes techniques de capture (libpcap, socket raw). De plus, les outils de manipulation de paquets tels que scapy offrent une flexibilité très intéressante grâce à l'utilisation d'un langage de script interprété, Python. Nous nous proposons donc de réaliser la base architecturale permettant d'une part d'amener les paquets en espace utilisateur, d'ouvrir une interface simplifiée pour le traitement à l'aide d'un langage de script, puis d'une fonction permettant la réémission des paquets, à la fois sur la carte sans fil, et d'autre part au sein du système lui-même, pour les applicatifs qu'ils hébergent. A partir de cet outil, nous élaborons plusieurs scénarios où les techniques inter-couches seront mises en oeuvre pour illustrer la flexibilité de cet outil.

6.2 PACket MAniPulation : un outil de traitement de paquet

Nous avons nommé notre outil PACMAP, pour PACket MAniPulation. Comme annoncé, cet outil sera la base pour la réalisation d'architectures inter-couches. L'objectif de PACMAP est la récupération des paquets bruts, à la fois issus du système et de ses applications, et à la fois des interfaces sans fil installées sur le système. PACMAP fournit aussi un interfaçage facilité pour la manipulation des paquets et notamment des paquets sans fil : PACMAP intègre une majeure partie du protocole 802.11 en son

sein et permet l'interfaçage à tous les niveaux du protocole afin d'utiliser celui-ci dans des scénarii innovants basés sur des techniques inter-couches.

6.2.1 Architecture

L'architecture retenue pour PACMAP est décrite suivant la figure 6.3.

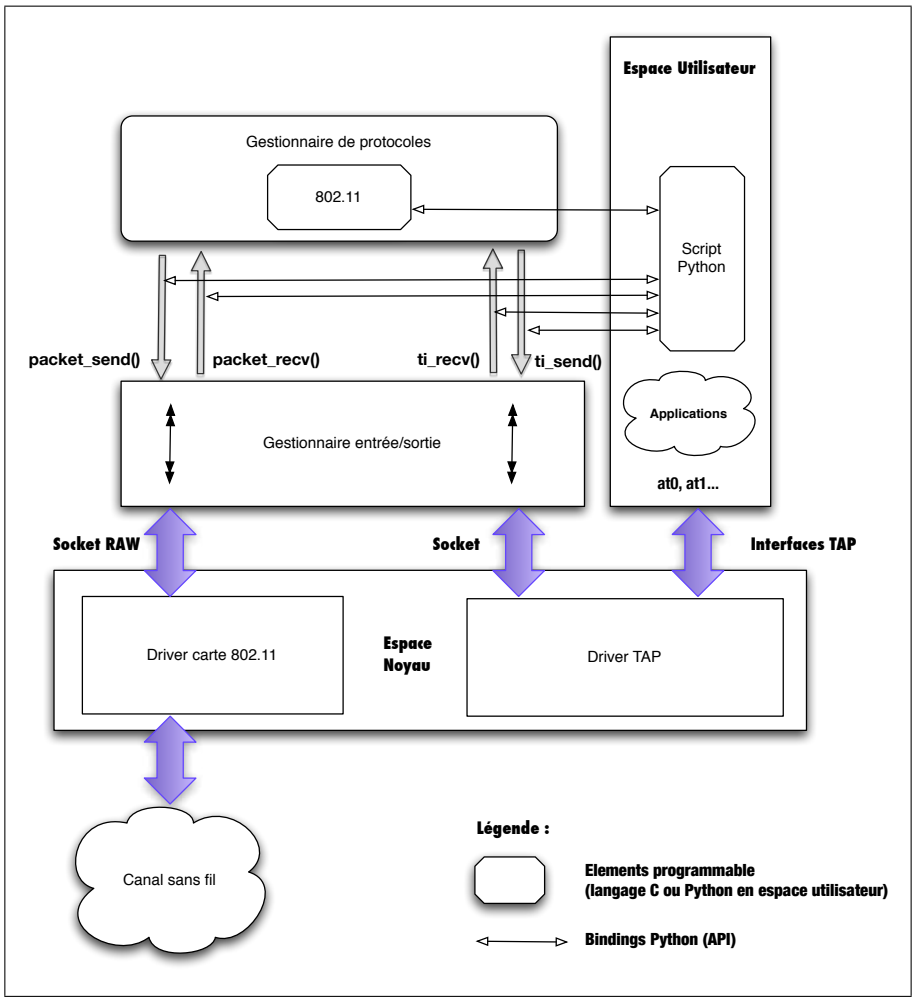


Figure 6.3 – Architecture de PACMAP.

PACMAP s'exécute totalement en espace utilisateur. A l'aide de son gestionnaire de connexion, il utilise des mécanismes de socket pour récupérer les données du système d'exploitation, à la fois des données issues et à destination de l'interface 802.11, et à la fois les données issues et à destination des applications du système. A noter que les mécanismes de socket peuvent être échangés contre des mécanismes plus

spécifiques à chaque système, ou même par cette librairie d'entrées/sorties, comme libpcap, grâce au gestionnaire de connexion qui peut masquer totalement les mécanismes de récupération et d'injection des données. L'intérêt du mécanisme de socket est qu'il garantit le fonctionnement de PACMAP sur n'importe quel système utilisant des mécanismes de socket.

PACMAP utilise 2 types de paquets :

- d'une part, les paquets issus de la carte sans fil, qui sont habituellement encapsulés avec une en-tête RadioTap, utilisé pour la récupération de statistiques (Modulation du paquet reçu, rapport signal bruit, canal de réception). PACMAP récupère ces paquets, stocke les statistiques, et génère des évènements de réception ;

- d'autre part, les paquets issus de l'espace utilisateur où s'exécutent les applications usuelles (serveur web, browser etc...). Afin de pouvoir utiliser les applications usuelles, nous utilisons une interface virtuelle reposant sur le driver TAP, qui permet de simuler une interface Ethernet basique. Tout le trafic issu de cette interface est récupéré dans PACMAP et génère là aussi des évènements.

6.2.2 Traitement d'évènements

Le modèle retenu pour PACMAP est en effet évènementiel : chaque packet reçu génère un événènment, qui est intercepté par deux ensembles d'éléments reprogrammables :

- d'une part le gestionnaire de protocole qui s'execute au sein de PACMAP, en langage C, qui permet l'implémentation optimale de protocoles. Pour l'heure, PACMAP embarque la gestion du protocole 802.11, mais peut très bien exécuter une pile TCP/IP, ou même des protocoles plus applicatifs (DNS, DHCP, HTTP...).

- d'autre part un script Python qui présente des fonctions qui sont appelées en fonction de l'évènement. Un utilisateur de PACMAP peut donc implémenter, en langage Python, l'intégralité d'un protocole qui sera géré en fonction des évènements issus de PACMAP : il peut ensuite utiliser ces évènements et traiter les données à l'aide d'une bibliothèque telle que Scapy, qui est alors utilisée uniquement pour ses fonctions de formatage de paquet, la réception et l'envoi de paquets étant géré par PACMAP en code natif.

Nous présentons plus loin les fonctions offertes et les évènements générés par PACMAP et accessibles depuis l'interpréteur Python exécutés par PACMAP.

6.2.3 Protocoles embarqués

Le gestionnaire de protocole de PACMAP permet l'exécution de protocole à partir des évènements issus des périphériques d'entrées/sorties. Dans un premier temps, nous avons implémenté un sous-ensemble du protocole 802.11 au sein de PACMAP, afin d'illustrer ce mécanisme.

Il s'agit donc d'une machine à état 802.11, qui peut gérer un ou plusieurs points d'accès virtuels (au sens où ils s'exécutent sur une même interface physique) et l'en-

semble de leurs clients respectifs. Chaque AP virtuel présente une interface TAP associée (appelée vap0, vap1...) sur lequel le trafic reçu par le point d'accès est reçu. De cette manière, on obtient un fonctionnement similaire au mode point d'accès présent sur certaines interfaces 802.11, sauf que le fonctionnement du point d'accès est altérable à souhait et permet l'implémentation de techniques d'optimisation inter-couches, soit par modification directe dans le gestionnaire de connexion, soit par l'intermédiaire des évènements générés au niveau de l'interpréteur embarqué Python.

Tout protocole peut être implémenté en parallèle ou au dessus du protocole 802.11 déjà présent. L'implémentation d'une pile complète TCP/IP est par exemple possible, ce qui permet la mise en oeuvre d'approche inter-couches sur l'ensemble de la pile réseau : action sur le routage, sur le comportement de TCP, ou même le comportement des applications.

6.2.4 Réalisation

La réalisation d'une telle architecture n'a pas posé de problème particulier, grâce à une conception progressive.

Dans un premier temps, nous nous sommes attachés à vérifier la capacité des cartes sans fil à envoyer et recevoir des données arbitraires sur le canal sans fil. Cette opération est généralement opérée lorsque la carte est placée en mode dit moniteur, où l'ensemble des filtres logiciels et matériels sont désactivés. Cette étape est la plus contraignante, dans le sens où chaque carte dispose de ses propres capacités et méthodes pour effectuer l'envoi et la réception du trafic. Nous nous sommes donc focalisés sur l'utilisation des cartes Atheros, car la mise en oeuvre du mode moniteur sur cette carte est particulièrement efficace. Pour l'injection, l'étude du code du pilote libre MadWiFi[6] a permis de déterminer le formatage des trames permettant un envoi des trames sans aucune modification par le matériel.

Ensuite, nous avons effectué le même travail au niveau du pilote TUN/TAP, pour vérifier son bon fonctionnement. Ces deux étapes étant terminées, nous avons réalisé le gestionnaire de protocole grâce à un mécanisme de processus (thread) permettant des traitements parallèles en fonction de l'interface utilisée pour le traitement du paquet. Nous avons également ajouté un processus indépendant, dont le réveil périodique grâce à l'horloge temps réel permet d'effectuer des opérations selon un échéancier donné.

Dès lors, nous avons réalisé une implémentation simplifiée du protocole 802.11, qui utilise les évènements de réception et d'émission de paquets pour traiter le contenu des paquets. Ce morceau de code effectue un découpage ("parsing") du paquet suivant le protocole 802.11 et génère le cas échéant les réponses protocolaires adéquates, et un ensemble d'évènements représentatifs de l'exécution du protocole lui-même.

Enfin, nous avons introduit une instance d'un interpréteur Python au sein de

[6] http ://madWiFi.org/

PACMAP, que nous avons enrichi de méthodes supplémentaires grâce à la fonction *Py_InitModule*(...), qui permet l'appel de fonctions écrites en C à partir d'un script Python. Cela nous permet d'enrichir les possibilités programmatiques, que nous définissons ci-après.

6.2.5 Possibilités programmatiques

Comme nous l'avons déjà souligné, PACMAP présente à la fois des évènements accessibles en langage C ou en langage Python. Cette dualité dans les possibilités d'extension permettent à PACMAP d'être très souple en fonction du scénario inter-couches à exploiter. En effet, de l'objectif final, l'utilisateur peut privilégier une implémentation rigoureuse en langage C, plus rapide à l'exécution, ou en langage interprêté Python, plus lente, mais qui facilite grandement le prototypage, mais surtout, permet la réutilisation de l'ensemble des outils déjà développés. PACMAP peut donc utiliser par exemple l'outil Scapy présenté précédemment.

Afin de permettre un large éventail d'applications l'utilisateur a accès à plusieurs groupes de fonctions. Tout d'abord, des fonctions élémentaires et génériques :

- *packet_recv*() : fonction appelée à la réception d'un paquet. Donne accès au contenu du paquet, à sa taille, ainsi qu'à l'en-tête radio issu d'une carte sans fil, ce qui permet la constitution de statistiques. De cette manière, nous obtenons le même degré de raffinement que d'autres architectures inter-couches, telles que XIAN, qui exploitent cette même source de statistiques.

- *packet_send*() : fonction appelée à l'émission d'un paquet. Permet l'émission d'un buffer arbitraire sur la carte sans fil.

- *createtap*() : fonction appelée pour la création d'une interface virtuelle, apparaissant sous forme d'interface Ethernet dans le système hôte. Utilise le driver TUN/TAP.

- *packet_ti_recv*() : fonction appelée à la réception d'un paquet sur une interface virtuelle. Permet la récupération d'un paquet généré par le système d'exploitation.

- *packet_ti_send*() : fonction appelée pour l'envoi d'un paquet sur une interface virtuelle. Permet l'envoi d'un paquet arbitraire dans le système d'exploitation.

- *periodic*() : fonction appelée à intervalles réguliers par un processus autonome de PACMAP, en fonction de l'horloge temps réel du système. Permet l'utilisation d'actions périodiques régulières.

A ce premier jeu de fonctions, se rajoutent des fonctions spécifiques à chaque protocole implémenté dans le gestionnaire de protocole. Dans notre cas, nous avons dans un premier temps uniquement le protocole 802.11. Les fonctions associées sont :

- *createvap*() : permet la création d'un point d'accès complet, géré par le gestionnaire de protocole en C.

- *proto*80211_*association*() : génère un évènement à une requête d'association.

- *proto*80211_*deassociation*() : génère un évènement à une requête de désassociation.

- *proto*80211_*authentification*() : génère un évènement à une requête d'authentification. Pour l'heure, nous n'avons pas implémenté de protocole de chiffrement de type WEP ou WPA. Cette fonction pourra servir à cet effet.

- *proto*80211_*deauthentification*() : génère un évènement à une requête de désauthentification. Pour l'heure, nous n'avons pas implémenté de protocole de chiffrement de type WEP ou WPA. Cette fonction pourra servir à cet effet.

- *proto*80211_*control*() : cette fonction permet la récupération d'un acquittement à un paquet qui a été correctement transmis. Peut être utilisé pour l'implémentation de nouveaux algorithmes de gestion du débit.

D'autres fonctions présentes dans le protocole 802.11 sont également introduites, nous n'avons conservé ici que les principales à fin d'illustration. L'ensemble de ces fonctions ont un équivalent en Python.

Nous allons maintenant illustrer l'utilisation de cette architecture dans plusieurs scénarii visant l'utilisation de techniques inter-couches.

6.3 Utilisation de PACMAP dans le cadre des réseaux de nouvelle génération

Au cours de nos travaux de thèse, nous avons été amené à travailler sur plusieurs projets visant à définir de nouvelles architectures pour des réseaux intégralement sans fil. Nous retenons plus particulièrement le projet européen WIP - Wireless Internet Protocol - et le projet Airnet qui nous a amené à participer au sein de l'équipe DRAKKAR à des propositions d'architecture telle que [110]. Dans cette proposition, notre équipe a essayé de couvrir l'ensemble des problématiques liées à la réalisation intégralement sans fil, couvrant ainsi les problèmes liés à la nomenclature (qu'est-ce qu'un noeud...), l'adressage, le routage ainsi que la gestion de la mobilité. Cependant, les solutions proposées dans cette publication restent avant tout théoriques.

Nous proposons donc d'utiliser PACMAP pour réaliser certains éléments de l'architecture proposée, ce qui permettra d'une part d'illustrer l'implémentation de prototypes avec PACMAP tout en validant l'utilisant d'une telle architecture de traitement de paquet sur des réseaux de nouvelle génération.

6.4 Exemple 1 : Une interface matérielle unique, des rôles multiples

Lors de notre étude de l'implémentation actuelle des réseaux sans fil 802.11 dans notre état de l'art, nous avons pu voir à quel point la relation entre adaptateur physique et interface logique était établie au sein des systèmes d'exploitation actuels, et plus particulièrement, au sein de la pile réseau des noyaux de ces mêmes systèmes d'exploitation. Les études et implémentations d'architecture inter-couches ont montré

la nécessité d'un espace partage pour faciliter l'échange de données entre les couches ; de même, notre étude des outils de la recherche en sécurité nous a conduit à adopter pour notre outil PACMAP un fonctionnement basé sur une architecture de traitement de paquets en espace utilisateur.

Dès lors, dans PACMAP le rôle de l'adaptateur réseau est réduit à sa plus simple expression : l'envoi et la réception de paquets sur un canal sans fil donné. La pile réseau doit donc évoluer.

L'analogie qui nous semble évidente ici est le récent développement des techniques de virtualisation ou même, de paravirtualisation (comme le projet Xen [111]), où un ordinateur exécute parallèlement plusieurs systèmes d'exploitation sur une unique plate-forme physique.

PACMAP permet de réaliser, de manière similaire, une exécution en parallèle de plusieurs piles réseaux : la ressource partagée étant ici la bande passante du réseau sans fil, et l'abstraction étant réalisée à l'échelle du paquet. La figure 6.4 traduit l'équivalence entre le modèle rendu possible par PACMAP et les modèles classiques issus de la virtualisation. Déjà, une telle comparaison avait été proposée par Zec dans [106], où l'auteur proposait d'introduire dans le noyau FreeBSD la notion de pile réseau clonable, afin d'améliorer entre autres les performances des solutions de virtualisation, développer l'usage de techniques de VPN transparents. Cependant, ce travail a depuis été intégré dans la réalisation de solution de virtualisation sous FreeBSD pour l'ensemble du système, et semble avoir abandonné le domaine initial du réseau.

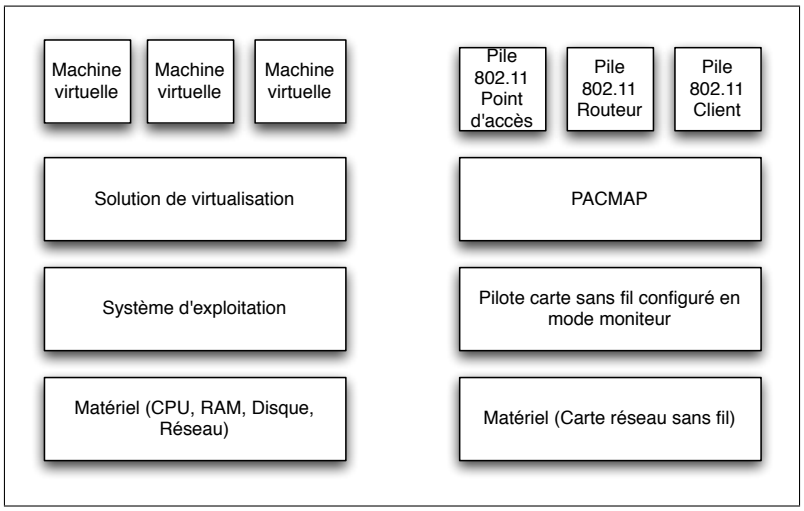


Figure 6.4 – Analogie entre la virtualisation dans les systèmes d'exploitation et la vision réseau introduite par PACMAP.

Dans une optique de mise en réseau sur un support sans fil, il est donc tout à fait possible de faire assumer à un seul adaptateur physique un ensemble de rôles

simultanés :

- être un point d'accès pour des clients locaux 802.11
- être un routeur pour du trafic dans un mesh
- être client d'un réseau 802.11
- participer à un réseau ad hoc 802.11

De plus, chacun de ces rôles peut s'exécuter au sein d'un espace partagé, ou alors dans des espaces distincts : on peut donc introduire des notions similaires au concept dit de *jail* (emprisonnement), vu dans les systèmes BSD : un programme peut s'exécuter dans un espace limité où l'accès aux ressources du système est isolé. Dès lors, les éventuelles failles de sécurité d'un protocole peuvent être isolées dans PACMAP, voir même dans un sous-processus de PACMAP.

Les seules restrictions sont d'ordre matériel. D'une part, la carte sans fil ne peut fonctionner que sur un seul canal ; la bande passante est donc limitée. A cela, il faut ajouter que chaque carte dispose de techniques différentes pour gérer les aspects temps réel, comme notre étude de l'implémentation d'une méthode d'accès l'a souligné. Ainsi, les cartes Atheros sont capables d'acquitter dans le temps imparti tout paquet 802.11 de manière autonome, du moment que l'adresse du réseau 802.11 (le BSSID) soit correctement inscrit dans le registre ad hoc de la carte. Les cartes 802.11 utilisant ce chipset ont donc notre préférence, car elles permettent les mécanismes d'envoi et de réception du traitement des paquets, ce qui permet d'atteindre les objectifs fixés par PACMAP : une architecture de traitement de paquets.

Un autre facteur limitatif dans le traitement de ces rôles simultanés est le système hôte lui-même, qui doit être capable de gérer les entrées/sorties de paquets, ainsi que leur traitement. Afin d'évaluer au mieux les performances de PACMAP, nous nous attachons maintenant à traiter un exemple complet de pile réseau 802.11 réalisée à l'aide de PACMAP.

6.5 Exemple 2 : Gestion des clients 802.11 actuels

6.5.1 Utilisation du protocole 802.11 pré-implémenté

Le nombre considérable des équipements 802.11 actuellement en circulation sur le marché des périphériques mobiles rend obligatoire la prise en compte de ces périphériques originels dans la mise en oeuvre de réseaux sans fil. Il convient donc de recréer, pour ces terminaux, les conditions habituelles de connectivité, c'est-à-dire, la mise à disposition d'un point d'accès et la gestion du protocole 802.11.

L'outil PACMAP incorpore d'ores et déjà le protocole 802.11 dans son gestionnaire de protocole. La création d'un point d'accès se résume dans ce cas à un script minimal pour piloter PACMAP :

```
1 #!/usr/bin/python
2
```

```
3  import pacmap
4
5  def background():
6      bssid = pacmap.getbssid()
7      essid = "Test"
8      pacmap.createvap(essid,bssid)
9      return 0
```

Dans cet exemple, nous utilisons la fonction *background*() qui permet d'appeler au démarrage de PACMAP des opérations de tâches de fond. Ici, nous créons un point d'accès avec pour nom "Test" et utilisant l'adresse matérielle de la carte sans fil utilisée.

6.5.2 Utilisation du protocole 802.11 par réimplémentation externe

Une autre possibilité de PACMAP est de reconstituer l'intégralité d'une machine à état 802.11 grâce à l'interface en langage Python. Nous pouvons ensuite utiliser la bibliothèque Scapy pour formater nos messages 802.11.

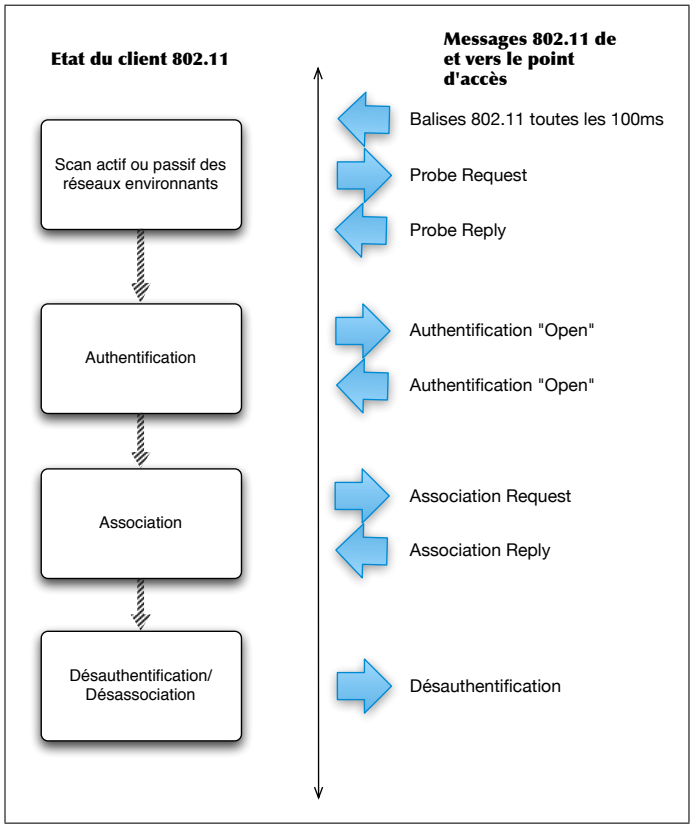


Figure 6.5 – Machine à état d'une client 802.11 et messages 802.11 correspondants.

Dans ce cas, le script de pilotage consiste à traiter chaque évènement 802.11 de la machine état. Le comportement d'un client 802.11 est décrit sur la figure 6.5.

Il s'agit donc d'implémenter l'ensemble des messages concernés, il y a donc :

- l'émission périodique des balises (beacons) par le point d'accès, indiquant le nom du réseau (SSID), l'adresse du réseau (BSSID) ainsi que les modulations autorisées ; cette opération étant périodique, elle sera exécutée à intervalles réguliers.

- la gestion des sondes actives (Probe) du client : à chaque requête (Probe Request), le point d'accès doit répondre par une réponse indiquant les mêmes éléments qu'une balise (SSID, BSSID, modulations)

- la gestion de l'authentification : les stations utilisent une phase d'authentification, même si le réseau est non-chiffré. Dans ce cas, l'authentification consiste à renvoyer le message d'origine sans modification, si ce n'est l'incrémentation d'un numéro de séquence.

- la gestion de l'association : la station s'adresse au point d'accès pour demander une association. Le point d'accès répond en indiquant une numéro d'association.

- la désassociation/désauthentification : les implémentations 802.11 correctes envoient un paquet de désauthentification sans utiliser l'expiration d'une association. Le point d'accès n'a pas à échanger de paquet.

L'implémentation d'une telle machine à état est triviale avec la combinaison PACMAP et Scapy. Tout d'abord, nous créeons en tâche de fond un périphérique Ethernet qui servira à la communication avec le système, puis nous retournons un simple index qui servira d'identificateur pour la suite :

```
1  #!/usr/bin/python
2  import pacmap
3  from scapy import *
4
5  bssid = ""
6  current_timestamp = time.mktime(...
7  index = 0
8
9  # fonction appelée en tâche de fond au démarrage
10 def background():
11     global bssid
12     global index
13     bssid = pacmap.getbssid()
14     index = pacmap.createtap()
15     return 0
```

Puis, nous utilisons la fonction *periodic*() pour gérer l'envoi périodique des balises. Les balises sont des messages créés à l'aide de la bibliothèque Scapy, grâce aux objets *Dot*11 et *Dot*11*Elt* qui représentent respectivement un paquet 802.11 et un champ "Information Element" utilisé pour le transport d'information dans les balises. Nous utilisons ensuite la fonction d'envoi de paquet de PACMAP à la vitesse nominale (1 MBits/s) pour envoyer le paquet formé sur le canal sans fil. Le code résultant est donc

minimal :

```
# fonction appelée périodiquement
def periodic():

    dot11_answer = Dot11(
            FCfield ="retry",
            SC= (seq<<4 & 0xff) |( (seq>>4 & 0xff) << 8 & 0xff00) ,
                type = "Management",
            addr1 = "ff:ff:ff:ff:ff:ff",
                addr2 = bssid,
                addr3 = bssid)/
        Dot11Beacon(timestamp = current_timestamp,
        cap = 0x0104,beacon_interval = beacon_interval )/
        Dot11Elt(ID=0,info="Test")/
        Dot11Elt(ID=1,info="\x82\x84\x8b\x96")/
        Dot11Elt(ID=3,info="\x06")/
        Dot11Elt(ID=50,info="\x0c\x12\x18\x24\x30\x48\x60\x6c")

    pacmap.sendpacket(str(dot11_answer),
         len(str(dot11_answer)),1)

    return 0
```

Il faut maintenant implémenter la gestion des requêtes de détection de réseau émanant des clients. Pour ce faire, nous utilisons l'évènement $proto80211_probereq$ qui correspond à une requête de détection émise par un client, intercepté par notre gestionnaire de protocole intégré et qui appelle à son tour cette fonction en Python. Il suffit ensuite de récupérer dans le paquet d'origine l'adresse de l'émetteur, puis d'envoyer une réponse correctement formatée :

```
def proto80211_probereq(packet, length):
    dot11_frame = Packet(packet)
    dot11_frame.decode_payload_as(Dot11)
     client  = dot11_frame.getlayer(Dot11).addr2
    dot11_answer = Dot11(
    type = "Management",
    addr1 = dot11_frame.getlayer(Dot11).addr2,
    addr2 = bssid,
    addr3 = bssid)/
    Dot11ProbeResp(timestamp=current_timestamp,cap = 0x0104)/
    Dot11Elt(ID=0,info="Test")/
    Dot11Elt(ID=1,info="\x82\x84\x8b\x96")/
    Dot11Elt(ID=3,info="\x06")/
    Dot11Elt(ID=50,info="\x0c\x12\x18\x24\x30\x48\x60\x6c")
    pacmap.sendpacket(str(dot11_answer),len(str(dot11_answer)),1)
     return 1
```

Même chose pour la phase d'authentification :

```
def proto80211_auth(packet, length):
    dot11_frame = Packet(packet)
    dot11_frame.decode_payload_as(Dot11)
    dot11_answer = dot11_frame
    addr1=  dot11_frame.getlayer(Dot11).addr2
    dot11_answer.getlayer(Dot11).addr1= addr1
    dot11_answer.getlayer(Dot11).addr2= bssid
    dot11_answer.getlayer(Dot11Auth).seqnum=0x02
    pacmap.sendpacket(str(dot11_answer),len(str(dot11_answer)),1)
    return 1
```

Et enfin l'association proprement dite ;

```
def proto80211_assocreq(packet, length):
    print "Association␣request␣received"
    dot11_frame = Packet(packet)
    dot11_frame.decode_payload_as(Dot11)
    dot11_answer = dot11_frame
     client  = dot11_frame.getlayer(Dot11).addr2
    dot11_answer = Dot11(
    type = "Management",
    addr1 = dot11_frame.getlayer(Dot11).addr2,
    addr2 = bssid, addr3 = bssid)/
    Dot11AssoResp(cap = 0x0104, AID=0xc001)/
    Dot11Elt(ID=0,info=ssid)/Dot11Elt(ID=1,info="\x02\x04\x0b\x16\x0c\x12\x18\x24")/
    Dot11Elt(ID=50,info="\x30\x48\x60\x6c")
    pacmap.sendpacket(str(dot11_answer),len(str(dot11_answer)),1)
    return 1
```

Le protocole 802.11 étant implémenté grossièrement, il suffit maintenant de gérer les entrées/sorties de paquets. Nous offrons une interface Ethernet au niveau du système que nous avons créé au début par l'utilisation de la fonction *createtap*(), il faut donc gérer la réception de paquets sur cette interface et la réémission sur le lien sans fil. La fonction *sendpacket* ajoute automatiquement une en-tête 80211 au paquet d'origine s'il n'est pas présent. Le code est là aussi donc minimal :

```
# fonction appelée à la réception de donnée sur l'interface Ethernet
def packet_ti_recv(packet, length,index):
    ether_frame = Packet(packet)
    ether_frame.decode_payload_as(Ether)

    dot11_answer=Dot11(type = "Data",
    addr1 = ether_frame.getlayer(Ether).dst,
    addr2 = ether_frame.getlayer(Ether).src,
    addr3 = bssid,
    FCfield=0x02)/LLC()/SNAP(code=ether_frame.getlayer(Ether).type)/
    ether_frame.getlayer(Ether).payload

    pacmap.sendpacket(str(dot11_answer),len(str(dot11_answer)),speed)
```

```
14      return 1
```

Enfin, il suffit maintenant de gérer les paquets entrants par l'interface sans fil, qui seront ensuite écrits sur l'interface Ethernet offerte au système d'exploitation. Cette étape consiste à enlever l'en-tête 802.11 ainsi que les couches de liaison pour les remplacer par une simple couche de liaison Ethernet. Scapy offre également le formatage de ce type de message, il ne reste ensuite qu'à l'envoyer :

```
1  def proto80211_data(packet, length):
2      dot11_frame = Packet(packet)
3      dot11_frame.decode_payload_as(Dot11)
4      if  dot11_frame.haslayer(SNAP):
5              eth_sent_frame = Ether(
6              dst=dot11_frame.getlayer(Dot11).addr3,
7              src=dot11_frame.getlayer(Dot11).addr2,
8              type=dot11_frame.getlayer(SNAP).code)
9              eth_sent_frame.payload =
10             dot11_frame.getlayer(SNAP).payload
11
12             pacmap.sendtipacket(str(eth_sent_frame),
13             len( str (eth_sent_frame)),index)
14     return 1
```

À ce stade, nous disposons d'un point d'accès minimaliste, à l'aide de quelques lignes de script écrites en Python.

6.5.3 Remarques sur les différentes implémentations et performances obtenues

À l'aide de PACMAP nous avons pu implémenter un point d'accès minimaliste en utilisant d'une part une machine à état intégrée écrite en langage C, et d'autre part une machine à états écrite en Python à l'aide de la bibliothèque Scapy. Vis-à-vis d'un point d'accès traditionnel implémenté dans le noyau du système d'exploitation, nous nous attendons à des performances dégradées.

Un premier facteur de dégradation des performances est l'exécution en espace utilisateur : au sein de celui-ci, les tâches sont interruptibles par le noyau du système d'exploitation. De plus, PACMAP récupère les données de deux sources : d'une part la carte sans fil, et d'autre part du driver TUN/TAP responsable des échanges avec les interfaces virtuelles éventuelles que nous créons pour les applications du système hôte. Ces drivers fonctionnant en espace noyau, des échanges mémoires sont donc nécessaires pour permettre l'accès aux données en espace utilisateur : ces recopies ont un coût non négligeable et ajoutent un temps de traitement supplémentaire.

Mais notre principal souci concerne l'utilisation du langage interprété Python. L'exécution de celui-ci est de part sa nature même considérablement plus lente que du code machine natif. De plus, la bibliothèque Scapy est connue pour ses lenteurs

à l'exécution, lié à la gestion d'un grand nombre de protocole. Nous nous attendons donc à des performances fortement dégradées en cas d'utilisation exclusive de fonctions écrites en Python.

Afin d'évaluer les performances, nous allons étudier plusieurs paramètres. D'une part, le comportement en régime saturé du point d'accès, en envoi et en réception depuis un client standard ; puis, nous étudierons la latence, en évaluant plus finement l'impact des différents choix d'implémentations.

Tout d'abord, nous effectuons un test en régime saturé, par envoi de paquets UDP de 1500 octets en envoi ("uplink" vers le point d'accès) et en réception ("downlink" vers le point d'accès) de et vers un client utilisant une carte classique de chipset Atheros avec le driver MadWiFi, sur la bande des 5Ghz, sur un canal libre de toute interférence issue d'équipements 802.11, en utilisant uniquement la modulation OFDM de 54Mbit/s.

Tout d'abord, la réception de paquets est présentée sur la figure 6.6 :

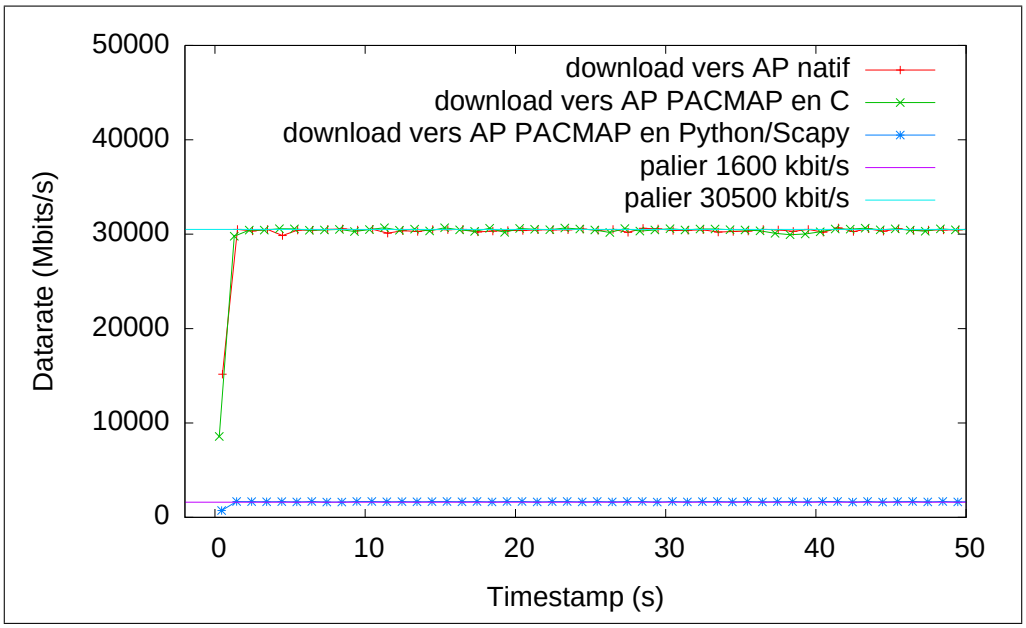


Figure 6.6 – Débits comparés entre point d'accès natif, point d'accès réalisés avec PACMAP en C, et point d'accès en Python avec PACMAP/Scapy depuis le point d'accès vers le client.

Nos premières constations sont intéressantes : nous obtenons un débit pratique quasi identique, de 30,5 Mbits/s, pour le point d'accès natif (carte en mode point d'accès) et pour notre point d'accès réalisé à l'aide de PACMAP en langage C. Par contre, le point d'accès réalisé à l'aide PACMAP en Python à l'aide de la bibliothèque Scapy offre un débit très faible de 1600 Kbit/s, qui s'explique par un très grand nombre

de paquets perdus : 100% des ressources de la machine sont occupées par PACMAP dans ce cas, à comparer au taux d'utilisation d'un code en C, que nous avons évalués entre 2 et 8%.

Puis, l'envoi de paquets est présenté sur la figure 6.7 :

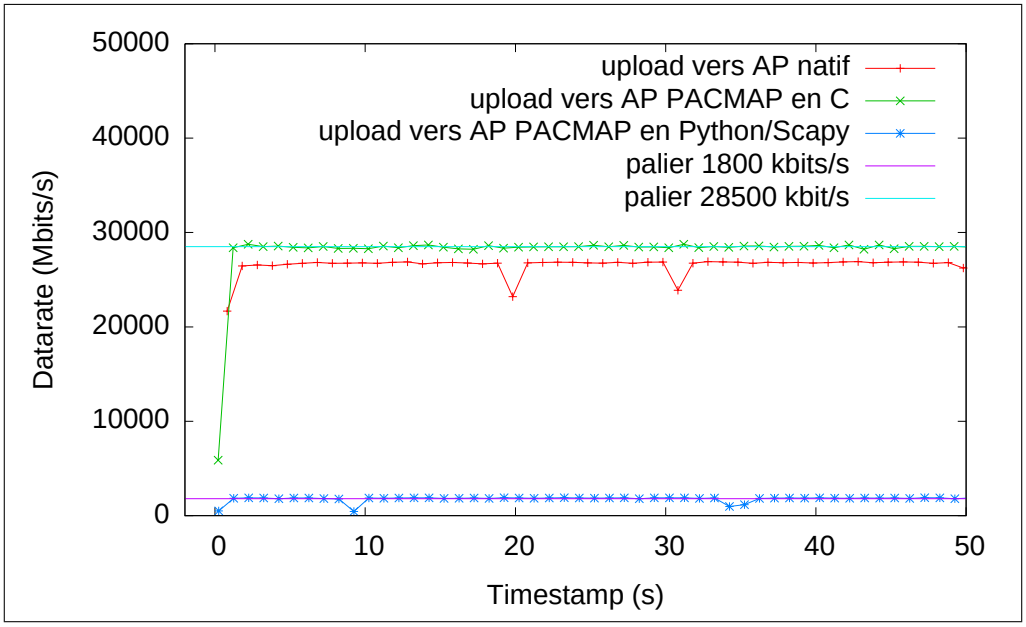


Figure 6.7 – Débits comparés entre point d'accès natif, point d'accès réalisés avec PACMAP en C, et point d'accès en Python avec PACMAP/Scapy, depuis le client vers le point d'accès.

Nous effectuons des constations similaires à la réception de données. Le point d'accès natif ainsi que PACMAP et son composant en C offrent des débits similaires de 28500 Kbit/s, avec un gain remarquable pour PACMAP. Cela est attribuable au fonctionnement de PACMAP, qui injecte les paquets avec la même méthode d'accès DCF, alors qu'un point d'accès classique adopte une priorité pour l'envoi des balises. Dans le cas de PACMAP, il n'y a pas de mécanisme de priorité, l'envoi de paquets n'est donc pas altéré par ces mécanismes, et le débit est donc maximal. La version écrite en Python à l'aide de Scapy offre, comme à la réception des débits très faibles de 1800 kbits/s, qui s'expliquent par des pertes de paquets conséquentes : le processus PACMAP occupe 100% des ressources de la machine, à comparer là aussi au taux d'utilisation de PACMAP avec du code écrit en C (entre 2 et 8%).

Afin d'expliquer les pertes observées par notre implémentation en Python, revenons sur les fonctions offertes par scapy, et plus particulièrement sur celle-ci :

```
1  dot11_frame = Packet(packet)
2  dot11_frame.decode_payload_as(Ether)
```

Cette fonction permet d'identifier, à partir d'un paquet brut, des structures de type objet permettant la manipulation du paquet et de ses champs. Nous nous en servons pour récupérer le contenu des champs. Cependant, son fonctionnement est complexe : à partir du paquet, elle détermine le type de celui-ci, ainsi que l'ensemble des couches encapsulées dans celui-ci. C'est donc une fonction récursive assez coûteuse en temps de traitement.

L'étude de la latence va apporter des réponses sur cette question.

Nous étudions le comportement de la commande *ping* usuelle dans les différents cas de figures soulevés ici, au moyen d'un histogramme.

Nous mesurons les résultats d'un ping de 64 octets, envoyés depuis un client vers un point d'accès à la modulation de 54Mbit/s toutes les 100 ms. Le calcul d'un histogramme sur ces résultats permet le tracé des figures suivantes 6.8 et 6.9 :

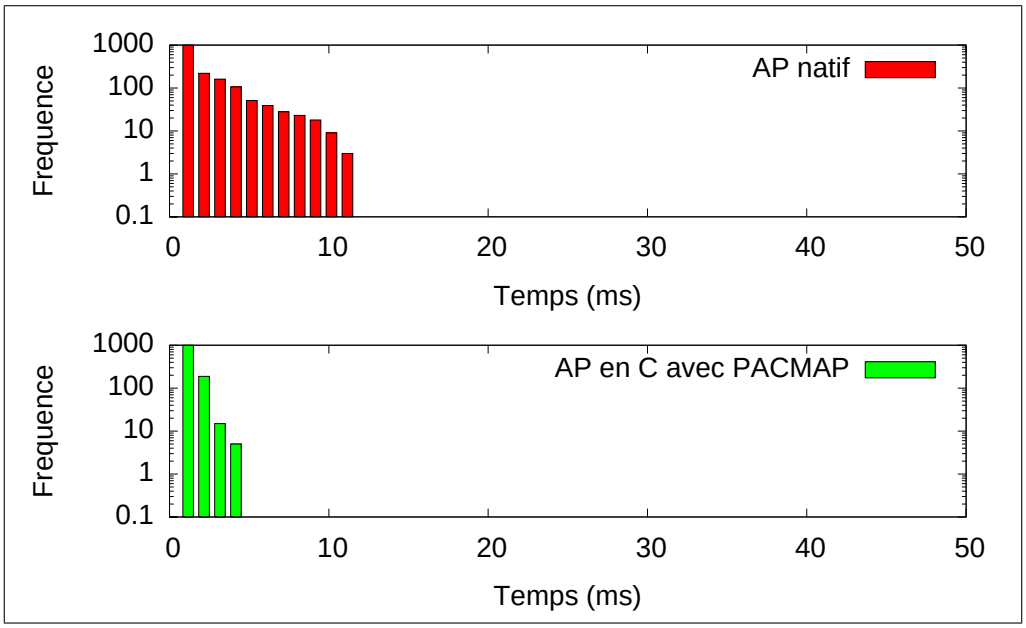


Figure 6.8 – Répartition des délais lors d'un ping entre client et point d'accès, en haut point d'accès natif, en bas point d'accès écrit avec PACMAP en C.

Sur la figure 6.8, nous observons que le délai offert en utilisant PACMAP est sensiblement meilleur que celui d'origine, là encore grâce à l'absence de mécanisme de files d'attente liée au fait que PACMAP injecte de manière brut les paquets. Les résultats observés sont en parfaite corrélation avec les observations effectuées sur le réseau.

Sur la figure 6.9, nous observons que le délai offert en utilisant la combinaison PACMAP/Scapy offre des délais beaucoup plus dispersés, indicateurs d'un temps de traitement individuel de chaque paquet très supérieur au code en C.

Afin de vérifier cette assertion, nous rajoutons au script d'exécution de PACMAP

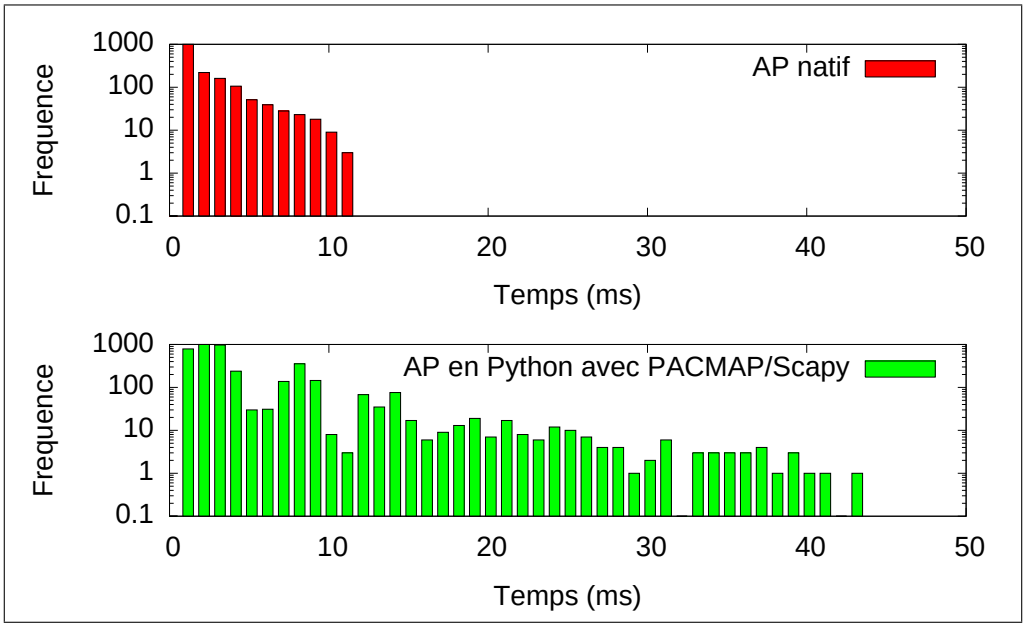


Figure 6.9 – Répartition des délais lors d'un ping entre client et point d'accès, en haut point d'accès natif, en bas point d'accès écrit avec PACMAP/Scapy en Python.

utilisant le protocole 802.11 intégré en C une composante Python à chaque arrivée de paquets, où nous exécutons le code suivant :

```
1  def packet_recv(packet, length):
2      dot11_frame = Packet(packet)
3      dot11_frame.decode_payload_as(Dot11)
4      return 0
5
6  def packet_ti_recv(packet, length,index):
7      ether_frame = Packet(packet)
8      ether_frame.decode_payload_as(Ether)
9      return 0
```

Par cet ajout, nous utilisons Scapy juste pour examiner le paquet, sans modifier quoi que ce soit. Nous réitérons l'expérience précédente avec l'utilisation de *ping*, et nous comparons à nos résultats précédents.

Sur la figure 6.10, nous observons que les fonctions en Python de Scapy à chaque arrivée/départ de paquet dans le code en C introduisent un délai de traitement supplémentaire qui impacte la latence des paquets de manière non négligeable.

De ces résultats, nous pouvons établir le cadre d'utilisation précis de PACMAP :

- PACMAP doit être utilisé avec du code en C pour les applications nécessitant un temps de traitement minimal et utilisant des taux d'entrées/sorties de paquets très importants (régime saturé). L'utilisation de code en espace utilisateur a très peu

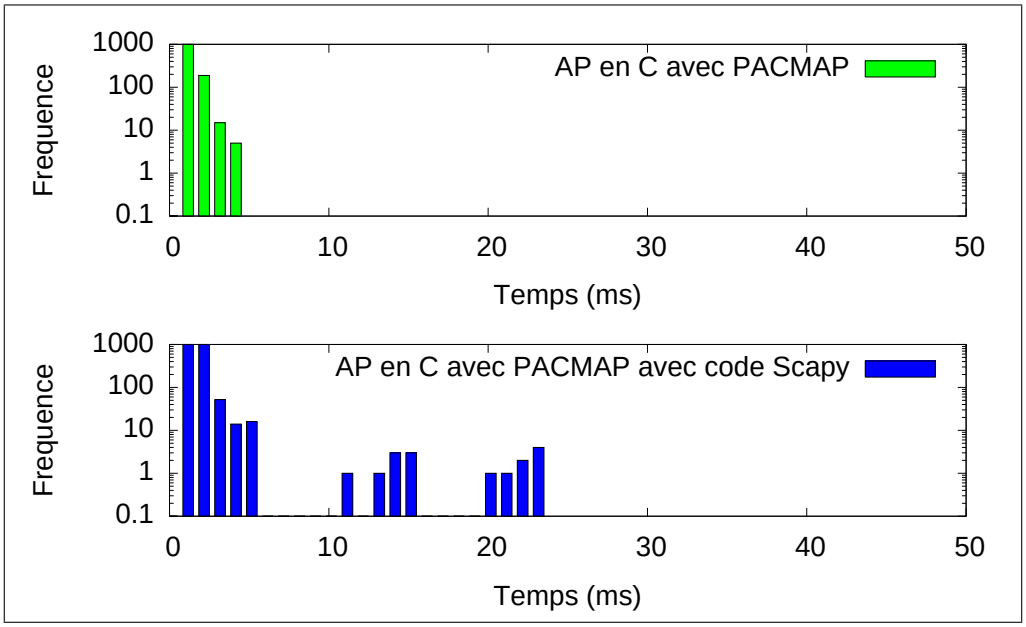


Figure 6.10 – Répartition des délais lors d'un ping entre client et point d'accès, en haut point d'accès en C utilisant PACMAP, en bas point d'accès écrit avec PACMAP en C avec des interceptions utilisant Scapy.

d'influence sur les débits et les latences obtenus, tant que le code ne fait pas appel à des fonctions en Python.

- PACMAP peut être utilisé avec du code Python avec des fonctionnalités identiques à du code en C, mais pour des débits limités (nous avons mesuré une saturation à 1,6 Mbits et 1,8 Mbits lors de nos tests précédents). Il convient donc de limiter les modulations utilisées au débit de 1Mbits/s, afin de garantir une absence de perte de paquets. Les applications utilisant du code écrit en Python et reposant sur Scapy devront prendre en compte la latence induite par l'utilisation de scapy, qui rajoute un délai de l'ordre de 40 ms lors du traitement des paquets.

Ces considérations en terme de performances étant établies, nous pouvons maintenant utiliser PACMAP sur des exemples d'architectures de nouvelle génération dans un cadre expérimental plus défini.

6.6 Exemple 3 : Adressage traditionnel

L'adressage est un sujet épineux récurrent dans les différentes propositions d'architecture de réseaux sans fil. La complexité provient de l'héritage des réseaux actuels, où le terme adresse désigne plusieurs éléments à la fois liés mais distincts. Plusieurs propositions [113] [107] [112] ont essayé de proposer d'encadrer l'usage de ce terme à une

définition précise : une adresse est un identifiant binaire qui permet de localiser un point de terminaison du réseau de telle manière que les routeurs peuvent acheminer le trafic vers le point donné.

Dans les réseaux actuels, le terme adresse est utilisé de manière parallèle dans plusieurs couches : au niveau Ethernet, les adresses sont fortement liées au matériel, et sont représentées par 6 octets. Au niveau IP, les adresses peuvent être composées d'un groupe de 4 octets (IP version 4) ou 16 octets (IP version 6). Le routage est réalisé habituellement au niveau IP. Les adresses matérielles sont utilisées dans un contexte d'interconnexion physique : elles sont généralement réservées dans le cadre des liaisons physiques, par exemple un réseau Ethernet repose sur l'utilisation des adresses Ethernet pour les échanges de données entre les cartes et les commutateurs.

Ainsi, pour les réseaux sans fil actuels, les adresses physiques sont utilisées de manière plus complète : chaque station dispose d'une adresse matérielle à 6 octets, et l'identifiant physique d'un réseau, le BSSID, est généralement l'adresse physique du point d'accès utilisé. Dans ce cadre, la création de nouvelles architectures pour des réseaux sans fil passe par la gestion de ces adresses et la mise en relation avec des identifiants utilisables par des routeurs. Un protocole assurant la correspondance entre ces adresses physiques et les adresses dites "routables" est donc nécessaire. Par héritage, le protocole ARP est utilisé.

Pour compléter ce dispositif, les stations utilisent un protocole d'auto-configuration pour l'assignation d'adresses routables. Pour les adresses IPv4, les stations dans les systèmes d'exploitation modernes utilisent une combinaison de deux systèmes : d'une part, un protocole d'autoconfiguration automatique, AutoIP, qui assigne automatiquement une adresse de la forme 169.254.X.Y, et qui repose sur ARP pour vérifier l'unicité de l'adresse choisie ; d'autre part, les stations utilisent le protocole DHCP (Dynamic Host Configuration Protocol) qui permet d'obtenir un bail sur une adresse de la part d'un serveur, grâce à un échange en 3 temps. Premier temps, découverte des serveurs par une trame diffusée ; ensuite, réponse des serveurs disponibles, puis demande d'une offre d'adresse à un serveur spécifique par le client, puis réponse du serveur choisi. Le protocole IPv6 utilise un mécanisme plus efficace de préfixe émis par le routeur local, qui émet sur le lien physique une trame d'autoconfiguration. Les clients ajoutent à ce préfixe leur adresse matérielle, ce qui garantit de bonnes propriétés d'unicité.

Dans un contexte de réseau basé sur du multi-sauts, ces considérations ne sont plus acceptables : les trames ARP et DHCP ne peuvent voyager sur l'intégralité du réseau sans fil. Elles doivent donc être associées à un contexte local, au sein de la relation qui unit un client et son point d'accès. Des propositions ont déjà été émises dans ce sens : avec LUNAR [114], Tschudin et al. proposent une interception efficace des requêtes ARP et DHCP pour les substituer par des mécanismes plus adaptés à des réseaux de type mesh. Dans LUNAR, l'interception du trafic DHCP passe par la simulation d'un serveur DHCP dans l'application. Par contre, le trafic ARP passe par une récupération

du trafic ARP issu du noyau, et une altération de la table ARP du noyau.

Nous pensons qu'il est souhaitable, dans les réseaux de nouvelle génération, d'avoir une granularité plus fine et un contrôle exclusif dans la gestion de ces mécanismes de compatibilité. PACMAP repose sur un fonctionnement isolé de la pile réseau dans un processus en espace utilisateur. Il est donc possible, à l'aide des mécanismes évènementiels, d'intercepter le trafic ARP et DHCP et de fournir des réponses adaptées.

Pour illustrer la facilité avec laquelle PACMAP peut intercepter ce type de trafic, nous utilisons quelques lignes de code Python écrite à l'aide de Scapy. Tout d'abord, nous interceptons toutes les requêtes ARP :

```
def proto80211_data(packet, length):
  dot11_frame = Packet(packet)
  dot11_frame.decode_payload_as(Dot11)
  mac = dot11_frame.getlayer(Dot11).addr2

  # détection du trafic ARP et identification des requêtes
  if dot11_frame[ARP] and dot11_frame[ARP].op == 1:

    dot11_answer=Dot11(type = "Data", addr1 = mac,
        addr2 = bssid,addr3 = bssid,FCfield=0x02)/
        LLC()/SNAP()/
        ARP(op=2,hwsrc=bssid,psrc="192.168.10.1",
        hwdst=dot11_frame[ARP].hwsrc,pdst=dot11_frame[ARP].psrc)

    # envoi de la trame
    pacmap.sendpacket(str(dot11_answer),len(str(dot11_answer)),speed)
```

De manière similaire, le même type d'interception peut avoir lieu pour le protocole DHCP :

```
  mac = dot11_frame.getlayer(Dot11).addr2
  ip_client = "192.168.10.2"
  ip_server = "192.168.10.1"


  if dot11_frame[DHCP] and dot11_frame[DHCP].options[0][1] == 1:
    # paquet de type "DHCPDISCOVER"
    # Réponse : on signale le pseudo serveur DHCP au client

    dot11_answer=Dot11(type = "Data", addr1 = mac,
        addr2 = bssid, addr3 = bssid,FCfield=0x02)/
        LLC()/SNAP()/
        IP(src=ip_server)/UDP(sport=67,dport=68)/
        BOOTP(op=2,giaddr=ip_server,
        chaddr=mac2str(mac), xid=dot11_frame[BOOTP].xid)/
        DHCP(options=[('message-type','offer'),('server_id',ip_server),
        ('lease_time',43200), ('renewal_time',21600),('rebinding_time',37800),
        ('subnet_mask',"255.255.255.0"),('router',ip_server),
        ('name_server',ip_server),('end')])
```

```
20
21  if dot11_frame[DHCP] and dot11_frame[DHCP].options[0][1] == 3:
22    # paquet de type "DHCPREQUEST"
23    # Réponse : acquittement "ACK"
24
25    dot11_answer=Dot11(type = "Data",
26         addr1 = mac,
27         addr2 = bssid,
28         addr3 = bssid,
29         FCfield=0x02)/
30         LLC()/SNAP()/
31         IP(src=ip_server,dst=ip_client)/
32         UDP(sport=67,dport=68)/
33         BOOTP(op=2, ,yiaddr= ip_client ,giaddr=ip_server,
34         chaddr=mac2str(mac), xid=dot11_frame[BOOTP].xid)/
35         DHCP(options=[('message-type','ack'),
36         ('server_id',"ip_server"), ('lease_time',43200),
37         ('renewal_time',21600),('rebinding_time',37800),
38         ('subnet_mask',"255.255.255.0"),('router',"ip_server"),
39         ('name_server',"ip_server"),('end')])
40
41  # Envoi de la réponse au client
42    pacmap.sendpacket(str(dot11_answer),len(str(dot11_answer)),speed)
```

Là encore, la combinaison PACMAP/Scapy permet de résoudre assez rapidement le problème de l'interception protocolaire. Bien entendu, ici nous avons illustré seulement la partie interception, mais il serait tout à fait possible d'implémenter grâce à PACMAP des protocoles permettant un adressage plus efficace dans le coeur du réseau sans fil, comme la proposition XRP de LUNAR, citée précédemment.

6.7 Exemple 4 : Gestion de la mobilité - AP virtuel

Nous cherchons à mettre en oeuvre PACMAP dans un scénario plus complet cette fois-ci, pour résoudre le problème de la mobilité transparente. Comme nous l'avons vu dans notre autopsie des réseaux 802.11, la mobilité est un problème récurrent dans les réseaux sans fil et d'autant plus prééminent désormais avec la généralisation des solutions de voix sur IP (VoIP). Ce trafic, certes de débits faibles, exige cependant une faible latence et une jigue contrôlée, de manière à assurer une restitution correcte de la voix humaine. Plusieurs publications ont analysé les caractéristiques de ce type de flux, et ont permis de quantifier les besoins des communications temps réel en voix sur IP. La mobilité dans les réseaux 802.11 vient tout juste d'être ajoutée dans l'amendement 802.11r, en brouillon depuis 2004 et ratifié par l'IEEE en Août 2008.

6.7.1 Principes des points d'accès virtuels

Les réseaux 802.11 représentent donc un défi concernant la mobilité. D'une part, les communications entre les points d'accès d'une même zone de service (BSS) doivent être en mesure de communiquer et d'échanger des informations entre eux. IAPP a été la première proposition de protocole de gestion de la mobilité dans IEEE 802.11 concernant la communication entre point d'accès, mais qui est désormais abandonnée et laissée donc au libre choix des constructeurs de point d'accès. Cependant, le problème principal reste la lourdeur protocolaire de 802.11, qui, lors d'un déplacement, se traduit par trois étapes : recherche d'un nouveau point d'accès, authentification puis association. Ces deux contraintes associées (lourdeur protocolaire + échange d'information entre les points d'accès) sont un véritable frein à une mobilité réelle des utilisateurs au sein d'un réseau tout sans fil.

Une première remarque qui découle de l'examen des propositions en matière de mobilité dans les réseaux sans fil de type 802.11, c'est le choix de positionner la décision de mobilité au sein du terminal lui-même. Ce choix paraît pertinent dans le sens où le terminal est le plus à même d'identifier les réseaux disponibles à proximité. Cependant, le but ultime d'un réseau supportant la mobilité est d'assurer le transport des données issues du terminal mobile.

Dans cette optique, l'intérêt réside dans la continuité de la connectivité. Or, dans les faits, seuls les points d'accès d'une même zone contribuent au support de la connectivité. Il est donc plus logique de placer la décision de la mobilité au sein du coeur de réseau. Pour cela, nous allons prendre en compte les caractéristiques intrinsèques des ondes radio, à savoir leur diffusion : un client transmettant des données vers le point d'accès auquel est associé envoie également des données tout autour de lui, et notamment aux autres points d'accès. L'idée étant de profiter de la réception par les points d'accès du trafic radio des clients pour anticiper les déplacements de mobilité.

Deux problèmes apparaissent donc :

- il faut d'une part élaborer une technique permettant d'évaluer le déplacement du terminal ;

- d'autre part il faut faire croire au terminal que la connectivité est assurée en permanence.

Pour le premier problème lié à l'évaluation du déplacement du terminal, il s'agit avant tout de maintenir un niveau de signal suffisant pour assurer la bonne réception des paquets. La solution qui s'impose donc est donc une surveillance du niveau de signal par les points d'accès, ainsi qu'une politique pour agir en cas de forte diminution ou augmentation. Dès lors, l'évolution physique du terminal peut permettre la génération d'évènements.

L'autre problème est plus épineux : les terminaux 802.11 standard utilisent deux éléments pour évaluer leur connectivité : d'une part, l'acquittement de tout paquet radio transmis est utilisé pour déterminer si la connexion est toujours active, et sinon, en cas d'absence de trafic, les terminaux utilisent les balises du point d'accès pour

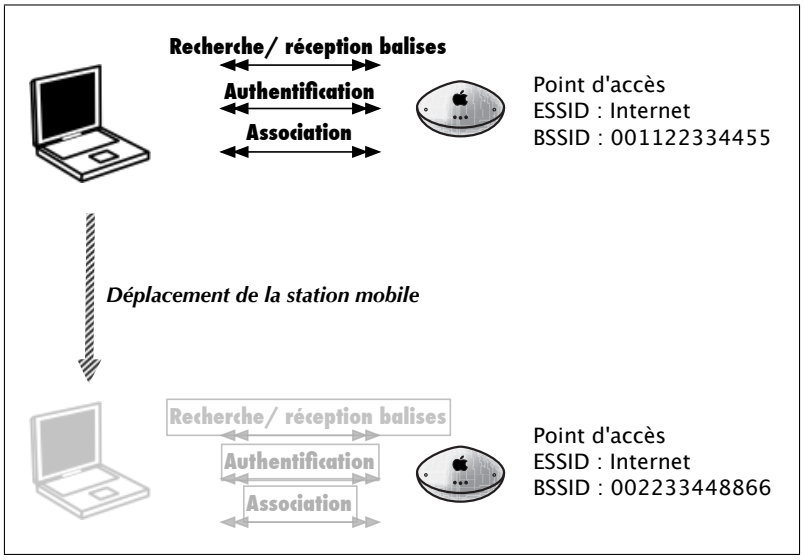


Figure 6.11 – Mobilité dans les réseaux 802.11 actuels.

évaluer la disponibilité du réseau. Si l'on arrive à assurer la présence continuelle de ces deux éléments au niveau d'une station mobile, sa connectivité sera donc assurée de son point de vue.

C'est à ce niveau que l'outil PACMAP va nous être utile. Grâce aux capacités de manipulation du protocole 802.11 incluses, nous pouvons tout à fait simuler la présence d'un point d'accès dès lors que la station est à portée radio. Dès lors, du point de vue de la station, il n'y a plus de déplacement : c'est le concept de points d'accès virtuels ("Virtual AP").

Nous illustrons ce concept dans les schémas suivant, qui illustrent les techniques de mobilité actuelles (figure 6.11) et les techniques basées sur des points d'accès virtuels (figure 6.12).

Une technique proche de celle-ci est d'ores et déjà utilisée par les points d'accès actuels, qui simulent des réseaux multiples en émettant plusieurs types de balises sur un même canal. Cependant, ces techniques sont limitées et chaque point d'accès virtuel constitue une unité indépendante. Ici, en utilisant PACMAP, nous pouvons récupérer les informations d'un point d'accès virtuel et les transférer d'un point d'accès réel à l'autre.

L'utilisation de points d'accès virtuels permet d'éviter toute perte de connectivité au niveau du noeud mobile, ce qui facilite d'une part le support de la mobilité dans des applications critiques (téléphonie sur IP par exemple), puisque celle-ci est alors gérée par le réseau lui-même. De plus, le contrôle de la connectivité est restreint au coeur de réseau, ce qui facilite la gestion dans le cadre d'un réseau purement sans fil.

En outre, l'utilisation de points d'accès virtuels permet d'associer aux déplace-

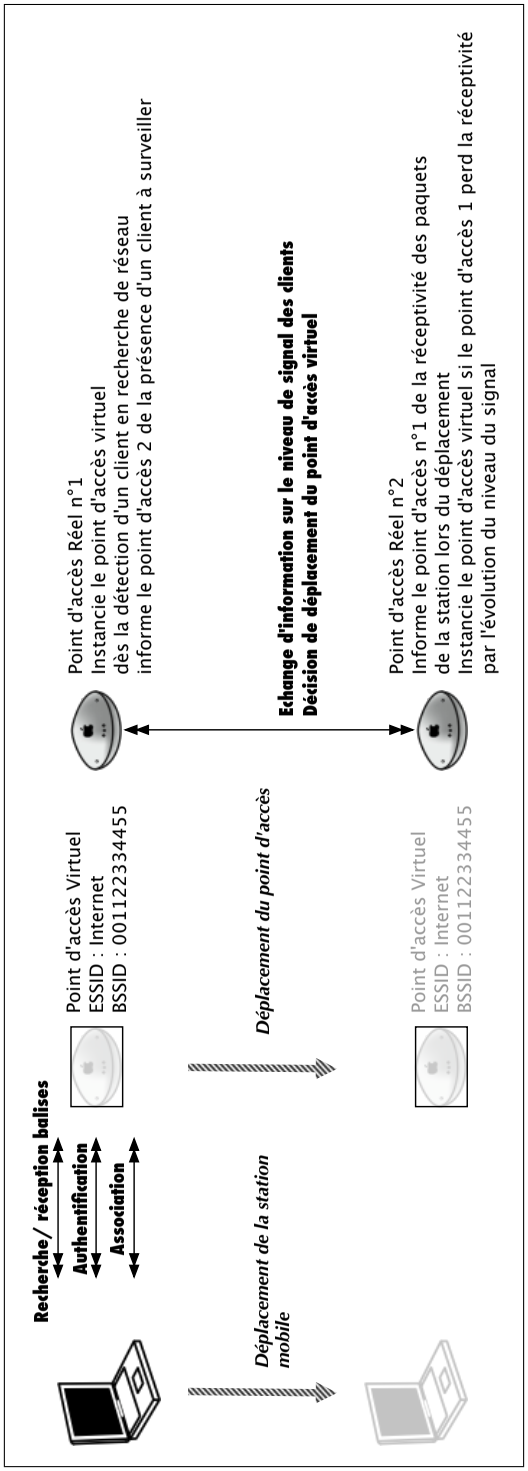


Figure 6.12 – Mobilité avec l'utilisation de point d'accès virtuels.

ments du point d'attache du mobile l'intégralité de ses paramètres : à savoir information sur le routage, caches éventuels... Nous obtenons donc une optimisation intercouches de manière très simple dans cet exemple précis.

6.7.2 Réalisation avec PACMAP

Nous nous proposons d'évaluer la faisabilité du concept d'AP virtuels en utilisant PACMAP, en utilisant le langage Python. Nous disposons pour cela de la base de code des exemples précédents, qui assure la gestion des terminaux IEEE 802.11, ainsi que les problèmes liés à l'adressage et à l'auto-configuration des clients traditionnels. Nous allons simplement compléter les exemples précédents pour implémenter le mécanisme de mobilité transparente.

Selon le principe de fonctionnement des AP virtuels, un client qui cherche à se connecter doit se voir offrir un point d'accès dédié. La première étape consiste à intercepter les requêtes de sondage des clients 802.11 pour envoyer une réponse spécifique :

```
1  def proto80211_probereq(packet, length):
2      # Réception d'une trame de découverte des points d'accès
3      dot11_frame = Packet(packet)
4      dot11_frame.decode_payload_as(Dot11)
5      # Récupération de l'adresse du client
6       client  = dot11_frame.getlayer(Dot11).addr2
7      # Génération d'un nom de réseau de la forme "Client-XX:XX:XX:XX:XX:XX" où
8      # XX:XX:XX:XX:XX:XX est l'adresse matérielle unique du client
9      ssid  = "Client-%s" % client
10     current_timestamp = time.mktime(datetime.datetime.now().timetuple())*1e6+datetime.
           datetime.now().microsecond
11     # Préparation d'un paquet de réponse
12     dot11_answer = Dot11(
13     type = "Management",
14     addr1 = dot11_frame.getlayer(Dot11).addr2,
15     addr2 = bssid,
16     addr3 = bssid)/Dot11ProbeResp(timestamp=current_timestamp,cap = 0x0104)/Dot11Elt(
           ID=0,info=ssid)/Dot11Elt(ID=1,info="\x82")/Dot11Elt(ID=3,info="\x06")
17     # Envoi de la réponse
18     pacmap.sendpacket(str(dot11_answer),len(str(dot11_answer)),1)
19     return 1
```

Le client perçoit donc la présence d'un point d'accès personnalité, dont le nom est de la forme "$Client - 00:11:22:33:44:55$" où $00:11:22:33:44:55$ est l'adresse matérielle du client.

Le client cherche ensuite à se connecter suivant la procédure décrite précédemment en figure 6.5. Afin de gérer les enregistrements, nous utilisons un objet *Client* et un objet *ClientList*.

```
1  class Client:
2      def __init__(self):
```

```
3           self.bssid = bssid # adresse du point d'accès associé
4           self.mac = mac # adresse MAC
5           self.timestamp = # tampon indiquant la dernière communication
6           self.sequence = 0 # numéro de séquence pour suivi paquet par paquet
7           self.rssi  = 0 # niveau de signal du dernier paquet reçu
8
9   class ClientList:
10      def __init__(self):
11          self.data = [] # tableau d'objet Client
12          self.mac = [] # tableau servant pour l'ajout/suppression par adresse mac
13          self.index = -1 # index de l'interface virtuelle associée
```

Dès qu'un client est associé, on crée un enregistrement de type Client dans une liste de type ClientList. A partir de là, la fonction *periodic*() envoie une balise à intervalles réguliers, indiquant que le point d'accès est toujours présent, ce qui permet au client de savoir que le point d'accès est toujours visible.

Nous profitons de cette balise pour y insérer le niveau de signal du dernier paquet reçu, que nous stockons à chaque arrivée d'un paquet :

```
1   def proto80211_data(packet, length):
2       # ../..
3       client.setTimestamp(timestamp)
4       client.setLastRSSI(pacmap.getrxinfo(1))
5       # ../..
6
7
8   def periodic(float1, float2, float3):
9       # ../..
10      # pour chaque client associé :
11      for client in clients_list :
12      # on extrait le dernier niveau de signal reçu
13      rssi = "%d" % client.getLastRSSI()
14      # on l'introduit en paramètre supplémentaire de la balise
15      dot11_answer = Dot11(...)/Dot11Elt(ID=221,info=rssi)
16      # ../..
17      # on procède à l'envoi
18      pacmap.sendpacket(str(dot11_answer),len(str(dot11_answer)),1)
```

Désormais, nous disposons d'un point d'accès qui pour chaque client diffuse dans la balise associée le niveau du signal reçu.

Il ne reste plus qu'à traiter le point de vue d'un point d'accès secondaire, situé sur le même canal radio : celui-ci reçoit les balises et doit les traiter. Nous maintenons deux listes par point d'accès : une liste des clients, *clients_list*, et une liste des clients à surveiller, *watch_list* :

```
1   def proto80211_beacon(packet, length, essid):
2       # ../..
3       # si on détecte une balise pour un client dédié
4       if (essid.startswith("Client-")):
```

```
5       # alors on l'ajoute à la liste de surveillance
6           watch_list.add(clientmac)
7       # ../..
8   return 0
```

Il suffit alors de surveiller le dernier niveau de signal reçu avec le niveau de signal rapporté au niveau du point d'accès initial :

```
1   # Seuil de migration en dBm
2   floor = 15
3
4   def proto80211_beacon(packet, length, essid):
5       # ../..
6       # si le client surveillé à un niveau de signal supérieur de plus de 15 dBm à celui
            rapporté par son point d'accès d'origine, la décision de migration est prise
7       if (watched_client.getLastRSSI()-lastRSSI) > floor:
8           clients_list .add(client)
9       # ../..
10  return 0
```

La migration est un processus sans signalisation particulière : à partir du moment où le nouveau point d'accès enregistre un nouveau client, il émule à son tour le point d'accès par émission de balises. Le point d'accès initial en est donc averti par les balises, et peut donc détruire l'association :

```
1   # Seuil de migration en dBm
2   floor = 15
3
4   def proto80211_beacon(packet, length, essid):
5       # ../..
6       # on vérifier si la balise correspond à un de nos clients associés
7       client = clients_list .getClient(clientmac)
8       # si c'est le cas, cela signifie qu'un autre point d'accès prend en charge ce client, on
            peut donc l'abandonner
9           if client :
10              clients_list .delete(clientmac)
11      # ../..
12  return 0
```

A ce stade, nous disposons d'une architecture de mobilité fonctionnelle en quelques lignes de code Python. Nous complétons ce code par les exemples précédents liés à l'adressage : nous incorporons une gestion des trames DHCP, ainsi qu'une gestion des trames ARP. Ainsi, toute requête ARP sera interceptée et fera l'objet d'une réponse en fonction de la situation de la station concernée. L'ensemble du code complet et fonctionnel est présenté en annexe E.

6.7.3 Tests de mobilité

La mise en oeuvre de notre prototype nous a conduit à considérer un certain nombre de paramètres ajustables pour la gestion de mobilité. Ces paramètres sont principalement liés à la réactivité de notre solution, évaluée dans notre laboratoire pour deux points d'accès équipés de PACMAP et du script Virtual AP, selon la figure.

Le premier paramètre est le seuil à partir duquel il est plus intéressant de faire transiter le trafic sur un point d'accès plutôt qu'un autre. Dans notre approche, le seuil est empirement déterminé à partir des seuils de réception obtenus par le placement des points d'accès vis à vis du client. Mais ce seuil pourrait être complété par d'autres éléments : la charge du point d'accès, le niveau de trafic ambiant par exemple.

Dans notre cas, nous avons déterminé empiriquement pour la situation que le seuil de signal assurant une bonne couverture permettant une absence de perte de paquet était une différence de 15 dB dans le niveau de signal reçu : dès que le niveau de signal des paquets reçus par un AP dépasse celui d'un autre de ce seuil, nous amorçons l'enregistrement du terminal sur le nouvel AP et le désenregistrement sur l'ancien.

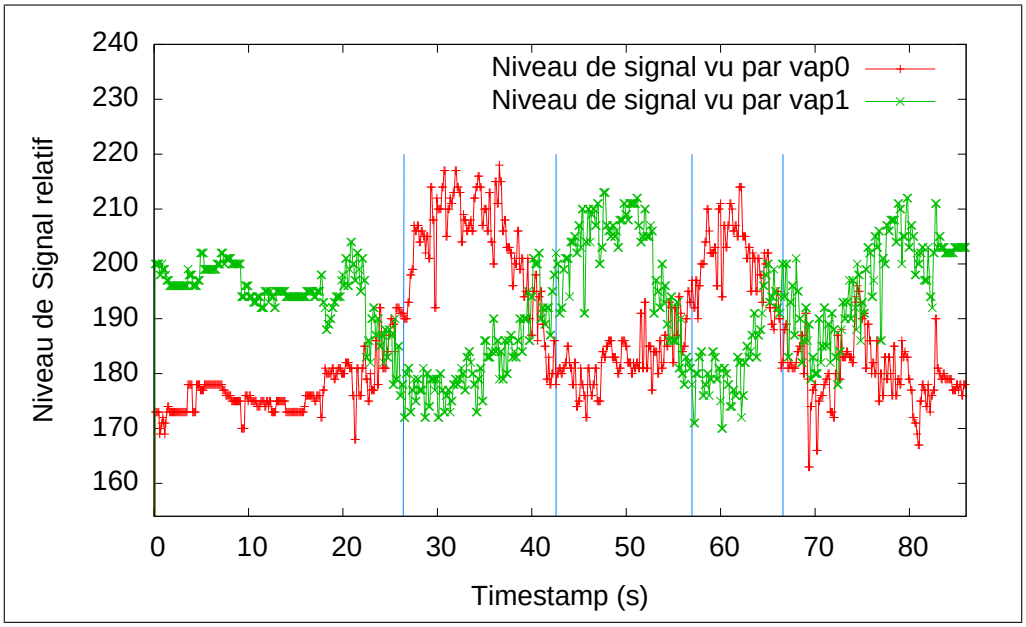


Figure 6.13 – Evolution du niveau de signal du client vu par les deux points d'accès et décisions de mobilité.

Un exemple de l'évolution du signal reçu par deux points d'accès exécutant PACMAP et Virtual AP est présenté figure 6.13. Dans une première phase, le mobile est statique, et son trafic est géré par le point d'accès vap1. Puis, il y a déplacement, le niveau de signal est beaucoup plus fluctuant, et ceci, au niveau des deux points d'accès. Une fois la valeur de seuil atteinte, le trafic du client est écoulé par vap0. Le client se

déplaçant entre les deux points d'accès, il y a 3 autres décisions de mobilité qui sont prises, de tel sorte que pour le terminal client est toujours associé au point d'accès ayant la meilleure réception.

De plus, nous observons que la valeur du signal est corrélée entre les deux mesures (sur le point d'accès actuel et sur le point d'accès distant) : l'utilisation de la différence entre les deux mesures permet donc de profiter de cette corrélation pour ne conserver que la différence de signaux, et non les fluctuations qui touchent le terminal mobile. Ainsi, nous obtenons un bon compromis entre la stabilité de la décision et la garantie d'assurer la connectivité.

Le désenregistrement est également un paramètre de réactivité de notre solution. En effet, le désenregistrement qui correspond à l'abandon par un point d'accès de la gestion des paquets d'un client, est fonction de la réception des balises émises par le point d'accès qui prend désormais la gestion des paquets de ce client.

Dans notre cas, ces balises sont envoyées au rythme d'une balise toutes les 100ms, ce qui correspond à notre temps de réaction pour mesurer le niveau du signal. Cependant dans la pratique, il est possible qu'une balise soit transmise par l'ancien point d'accès alors que la migration vient d'avoir lieu. Cela est imputable au temps de traitement des balises par PACMAP ; afin de s'assurer que la migration a bien eu lieu, nous introduisons un paramètre *delta* qui correspond au nombre de balises issues du nouveau point d'accès. Dans la pratique, 2 balises consécutives suffisent pour éviter tout désenregistrement intempestif. Le problème et la solution retenue sont présentés sur la figure 6.14 : l'arrêt des balises correspond au désenregistrement effectif du client grâce à l'utilisation de l'intervalle *delta*.

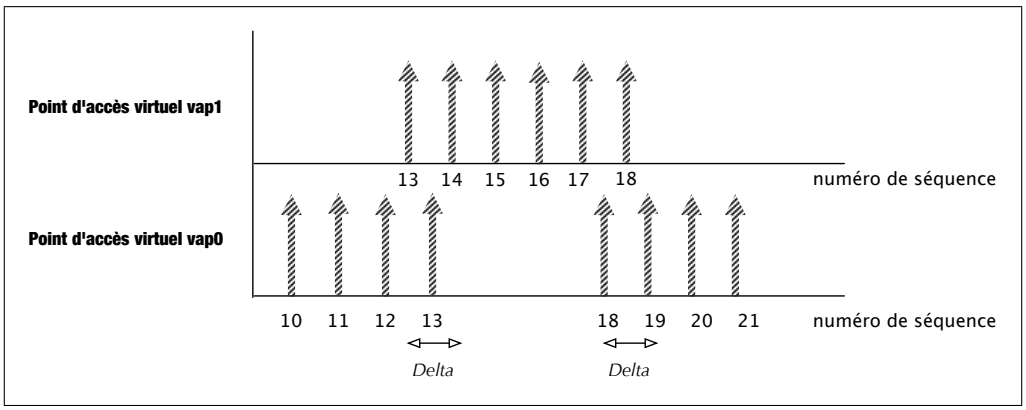


Figure 6.14 – Emission des balises lors d'une détection de mobilité.

En terme d'usage, nous avons effectué des tests de trafic bidirectionnel de taille faible et de fréquence régulière, simulant ainsi un trafic VoIP.

La plate-forme se réduit à deux ordinateurs équipés de carte sans fil et exécutant PACMAP et le script Virtual AP ; les interfaces Ethernet créées par PACMAP sont

ensuite reliées par pont Ethernet l'une à l'autre. Ainsi nous reproduisons le schéma classique d'un réseau doté de plusieurs points d'accès.

Nous utilisons un *ping* de 64 octets toutes les 200ms sur un client se déplaçant entre 2 points d'accès utilisant Virtual AP et nous étudions l'évolution de la latence en fonction de la migration du terminal. Le *ping* est lancé depuis le point d'accès initial.

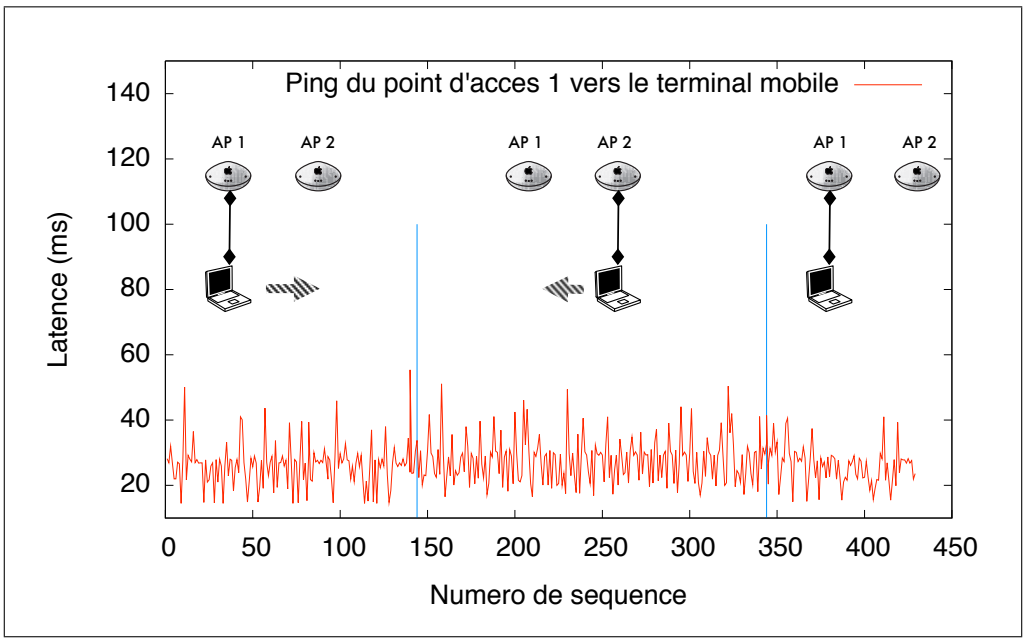


Figure 6.15 – Evolution de la latence en fonction de l'état de mobilité du terminal (version Python).

La figure 6.15 présente ainsi l'évolution de la latence. Les instants où le terminal migre d'un point d'accès à un autre sont indiqués par les traits verticaux, entre les séquences $seq = 144$ et $seq = 344$ où la station est connectée sur le point d'accès distant. Comme nous pouvons nous en rendre compte, les effets du pont Ethernet sont visibles au niveau de la latence moyenne, qui est naturellement plus importante quand le terminal est distant. Il faut noter que le pont réseau est une cause de soucis, car celui-ci maintient des tables ARP indiquant l'association d'une adresse Ethernet et d'une interface. Lors d'un déplacement géré par Virtual AP, la transition est trop rapide pour le délai d'expiration habituel des tables ARP. Il a donc fallu vider ces tables manuellement (le script de Virtual AP comporte donc un appel externe à la table ARP).

Mis à part les effets du pont réseau, nous constatons qu'il n'y a pas de réelle augmentation de la latence lors des phases de transition : nous atteignons ainsi l'objectif fixé par Virtual AP, à savoir une mobilité transparente pour les terminaux mobiles.

Les niveaux de latence relevés et leurs fluctuations en régime continu sont impu-

tables à l'utilisation du code Scapy dans le script Virtual AP. Néanmoins, les résultats obtenus précédemment lors de l'examen comparé d'une implémentation d'une machine à état 802.11 en langage natif vis à vis de son équivalent en Python semblent indiquer qu'une implémentation native du concept de Virtual AP, plus difficile à réaliser, bénéficierait cependant d'un gain probable au niveau de la latence moyenne observée.

Enfin de vérifier cette assertion, nous avons implémenté Virtual AP en langage natif en utilisant l'implémentation en Python comme canevas. Le code représente 1000 lignes contrairement à l'implémentation Python qui représente seulement 250 lignes de codes, grâce à l'utilisation de la bibliothèque Scapy. Nous effectuons sur ce prototype les mêmes mesures que précédemment, dont les résultats sont présentés sur la figure 6.16.

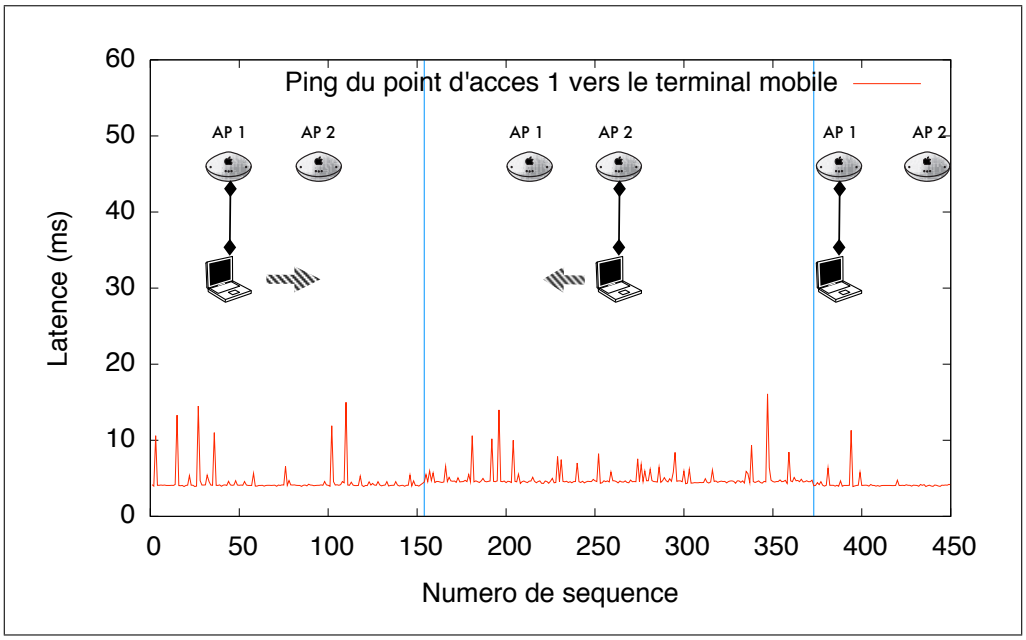


Figure 6.16 – Evolution de la latence en fonction de l'état de mobilité du terminal (version native).

Nous observons que la latence a très fortement diminué grâce à l'utilisation du code natif. Les perturbations observées sont alors uniquement dues à l'accès au médium, mais nous observons les mêmes propriétés de conservation de la connectivité et d'absence d'augmentation de la latence durant les phases de mobilité. Le coût du pont réseau filaire entre les deux points d'accès est dorénavant identifiable (ajout de 0.4 ms sur la latence moyenne observée de 4ms).

6.7.4 Discussion sur Virtual AP

La solution présentée est victime d'un inconvénient majeur, la multiplication des balises en fonction du nombre de clients mobiles à gérer. En effet, chaque nouveau client engendre la création d'un point d'accès virtuel, ce qui rajoute une signalisation périodique supplémentaire.

Il est possible de limiter l'effet néfaste de cette signalisation en réduisant la fréquence d'émission des balises, plus exactement en la rendant inversement proportionnelle au nombre de clients associés. Dès lors, le nombre de balises ne dépasse pas celui d'un point d'accès traditionnel. Cependant cette solution présente le désagrément de réduire l'information obtenue sur la qualité du signal distant ; la réactivité pour la mobilité s'en trouve forcément réduite.

Cependant, les pistes d'améliorations sont nombreuses : les mécanismes de piggybacking, comme par exemple l'utilisation des acquittements émis par le point d'accès destinataire. Ceux-ci peuvent être complétés pour ajouter une information supplémentaire concernant le niveau de signal, permettant ainsi la réception passive d'informations entre les points d'accès composant le réseau sans fil. Une autre piste pourrait être l'utilisation de la géolocalisation, et de la prédiction de mouvement : ainsi le déclencheur ne serait pas le niveau de signal, mais la position estimée du terminal.

Les inconvénients de Virtual AP sous forme actuelle peuvent donc être facilement contournés, ce qui en fait une solution intéressante et une bonne illustration des capacités de l'outil PACMAP pour la création de nouvelles architectures.

6.8 Conclusion

Dans ce chapitre, nous avons présenté PACMAP, un outil conçu pour la manipulation de paquets en espace utilisateur, et plus généralement, au prototypage de solutions pour les réseaux sans fil de nouvelle génération sur du matériel sans fil générique. En isolant les problèmes liés aux aspects temps réel du traitement de paquets, nous avons pu simplement nous affranchir des contraintes habituelles des architectures actuelles qui sont trop liées au système d'exploitation hôte et aux pilotes du périphérique utilisé, et nous concentrer sur les fonctionnalités offertes par l'architecture.

En éliminant ce lien, nous espérons ouvrir la voie à de nouvelles architectures plus indépendantes du système d'exploitation, et par la même, plus flexibles pour des approches novatrices aux problèmes usuels posés par les réseaux sans fil. Au détour de l'exemple de la mobilité, nous avons pu proposer Virtual AP, une solution originale au problème de la mobilité transparente, en couplant la mobilité du terminal avec une mobilité de la signalisation qui lui est attachée.

Cependant, comme nous l'avons vu sur le comportement d'un simple point d'accès, les performances d'une architecture comme PACMAP sont plus que jamais liées à la qualité de l'implémentation. L'utilisation d'un langage de script, Python, et de

bibliothèques ont considérablement réduit la durée de développement du prototype, au détriment des performances pures. C'est pourquoi la dualité de l'outil PACMAP, qui permet une implémentation en langage scripté (Python) ou en langage compilé (C) nous semble plus que jamais pertinente.

Au delà de la simple réalisation de prototype, PACMAP nous permet de reconsidérer surtout notre vision traditionnelle du réseau, trop monolithique, reposant trop sur le modèle en couches. En recréant une pile réseau en espace utilisateur, et en nous donnant la possibilité de voir et de manipuler une pile réseau comme une instance reprogrammable dynamiquement, c'est un éventail de possibilités qui s'ouvrent désormais. Par exemple, chaque client peut invoquer une pile 802.11 disposant des caractéristiques qu'il désire, en teme de modulation ou de chiffrement. Mais allons plus loin : chaque application peut désormais disposer d'une pile réseau adaptée à ses besoins, et ainsi obtenir la vision du réseau qui l'intéresse. Par exemple, dans Virtual AP, les applications tournant sur les stations mobiles percoivent désormais le réseau comme un élément fixe, et non plus mobile.

Dès lors, cette nouvelle approche architecturale atteint l'un des objectifs et l'un du succès du modèle en couches : la capacité d'abstraction. Au delà des couches du bas niveau, nous pouvons désormais disposer d'une réelle abstraction architecturale.

CHAPITRE 7

CONCLUSIONS & PERSPECTIVES

L'objectif initial de cette thèse était d'étudier les architectures spontanées pour les réseaux sans fil de nouvelle génération. Suite à notre autopsie des réseaux sans fil actuels fonctionnant suivant la norme IEEE 802.11, il est rapidement apparu que ces réseaux souffraient de multiples anomalies, liées d'une part aux choix technologiques mis en oeuvre pour leur fonctionnement (gestion de plusieurs modulations physiques, choix de la méthode d'accès CSMA/CA) et d'autre part aux choix architecturaux effectués (fonctionnement du protocole 802.11, modes d'utilisation retenus). L'origine de ces limitations est rapidement apparue comme un héritage de l'histoire de la conception des réseaux modernes, basés sur une structuration en couches et une implémentation indépendante de celles-ci au sein des systèmes d'exploitation modernes.

A partir de l'observation de l'ensemble des contributions, nous avons pu identifier que les pistes d'optimisation retenues reposaient principalement sur des violations locales de l'architecture en couches. A partir de là, nous avons évalué les propositions plus génériques basées sur des architectures inter-couches. Cependant, ces architectures et les éventuels prototypes expérimentaux qui en ont résulté souffrent d'une conception reposant sur des modifications et des interactions du modèle actuel de la pile réseau telle qu'elle existe dans les systèmes d'exploitation actuels.

Afin de dépasser ce modèle actuel, nous nous sommes penchés sur les contributions d'un autre domaine de la recherche en informatique, la recherche en sécurité, pour nous apercevoir que les outils développés dans cette discipline offraient une réelle alternative au modèle intégré de la couche réseau, et facilitaient de ce fait l'implémentation d'une réelle vision inter-couches.

Dès lors, nous nous sommes attachés à définir une nouvelle approche pour l'architecture réseau, reposant toujours sur les matériels actuellement disponibles, mais offrant une approche inédite, basée sur une séparation temporelle de la couche réseau respectant la temporalité des opérations mises en oeuvre dans son fonctionne-

ment. Ainsi, nous avons proposé une segmentation avec d'une part un traitement lié au temps réel, correspondant aux phénomènes physiques et à la méthode d'accès, et d'autre part un traitement à l'échelle du paquet, plus adapté à l'approche protocolaire utilisée dans les réseaux.

Afin de valider cette nouvelle approche, nous avons contribué sur deux aspects distincts.

D'une part, nous avons étudié la capacité du matériel actuel à permettre des modifications de ses composantes temps réel. Pour cela, nous avons étudié l'implémentation d'une méthode d'accès développée précédemment sur simulateur, Idle Sense, et valider celle-ci au travers d'une vérification expérimentale des performances attendues théoriquement. Cette implémentation a permis de vérifier que les matériels actuellement disponibles sont capables d'être dynamiquement adaptables à de nouveaux usages.

D'autre part, nous avons concentré nos efforts sur le développement d'un outil de manipulation de paquets, PACMAP, destiné à permettre la concrétisation d'une nouvelle approche du fonctionnement des réseaux. Nous espérons que cette nouvelle approche, basée sur un traitement des paquets en espace utilisateur par des outils en langages natifs et en langages interprétés, permettra un développement de nouvelles réponses aux défis posés par les nouveaux usages offerts par la connectivité sans fil. Pour valider cette hypothèse, nous avons utilisé cet outil avec succès non seulement pour reconstituer l'architecture des réseaux actuels à la norme IEEE 802.11, mais pour proposer une nouvelle solution alternative au problème ponctuel de la mobilité transparente au sein des réseaux sans fil, Virtual AP.

Ces contributions valident donc de manière expérimentale notre proposition d'une séparation temporelle pour la réalisation d'architecture pour les réseaux de nouvelles générations. Pour l'heure, les outils développés et utilisés pour cette validation expérimentale sont intrinsèquement limités par l'isolement des traitements temps réel au sein de l'adaptateur réseau.

Cependant, nous fondons de grands espoirs dans le développement des architectures radio définies logiciellement : ainsi, même si une approche respectant les temporalités sera toujours nécessaire, il sera possible d'interagir au mieux entre les deux domaines temporels, et ainsi atteindre les optimaux théoriques en accord avec les usages souhaités pour ces nouveaux réseaux sans fil.

Déjà, les derniers travaux autour du projet GNU Radio semblent confirmer cette perspective. En effet, outre l'arrivée d'une nouvelle plate-forme matérielle, l'USRP2[1], dont les capacités de traitement ont été considérablement augmentées et rendent possible le traitement logiciel des signaux radio les plus complexes, c'est avant tout la collaboration d'entités historiquement présentes à la génèse d'Internet, comme BBN, qui autrefois avait soutenu et participé aux travaux fondateurs d'Internet[2] (Poursuite

[1] http ://www.gnuradio.org/trac/wiki/USRP2
[2] http ://www.bbn.com/about/timeline/

des idées de J.C.R Licklider dans les années 60, ou encore la première implémentation TCP sous Unix en 1977) et qui aujourd'hui a contribué à fournir une première implémentation fonctionnelle de 802.11 au sein du projet GNU Radio[3].

Les travaux présentés ici ne sont donc qu'un simple pierre préparatoire à l'édifice constitué par les réseaux futurs : néanmoins, nous espérons qu'ils accéléreront l'intégration des futurs développements des radio logicielles au sein des architectures des systèmes d'exploitation, perpétuant ainsi la tradition d'amélioration perpétuelle en vigueur dans le monde du réseau, comme Internet le démontre aujourd'hui.

[3]http ://gnuradio.org/trac/wiki/OtherCode

Troisième partie

Annexes

ANNEXE A

RÉPARTITION DES CANAUX IEEE 802.11

Canal	1	2	3	4	5	6	7
Fréquences	2.412	2.417	2.422	2.427	2.432	2.437	2.442
Canal	8	9	10	11	12	13	14
Fréquences	2.447	2.452	2.457	2.462	2.467	2.472	2.484

Table A.1 – Canaux de transmission 802.11 sur la bande 2.4GHz : répartition des fréquences porteuses.

Canal	34	36	38	40	42	44	46	48
Fréquences	5.170	5.180	5.190	5.200	5.210	5.220	5.230	5.240

Table A.2 – Canaux de transmission 802.11 sur la bande 5GHz : répartition des fréquences porteuses, bande U-II basse.

Canal	52	56	60	64
Fréquences	5.260	5.280	5.300	5.320

Table A.3 – Canaux de transmission 802.11 sur la bande 5GHz : répartition des fréquences porteuses, bande U-II moyenne.

Canal	149	153	157	161
Fréquences	5.745	5.765	5.785	5.805

Table A.4 – Canaux de transmission 802.11 sur la bande 5GHz : répartition des fréquences porteuses, bande U-II haute.

ANNEXE B

FORMAT DE L'EN-TÊTE PLCP

B.1 Famille DSSS

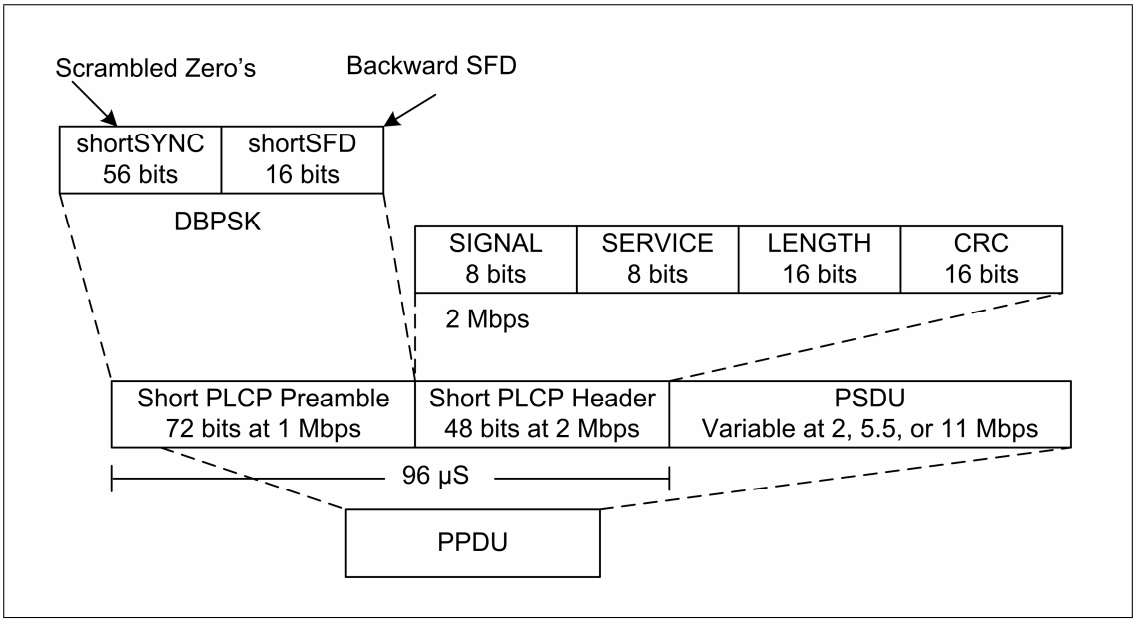


Figure B.1 – Format d'une trame radio 802.11 suivant le standard original 802.11, avec préambule court.

Pour les familles DSSS (voir figure B.1) ou en cas de compatibilité (802.11g en mode dit mixte, voir figure B.2) le préambule est constitué d'une séquence de longueur variable, transmise à 1Mbits/s, constituée d'une séquence de synchronisation (128 bits

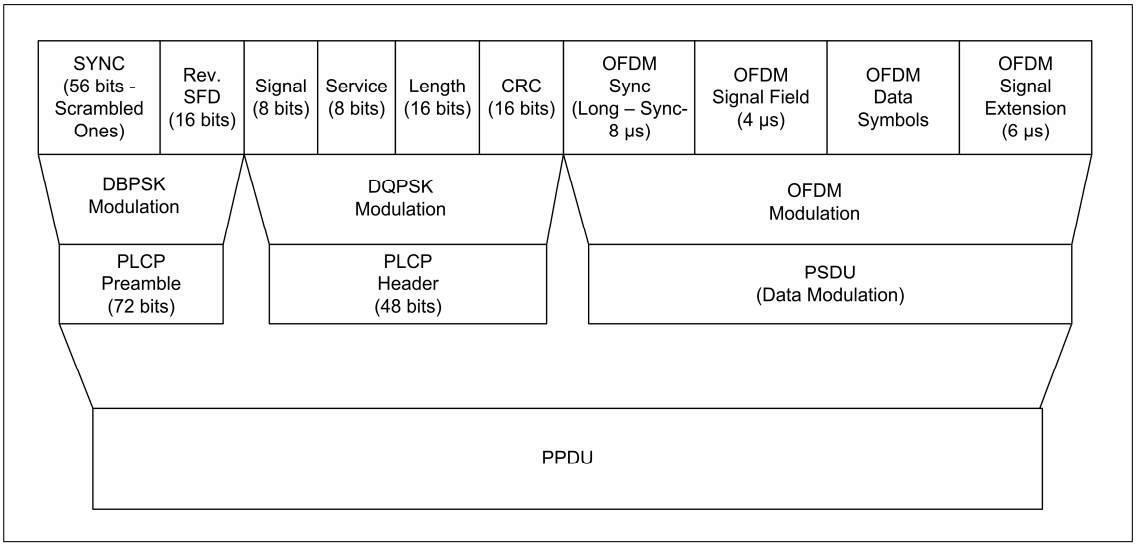


Figure B.2 – Format d'une trame radio 802.11 en mode de compatibilité avec préambule court.

si le préambule est long ou 56 bits si le préambule est court (introduit dans 802.11b)) et d'une marque de début de trame (Start Frame Delimiter) de 16 bits.

Après le préambule, suit immédiatement l'en-tête PLCP, qui lui aussi sera fonction du mode de modulation. Pour le mode DSSS ou le mode de compatibilité de 802.11g, il s'agit une en-tête de 48 bits transmis à une vitesse de 1Mbits (préambule long) ou 2Mbits (préambule court). Il contient 4 champs : signal, service, longueur et HEC (une somme de contrôle pour l'en-tête). Le signal indique la vitesse à laquelle les données qui suivent seront envoyées. Le service est réservé à un usage ultérieur. le HEC est un CRC à 16 bits.

A la fin du paquet transmis sur le canal sont insérées les données à transmettre, d'une longueur et d'une vitesse variable, mais connue des récepteurs grâce à l'en-tête PLCP.

B.2 Famille OFDM

Pour la famille de modulation OFDM (802.11a et 802.11g en mode pur), le préambule consiste en la transmission de 12 symboles permettant l'établissement du contrôle de gain (voir figure B.3). La durée de cette opération pour les modes OFDM est de 16 μs. Après le préambule, suit immédiatement l'en-tête PLCP, qui lui aussi sera fonction du mode de modulation. Pour les familles OFDM, l'en-tête est constitué d'un symbole envoyé à la vitesse minimale, soit 6MBits/s, qui véhicule un champ rate de 4 bits, un champ réservé de 1 bit, un champ longueur de 12 bits, un champ de parité d'1 bit, et un champ de fin de 6 bits, soit 24 bits au total. Le champ rate indique ici aussi la

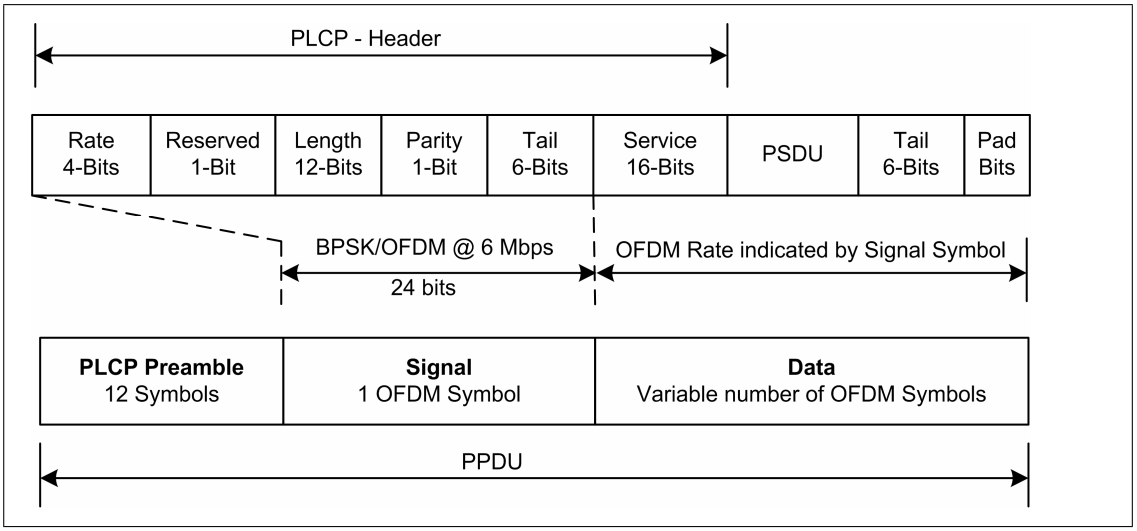


Figure B.3 – Format d'une trame radio 802.11 suivant le standard 802.11a ou 802.11g pur.

vitesse des données qui vont suivre.

A la fin du paquet transmis sur le canal sont insérées les données à transmettre, d'une longueur et d'une vitesse variable, mais connue des récepteurs grâce à l'en-tête PLCP. Pour les modulations OFDM, un champ service de 16 bits est ajouté aux données à transmettre, ainsi d'un champ de fin de 6 bits, et de bits de padding pour completer un symbole partiellement utilisé. A noter qu'en cas d'utilisation du mode de compatitbilité de 802.11g avec une modulation OFDM, la fin du paquet est constituée d'une en-tête OFDM, avec une synchronisation de 8 μs, d'un champ signal de 4 μs, de symboles constituant les données à transmettre, et d'un signal d'extension de 6 μs.

ANNEXE C

FORMAT DE L'EN-TÊTE IEEE 802.11

La table C.1 explicite les différents champs constitutifs de l'en-tête.

FC (2)	D/ID (2)	Adresse 1 (4 octets)	Adresse 2 (4 octets)
Adresse 3 (4 octets)	SC (2)	Adresse 4 (4 octets)	Corps de la trame
Corps de la trame (0 à 2312 octets)	FCS (2)		

Table C.1 – Entête des paquets suivant le protocole 802.11

Le champs FC - Frame Control ou contrôle de trame explicite la nature de la trame (voir tableau C.2). Il explicite :

Version du protocole (2bits)	Type (2bits)	Sous-type (4 bits)	To DS (1 bit)
From DS (1 bit)	More Frag (1 bit)	Retry (1 bit)	Power Mgt (1 bit)
More data (1 bit)	WEP (1 bit)	Order (1 bit)	

Table C.2 – Champ Frame Control - FC de la trame 802.11.

- la version du protocole (2 bits), dont la valeur est 0 pour la première version du protocole ;

- le type et sous-type, de 2 et 4 bits, qui définissent le type et le sous-type de trame (voir tableau récapitulatif C.3). Le type gestion correspond au demande d'association. Le type contrôle correspond à l'accès au médium. Le type donnée constitue le transport des données en lui-même (IP ou autre).

- To DS : ce bit vaut 1 quand la trame est destinée à un système de distribution, c'est à dire à un ensemble de point d'accès (Access Point - AP). Toute trame destinée à un AP a ce champ positionné à 1.

- From DS : ce bit vaut 1 quand la trame provient d'un système de distribution (AP).

Type	Description	Sous-type	Description
00	Gestion	0000	Requête d'association
00	Gestion	0001	Réponse d'association
00	Gestion	0010	Requête de réassociation
00	Gestion	0011	Réponse d'association
00	Gestion	0100	Requête d'enquête
00	Gestion	0101	Réponse d'enquête
00	Gestion	1000	Balise (Beacon)
00	Gestion	1001	Annonce de trafic (ATIM)
00	Gestion	1010	Désassociation
00	Gestion	1011	Authentification
00	Gestion	1100	Désauthentification
01	Contrôle	1010	Economie d'énergie (mode d'interrogation)
01	Contrôle	1011	Request To Send (RTS)
01	Contrôle	1100	Clear To Send (CTS)
01	Contrôle	1101	Acquittement (ACK)
01	Contrôle	1110	Fin de la période d'absence de contention (CF-Free)
01	Contrôle	1111	CF-Free + ACK
10	Données	0000	Données
10	Données	0001	Données + CF-Ack
10	Données	0010	Données + CF-Poll
10	Données	0011	Données + CF-Ack+CF-Pool
10	Données	0100	Pas de données (NULL)
10	Données	0101	CF-Ack
10	Données	0110	CF-Poll
10	Données	0111	CF-Ack+CF-Poll

Table C.3 – Champ Type et Sous type - Nature de la trame 802.11.

- Si les deux champs précédents sont à 0, la trame est envoyée d'une station à une autre.

- More Fragments : permet d'indiquer la présence de fragments d'une même transmission. Le suivi est assuré par le champ SC (Sequence Control, que nous détaillons juste après).

- Retry : indique qu'il s'agit d'une retransmission d'un paquet vraisemblablement perdu

- Power Management indique que la station ayant envoyé le paquet est en mode de gestion d'énergie. Cela impacte la mise en cache des paquets à destination de cette machine, le cas échéant.

- More Data : Indique à une station vraisemblablement en mode d'énergie que du trafic est toujours en attente pour elle.

- WEP : indique que le corps de la trame est chiffré au standard WEP.

- Order : indique que la trame a été envoyée suivant une classe de service ordonnée.

Les champs supplémentaires sont Durée/ID, qui indiquent la durée d'utilisation du canal de transmission en microsecondes. Les champs adresses contiennent des adresses de 48 bits indiquant les émetteurs/destinataires (la nature des champs dépend du mode de 802.11 en cours d'utilisation). Enfin, le contrôle de séquence CS permet de réordonner les fragments.

Enfin, une somme de contrôle permet de vérifier l'intégrité d'une trame.

ANNEXE D

CALCULS DE DÉBITS IEEE 802.11

PHY (Physique)	Appellation		802.11	802.11	802.11b	802.11b
	Technique de modulation		DSSS	DSSS	ERP-DSSS	ERP-DSSS
	Modulation I/Q		BPSK	QPSK	CCK	CCK
	Nombre de bit encodés		1	2	4	8
	(réservé aux modulations DSSS)					
	Nombre de bit par chip		1	2	4	8
	Nombre de bit de chipping		11	11	8	8
	Taux de Mchips/sec		11	11	11	11
	Débit théorique (Mbits/s)		**1**	**2**	**5,5**	**11**
PLCP	Long (microsecs)		192			
	Court (microsecs)		96			
MAC	Slot (microsecs)		20			
	SIFS (microsecs)		10			
	DIFS (microsecs)		50			
	Cwmin (slots)		31			
	Cwmax (slots)		1023			
802.11	Normal (octets)		28			
	ACK (octets)		14,00			
	Vitesse ACK (Mbits/s)		1,00	2,00	5,50	11,00
Charge (octets)	IP		20			
	UDP		8			
	Raw		1472			
	Total		1500			
Débits UDP	Long		0,89961803	1,70124242	3,92938179	6,27901115
	Court		0,90962459	1,74355937	4,16273539	6,8968161
Débits 802.11	Long		**0,9167303**	**1,733603**	**4,0041255**	**6,3984489**
	Court		**0,9269272**	**1,7767249**	**4,2419179**	**7,0280055**

Figure D.1 – Débits pratiques calculés pour la technologie 802.11b.

PHY (Physique	Appellation								
	Technique de modulation	802.11a OFDM	802.11a OFDM	802.11a OFDM	802.11a OFDM	802.11a OFDM	802.11a OFDM	802.11a OFDM	802.11a OFDM
	Modulation I/Q	BPSK	BPSK	QPSK	QPSK	QAM16	QAM16	QAM64	QAM64
	Nombre de bit encodés	1	1	2	2	4	4	6	6
	(réservé aux modulations OFDM)	1	1	2	2	4	4	6	6
	Nombre de bit par porteuse	48	48	48	48	48	48	48	48
	Nombre de porteuses	0,5	0,75	0,5	0,75	0,5	0,75	0,66666667	0,75
	Taux de correction FEC	24	36	48	72	96	144	192	216
	Nombre de bits par symboles	4	4	4	4	4	4	4	4
	Débits théoriques	**6**	**9**	**12**	**18**	**24**	**36**	**48**	**54**
PLCP	(microsec)	20							
MAC	Slot (microsecs)	9							
	SIFS (microsecs)	16							
	DIFS (microsecs)	34							
	Cwmin (slots)	15							
	Cwmax (slots)	1023							
802.11	Normal (octets)	28							
	ACK (octets)	14,00							
	Vitesse ACK (Mbits/s)	6	6,00	6	6	6	24	24	24
Charge (octets	IP	20							
	UDP	8							
	Raw	1472							
	Total	1500							
Débits UDP		**5,3200813**	**7,6747167**	**9,8557679**	**13,768626**	**17,178702**	**23,471155**	**28,2511**	**30,308565**
Débits 802.11		**5,4212785**	**7,8207031**	**10,043242**	**14,030529**	**17,50547**	**23,917617**	**28,788485**	**30,885087**

Figure D.2 – Débits pratiques calculés pour la technologie 802.11a.

PHY (Physique	Appellation								
	Technique de modulation	802.11g OFDM	802.11g OFDM	802.11g OFDM	802.11g OFDM	802.11g OFDM	802.11g OFDM	802.11g OFDM	802.11g OFDM
	Modulation I/Q	BPSK	BPSK	QPSK	QPSK	QAM16	QAM16	QAM64	QAM64
	Nombre de bit encodés	1	1	2	2	4	4	6	6
	(réservé aux modulations OFDM)	1	1	2	2	4	4	6	6
	Nombre de bit par porteuse	48	48	48	48	48	48	48	48
	Nombre de porteuses	0,5	0,75	0,5	0,75	0,5	0,75	0,66666667	0,75
	Taux de correction FEC	24	36	48	72	96	144	192	216
	Nombre de bits par symboles	4	4	4	4	4	4	4	4
	Débits théoriques	**6**	**9**	**12**	**18**	**24**	**36**	**48**	**54**
PLCP	(microsec)	20							
MAC	Slot (microsecs)	9							
	SIFS (microsecs)	16							
	DIFS (microsecs)	34							
	Cwmin (slots)	15							
	Cwmax (slots)	1023							
802.11	Normal (octets)	28							
	ACK (octets)	14,00							
	Vitesse ACK (Mbits/s)	6	6,00	6	6	6	24	24	24
Charge (octets	IP	20							
	UDP	8							
	Raw	1472							
	Total	1500							
Débits UDP		**5,3490802**	**7,7352115**	**9,955756**	**13,964556**	**17,484781**	**24,046285**	**29,088514**	**31,27448**
Débits 802.11		**5,450829**	**7,8823486**	**10,145132**	**14,230186**	**17,817372**	**24,503687**	**29,641828**	**31,869375**

Figure D.3 – Débits pratiques calculés pour la technologie 802.11g.

PHY (Physique)									
	Appellation	802.11bg OFDM BPSK	802.11bg OFDM BPSK	802.11bg OFDM QPSK	802.11bg OFDM QPSK	802.11bg OFDM QAM16	802.11bg OFDM QAM16	802.11bg OFDM QAM64	802.11bg OFDM QAM64
	Modulation I/Q	1	1	2	2	4	4	6	6
		1	1	2	2	4	4	6	6
	(réservé aux modulations OFDM)	48	48	48	48	48	48	48	48
	Nombre de bit par porteuse	0,5	0,75	0,5	0,75	0,5	0,75	0,66666667	0,75
	Nombre de porteuses	24	36	48	72	96	144	192	216
	Taux de correction FEC	4	4	4	4	4	4	4	4
	Débits théoriques	**6**	**9**	**12**	**18**	**24**	**36**	**48**	**54**
PLCP	(microsec) court	210							
	long	114							
MAC	Slot (microsecs)	9							
	SIFS (microsecs)	10							
	DIFS (microsecs)	28							
	Cwmin (slots)	15							
	Cwmax (slots)	1023							
802.11	Normal (octets)	28							
	ACK (octets)	14,00							
	Vitesse ACK (Mbits/s)	6	6,00	6	6	24	24	24	24
Charge (octet:	IP	20							
	UDP	8							
	Raw	1472							
	Total	1500							
Débits UDP		**4,5616889**	**6,1901118**	**7,5350325**	**9,6265952**	**11,328523**	**13,539955**	**15,00446**	**15,565662**
		4,9282277	**6,8849839**	**8,5903951**	**11,418844**	**13,894985**	**17,37585**	**19,86393**	**20,85957**
Débits 802.11		**4,6484602**	**6,3078585**	**7,6783619**	**9,8097098**	**11,544012**	**13,797509**	**15,28987**	**15,861748**
		5,0219711	**7,0159483**	**8,7537994**	**11,63605**	**14,159292**	**17,706369**	**20,241777**	**21,256356**

Figure D.4 – Débits pratiques calculés pour la technologie 802.11b/g.

ANNEXE E

CODE DE VIRTUAL AP

```
1  #!/usr/bin/python
2
3  import psyco
4  import pacmap
5  from scapy import Dot11,Packet,Dot11Elt,Dot11Auth,Ether,LLC,SNAP,Dot11ProbeResp,
           Dot11AssoResp,ARP,DHCP,IP,UDP,BOOTP,mac2str,str2mac,Dot11Beacon,MACField
6  import signal
7  import time, calendar,datetime
8  import random
9  import os
10
11 # Paramètres de démarrage de l'AP
12 tap_ip_address="192.168.10.1"
13 tap_mac_address="00:11:22:33:44:55"
14 bssid = " 00:11:22:33:44:55 "
15 floor = 15 # seuil en dB
16 delta = 5 # nombre de balises reçues avant disparition locale du terminal
17 expiracy = 10 # délai avec déassociation automatique
18
19 # Variables globales
20 timestamp = 0
21 index=-1
22
23 # Classe décrivant un client (adresse mac, bssid, numéro de séquence du dernier paquet
           transmis...)
24 class Client:
25
26     def __init__(self, mac, bssid):
27         self.bssid = bssid
28         self.mac = mac
29         self.timestamp = time.mktime(datetime.datetime.now().timetuple())
```

```
30            self.sequence = 0
31            self.lastsequence = 0
32            self.state  = "unknown"
33            self.rssi  = 0
34            r  = random.randint(10,100)
35            self.ip  = "192.168.10.%d"  % r
36
37        def getIP(self):
38            return self.ip
39
40        def setIP(self,ip):
41            self.ip=ip
42
43        def getBSSID(self):
44            return self.bssid
45
46        def setBSSID(self,bssid):
47            self.bssid  = bssid
48
49        def getLastRSSI(self):
50            return self.rssi
51
52        def setLastRSSI(self, rssi):
53            self.rssi=rssi
54
55        def getTimestamp(self):
56            return self.timestamp
57
58        def setTimestamp(self,timestamp):
59            self.timestamp=timestamp
60
61        def getLastSeq(self):
62                return self.lastsequence
63
64        def setLastSeq(self,seq):
65                self.lastsequence=seq
66
67        def getSeq(self):
68            return self.sequence
69
70        def setSeq(self,seq):
71            self.sequence=seq
72
73        def getMAC(self):
74            return self.mac
75
76        def getBSSID(self):
77            return self.bssid
78
79        def setBSSID(self,  remotebssid):
80            self.bssid  = remotebssid
```

```
81
82      def getState(self):
83          return self.state
84
85      def setState(self,state):
86          self.state=state
87
88  class ClientList:
89
90      def __init__(self):
91          self.data = []
92          self.mac = []
93          self.index = -1
94
95      def isPresent(self, mac):
96          i = -1
97          try:
98              while 1:
99                  i = self.mac.index(mac, i+1)
100         except ValueError:
101             pass
102         return i
103
104     def add(self,mac,current_bssid):
105         i = self.isPresent(mac)
106         if i < 0:
107             #print "Adding new client %s" % mac
108             client = Client(mac,current_bssid)
109             self.mac.append(mac)
110             self.data.append(client)
111
112     def delete(self,mac):
113         i = self.isPresent(mac)
114         if i>=0:
115             #print "Remove client %s" % mac
116             del self.mac[i]
117             del self.data[i]
118
119
120     def __iter__(self):
121         self.index = len(self.data)
122         return self
123
124     def next(self):
125         if self.index == 0:
126             raise StopIteration
127         self.index = self.index - 1
128         return self.data[self.index]
129
130     def getClient(self,mac):
131         i = self.isPresent(mac)
```

```
132             if i>=0:
133                 return self.data[i]
134             else:
135                 return
136
137         def getLength(self):
138             return len(self.mac)
139
140 # Liste des clients connectés à notre AP
141  clients_list = ClientList()
142
143 # Liste des clients à surveiller
144 watch_list = ClientList()
145
146 # Appel de ces routines au démarrage de PACMAP
147 def background():
148         global bssid
149         global index
150         global tap_ip_address
151         global tap_mac_address
152         print "BSSID␣from␣MAC␣Adapter␣is␣%s" % bssid
153         print "Virtual␣AP␣0.1␣by␣Yan␣Grunenberger\nWritten␣using␣PacMap"
154         index=pacmap.createtap()
155         pacmap.settapmac(index,mac2str(tap_mac_address)) #bssid))
156         pacmap.settapip(index,tap_ip_address)
157         return 0
158
159 # Fonction périodique de PACMAP (résolution : 1/512 de seconde)
160 def periodic(float1, float2, float3):
161         global timestamp
162         i = clients_list.getLength()
163         if i == 0:
164             i = 1
165         beacon_interval = 0x0064 * i
166         periodicity = 11.25 # period in 1/x seconds
167         result = int(float1/512 * periodicity)
168         if result > timestamp:
169             timestamp = result
170
171             # Pour chaque client, on envoie une balise
172             for client in clients_list:
173                 current_timestamp = time.mktime(datetime.datetime.now().timetuple())*1e6+
174                                     datetime.datetime.now().microsecond
175                 # Dernier numéro de séquence utilisé dans le paquet, nécessaire pour la continuité
176                 seq = client.getSeq()
177                 ssid = "Client-%s" % client.getMAC()
178                 rssi = "%d" % client.getLastRSSI()
179                 current_bssid = client.getBSSID()
180
181                 # Formatage de la balise, avec ajout du niveau de signal et de l'adresse IP
                attribuée du client
```

```
182                 dot11_answer = Dot11(
183                 FCfield ="retry",
184                 SC= (seq<<4 & 0xff) |( (seq>>4 & 0xff) << 8 & 0xff00) ,
185                 type = "Management",
186                 addr1 = "ff:ff:ff:ff:ff:ff",
187                 addr2 = current_bssid,
188                 addr3 = current_bssid)/Dot11Beacon(timestamp = current_timestamp,cap = 0
                x0104,beacon_interval = beacon_interval )/Dot11Elt(ID=0,info=ssid)/Dot11Elt(ID=1,
                info="\x82")/Dot11Elt(ID=3,info="\x06")/Dot11Elt(ID=221,info=rssi)/Dot11Elt(ID
                =221,info=client.getIP())
189
190                 pacmap.sendpacket(str(dot11_answer),len(str(dot11_answer)),0x02)
191
192                 if seq == 4095:
193                     client.setSeq(0)
194                 else:
195                     client.setSeq(seq+1)
196                 diff = timestamp - client.getTimestamp()
197
198                 # Délai d'expiration des clients (pour les clients n'utilisant pas la
              déconnexion)
199                 if diff>expiracy:
200                     clients_list.delete(client.getMAC())
201     return 0
202
203 # Fonction inutilisée : traite l'ensemble des paquets reçus
204 def packet_recv(packet, length):
205     return 0
206
207 # Fonction exécutée à la réception d'un paquet sur l'interface Ethernet du système (vap0,
            vap1...)
208 def packet_ti_recv(packet, length,index):
209     ether_frame = Packet(packet)
210     ether_frame.decode_payload_as(Ether)
211
212     addr1 = ether_frame.getlayer(Ether).dst
213     addr2 = bssid
214     addr3 = ether_frame.getlayer(Ether).src
215
216     # Interception des trames ARP pour alléger le trafic sans fil : on répond directement au
            système grâce à la liste des clients connectés
217     if ether_frame[ARP] and ether_frame[ARP].op == 1:
218         print "TAP␣ARP␣who-has␣%s␣"%(ether_frame[ARP].pdst)
219         for client in clients_list :
220             if client and (client.getIP() == ether_frame[ARP].pdst):
221                 print "TAP␣ARP␣who-has␣%s␣->␣reply␣%s" %(client.getIP(),client.
              getMAC())
222                 eth_sent_frame = Ether(dst=addr3,src=client.getMAC(),type=ether_frame.
              getlayer(Ether).type)/ARP(op=2,hwsrc=client.getMAC(),psrc=client.getIP(),hwdst=
              addr3,pdst=ether_frame[ARP].psrc)
223                 pacmap.sendtipacket(str(eth_sent_frame),len(str(eth_sent_frame)),index)
```

```
224                     return 1
225
226         client = clients_list.getClient(addr1)
227         if client:
228             addr2 = client.getBSSID()
229
230         dot11_answer=Dot11(type = "Data",
231         addr1 = addr1,
232         addr2 = addr2,
233         addr3 = addr3,
234         FCfield=0x82)/LLC()/SNAP(code=ether_frame.getlayer(Ether).type)/ether_frame.getlayer(
                Ether).payload
235
236         # Détection du trafic à destination d'un client sans fil et envoi si client associé
237         if (client and addr1 != "ff:ff:ff:ff:ff:ff"):
238             pacmap.sendpacket(str(dot11_answer),len(str(dot11_answer)),0x02)
239         return 1
240
241 # Détection des requêtes de détection de point d'accès, et réponse adéquate
242 def proto80211_probereq(packet, length):
243         dot11_frame = Packet(packet)
244         dot11_frame.decode_payload_as(Dot11)
245         clientmac = dot11_frame.getlayer(Dot11).addr2
246
247         # Génération d'un SSID de la forme "Client-XX:XX..."
248         ssid = "Client-%s" % clientmac
249         current_timestamp = time.mktime(datetime.datetime.now().timetuple())*1e6+datetime.
                datetime.now().microsecond
250
251         # Réponse à une requête générale avec notre BSSID
252         if dot11_frame.getlayer(Dot11).addr3 == "ff:ff:ff:ff:ff:ff":
253             dot11_answer = Dot11(
254         type = "Management",
255         addr1 = dot11_frame.getlayer(Dot11).addr2,
256         addr2 = bssid,
257         addr3 = bssid)/Dot11ProbeResp(timestamp=current_timestamp,cap = 0x0104)/Dot11Elt(
                ID=0,info=ssid)/Dot11Elt(ID=1,info="\x82")/Dot11Elt(ID=3,info="\x06")
258
259             pacmap.sendpacket(str(dot11_answer),len(str(dot11_answer)),0x02)
260         # Réponse à une requête ciblée si le client est associé (ce cas se produit pour des
                clients qui cherche périodiquement les points d'accès)
261         else:
262             client = clients_list.getClient(clientmac)
263             if client:
264                 dot11_answer = Dot11(
265             type = "Management",
266             addr1 = dot11_frame.getlayer(Dot11).addr2,
267             addr2 = client.getBSSID(),
268             addr3 = client.getBSSID())/Dot11ProbeResp(timestamp=current_timestamp,cap = 0
                x0104)/Dot11Elt(ID=0,info=ssid)/Dot11Elt(ID=1,info="\x82")/Dot11Elt(ID=3,info="\
                x06")
```

```
269                 pacmap.sendpacket(str(dot11_answer),len(str(dot11_answer)),0x02)
270
271     return 1
272
273 # Interception des demandes d'authentification
274 def proto80211_auth(packet, length):
275     global bssid
276     print "Authentification␣received"
277     dot11_frame = Packet(packet)
278     dot11_frame.decode_payload_as(Dot11)
279
280     if (dot11_frame.getlayer(Dot11).addr3 == bssid):
281         if (dot11_frame.getlayer(Dot11Auth).seqnum == 0x01):
282             dot11_answer = dot11_frame
283             addr1= dot11_frame.getlayer(Dot11).addr2
284             dot11_answer.getlayer(Dot11).addr1= addr1
285             dot11_answer.getlayer(Dot11).addr2= bssid
286             dot11_answer.getlayer(Dot11Auth).seqnum=0x02
287             pacmap.sendpacket(str(dot11_answer),len(str(dot11_answer)),0x02)
288
289     return 1
290
291 # Interception des demandes de déconnexion
292 def proto80211_deauth(packet, length):
293     print "Deauthentification␣received"
294     dot11_frame = Packet(packet)
295     dot11_frame.decode_payload_as(Dot11)
296     client = dot11_frame.getlayer(Dot11).addr2
297     clients_list .delete( client )
298     return 1
299
300 # Interception des demandes d'association
301 def proto80211_assocreq(packet, length):
302     print "Association␣request␣received"
303     dot11_frame = Packet(packet)
304     dot11_frame.decode_payload_as(Dot11)
305     if (dot11_frame.getlayer(Dot11).addr3 == bssid):
306         dot11_answer = dot11_frame
307         client = dot11_frame.getlayer(Dot11).addr2
308         ssid = "Client−%s" % client
309         dot11_answer = Dot11(type = "Management",addr1 = dot11_frame.getlayer(Dot11).
              addr2,addr2 = bssid, addr3 = bssid)/Dot11AssoResp(cap = 0x0104, AID=0xc001)/
              Dot11Elt(ID=0,info=ssid)/Dot11Elt(ID=1,info="\x02")
310         pacmap.sendpacket(str(dot11_answer),len(str(dot11_answer)),0x02)
311         clients_list .add(client,bssid)
312
313     return 1
314
315 # Traitement des balises reçues
316 def proto80211_beacon(packet, length, essid):
317     dot11_frame = Packet(packet)
```

```
318     dot11_frame.decode_payload_as(Dot11)
319
320     # Détection des trames issues d'un point d'accès virtuel
321     if (essid.startswith("Client-")):
322
323         clientmac = essid.replace("Client-", "")
324         seq = (dot11_frame[Dot11].SC >> 4 )
325
326         client = clients_list.getClient(clientmac)
327         if client:
328        # On reçoit une balise d'un client enregistré.
329        # On se sert du numéro de séquence pour s'assurer que les balises sont transmises en
            continu.
330             if (seq > client.getLastSeq()+delta ):
331                 print "Client with seq %d was transfered, cancel registration here with
            seq %d" % (client.getLastSeq(),seq)
332                 arp_flush = "arp -d %s" % client.getIP()
333                 clients_list.delete(clientmac)
334                 # Purge de la table ARP des interfaces TAP et des bridges éventuels
335                 os.system(arp_flush)
336
337         # Récupération du RSSI et de l'IP du client
338         current_payload = dot11_frame.getlayer(Dot11Beacon).payload
339         current_bssid = dot11_frame.getlayer(Dot11).addr3
340         while (current_payload[Dot11Elt] and current_payload.getlayer(Dot11Elt).ID != 221):
341             current_payload = current_payload.payload
342         lastRSSI = int(current_payload.getlayer(Dot11Elt).info)
343         current_payload = current_payload.payload
344         ip = current_payload.getlayer(Dot11Elt).info
345
346         # Ajout du client à la liste des clients à surveiller
347         watch_list.add(clientmac,current_bssid)
348         watched_client = watch_list.getClient(clientmac)
349         watched_client.setBSSID(current_bssid)
350
351         # Détection de la mobilité par seuil (différence entre le signal reçu local et le
            signal reçu distant)
352         if (watched_client.getLastRSSI()-lastRSSI) > floor:
353             # Migration du client sur le point d'accès
354             print "Transfering terminal to this AP"
355             clients_list.add(clientmac,current_bssid)
356             client = clients_list.getClient(clientmac)
357             timestamp = time.mktime(datetime.datetime.now().timetuple())
358                         +datetime.datetime.now().microsecond/1e6
359             client.setTimestamp(timestamp)
360             client.setBSSID(current_bssid)
361             print "previous AP was : %s with ip %s" % (current_bssid ,ip)
362             client.setLastRSSI(lastRSSI)
363             client.setIP(ip)
364             arp_flush = "arp -d %s" % client.getIP() # Purge de la table ARP
365             os.system(arp_flush)
```

```
366                 if seq == 4095:
367                     client.setSeq(0)
368                     client.setLastSeq(0)
369                 else:
370                     client.setSeq(seq)
371                     client.setLastSeq(seq)
372                 watch_list.delete(clientmac)
373 
374     return 0
375 
376 # Fonction inutilisée : récupération des trames de contrôles (ACK)
377 def proto80211_ctrl(packet, length):
378     return 0
379 
380 # Traitement des trames de données entrantes
381 def proto80211_data(packet, length):
382     global index
383     dot11_frame = Packet(packet)
384     dot11_frame.decode_payload_as(Dot11)
385 
386     mac = dot11_frame.getlayer(Dot11).addr2
387 
388     watched_client = watch_list.getClient(mac)
389 
390     # Si la station émettrice est surveillée, on récupère le niveau de signal reçu
391     if watched_client:
392         watched_client.setLastRSSI(pacmap.getrxinfo(1))
393 
394    # Si le BSSID de destination est proche du notre, on traite le trafic
395     if (dot11_frame.getlayer(Dot11).addr1[2:] == bssid[2:]):
396 
397         client = clients_list.getClient(mac)
398         if client:
399             timestamp = time.mktime(datetime.datetime.now().timetuple())+datetime.datetime
        .now().microsecond/1e6
400             client.setTimestamp(timestamp)
401             client.setLastRSSI(pacmap.getrxinfo(1))
402 
403             # Interception du trafic ARP : si l'adresse demandée est celle de l'AP, nous
        répondons
404             if dot11_frame[ARP] and dot11_frame[ARP].op == 1 and dot11_frame[ARP].pdst
        ==tap_ip_address:
405                 print "Wireless␣ARP␣who-has␣->␣reply"
406                 dot11_answer=Dot11(type = "Data",addr1 = mac,addr2 = client.getBSSID(),
        addr3 = tap_mac_address,FCfield=0x02)/LLC()/SNAP()/ARP(op=2,hwsrc=
        tap_mac_address,psrc=tap_ip_address,hwdst=dot11_frame[ARP].hwsrc,pdst=
        dot11_frame[ARP].psrc)
407                 pacmap.sendpacket(str(dot11_answer),len(str(dot11_answer)),0x02)
408                 return 1
409 
410             # Interception des requêtes DHCP du client
```

```
411             if dot11_frame[DHCP] and dot11_frame[DHCP].options[0][1] == 1:
412                 dot11_answer=Dot11(type = "Data",addr1 = mac,addr2 = client.getBSSID(),
        addr3 = tap_mac_address,FCfield=0x02)/LLC()/SNAP()/IP(src=tap_ip_address,dst=
        client.getIP())/UDP(sport=67,dport=68)/BOOTP(op=2, ciaddr="0.0.0.0",yiaddr=client.
        getIP(),siaddr="0.0.0.0",giaddr=tap_ip_address,chaddr=mac2str(mac), xid=
        dot11_frame[BOOTP].xid)/DHCP(options=[('message-type','offer'),('server_id',
        tap_ip_address), ('lease_time',43200), ('renewal_time',21600),('rebinding_time',37800),(
        'subnet_mask',"255.255.255.0"),('end')]) #('router',tap_ip_address),('name_server',
        tap_ip_address)
413
414                 pacmap.sendpacket(str(dot11_answer),len(str(dot11_answer)),0x02)
415                 return 1
416
417             if dot11_frame[DHCP] and dot11_frame[DHCP].options[0][1] == 3:
418                 print "DHCPREQUEST -> REPLY with %s" % client.getIP()
419                 dot11_answer=Dot11(type = "Data",addr1 = mac,addr2 = client.getBSSID(),
        addr3 = tap_mac_address,FCfield=0x02)/LLC()/SNAP()/IP(src=tap_ip_address,dst=
        client.getIP())/UDP(sport=67,dport=68)/BOOTP(op=2, ciaddr="0.0.0.0",yiaddr=client.
        getIP(),siaddr="0.0.0.0",giaddr=tap_ip_address,chaddr=mac2str(mac), xid=
        dot11_frame[BOOTP].xid)/DHCP(options=[('message-type','ack'),('server_id',
        tap_ip_address), ('lease_time',43200), ('renewal_time',21600),('rebinding_time',37800),(
        'subnet_mask',"255.255.255.0"),('end')])
420
421                 pacmap.sendpacket(str(dot11_answer),len(str(dot11_answer)),0x02)
422                 return 1
423
424             # Le trafic est générique, on enlève l'en-tête SNAP pour le remplacer par une
        en-tête Ethernet et on l'envoie sur l'interface Ethernet
425             if dot11_frame.haslayer(SNAP):
426                 eth_sent_frame = Ether(dst=dot11_frame.getlayer(Dot11).addr3,src=
        dot11_frame.getlayer(Dot11).addr2,type=dot11_frame.getlayer(SNAP).code)
427                 eth_sent_frame.payload = dot11_frame.getlayer(SNAP).payload
428
429                 pacmap.sendtipacket(str(eth_sent_frame),len(str(eth_sent_frame)),index)
430
431         else:
432             # "Data from unknown station"
433     else:
434         #"Data for another BSSID"
435     return 1
```

Quatrième partie

Bibliographie

[1] L Kleinrock. Communication Nets : Stochastic Message Flow and Delay. *Communication Nets. Stochastic Message Flow and Delay*, page 288, Jan 1964.

[2] R Metcalfe et D Boggs. Ethernet : distributed packet switching for local computer networks. *Communications of ACM*, 19, Jan 1976.

[3] L Pouzin. Presentation and major design aspects of the CYCLADES computer network. *Proceedings of the third ACM symposium on Data communications and Data networks : Analysis and design*, pages 80–87, Jan 1973.

[4] J.C.R Licklider et Welden E Clark. On-Line Man-Computer Communication. *Proceedings of the AFIPS 1962 Spring Joint Computer Conference*, 21:113–128, 1962.

[5] D Cohen. A Protocol for Packet Switching Voice Communication. *Proceedings of Computer Network Protocols Symposium*, 2(4-5):320–331, 1978.

[6] L Roberts. Multiple computer networks and intercomputer communication. *Proceedings of the first ACM symposium on Operating System Principles*, pages 3.1 – 3.6, Jan 1967.

[7] A McKenzie. Host/Host Protocol for the ARPA Network. *Current Network Protocols, Network Information Center, Stanford Research Institute, Menlo Park, California*, Jan 1972.

[8] Thomas Merill et Lawrence G Roberts. Toward a Cooperative Network of Time-Shared Computers. *Proceedings of the Fall AFIPS Conference*, pages 425–431, 1966.

[9] A Kamerman et Leo Monteban. WaveLAN®-II : A high-performance wireless LAN for the unlicensed band : Wireless. *Bell Labs technical journal*, Jan 1997.

[10] N Abramson. The ALOHA System – Another Alternative for Computer Communications. *AFIPS Conference Proceedings*, 36:295–298, Sep 1970.

[11] R Kahn. The Organization of Computer Resources into a Packet Radio Network. *IEEE Transactions on Communications*, 25(1):169–178, Jan 1977.

[12] Leonard Kleinrock. Information Flow in Large Communication Nets. *RLE Quarterly Progress Report*, 1961.

[13] B Crow, I Widjaja, L Kim et P Sakai. IEEE 802.11 Wireless Local Area Networks. *IEEE Communications Magazine*, 35(9):116–126, Jan 1997.

[14] P Baran. On Distributed Communications Networks. *IEEE Transactions on Information Theory*, 12(1-9), Mar 1964.

[15] V Cerf et R Kahn. A Protocol for Packet Network Intercommunication. *IEEE Transactions on Communications*, 22(5):637– 648, Jan 1974.

[16] A Mishra, M Shin et W Arbaugh. An empirical analysis of the IEEE 802.11 MAC layer handoff process. *ACM SIGCOMM Computer Communication Review*, 33(2):93–102, Jan 2003.

[17] R Draves, J Padhye et B Zill. Routing in multi-radio, multi-hop wireless mesh networks. *Proceedings of the 10th Annual International Conference on Mobile Computing and Networking*, pages 114–128, Jan 2004.

[18] Z Haas. A new routing protocol for the reconfigurable wireless networks. *IEEE 6th International Conference on Universal Personal Communications Record*, 2:562 – 566 vol.2, Sep 1997.

[19] G Holland, N Vaidya et P Bahl. A rate-adaptive MAC protocol for multi-Hop wireless networks. *Proceedings of the 7th Annual International Conference on Mobile Computing and Networking*, pages 236–251, Jan 2001.

[20] M Fiore, J Haerri, F Filali et C Bonnet. Vehicular Mobility Simulation for VANETs. *40th Annual Simulation Symposium (ANSS'07)*, pages 301–309, Jan 2007.

[21] D Johnson, D Maltz et J Broch. DSR : The Dynamic Source Routing Protocol for Multi-Hop Wireless Ad Hoc Networks. *Ad Hoc Networking by Charles E. Perkins*, 5, Jan 2001.

[22] S Shin, G Forte, A Rawat et H Schulzrinne. Reducing MAC layer handoff latency in IEEE 802.11 wireless LANs. *Proceedings of the Second International Workshop on Mobility Management & Wireless Access Protocols*, pages 19–26, Jan 2004.

[23] A Kochut, A Vasan, A Shankar et A Agrawala. Sniffing out the correct physical layer capture model in 802.11 b. *Proceedings of the 12th IEEE International Conference on Network Protocols ICNP 2004.*, pages 252–261, Jan 2004.

[24] Y Ko et N Vaidya. Location Aided Routing (LAR) in mobile ad hoc networks. *Wireless Networks*, 6:307–321, Jan 2000.

[25] F Guillemin, P Robert et B Zwart. AIMD algorithms and exponential functionals. *The Annals of Applied Probability*, 14(1):90–117, Jan 2004.

[26] F Calì, M Conti et E Gregori. IEEE 802.11 Protocol : Design and Performance Evaluation of an Adaptive Backoff Mechanism. *IEEE Journal on Selected Areas on Communications*, 18(9):1774–1786, Jan 2000.

[27] H Ma, X Li, H Li, P Zhang, S Luo et C Yuan. Dynamic optimization of IEEE 802.11 CSMA/CA based on the number of competing stations. *IEEE International Conference on Communications*, 1:191–195, Jan 2004.

[28] G Bianchi et I Tinnirello. Kalman filter estimation of the number of competing terminals in an IEEE 802.11 network. *IEEE INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications Societies.*, 2:844–852, 2003.

[29] Guangyu Pei, M Gerla et Tsu-Wei Chen. Fisheye state routing : a routing scheme for ad hoc wireless networks. *IEEE International Conference on Communications - ICC 2000*, 1:70–74, May 2000.

[30] L Kleinrock et S Lam. Packet Switching in a Multiaccess Broadcast Channel : Performance Evaluation. *IEEE Transactions on Communications*, 23(4):410–423, Jan 1975.

[31] B Sadeghi, V Kanodia, A Sabharwal et E Knightly. Opportunistic media sccess for multirate ad hoc networks. *Proceedings of the 8th Annual International Conference on Mobile Computing and Networking*, pages 24–25, Jan 2002.

[32] H Velayos et G Karlsson. Techniques to Reduce IEEE 802.11 b MAC Layer Handover Time. *IEEE International Conference on Communications*, 2004.

[33] IEEE SA Standards Board. IEEE 802.11g. *IEEE*, 2003.

[34] L Bononi, M Conti et E Gregori. Runtime optimization of IEEE 802.11 wireless LANs performance. *IEEE Transactions on Parallel and Distributed Systems*, 15(1): 66–80, Jan 2004.

[35] A Mishra, M Shin et W Arbaush. Context caching using neighbor graphs for fast handoffs in a wireless network. *INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies*, 1(7):361, Jan 2004.

[36] M Heusse, F Rousseau, R Guillier et A Duda. Idle sense : an optimal access method for high throughput and fairness in rate diverse wireless LANs. *Proceedings of the 2005 Conference of Special Interest Group on Data Communication - SIGCOMM'05*, 35(4):121–132, Jan 2005.

[37] Paramvir Bahl, Ranveer Chandra et John Dunagan. SSCH : slotted seeded channel hopping for capacity improvement in IEEE 802.11 ad-hoc wireless networks. *MobiCom '04 : Proceedings of the 10th Annual International Conference on Mobile Computing and Networking*, pages 216–230, Sep 2004.

[38] C Perkins et P Bhagwat. Highly dynamic Destination-Sequenced Distance-Vector routing (DSDV) for mobile computers. *ACM SIGCOMM Computer Communication Review*, 24(4):234–244, Jan 1994.

[39] M Lacage, M Manshaei et T Turletti. IEEE 802.11 rate adaptation : a practical approach. *Proceedings of the 7th ACM international Symposium on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, pages 126–134, Jan 2004.

[40] V Mhatre et K Papagiannaki. Using smart triggers for improved user performance in 802.11 wireless networks. *Proceedings of the 4th International Conference on Mobile Systems, Applications and Services*, pages 246–259, Jan 2006.

[41] Luigi Iannone et Serge Fdida. SDT. 11b : Un Schéma à Division de Temps pour éviter l'anomalie de la couche MAC 802.11 b. *Colloque Français sur l'Ingénierie des Protocoles - CFIP*, 2005.

[42] H Lundgren, E Nordströ et C Tschudin. Coping with communication gray zones in IEEE 802.11 b based ad hoc networks. *Proceedings of the 5th ACM International Workshop on Wireless Mobile Multimedia*, pages 49–55, Jan 2002.

[43] C Perkins et E Royer. Ad-hoc on-demand distance vector routing. *Second IEEE Workshop on Mobile Computing Systems and Applications, 1999. Proceedings. WMCSA '99.*, pages 90–100, Jan 1999.

[44] D Couto, D Aguayo, J Bicket et R Morris. a high-throughput path metric for multi-hop wireless routing. *Wireless Networks*, 11:419–434, Jan 2005.

[45] F Cali, M Conti et E Gregori. IEEE 802.11 wireless LAN : capacity analysis and protocol enhancement. *Proceedings IEEE INFOCOM '98. Seventeenth Annual Joint Conference of the IEEE Computer and Communications Societies.*, 1:142–149, 1998.

[46] B Karp et H Kung. GPSR : greedy perimeter stateless routing for wireless networks. *Proceedings of the 6th Annual International Conference on Mobile Computing and Networking*, pages 243–254, Jan 2000.

[47] G Aggelou et R Tafazolli. RDMAR : a bandwidth-efficient routing protocol for mobile ad hoc networks. *Proceedings of the 2nd ACM International Workshop on Wireless Mobile Multimedia*, Jan 1999.

[48] L Kleinrock et F Tobagi. Packet Switching in Radio Channels : Part I–Carrier Sense Multiple-Access Modes and Their Throughput-Delay Characteristics. *IEEE Transactions on Communications*, 23(12):1400–1416, Jan 1975.

[49] IEEE SA Standards Board. IEEE 802.11. *IEEE*, 1999.

[50] Q Ni, I Aad, C Barakat et T Turletti. Modeling and analysis of slow CW decrease IEEE 802.11 WLAN. *IEEE Proceedings on Personal, Indoor and Mobile Radio Communications. PIMRC 2003*, Jan 2003.

[51] IEEE SA Standards Board. IEEE 802.11a. *IEEE*, 1999.

[52] B Sadeghi, V Kanodia, A Sabharwal et E Knightly. OAR : An Opportunistic Auto-Rate Media Access Protocol for Ad Hoc Networks. *Wireless Networks*, 11(1-2):39–53, Jan 2005.

[53] T Razafindralambo, I Lassous, L Iannone et S Fdida. Dynamic packet aggregation to solve performance anomaly in 802.11 wireless networks. *Proceedings of the 9th ACM International Symposium on Modeling Analysis and Simulation of Wireless and Mobile Systems*, pages 247–254, Jan 2006.

[54] A Mishra, M Shin, N Petroni Jr et T Clancy. Proactive key distribution using neighbor graphs. *IEEE Wireless Communications*, 11(1):26–36, Jan 2004.

[55] V Park et M Corson. A highly adaptive distributed routing algorithm for mobile wireless networks. *INFOCOM '97. Sixteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, 3:1405 – 1413, Mar 1997.

[56] T Clausen, G Hansen, L Christensen et G Behrmann. The Optimized Link State Routing Protocol, Evaluation through Experiments and Simulation. *Proceeding of Wireless Personal Multimedia Communications. MindPass Center for Distributed Systems, Aalborg University, Fourth International Symposium on Wireless Personal Multimedia Communications*, Jan 2001.

[57] IEEE SA Standards Board. IEEE 802.11b. *IEEE*, 1999.

[58] A Iwata, Ching-Chuan Chiang, Guangyu Pei, M Gerla et Tsu-Wei Chen ;. Scalable routing strategies for ad hoc wireless networks. *Selected Areas in Communications, IEEE Journal on*, 17(8):1369 – 1379, Aug 1999.

[59] F Theoleyre et F Valois. Virtual structure routing in ad hoc networks. *IEEE International Conference on Communications. ICC 2005*, 5:3078–3082, Jan 2005.

[60] P Jacquet, P Muhlethaler, T Clausen, A Laouiti, A Qayyum et L Viennot. Optimized link state routing protocol for ad hoc networks. *Proceedings of the IEEE INMIC 2001 Conference*, pages 62–68, Nov 2001.

[61] Nicolas Montavont et Thomas Noël. Analysis and evaluation of mobile IPv6 handovers over wireless LAN. *Mobile Networks and Applications*, 8(6):643–653, Dec 2003.

[62] G Bianchi, L Fratta et M Oliveri. Performance Evaluation and Enhancement of the CSMA/CA MAC Protocol for 802.11 Wireless LANs. *Seventh IEEE International Symposium on Personal, Indoor and Mobile Radio Communications, PIMRC'96.*, 2:392–396, Jan 1996.

[63] M Heusse, F Rousseau, G Berger-Sabbatel et A Duda. Performance anomaly of 802.11 b. *IEEE INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications Societies.*, 2:836–843, 2003.

[64] J Macker et M Corson. Mobile ad hoc networking and the IETF. *ACM SIGMOBILE Mobile Computing and Communications Review*, Jan 1998.

[65] Y Kwon, Y Fang et H Latchman. A novel MAC protocol with fast collision resolution for wireless LANs. *IEEE INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications Societies.*, 2:853–862, 2003.

[66] S Basagni, I Chlamtac, V Syrotiuk et B Woodward. A distance routing effect algorithm for mobility (DREAM). *Proceedings of the 4th annual ACM/IEEE International Conference on Mobile Computing and Networking*, pages 76–84, Jan 1998.

[67] LS Paun. Gestion de la mobilité dans les réseaux ambiants. *Thèse*, 2005.

[68] IEEE. Recommended Practice for Multi-Vendor Access Point Interoperability via an Inter-Access Point Protocol Acress Distribution Systems Supporting 802.11 operation. *IEEE Standard 802.11*, 1999.

[69] E Kohler, R Morris, B Chen et J Jannotti. The click modular router. *ACM Transactions on Computer Systems (TOCS)*, 18(3):263–297, Jan 2000.

[70] V Srivastava et M Motani. Cross-layer design : a survey and the road ahead. *IEEE Communications Magazine*, 43(12):112–119, Jan 2005.

[71] W Zhu, D Browne et M Fitz. An open access wideband multiantenna wireless testbed with remote control capability. *First International Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities. Tridentcom 2005.*, pages 72–81, Jan 2005.

[72] J Bicket, D Aguayo, S Biswas et R Morris. Architecture and evaluation of an unplanned 802.11 b mesh network. *Proceedings of the 11th Annual International Conference on Mobile Computing and Networking*, pages 31–42, Jan 2005.

[73] I Busse, B Deffner et H Schulzrinne. Dynamic QoS control of multimedia applications based on RTP. *Computer Communications*, 19:49–58, Jan 1996.

[74] T Kobayashi, A Batista, A Brito et P Motta Pires. Using a packet manipulation tool for security analysis of industrial network protocols. *IEEE Conference on Emerging Technologies & Factory Automation. ETFA.*, pages 744–747, Jan 2007.

[75] B White, J Lepreau, L Stoller et R Ricci. An integrated experimental environment for distributed systems and networks. *ACM SIGOPS Operating Systems Review*, 36(SI):255–270, Jan 2002.

[76] D Aguayo, J Bicket, S Biswas, G Judd et R Morris. Link-level measurements from an 802. 11 b mesh network. *ACM SIGCOMM Computer Communication Review*, pages 121–132, Jan 2004.

[77] M Mendonca et N Neves. Fuzzing Wi-Fi Drivers to Locate Security Vulnerabilities. *10th IEEE High Assurance Systems Engineering Symposium. HASE '07.*, pages 379–380, Jan 2007.

[78] S Floyd. TCP and explicit congestion notification. *ACM SIGCOMM Computer Communication Review*, 24(5):8–23, Jan 1994.

[79] H Aiache, V Conan, J Leguay et M Levy. XIAN : Cross-Layer Interface for Wireless Ad hoc Networks. *Mediterranean Ad Hoc Networking Workshop (Med-Hoc-Net)*, 2006.

[80] J Mitola. The software radio architecture. *IEEE Communications Magazine*, 33(5): 26–38, Jan 1995.

[81] S Keil et C Kolbitsch. Stateful Fuzzing of Wireless Device Drivers in an Emulated Environment. *Black Hat Japan*, 2007.

[82] M Ott, I Seskar, R Siraccusa et M Singh. ORBIT testbed software architecture : supporting experiments as a service. *First International Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities. Tridentcom 2005.*, pages 136–145, Jan 2005.

[83] R Patra, S Nedevschi, S Surana et A Sheth. WiLDNet : Design and Implementation of High Performance WiFi Based Long Distance Networks. *USENIX NSDI*, 2007.

[84] J Pavon et S Chio. Link adaptation strategy for IEEE 802.11 WLAN via received signal strength measurement. *IEEE International Conference on Communications. ICC '03.*, 2:1102–1113, Jan 2003.

[85] E Tews, R Weinmann et A Pyshkin. Breaking 104 bit WEP in less than 60 seconds. *Lecture Notes in Computer Science*, 4867(Information Security Applications):188–202, Jan 2008.

[86] Russ Housley et William Arbaugh. Security problems in 802.11-based networks. *Communications of the ACM*, 46(5):31–34, May 2003.

[87] T ElBatt et A Ephremides. Joint scheduling and power control for wireless ad hoc networks. *Wireless Communications*, 3(1):74–85, Jan 2004.

[88] V Kawadia et P Kumar. A cautionary perspective on cross-layer design. *IEEE Wireless Communications*, 12(1):3–11, Jan 2005.

[89] M Handley et A Greenhalgh. XORP : Breaking the Mould in Router Software. *Proc. London Communications Symposium 2004 (LCS 2004)*, Jan 2004.

[90] F Abdesslem, L Iannone, M de Amorim et K Obraczka. A Prototyping Environment for Wireless Multihop Networks. *Lectures Notes in Computer Science*, 4866(Sustainable Internet):33–47, Jan 2007.

[91] W Arbaugh, N Shankar et Y Wan. Your 802.11 wireless network has no clothes. *IEEE Wireless Communications*, 9(6):44–51, Jan 2002.

[92] Q Liu, S Zhou et G Giannakis. Cross-Layer Combining of Adaptive Modulation and Coding With Truncated ARQ Over Wireless Links. *IEEE Transactions on Wireless Communications*, 3(5):1746–1755, Jan 2004.

[93] Nikita Borisov, Ian Goldberg et David Wagner. Intercepting mobile communications : the insecurity of 802.11. *MobiCom '01 : Proceedings of the 7th Annual International Conference on Mobile Computing and Networking*, pages 180–189, Jul 2001.

[94] A Rao et I Stoica. An overlay MAC layer for 802.11 networks. *Proceedings of the 3rd International Conference on Mobile Systems, Applications, and Services*, pages 135–148, Jan 2005.

[95] R Dhar, G George, A Malani et P Steenkiste. Supporting Integrated MAC and PHY Software Development for the USRP SDR. *1st IEEE Workshop on Networking Technologies for Software Defined Radio Networks. SDR '06.*, pages 68–77, Jan 2006.

[96] H Anouar, C Bonnet, D Câmara, F Filali et R Knopp. An overview of OpenAirInterface wireless network emulation methodology. *ACM SIGMETRICS Performance Evaluation Review*, 36(2):90–94, Jan 2008.

[97] S Biswas et R Morris. ExOR : opportunistic multi-hop routing for wireless networks. *Proceedings of the 2005 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, pages 133–144, Jan 2005.

[98] S Park, K Kim, D Kim, S Choi et S Hong. Collaborative QoS architecture between DiffServ and 802.11 e wireless LAN. *Vehicular Technology Conference. VTC 2003*, 2:945–949, Jan 2003.

[99] G Chelius et É Fleury. Ananas : A New Adhoc Network Architectural Scheme. *Fourth International Conference on Mobile and Wireless Communications Networks (MWCN 2002)*, Jan 2002.

[100] R Bernasconi, Silvia Giordano, Alessandro Puiatti, Raffaele Bruno et Enrico Gregori. Design and Implementation of an Enhanced 802.11 MAC Architecture for Single-Hop Wireless Networks. *EURASIP Journal on Wireless Communications and Networking*, Jan 2007.

[101] Elena Lopez-Aguilera, Martin Heusse, Yan Grunenberger, Franck Rousseau, Andrzej Duda et Jordi Casademont. An Asymmetric Access Point for Solving the Unfairness Problem in WLANs. *IEEE Transactions on Mobile Computing*, 2008.

[102] P Coulton et D Carline. An SDR inspired design for the FPGA implementation of 802.11 a baseband system. *IEEE International Symposium on Consumer Electronics*, pages 470–475, Jan 2004.

[103] T Henderson, S Roy, S Floyd et G Riley. ns-3 project goals. *Proceeding from the Workshop on ns-2 : the IP network simulator*, Jan 2006.

[104] G Judd et P Steenkiste. Using Emulation to Understand and Improve Wireless Networks and Applications. *Proceedings of NSDI*, Jan 2005.

[105] A Jow, C Schurgers et D Palmer. CalRadio : a portable, flexible 802.11 wireless research platform. *Proceedings of the 1st International Workshop on System Evaluation for Mobile Platforms*, pages 49–54, Jan 2007.

[106] M Zec. Implementing a Clonable Network Stack in the FreeBSD Kernel. *Proceedings of the USENIX Annual Technical Conference*, 2003.

[107] P Francis. Addressing in Internetwork Protocols. *PHD Thesis*, Jan 1994.

[108] Eddie Kohler, Robert Morris, Benjie Chen, John Jannotti et M Frans Kaashoek. The click modular router. *ACM Transactions on Computer Systems (TOCS)*, 18(3):263–297, Jan 2000.

[109] J Mogul et K Ramakrishnan. Eliminating receive livelock in an interrupt-driven kernel. *USENIX Annual Technical Conference*, Jan 1996.

[110] Franck Rousseau, Yan Grunenberger, Vincent Untz, Eryk Schiller, Paul Starzetz, Fabrice Theoleyre, Martin Heusse, Olivier Alphand et Andrzej Duda. An architecture for seamless mobility in spontaneous wireless mesh networks. *Proceedings of the ACM International Workshop on Mobility in the Evolving Internet Architecture — Mobiarch'07. Kyoto, Japan*, Sep 2007.

[111] P Barham, B Dragovic, K Fraser, S Hand et T Harris. Xen and the art of virtualization. *ACM SIGOPS Operating Systems Review*, pages 164–177, Jan 2003.

[112] H Balakrishnan, K Lakshminarayanan et S Ratnasamy. A layered naming architecture for the internet. *Proceedings of the 2004 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, pages 343–352, Jan 2004.

[113] I Castineyra, N Chiappa et M Steenstrup. RFC1992 : The Nimrod Routing Architecture. *Internet RFCs*, Jan 1996.

[114] C Tschudin, R Gold, O Rensfelt et O Wibling. LUNAR : a Lightweight Underlay Network Ad-hoc Routing Protocol and Implementation. *Proceedings of NEW2AN'04*, Jan 2004.

Cinquième partie

Liste des publications scientifiques et brevets

E.1 Publications

[115] Y Grunenberger, P Nguyen, G Privat et J Hatala. Distributed Multi-source Video Composition on High Capacity Networks. *Proceedings of MIPS 2004*, LNCS 3311, Jan 2004.

[116] Martin Heusse, Yan Grunenberger, Elena Lopez-Aguilera et Andrzej Duda. An Approach for Solving the Unfairness Problem in WLANs. *Proceedings of the 7th Scandinavian Workshop on Wireless Ad-hoc Networks — ADHOC'07. Stockholm, Sweden*, May 2007.

[117] M Heusse, Y Grunenberger, E López-Aguilera et A Duda. Iniquité dans les réseaux locaux sans fil : rétablir l'ordre juste. *Algotel 2007, Ile d'Oléron*, 2007.

[118] Ibars, A del Coso, Y Grunenberger et F Theoleyre. Increasing the Throughput of Wireless Mesh Networks with Cooperative Techniques. *Mobile and Wireless Communications Summit*, Jan 2007.

[119] F Rousseau, Y Grunenberger, V Untz et E Schiller. An architecture for seamless mobility in spontaneous wireless mesh networks. *Proceedings of the ACM International Workshop on Mobility in the Evolving Internet Architecture — Mobiarch'07. Kyoto, Japan*, Jan 2007.

[120] Y Grunenberger, M Heusse, F Rousseau et A Duda. Experience with an implementation of the Idle Sense wireless access method. *Proceedings of the 2007 ACM CoNEXT conference*, Oct 2007.

[121] Elena Lopez-Aguilera, M Heusse, Y Grunenberger et F Rousseau. An Asymmetric Access Point for Solving the Unfairness Problem in WLANs. *IEEE Transactions on Mobile Computing*, 7(10):1213–1227, Jan 2008.

E.2 Brevets

Dépôt de brevet en cours sur le concept de mobilité transparente à l'aide de Virtual AP.

Réseaux sans fil de nouvelle génération : architectures spontanées et optimisations inter-couches

Résumé

Cette thèse propose une nouvelle approche pour la concrétisation des réseaux sans fil de nouvelle génération et de leurs nouveaux usages. Elle privilégie une séparation temporelle de la pile réseau, avec une première partie dédiée aux opérations temps réel, et une seconde partie organisée autour d'une architecture de traitement de paquets, construite dans un même espace permettant ainsi des optimisations inter-couches. La faisabilité de cette approche est vérifiée tout d'abord par l'étude des plates-formes flexibles permettant la réalisation de nouvelle méthode d'accès, puis au travers de PACMAP, un outil de manipulation de paquet en espace utilisateur, dont les fonctionnalités seront illustrées au travers du prototypage d'une solution originale au problème de la mobilité, Virtual AP.
Mots clés : réseaux sans fil, modèle en couche, inter-couches, prototype, WiFi, méthodes d'accès, mobilité .

Next-gen Wireless Networks : spontaneous architectures and crosslayer optimizations

Abstract

This thesis proposes a new approach to the design of next-generation wireless networks. It consists of a temporal split between realtime operation, and packet-based operations, the last ones running in the userspace thus enabling crosslayer optimizations. This approach is validated through the study of flexible platforms for implementing new access methods, and by introducing PACMAP, a Packet Manipulation tool, which will be used to solve the problem of mobility with the original concept of Virtual AP.
Keywords: wireless networks, layer model, crosslayer, prototype, WiFi, access method, mobility.

Discipline : Informatique
Laboratoire : LIG – Équipe Drakkar
BP 72, 38402 Saint Martin d'Hères CEDEX, France