TRAVAIL DE DIPLÔME 2004
INSTITUT TCOM

EXTENSION DU PILOTE LINUX MADWIFI POUR LES RÉSEAUX WLAN DEVANT SUPPORTER UNE CERTAINE QUALITÉ DE SERVICE

Diplômant
FRÉDÉRIC MIREMAD

HES • SO - EIVD
PROF. DR. STEPHAN ROBERT

16 décembre 2004

Table des matières

1 Résumé du projet **4**
1.1 Introduction . . . 4
1.2 Analyses applicatives et tests . . . 4
1.3 Analyse du code source . . . 5
1.4 Modification du code source et recherche de solution . . . 5
1.5 Outils de travail . . . 5

2 Article sur le support de la QoS dans WLAN[5] **6**
2.1 Introduction . . . 6
2.2 Définition du MSDU . . . 6
2.3 Définition de la terminologie CFP-CP . . . 6
2.4 Définition de la terminologie PCF et DCF . . . 6
2.5 La QoS gérée par un coordinateur central (PCF) . . . 6
2.5.1 Introduction . . . 6
2.5.2 Contention-Free Period . . . 6
2.5.3 Problèmes non résolus avec PCF . . . 7

3 HCF résumé de la norme HCF (p 63[3]) **8**
3.1 Introduction . . . 8
3.2 HCF contention-based channel access (EDCA) . . . 8
3.2.1 Introduction . . . 8
3.2.2 EDCA TXOPs . . . 10
3.3 HCF controlled channel access (HCCA) . . . 11
3.3.1 Introduction . . . 11
3.3.2 CAP/CFP/CP . . . 11

4 Analyse du driver Madwifi **12**
4.1 Introduction . . . 12
4.2 Matériel supporté . . . 12
4.3 Kernel . . . 12
4.4 Téléchargement et suivi du développement des sources de MadWiFi . . . 13
4.5 Architecture grossière du driver . . . 14
4.5.1 Introduction . . . 14
4.5.2 Répertoire ath . . . 17
4.5.3 Répertoire ath_hal . . . 17
4.5.4 Répertoire ath_rate . . . 17
4.5.5 driver . . . 18
4.5.6 Répertoire hal . . . 18
4.5.7 Répertoire include . . . 18
4.5.8 Répertoire net80211 . . . 18
4.5.9 Répertoire patches . . . 18
4.5.10 Répertoire tools . . . 19
4.5.11 Répertoire wlan . . . 19
4.6 Analyse syntaxique . . . 20
4.6.1 Introduction . . . 20
4.6.2 ath/ah.h . . . 20
4.6.3 Liste des dépendances . . . 20
4.7 Analyse sémantique . . . 23
4.7.1 Introduction . . . 23
4.7.2 Modèle-OSI / structure sous linux . . . 23
4.8 Analyse ciblée . . . 24
4.8.1 Introduction . . . 24

4.8.2 Activation du WME . . . 24
4.8.3 Différenciation des services . . . 27
4.8.4 Traitement des quatres files d'attente . . . 28
4.8.5 Mise à jour des flags . . . 32
4.8.6 Format des trames . . . 32
4.8.7 Gestion du beacon . . . 34
4.8.8 Interaction `if_ath.c hal.o` . . . 36
4.8.9 Conclusion de l'analyse . . . 39

5 Paramétrage de la carte **40**
5.1 Introduction . . . 40
5.2 Préparation de l'AP et activation du WME . . . 40
5.3 Banc de test . . . 44
5.3.1 Introduction . . . 44
5.3.2 Configuration de l'AP . . . 45
5.3.3 Configuration des stations mobiles . . . 45
5.3.4 Configuration de la station de monitoring . . . 46
5.3.5 Conception de script . . . 48
5.3.6 Logiciels utilisés . . . 49

6 Test des performances **55**
6.1 Introduction . . . 55
6.2 Méthode de mesure . . . 55
6.3 Traitement des données . . . 57
6.4 Test sans WME . . . 59
6.4.1 Test avec kphone . . . 59
6.4.2 Test avec pckETH . . . 59
6.5 Test avec WME . . . 59
6.5.1 Test avec kphone . . . 59
6.5.2 Test avec pckETH . . . 59
6.6 Conclusion . . . 59

7 Modification du driver **61**
7.1 Introduction . . . 61
7.2 Rajout de nouveaux paramètres . . . 61
7.3 Test des paramètres rajoutés . . . 63
7.4 Interception des fonctions de bas niveau . . . 65
7.4.1 Introduction . . . 65
7.4.2 Méthodologie et interception de la fonction . . . 65

8 Conclusion générale **67**
8.1 Introduction . . . 67
8.2 Nouvelles stations sur le ESSID . . . 67
8.3 Filtrage des flux prioritaires . . . 67
8.4 Limite des chipsets en général . . . 69
8.5 Appréciation de EDCA (sans HCCA) . . . 69
8.6 Remarque concernant la documentation . . . 69
8.7 Outils manquants et diverses remarques . . . 70
8.8 Conclusion finale et personnelle . . . 70

9 Remerciements **72**

10 Annexes **73**

11 Glossaire **74**
11.1 Abrévations 74

Table des figures

1 CAP/CFP/CP periods[3] 11
2 Schéma global 12
3 fic :arborescenceAh 20
4 Correspondance entre le modèle OSI, notre structure (WLAN) et le niveau auquel intervient les commandes systèmes 23
5 Schéma logique de l'interaction entre les sources visibles et le binaire hal.o 37
6 Interface intermédiaire conçu pour ajouter une coordinateur au pilote 38
7 Illustation du pont entre la carte réseau Ethernet et la carte wireless 42
8 Illustration du banc de test 44
9 Un exemple de beacon disséqué par Ethereal 47
10 Définition des règles de coloration d'Ethereal 51
11 Agencement et configuration sur mesure d'Ethereal 52
12 Capture de la fenêtre de configuration principale de packETH [2] 53
13 Capture de la fenêtre de configuration des séquences de packETH [2] 54
14 Temps d'attente après libération du canal, sans WME (Chronogramme inspiré de l'article[5]) 55
15 Temps d'attente après libération du canal, sans WME (Chronogramme inspiré de l'article[5]) 56
16 Gestion de bas niveau faite par le `hal.o` 68

1 Résumé du projet

1.1 Introduction

Selon le cahier des charges, le projet consiste à évaluer et étendre la gestion de la QoS du pilote des cartes wireless disposant des chipset ATHEROS®. Ce pilote s'appelle MadWiFi[1]. Ce pilote fonctionne sous linux, ce qui est un avantage car la source du kernel linux est libre (sous licence GPL) ce qui signifie qu'il est possible d'agir à tout niveau. De plus, les outils d'analyse et les informations issus du noyau linux permettent d'établir des diagnostics précis en matière d'analyse réseau. MadWiFi est principalement sous licence GPL excepté un fichier appelé hal (Hardware Access Layer). Ce dernier est fourni sous forme de binaire.

La QoS est principalement gérée au niveau IP. Les routeurs choisissent les paquets à router dans l'ordre de priorité (selon les champ DSCP). Cependant, au sein d'un réseau wireless, lorsque plusieurs transmissions sont en cours et que le nombre de stations faisant partie d'un seul ESSID n'est pas négligeable, il est nécessaire de coordonner l'accès au canal selon la priorité. Le but est d'assurer que les transmissions de type voix sur IP ou vidéo puissent obtenir un débit garanti.

Des solutions propriétaires pour gérer la qualité de service au sein d'ESSID existent déjà, mais actuellement, elle ne gère la priorité que dans un sens, du point d'accès à la station mobile. Aussi, il est nécessaire d'acheter un AP supportant cette gestion (matériel embarqué) ainsi qu'une carte de la même marque. Il est inutile de rappeler que l'utilisation d'un matériel provenant d'une autre marque ne sera pas compatible.

Actuellement, une norme conçue pour gérer la QoS au sein des WLAN basé sur IEEE802.11 est en phase d'élaboration. Il s'agit de la norme IEEE802.11e. Elle propose deux principales méthodes de coordination pour accéder au canal (ces deux méthodes sont complémentaires).

En utilisant le pilote MadWiFi pour intégrer cette gestion, il est possible de traiter à la fois l'AP ainsi que la station mobile car, sous linux, il est possible de transformer la station fixe en point d'accès, ceci avec le même pilote.

L'équipe de développement de MadWiFi a intégré dans le pilote une des deux méthodes de coordination spécifiée dans IEEE802.11e (il s'agit d'EDCA), ce qui a permis, lors l'élaboration de ce projet de diplôme, de tester EDCA dans des conditions réelles.

1.2 Analyses applicatives et tests

La mise en place d'un banc de test est indispensable si l'on veut apprécier le gain de performance selon EDCA. Le banc de test est composé de deux stations mobiles (laptop disposant d'un slot PCMCIA) équipées de la carte NetGEAR WAG511 (chipset ATHEROS®) AR5212) et d'une station fixe (DELL 4550, pentium 4) équipée d'une carte pci DLINK DWL-G520 (Chipset AR5212). Les tests ont étés fait selon la norme IEEE802.11b, car le poste de mesure est équipé d'une carte pouvant se mettre en mode monitor (cela permet de capturer toutes les trames de contrôle d'un WLAN tout en étant passif) ne peut dépasser la norme 802.11b (maximum 11Mbit/s).

Lors des tests il a été démontré que l'intégration de EDCA dans le pilote MadWiFi est fonctionnelle.

[1] MadWiFi : Multiband Atheros Driver for WiFi

1.3 Analyse du code source

L'analyse du code source de MadWiFi est relativement complexe du fait que tout est conçu afin de préserver les couches d'abstraction. Par exemple, sous linux, lorsque l'on utilise les commandes système pour configurer une carte wireless, elle est vue du système comme une carte wireless générique, type. La configuration de cette carte doit se faire avec les mêmes commandes que pour une autre carte. Or, au sein de la carte proprement dite, il y a des propriétés propres au chipset qu'on ne peut ignorer. C'est pourquoi, il y a toute une structure pour redéfinir, réagencer les fonctions provenant des couches supérieures afin qu'elle soit interprétables directement par le chipset.

Aussi, le code source du pilote se sert d'interfaces prédéfinies dans les sources du kernel. Par exemple, la notion de buffer (pour transporter le flux de donnée provenant des couches supérieures) est défini dans les sources du kernel linux.

L'analyse a permis de mettre en lumière la structure du pilote, son interaction avec les source du noyau linux, les relations avec le fichier binaire hal.o et les limitations qui en découlent.

Le choix du kernel linux porte finalement sur la version 2.6.9 (au départ 2.6.5, mais en cours de route, il a été possible d'utiliser la 2.6.9). Les sources étant libres sont téléchargeables sur `http://www.kernel.org/`.

1.4 Modification du code source et recherche de solution

Il a été possible, lors du projet, de rajouter des fonctionnalités de paramétrage visibles depuis un outil de configuration système (iwpriv). Ceci est très pratique car l'on peut activer, désactiver des sections que l'on a rajouter dans le code, sans recompiler ni recharger le pilote.

Il a également été possible de court-circuiter l'accès aux fonctions de bas-niveau traitées par le `hal.o` et ceci pourrait permettre d'intercaler un gestionnaire conçu sur mesure.

1.5 Outils de travail

Les outils utilisés sont :
- Ethereal
- packETH
- wireless-tools de Jean Tourrilhes (iwconfig, iwpriv, iwevent, iwlist, iwspy..).

Les outils développés sont :
- wlanConfig et son fichier de configuration wlanconfigrc (script pour auto-configurer chaque station wireless y compris l'AP)
- traitcapteth.sh est un script conçu pour convertir les captures ethereal exportés sous format texte en fichier csv importable depuis un tableur

Remarque : On peut donc constater que les outils utilisés sont libres et sous licence GPL ce qui autorise leur modification.

2 Article sur le support de la QoS dans WLAN[5]

2.1 Introduction

Citons d'abord quelques notions pour mieux comprendre les articles qui suivent.

2.2 Définition du MSDU

MAC Service Data Unit, contient les données des couches supérieures. Elle peut être fragmentée en plusieurs MPDU (MAC Protocol Data Unit). Les stations peuvent envoyer des MSDU de taille variable (jusqu'à 2304 bytes). Une station peut fragmenter un long MSDU en plusieurs MPDU.
Lorsque deux stations (ou plus) détectent qu'il n'y a plus de trafic sur le medium, elles risquent d'initier leur transmission en même temps, ce qui provoque des collisions. Pour réduire la probabilité de collision, DCF utilise le mécanisme appelé CA (Collision Avoidance), qui n'est autre que le rajout d'un temps aléatoire avant toute transmission (backoff).

2.3 Définition de la terminologie CFP-CP

Il s'agit des périodes de contention (CP) et des périodes libres de contention (CFP). Durant les CP, les stations sont en concurrence, durant une CFP, c'est l'inverse, le canal est réservé pour certaines transmissions prioritaires.

2.4 Définition de la terminologie PCF et DCF

PCF signifie Point Coordination Function. Ce système gère la coordination d'accès au canal de manière centralisé.

DCF signifie Distributed Coordination Function. La coordination se fait de manière décentralisée, chaque type de transmission doit attendre un temps défini selon sa priorité avant d'avoir accès au canal. Nous verrons plus loin que cette méthode est couramment utilisée car elle est conçue pour les périodes de contention (CP).

2.5 La QoS gérée par un coordinateur central (PCF)

2.5.1 Introduction

La norme 802.11 utilise PCF pour réserver des périodes. PCF fournit des mécanismes pour définir les priorités d'accès au médium. C'est un coordinateur central (appelé le Point Coordinator) qui va se charger de cette tâche. Il est situé dans l'AP. L'accès par PCF a une priorité supérieure à celle délivrée par DCF.

2.5.2 Contention-Free Period

Avec PCF, les CP[2] et CFP[3] s'alternent périodiquement et lorsque qu'un CFP suit un CP, on appelle ceci une SuperTrame (Superframe). PCF est utilisé pour accéder au medium durant une CFP alors que DCF est utilisé durant les CP. Une supertrame (superframe) doit inclure une CP pour une taille minimum d'un MSDU (échange d'une trame).
Une supertrame commence par un beacon, transmis par l'AP (on verra plus loin que cette trame devra comprendre les paramètres EDCA) et a pour but de **synchroniser** les stations ! Chaque station doit donc savoir quand viendra le prochain beacon car elles sont en mesure de consulter le TBTT[4] qui n'est autre la

[2] Période de contention, les stations "se battent" pour accéder au medium

[3] C'est une période libre de concurrence. Seule une station désignée pour une seule séquence de données a accès au medium, selon la priorité. Par exemple de la voix

[4] Target Beacon Transmission Time : C'est le temps à attendre avant de voir le prochain beacon

durée séparant le beacon actuel avec le prochain beacon et ce champ doit être visible dans chaque beacon. Durant la CFP, seuls une QSTA[5] et le QAP[6] peuvent occuper le medium (WM[7]).

Le PC[8] va effectuer un polling vers la station prioritaire pour lui demander de transmettre son paquet en attente (s'il y en a un) et si, du côté du PC[9] un paquet prioritaire est destiné à cette station, le PC profitera pour l'insérer dans la trame de polling. Ainsi, il n'y a pas d'échange de trame inutile. Dès le moment où la station concernée reçoit le polling, elle enverra un acquittement en même temps qu'un éventuel MSDU. Si le PC ne reçoit aucune réponse de la QSTA «pollée», ceci après un PIFS[10], le PC va «poller» la station suivante. Il ne peut y avoir d'inutilisation du médium dépassant un PIFS durant un CFP. De plus, du fait que $SIFS < PIFS < DIFS$, une trame de polling ne peut interrompre un acquittement car ce dernier est prioritaire[11].

Remarque : dans le cas ou certains acronymes de sont pas explicités au bas de la page, consulter le glossaire, page 74.

2.5.3 Problèmes non résolus avec PCF

Le but est de garantir des CFP. Cependant, on a le risque d'avoir des délais imprévisibles pour les beacon et des durées de transmission inconnues pour les stations «pollées». Dans un TBTT, l'AP, en l'occurence le PC, prédit précisément le moment où le prochain beacon tombera, mais le beacon ne peut être transmis au WM que si le canal a été libéré après un PIFS. Or, les stations peuvent commencer leur transmission même si là, l'envoi du MSDU ne terminera pas avant le prochain TBTT. Ceci va donc retarder la transmission du prochain beacon par rapport au TBTT défini.

Un autre problème avec PCF est le temps de transmission indéterminé des stations «pollées». En effet, une station «pollée» a le droit de délivrer un MSDU fragmenté de manière arbitraire (jusqu'à 2304 bytes et 2312 bytes avec cryptage WEP). Aussi, différentes modulations sont définies avec 802.11a[12], ce qui aura pour effet que le MSDU délivré après polling sort du contrôle du PC, ce qui va par conséquent réduire la QoS réservée pour les autres stations durant le reste du CFP.

[5] QoS Station : Station mobile capable de comprendre la gestion de QoS
[6] QoS Access Point : Point d'accès capable de gérer la QoS
[7] WM : Wireless Medium
[8] **PC** : Point Coordinator
[9] PC : PCF Coordinator, c'est l'organe central gérant la coordination selon la méthode PCF
[10] PCF Interframe Space : temps réglementaire à attendre, après libération du canal, pour une trame PCF avant d'avoir accès au canal
[11] Du fait qu'une trame d'acquittement intervient après un SIFS, elle est prioritaire à une trame de polling devant attendre un PIFS
[12] L'article a axé ses exemples sur la norme 802.11a, mais ce genre de cas de figure est transposable avec 802.11g ou b

3 HCF résumé de la norme HCF (p 63[3])

3.1 Introduction

Hybrid Coordination Function est un système de gestion de la QoS basé sur DCF (Distribution Coordination Function) et PCF (Point Coordination Function), il s'agit donc d'une technique hybride. HCF utilise à la fois une méthode basée sur l'accès au canal durant la période de contention appelée EDCA (Enhanced Distributed Channel Access) et un mécanisme de contrôle d'accès au canal pour gérer les transferts durant les périodes libres de contention, appelé HCCA (HCF Controlled Channel Access).

Les QSTAs[13] reçoivent leur TXOPs[14]. Si l'autorisation d'émettrre (TXOP) est obtenue durant une période de contention (CP), c'est selon EDCA ou HCCA que l'autorisation est obtenue[15]. Si, par contre, l'opportunité d'émettre est obtenue durant une période de libre contention (CFP), le TXOP a été **uniquement** défini par la méthode HCCA.

Remarque : Afin de bien clarifier l'usage des différents mécanismes, EDCA est utilisé **uniquement** lorsque l'on est en période de contention (CP), HCCA peut-être utilisé dans les deux cas (CP et CFP), c'est-à-dire également en période de libre contention. Finalement **HCF** utilise à la fois des mécanismes de **PCF** et de **DCF**, ce qui explique son surnom de *hybride*.

3.2 HCF contention-based channel access (EDCA)

3.2.1 Introduction

Il s'agit ici de définir le mécanisme qui va gérer les transferts lors d'une période de contention. Pour ce faire, on définit huit niveaux de priorité et quatre catégories d'accès. On peut le voir sur le tableau (TAB. 1) à la page 8.

Priorité	Désignation	Catégorie	Description
1	BK	AC_BK	Background
2	-	AC_BK	Background
0	BE	AC_BE	Best Effort
3	EE	AC_BE	Best Effort
4	CL	AC_VI	Video
5	VI	AC_VI	Video
6	VO	AC_VO	Voice
7	NC	AC_VO	Voice

TAB. 1 – Liste des UPs[16] et des ACs[17][3]

Pour chaque AC, une variante améliorée de DCF, appelée "Fonction d'accès au canal"[18], contestera les TXOPs utilisant un set de paramètres EDCA défini dans le «`EDCA Parameter Set element`» ou défini avec des valeurs par défaut lorsqu'aucun de ces élément de coordination n'est reçu du QAP appartenant au QBSS[19] auquel notre station est associée et cette règle est valable dans les cas suivants :

[13] QSTAs : stations supportant le mécanisme de gestion de la QoS en WLAN
[14] TXOP : Opportunité d'émettre, plus simplement en recevant cette autorisation ils ont la liberté d'émettre de la part du QAP
[15] HCCA peut également avoir fourni l'autorisation car il peut également être utilisé en CP
[18] Traduit littéralement de l'anglais "channel access function"
[19] QBSS : QoS BSS, réseau WLAN supportant la QoS

a) Lorsque les paramètres utilisés par la "fonction d'accès au canal" pour contrôler l'opération sont définis dans la table "`dot11QAPEDCATable`" du **MIB** du QAP et dans la station QSTA (non-AP)

b) Lorsque le temps minimum d'attente n'est plus la constante DIFS mais une valeur déterminée dans table de la MIB `dot11EDCATableAIFSN` du QSTA (non-AP).

c) Lorsque les limites[20] de la fenêtre de contention (CW), dans laquelle le "random backoff"[21] est calculé, ne sont pas déterminées par la couche PHY mais sont variables et contenues dans la table de la MIB du QAP (attribut : `dot11QAPEDCACWmin` et `dot11QAPEDCACWmax`) ainsi que dans la table du QSTA (attribut : `dot11EDCATableCWmin` et `dot11EDCATableCWmax`). Ces valeurs sont assignées par une entité de gestion ou un QAP.

d) Lorsque les collisions entre les fonctions d'accès au canal de contention entre les **QSTA** sont résolues par ordre de priorité selon l'**AC** de la trame.

e) Durant l'utilisation d'EDCA, le TXOP est obtenu par une fonction d'accès au canal, une QSTA doit envoyer les trames multiples avec la même catégorie d'accès. Le nombre de trames possibles pour un envoi est défini dans l'attribut `dot11QAPEDCATXOPLimit` de la **MIB** du **QAP** et dans l'attribut `dot11EDCATableTXOPLimit` de la **QSTA**. La valeur **0** signifie que le EDCA TXOP est limité à un seul MSDU[22] ou MMPDU.

Le QAP émet les paramètres EDCA dans des beacons sélectionnés, et, dans toutes les réponses et (ré) associations, les trames doivent contenir le set de paramètres EDCA (EDCA Parameter Set information element). Si par contre le QAP ne reçoit en retour aucun élément EDCA, les QSTAs devront utiliser les valeurs définies par défaut. Les champs d'information suivant le champ d'information QoS du EDCA Parameter Set doivent toujours être inclus dans tous les beacons contenus dans l'intervalle de deux ou plus DTIM[23] suivant un changement d'AC, ceci afin de garantir que toutes les stations soient aptes à recevoir les nouveaux paramètres EDCA.
Une QSTA doit donc mettre à jour les paramètres EDCA dans sa table MIB. Un QAP pourrait changer les paramètres d'accès EDCA en modifiant le l'élément de paramètre EDCA (appelé **EDCA Parameter Set element**) dans le beacon et attend ensuite une réponse.

Cependant, les QAP ne devraient que rarement changer les paramètres EDCA. Pour savoir si ces derniers ont changé, la QSTA doit simplement comparer la *EDCA Parameter Set Update Count Value* (un sous-champs de la "QoS Capability") avec celle stockée dans sa table. Si elles diffèrent, alors la QSTA doit envoyer un "Probe Request", c'est-à-dire une demande de mise à jour au QAP. Cette technique économise de la bande passante ne demandant la retransmission que dans le cas où l'on découvre une différence entre les deux set de paramètres.
La gestion des types de trame **doit se faire en utilisant la catégorie d'accès AC_VO sans aucune restriction** de la part de procédures de contrôle d'admission, c'est-à-dire en utilisant la plus haute priorité. Une QSTA doit donc envoyer ses trames de gestion (management frame) en utilisant la catégorie d'accès AC_VO avant de s'associer avec un BSS même si ce BSS ne supporte pas la QoS. Les trames de contrôle **BlockAckReq** et **BlockAck** doivent être envoyées en utilisant les mêmes paramètres QoS que pour les trames (QoS) de données. Afin de réduire la probabilité de collision, les trames de contrôle PS-Poll doivent être envoyées avec la catégorie d'accès AC_BE (Best Effort). Dans le but de déterminer l'AC pour une trame RTS[24], la trame RTS doit hériter de l'UP[25] de la trame de donnée ou de contrôle qui est incluse dans la séquence d'échange dont la première trame est le RTS.

[20] Les limites de la CW sont appelées `aCWmin` et `aCWmax`

[21] Le random backoff est le temps aléatoire rajouté aux temps définis par les règles de synchronisation afin de diminuer le risque de collision

[22] MAC Service Data Unit

[23] DTIM : C'est le temps définis séparant chaque beacon

[24] RTS/CTS : Request-to-send, clear-to-send, méthode de réservation de canal utilisée afin d'éviiter que des stations qui ne se voient pas se collisionnent

[25] UP : User Priority

3.2.2 EDCA TXOPs

Il y a deux modes de TXOPs définis. Le premier est EDCA TXOP, il apparait lorsque les règles de EDCA autorisent l'accès au WM. Le second mode, **continuation EDCA TXOP** (littéralement traduit le mode **continu**), est utilisé lorsque la fonction d'accès au canal ne donne pas le droit d'y accéder, ceci après une séquence d'échange de trames nécessitant la réception d'une trame d'acquittement (Ack frame). La durée limite d'un TXOP est fournie par le QAP, dans l'EDCA Parameter Set Information Element. Lorsque l'on a comme valeur limite 0, cela signifie qu'un simple MSDU (ou MMPDU), en plus un échange RTS/CTS ou simplement CTS, peut être transmis à n'importe quel taux (débit) pour chaque TXOP. Les QSTAs (non-AP) doivent s'assurer que la durée des TXOPs obtenus n'excède pas la limite TXOP définie (dans les tables). En effet, une station ayant l'autorisation d'émettre en priorité durant une CFP a un temps maximum limité à disposition.
Une QSTA doit fragmenter un MSDU afin que la transmission du premier MPDU ne fasse pas que la limite du TXOP n'excède le taux fixé par la couche PHY pour la transmission initiale.

3.3 HCF controlled channel access (HCCA)

3.3.1 Introduction

Le mécanisme HCCA utilise un coordinateur central (HC) qui opère selon des règles différentes du PC[26]. Le HC est intégré au QAP du QBSS et dispose d'une grande priorité d'accès sur le WM[27] lorsqu'il veut établir une séquence d'échanges de trames ainsi que pour allouer les TXOPs.

3.3.2 CAP/CFP/CP

L'intérêt de notre technique hybride est de garantir des laps de temps réguliers libres de contention. Durant cette période appelée CFP, seul le trafic prioritaire nécessitant une qualité de service aura accès au WM. C'est le HC qui va déterminer les CFP et les CP. Avant de démarrer une CFP ou de donner une TXOP en CP, il doit déterminer d'abord si le WM est libre. Ce dernier est considéré comme libre s'il n'y pas d'activité après un PIFS[28]. Après un PIFS, le HC doit alors transmettre la première trame autorisée de la séquence d'échange, avec une durée[29] couvrant la CFP ou le TXOP délivré (durant le CP). Cependant, la première trame autorisée durant une CFP après un TBTT[30] n'est ni plus ni moins un Beacon. On peut voir le chronogramme repris de la norme[3] à la figure 1, à la page 11.

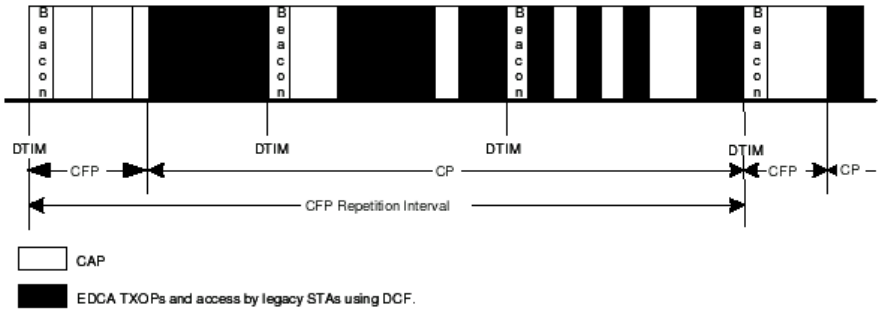


FIG. 1 – CAP/CFP/CP periods[3]

Durant une CFP, ou, durant une TXOP pendant une CP, après chaque trame de données ou de contrôle ayant un groupe d'adresses dans le champ d'adresse 1[31], le coordinateur (HC) doit attendre au moins un PIFS et ne peut continuer à transmettre que si le WM est libre. Après la dernière trame de chaque séquence (non finale), le possesseur du TXOP doit attendre un SIFS avant de transmettre la première trame de la prochaine séquence. Le HC doit réclamer le canal un PIFS après le TXOP à condition qu'il soit libre de trafic. Un CAP[32] termine lorsque le HC ne réclame pas le canal après un PIFS à la fin d'un TXOP.

Cette explication a été tirée de la page 77 du draft [3].

[26] **PC** : Point Coordinator, coordinateur selon le mécanisme PCF
[27] Wireless Medium
[28] PIFS : PCF Interframe space
[29] Duration value
[30] TBTT : Target Beacon Transmission Time
[31] Une station utilise le contenu du champ d'adresse 1, voir page 30 du draft 8[3]
[32] Controlled access phase

4 Analyse du driver Madwifi

4.1 Introduction

En se rapportant à l'analyse préalable effectuée par Dr. Lassaâd Gannoune et Dr. Stephan Robert[6], nous savons déjà que l'architecture principale est basée sur trois principaux modules :

1) ath_hal le noyau dur gérant le matériel (chipset Atheros), la puissance, la fréquence, etc.
2) ath_pci module faisant le pont entre les fonctionnalités WLAN et le noyau
3) wlan module intégrant les principes du wireless (norme)

Selon le rapport intermédiaire, ils interagissent selon le schéma (FIG. 2) à la page 12)

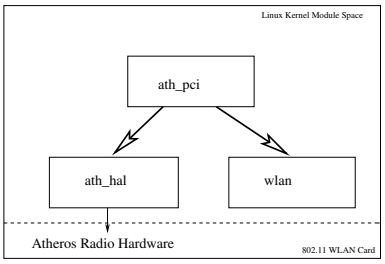


FIG. 2 – Schéma global

4.2 Matériel supporté

Avant de commencer l'étude et la modification du driver, il est nécessaire de voir dans le détail quels sont les chipsets supportés et sur quelles plateformes. En consultant le site de MadWiFi à la page `http://www.mattfoster.clara.co.uk/madwifi-hw.htm`, on y voit une liste de cartes disposant d'un chipset Atheros. L'avantage de travailler dans l'univers linux est que les drivers ne se préoccupent que du chipset utilisé, ce qui permet d'acheter une carte d'une marque quelconque (économie financière) et de bénéficier des mêmes performances qu'une carte d'une marque mieux cotée mais sans raison apparente plus coûteuse. Pour notre plateforme, le choix s'est porté sur une carte PCI DLINK-DWL520 (chipset Atheros AR5212) et d'une carte PCMCIA(PC-CARD) Netgear-WAG511 (Chipset AR5212). Les deux cartes fonctionnent avec les normes 802.11a,b,g.

Si l'on consulte `{$MADWIFIROOT}/hal/linux/Makefile.inc`, on découvre les lignes :

```
1  # AH_SUPPORT_AR5210     802.11a only MAC
2  # AH_SUPPORT_AR5211     802.11a+802.11b MAC (also pure g, but not yet supported)
3  # AH_SUPPORT_AR5212     802.11a+802.11b+802.11g MAC
4  # AH_SUPPORT_AR5312     5212 MAC WiSoC (requires AH_SUPPORT_AR5212)
```

Aussi, en listant ce répertoire, on remarque que le binaire (fichier objet .o) est disponible pour les processeurs : arm9, armv4, i386 (ce que nous utilisons), mips1, mipsisa32, powerpc, x86_64 et xscale, ce qui permet de voir que le driver est relativement bien porté sur différentes plateformes.

4.3 Kernel

Le driver est développé pour fonctionner sous linux. Actuellement il existe deux générations du kernel en constante évolution, le kernel v2.4 et le kernel v2.6. Leurs architectures internes diffèrent, le 2.6 étant nettement plus évolué, les divers projets tendent à être développés sur la nouvelle architecture. Le choix du

kernel va donc se porter sur le kernel 2.6. Aussi, en suivant, en parallèle l'évolution de l'implémentation MIPL[33] (Mobile IPv6), on remarque que la seconde version de ce MIPL tournera sur linux kernel v2.6.

Le kernel 2.6 est actuellement à la version 2.6.9. Le driver a été compilé sans problème jusqu'à la version 2.6.5. Au delà de la version 2.6.5, il n'est plus possible de compiler MadWiFi sans y apporter quelques modifications. Il existe un patch pour pallier à ce problème. Les informations relatifves aux patchs sont disponibles sur `http://www.mattfoster.clara.co.uk/madwifi-3.htm`. Comme le projet consiste en des modifications sensibles sur le driver, le choix portera sur une version d'origine du driver. Nous nous concentrerons sur un kernel 2.6.5.

4.4 Téléchargement et suivi du développement des sources de MadWiFi

La façon la plus efficace de suivre l'évolution du pilote est de télécharger les sources sur le serveur CVS. On créé d'abord un répertoire dans lequel on stocke les sources du pilote. Depuis un terminal (par ex. xterm), on se rend dans le répertoire en question via la commande `cd`. Ensuite on effectue les operations suivantes :

- cvs -d :pserver :anonymous@cvs.sourceforge.net :/cvsroot/madwifi login
- cvs -z3 -d :pserver :anonymous@cvs.sourceforge.net :/cvsroot/madwifi co modulename

modulename peut-être le répertoire courant "." ou, par exemple, madwifi, ce qui téléchargera juste le répertoire madwifi. Au départ, il est préférable de tout télécharger, donc de mettre un "." en lieu et place du **modulename**.

Au stade actuel, les sources du pilote sont chargées. Dans le répertoire racine que nous avions préalablement créer demeure un répertoire généré automatiquement, il s'agit du répertoire CVS comportant toutes les informations nécessaires concernant le projet cvs chargé dans le répertoire courant. Si l'on désire mettre à jour notre pilote, il n'est plus nécessaire d'effectuer les deux opérations préalablement décrites mais simplement de taper :

- `cvs diff` (pour obtenir les différences entre les fichiers du répertoire courant et le projet stocké sur sourceforge.net)
- `cvs update -d` (pour effectuer la mise à jour (différentielle) et l'argument "-d" permet de créer automatiquement les répertoires qui auraient été ultérieurement ajoutés sur le projet par rapport à notre version obsolète du pilote.

Le driver est dans le répertoire madwifi. Un fichier README détaille la procédure à suivre pour compiler le noyau. L'opération se résume en une ligne :
`make clean ; make KERNELPATH=/usr/src/linux<version> ; make install`

Cependant, il est conseillé de lire le fichier d'aide. Il est possible d'intégrer le pilote au noyau sans le compiler en tant que module. Toutefois, il est déconseillé de le faire tant que le driver n'est pas reconnu parfaitement stable (par exemple, le mode ad-hoc est totalement instable).

[33] Le projet de semestre était basé sur l'implémentation MIPL. Le projet actuel est essentiellement axé sur la gestion à bas niveau, mais par la suite il serait intéressant de tester MIPL sur un QWLAN

4.5 Architecture grossière du driver

4.5.1 Introduction

Afin d'approcher la structure du driver et d'identifier le rôle de chaque fichier source, il est nécessaire de voir, dans un premier temps, dans quel répertoire chacun d'eux est placé. Voici les répertoires principaux :

```
MadWiFi
|--ath                  Contient la gestion de l'interface PCI et l'interaction entre les données
|                       destinées à la carte et le kernel
|--ath_hal              Contient des scripts pour la compilation
|--ath_rate             Contient les algorithmes de gestion du taux de transfert
|--driver
|--hal                  Contient les binaires précompilés (sous forme de fichier UUencode)
|--include
|--net80211             Contient les sources définissant ce qui se rapporte à IEEE802.11
|--patches
|--tools                Contient des outils d'analyse, statistiques
|--wlan
```

Nous ferons régulièrement référence aux différents fichiers composant le pilote, afin de faciliter la recherche des ces fichiers, l'arborescence suivante permet de voir comment les sources sont classées :

```
madwifi/
|-- COPYRIGHT
|-- Makefile
|-- Makefile.inc
|-- README
|-- ath
|   |-- Kconfig
|   |-- Makefile
|   |-- Makefile.kernel
|   |-- if_ath.c
|   |-- if_ath_pci.c
|   |-- if_ath_pci.h
|   |-- if_athioctl.h
|   |-- if_athrate.h
|   |-- if_athvar.h
|   `-- version.h
|-- ath_hal
|   |-- Makefile
|   `-- Makefile.kernel
|-- ath_rate
|   |-- amrr
|   |   |-- Makefile
|   |   |-- Makefile.kernel
|   |   |-- amrr.c
|   |   `-- amrr.h
|   `-- onoe
|       |-- Kconfig
|       |-- Makefile
|       |-- Makefile.kernel
|       |-- onoe.c
|       `-- onoe.h
|-- driver
```

Dans notre étude préliminaire, soulignons déjà l'importance des fichiers `if_ath.c`, `if_ath_pci.c`, `if_athvar.h`. Nous verrons par la suite, que ces fichiers définissent l'interface faisant le pont entre le kernel et le firmware[34].

Voici maintenant la suite de l'arborescence, on y voit le `hal.o` précompilé pour plusieurs plateformes :

```
|-- hal
|   |-- COPYRIGHT
|   |-- README
|   |-- ah.h
```

[34] Au stade actuel, nous présumons que le binaire `hal.o` dont la source est fermée contient un firmware se chargeant dans la carte afin d'y effectuer la gestion au plus bas niveau des transmissions entre la couche physique et la couche MAC.

```
|   |-- ah_desc.h
|   |-- ah_devid.h
|   |-- linux
|   |   |-- Makefile.inc
|   |   |-- ah_osdep.c
|   |   |-- ah_osdep.h
|   |   |-- arm9-le-thumb-elf.hal.o.uu
|   |   |-- arm9-le-thumb-elf.inc
|   |   |-- arm9-le-thumb-elf.opt_ah.h
|   |   |-- armv4-be-elf.hal.o.uu
|   |   |-- armv4-be-elf.inc
|   |   |-- armv4-be-elf.opt_ah.h
|   |   |-- armv4-le-elf.hal.o.uu
|   |   |-- armv4-le-elf.inc
|   |   |-- armv4-le-elf.opt_ah.h
|   |   |-- i386-elf.hal.o.uu
|   |   |-- i386-elf.inc
|   |   |-- i386-elf.opt_ah.h
|   |   |-- mips-be-elf.hal.o.uu
|   |   |-- mips-be-elf.inc
|   |   |-- mips-be-elf.opt_ah.h
|   |   |-- mips-le-elf.hal.o.uu
|   |   |-- mips-le-elf.inc
|   |   |-- mips-le-elf.opt_ah.h
|   |   |-- mips1-be-elf.hal.o.uu
|   |   |-- mips1-be-elf.inc
|   |   |-- mips1-be-elf.opt_ah.h
|   |   |-- mips1-le-elf.hal.o.uu
|   |   |-- mips1-le-elf.inc
|   |   |-- mips1-le-elf.opt_ah.h
|   |   |-- mipsisa32-be-elf.hal.o.uu
|   |   |-- mipsisa32-be-elf.inc
|   |   |-- mipsisa32-be-elf.opt_ah.h
|   |   |-- mipsisa32-le-elf.hal.o.uu
|   |   |-- mipsisa32-le-elf.inc
|   |   |-- mipsisa32-le-elf.opt_ah.h
|   |   |-- powerpc-be-eabi.hal.o.uu
|   |   |-- powerpc-be-eabi.inc
|   |   |-- powerpc-be-eabi.opt_ah.h
|   |   |-- powerpc-le-eabi.hal.o.uu
|   |   |-- powerpc-le-eabi.inc
|   |   |-- powerpc-le-eabi.opt_ah.h
|   |   |-- x86_64-elf.hal.o.uu
|   |   |-- x86_64-elf.inc
|   |   |-- x86_64-elf.opt_ah.h
|   |   |-- xscale-be-elf.hal.o.uu
|   |   |-- xscale-be-elf.inc
|   |   `-- xscale-be-elf.opt_ah.h
|   `-- version.h
|-- include
|   |-- compat.h
|   `-- sys
|       `-- queue.h
```

Dans l'arborescence qui suit, nous verrons plus loin dans le rapport que ces fichiers définissent les trames IEEE802.11, le paramètrage, etc. :

```
|-- net80211
|   |-- Kconfig
|   |-- Makefile
|   |-- Makefile.kernel
|   |-- eapol.h
|   |-- ieee80211.c
|   |-- ieee80211.h
|   |-- ieee80211_acl.c
|   |-- ieee80211_crypto.c
|   |-- ieee80211_crypto.h
|   |-- ieee80211_crypto_ccmp.c
|   |-- ieee80211_crypto_none.c
|   |-- ieee80211_crypto_tkip.c
|   |-- ieee80211_crypto_wep.c
|   |-- ieee80211_dot1x.c
|   |-- ieee80211_dot1x.h
|   |-- ieee80211_input.c
|   |-- ieee80211_ioctl.h
|   |-- ieee80211_linux.c
|   |-- ieee80211_linux.h
```

```
|   |-- ieee80211_node.c
|   |-- ieee80211_node.h
|   |-- ieee80211_output.c
|   |-- ieee80211_proto.c
|   |-- ieee80211_proto.h
|   |-- ieee80211_radiotap.h
|   |-- ieee80211_radius.c
|   |-- ieee80211_radius.h
|   |-- ieee80211_var.h
|   |-- ieee80211_wireless.c
|   |-- ieee80211_xauth.c
|   |-- if_ethersubr.h
|   |-- if_llc.h
|   |-- if_media.c
|   |-- if_media.h
|   |-- if_wavelan_ieee.h
|   |-- linux
|   |-- md5.c
|   |-- md5.h
|   |-- rc4.c
|   |-- rc4.h
|   |-- test_ccmp.c
|   |-- test_tkip.c
|   |-- test_wep.c
|   `-- version.h
|-- patches
|   |-- 2.4
|   |   |-- Config.in.patch
|   |   |-- Config.in.wireless-2.4.24.patch
|   |   |-- Configure.help.patch
|   |   |-- Makefile.patch
|   |   `-- install.sh
|   |-- 2.6
|   |   |-- Kconfig.patch
|   |   |-- Makefile.patch
|   |   `-- install.sh
|   `-- README
|-- release.h
|-- tools
|   |-- 80211stats.c
|   |-- Makefile
|   |-- athchans.c
|   |-- athctrl.c
|   |-- athkey.c
|   `-- athstats.c
`-- wlan
```

4.5.2 Répertoire ath

Ce répertoire contient les fichiers `ath_pci.c if_ath_pci.c if_ath_pci.h`. Il contient également quelques includes `if_athioctl.h if_athrate.h if_athvar.h` contenant des définitions, des paramètres d'accès à la carte. Cet étage (`if_ath_pci`) permet de gérer les accès au bus pci, à la carte. Dans le fichier `if_athrate.h`, on définit le débit du flux. Il est expliqué que pour chaque interface montée, une instance du module de gestion des taux est chargée :

```
* Interface definitions for transmit rate control modules for the
 * Atheros driver.
 *
 * A rate control module is responsible for choosing the transmit rate
 * for each data frame.  Management+control frames are always sent at
 * a fixed rate.
 *
 * Only one module may be present at a time; the driver references
 * rate control interfaces by symbol name.  If multiple modules are
 * to be supported we'll need to switch to a registration-based scheme
 * as is currently done, for example, for authentication modules.
 *
 * An instance of the rate control module is attached to each device
 * at attach time and detached when the device is destroyed.  The module
 * may associate data with each device and each node (station).  Both
 * sets of storage are opaque except for the size of the per-node storage
 * whic * The rate control module is notified for each state transition and
 * station association/reassociation.  Otherwise it is queried for a
 * rate for each outgoing frame and provided status from each transmitted
 * frame.  Any ancillary processing is the responsibility of the module
 * (e.g. if periodic processing is required then the module should setup
 * it's own timer).
 *
 * In addition to the transmit rate for each frame the module must also
 * indicate the number of attempts to make at the specified rate.  If this
 * number is != ATH_TXMAXTRY then an additional callback is made to setup
 * additional transmit state.  The rate control code is assumed to write
 * this additional data directly to the transmit descriptor.
 */h must be provided when the module is attached.
 *
```

4.5.3 Répertoire ath_hal

Contient un Makefile, le fichier `opt_ah.h` dont le contenu cite les divers chipsets supportés :

```
#define AH_SUPPORT_AR5210 1
#define AH_SUPPORT_AR5211 1
#define AH_SUPPORT_AR5212 1
```

Nous verrons plus loin, que, pour support WME et les quatres files d'attentes `AC_VO`, `AC_VI`, `AC_BE` et `AC_BK`.

4.5.4 Répertoire ath_rate

Ce répertoire contient `amrr.h amrr.c`. Il s'agit d'une implémentation de l'algorithme AMRR[35] qui permet de gérer l'adaptation du débit du flux afin de le maximiser. D'après le commentaire contenu dans le fichier, on y trouve l'information :

```
 * AMRR rate control. See:
 * http://www-sop.inria.fr/rapports/sophia/RR-5208.html
 * "IEEE 802.11 Rate Adaptation: A Practical Approach" by
 *    Mathieu Lacage, Hossein Manshaei, Thierry Turletti
```

Voici l'explication telle qu'elle est donnée à l'url mentionné dans l'encadré ci-dessus, concernant l'algorithme de controle du flux :

[35] AMRR : Adaptive Multi Rate Retry

Les trois couches physiques IEEE 802.11a/b/g disponibles aujourd'hui offrent toutes des services multi-modes. Afin d'obtenir des performances élevées dans des conditions de transmission variables, les systèmes qui utilisent ces couches physiques doivent continuellement changer le mode de transmission utilisé pour maximiser le débit disponible au niveau applicatif. Bien que les algorithmes de sélection de mode constituent un composant important de la performance de ces réseaux locaux sans fil, aucun algorithme à l'exception de ARF (Auto Rate Fallback) et de RBAR (Receiver Based Auto Rate) n'a été publié et les problèmes fondamentaux liés à l'implémentation de tels algorithmes n'ont jamais été pris en considération dans des articles publiés. Nous présentons donc le paramètre de ces systèmes 802.11 qui a la plus grande influence sur la conception d'algorithmes de contrôle de mode de transmission et nous identifions deux types de systèmes : les systèmes à faible latence et ceux à forte latence de communication entre les couches physiques et les couches MACs. Puis, nous proposons deux nouveaux algorithmes de contrôle de mode de transmission dont les performances approchent l'optimum représenté par RBAR et qui sont, contrairement à RBAR, réalisables aujourd'hui sans modifications incompatibles du standard 802.11. AARF (Adaptive ARF) et AMRR (Adaptive Multi Rate Retry), conçus respectivement pour les système dis à faible latence et forte latence, ont été implémentés et simulés à l'aide de ns. Nous avons aussi évalué une implémentation de AMRR dans un driver Linux pour cartes 802.11 basées sur le chipset AR5212 qui confirme les résultats de simulation et montre une amélioration importante du débit obtenu au niveau applicatif.

Dans le même répertoire (`ath_rate`), il demeure encore une autre section gérant le flux (onoe). A ce propos, lorsque l'on charge les divers modules du pilote, un module nommé `ath_rate_onoe` se charge également.

4.5.5 driver

(vide)

4.5.6 Répertoire hal

Comme décrit dans le rapport intermédiaire[6], ce répertoire contient la gestion du chipset proprement dit. La gestion de la puissance, des fréquences (canaux), des temps (timing, définition des périodes). La source n'est pas disponible car elle est précompilée pour toute plateforme, mais on a accès à une interface `ah.h` qui permet de définir les paramètres que l'on désire (avant compilation). C'est le moteur central du driver ayant accès au plus bas niveau.

4.5.7 Répertoire include

Contient un répertoire "Sys" dans lequel il y a un fichier queue.h. Ce fichier offre une structure de donnée (queue) complète. En ce qui concerne le deuxième fichier "compat.h", sa fonctionnalité n'est pas encore déterminée. Il semblerait qu'il s'agisse d'un outil permettant de déterminer la compatibilité avec la source du kernel choisi.

4.5.8 Répertoire net80211

C'est ici que se trouve la description des trames, la liste des paramètres pour configurer la carte (avec iwconfig et iwpriv). On verra plus loin dans le rapport et plus en détail les parties qui sont déterminantes pour le projet.

4.5.9 Répertoire patches

Ce répertoire contient les patchs qui vont être appliqués au kernel, respectivement v2.4 ou 2.6. Dans le répertoire racine de la source du kernel linux, ces patchs créeent les répertoires suivants :

- net/net80211
- drivers/net/wireless/_ath_hal
- drivers/net/wireless/ath
- drivers/net/wireless/hal

Ces patchs sont utiles lorsque l'on désire intégrer le pilote au sein du kernel et, au lieu de charger un module à chaque fois que l'on veut utiliser la carte, le kernel supporte intrinsèquement le matériel.

Il est préférable de conserver le driver "expérimental" en module pour les raisons suivantes :

1. Si le driver est instable, le kernel peut le devenir. On ne charge le module qu'en cas de nécessité, lorsqu'on utilise le matériel.
2. Par expérience personnelle mais sans véritable "preuve" à l'appui, le fait de décharger et recharger le module permet de réinitialiser la carte et la remettre dans l'état initial et charge les paramètres par défaut selon les scripts prédéfinis. Aussi, dans le cas de la carte PCMCIA, le fait de retirer la carte lance un script qui va décharger le module proprement, et lorsque l'on remet la carte la détection d'un nouveau périphérique PCMCIA va lancer un script[36] qui chargera les modules et mettra les paramètres par défaut. La carte est dans son état initial, prête à fonctionner.
3. Pour notre projet, comme il est nécessaire de modifier, compiler, tester le pilote, il nous est indispensable de pouvoir le décharger et recharger **rapidement**, ce qui n'est possible que si le pilote est sous forme de module. Dans le cas contraire, nous serions systématiquement contraint de recompiler le noyau linux, de le réinstaller, de rélancer lilo pour finalement redémarrer l'ordinateur.

4.5.10 Répertoire tools

Ce répertoire contient des outils d'analyse (statistiques) de la carte lors de son utilisation. La commande **athstats** est un moniteur qui indique le nombre d'erreurs crc, les débit E/S, etc. Voici un exemple :

```
blues:/usr/src/prginst/madwifi-test/madwifi/tools# ./athstats ath0
   input   output altrate   short    long xretry crcerr  crypt  phyerr rssi rate
      22        9       0       0       0      0     81      0      49   35  11M
       1        1       0       0       0      0      0      0       0   36  11M
       1        1       0       0       0      0      0      0       0   36  11M
```

4.5.11 Répertoire wlan

Ce répertoire ne contient rien, même après avoir compilé le pilote. Il n'a apparament pas d'utilité directe.

[36] Sous debian, pour notre application, il s'agit de : `/etc/pcmcia/wireless`

4.6 Analyse syntaxique

4.6.1 Introduction

La première analyse consiste en une simple analyse syntaxique, c'est une première approche qui nous permettra de comprendre l'interdépendance entre les divers modules.

4.6.2 ath/ah.h

En analysant les divers fichiers, on remarque que le fichier `hal/ah.h` et `hal/linux/opt_ah.h` sont inclus par les fichiers `ath/if_ath_pci.c` et `hal/linux/ah_osdep.c`. On peut voir cette interdépendance à la figure 3 à la page 20.

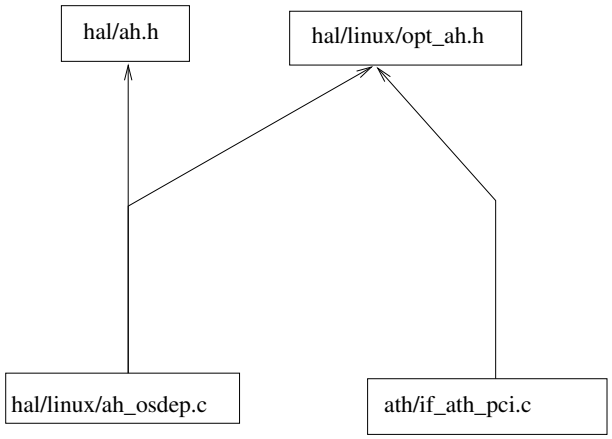


FIG. 3 – Interdépendance de `ah.h`, interface avec le HAL

4.6.3 Liste des dépendances

Les fichier sources ont été parcourus afin de déterminer quelles est l'importance des fichiers .h. Dans le fichier qui suit, le fichier inclus est en première ligne (après chaque séparation) et les fichiers qui l'incluent suivent le symbole : "`|-->`" :

```
1  ----------------------------
2  ./madwifi/ath/if_ath_pci.h
3  |-->./madwifi/ath/if_ath.c
4  ----------------------------
5  ./madwifi/ath/if_athioctl.h
6  |-->./madwifi/tools/athstats.c
7  ----------------------------
8  ./madwifi/ath/if_athrate.h
9  | -
10 ----------------------------
11 ./madwifi/ath/if_athvar.h
12 |-->./madwifi/ath/if_ath.c
13 |-->./madwifi/ath/if_ath_pci.c
14 |-->./madwifi/ath_rate/amrr/amrr.c
15 |-->./madwifi/ath_rate/onoe/onoe.c
16 |-->./madwifi/ath/version.h
17 ----------------------------
18 ./madwifi/hal/linux/ah_osdep.h
19 |-
20 ----------------------------
21 ./madwifi/hal/linux/i386-elf.opt_ah.h
22 |-
23 ----------------------------
24 ./madwifi/hal/ah.h
25 |./madwifi/hal/linux/ah_osdep.c
26 ----------------------------
```

```
./madwifi/hal/ah_desc.h
|-->./madwifi/ath/if_ath.c
|-->./madwifi/tools/athstats.c
|-->./madwifi/ath_rate/amrr/amrr.c
|-->./madwifi/ath_rate/onoe/onoe.c
----------------------------
./madwifi/hal/ah_devid.h
|-->./madwifi/ath/if_ath.c
Commentaire lors de l'inclusion| /* XXX to identify IBM cards */
----------------------------
./madwifi/hal/version.h
----------------------------
./madwifi/release.h
|-
----------------------------
./madwifi/include/sys/queue.h
|-
----------------------------
./madwifi/include/compat.h
|-
----------------------------
./madwifi/net80211/eapol.h
|-
----------------------------
./madwifi/net80211/ieee80211.h
|-->./madwifi/tools/80211stats.c#include "net80211/ieee80211.h"
|-->./madwifi/tools/athchans.c#include "net80211/ieee80211.h"
|-->./madwifi/tools/athkey.c#include "net80211/ieee80211.h"

----------------------------
./madwifi/net80211/ieee80211_crypto.h
|-->./madwifi/tools/80211stats.c
|-->./madwifi/tools/athchans.c
|-->./madwifi/tools/athkey.c
----------------------------
./madwifi/net80211/ieee80211_dot1x.h
|-->./madwifi/net80211/ieee80211.c
|-->./madwifi/net80211/ieee80211_dot1x.c
|-->./madwifi/net80211/ieee80211_proto.c
|-->./madwifi/net80211/ieee80211_radius.c
----------------------------
./madwifi/net80211/ieee80211_ioctl.h
|-->./madwifi/tools/80211stats.c
|-->./madwifi/tools/athchans.c
|-->./madwifi/tools/athkey.c
----------------------------
./madwifi/net80211/ieee80211_linux.h
|-
----------------------------
./madwifi/net80211/ieee80211_node.h
|-
----------------------------
./madwifi/net80211/ieee80211_proto.h
|-
----------------------------
./madwifi/net80211/ieee80211_radiotap.h
|-
----------------------------
./madwifi/net80211/ieee80211_radius.h
|./madwifi/net80211/ieee80211_dot1x.c
./madwifi/net80211/ieee80211_radius.c
----------------------------
./madwifi/net80211/ieee80211_var.h
|-->./madwifi/ath/if_ath.c
|-->./madwifi/ath/if_ath_pci.c
|-->./madwifi/net80211/ieee80211.c
|-->./madwifi/net80211/ieee80211_acl.c
|-->./madwifi/net80211/ieee80211_crypto.c
|-->./madwifi/net80211/ieee80211_crypto_ccmp.c
|-->./madwifi/net80211/ieee80211_crypto_none.c
|-->./madwifi/net80211/ieee80211_crypto_tkip.c
|-->./madwifi/net80211/ieee80211_crypto_wep.c
|-->./madwifi/net80211/ieee80211_dot1x.c
|-->./madwifi/net80211/ieee80211_input.c
|-->./madwifi/net80211/ieee80211_linux.c
|-->./madwifi/net80211/ieee80211_node.c
|-->./madwifi/net80211/ieee80211_output.c
|-->./madwifi/net80211/ieee80211_proto.c
|-->./madwifi/net80211/ieee80211_radius.c
|-->./madwifi/net80211/ieee80211_wireless.c
```

```
|-->./madwifi/net80211/ieee80211_xauth.c
|-->./madwifi/net80211/test_ccmp.c
|-->./madwifi/net80211/test_tkip.c
|-->./madwifi/net80211/test_wep.c
|-->./madwifi/ath_rate/amrr/amrr.c
|-->./madwifi/ath_rate/onoe/onoe.c
----------------------------
./madwifi/net80211/if_ethersubr.h
|-->./madwifi/ath/if_ath.c|#include "if_ethersubr.h"                /* for ETHER_IS_MULTICAST */
|-->./madwifi/net80211/ieee80211_crypto.c|#include "if_ethersubr.h"         /* XXX ETHER_HDR_LEN */
|-->./madwifi/net80211/ieee80211_dot1x.c|#include "if_ethersubr.h"
|-->./madwifi/net80211/ieee80211_input.c|#include "if_ethersubr.h"
|-->./madwifi/net80211/ieee80211_linux.c|#include "if_ethersubr.h"
|-->./madwifi/net80211/ieee80211_output.c|#include "if_ethersubr.h"
|-->./madwifi/net80211/ieee80211_radius.c|#include "if_ethersubr.h"         /* for ETHER_MAX_LEN */
|-->./madwifi/net80211/ieee80211_xauth.c|#include "if_ethersubr.h"
----------------------------
./madwifi/net80211/if_llc.h
|-->./madwifi/ath/if_ath.c|#include "if_llc.h"
|-->./madwifi/net80211/ieee80211_dot1x.c|#include "if_llc.h"
|-->./madwifi/net80211/ieee80211_input.c|#include "if_llc.h"
|-->./madwifi/net80211/ieee80211_output.c|#include "if_llc.h"
|-->./madwifi/net80211/ieee80211_radius.c|#include "if_llc.h"                      /* for LLC_SNAPFRAMELEN */
|-->./madwifi/net80211/ieee80211_xauth.c|#include "if_llc.h"
----------------------------
./madwifi/net80211/if_media.h
|-->./madwifi/ath/if_ath.c|
|-->./madwifi/ath/if_ath_pci.c|
|-->./madwifi/net80211/ieee80211.c|
|-->./madwifi/net80211/ieee80211_acl.c|
|-->./madwifi/net80211/ieee80211_crypto.c|
|-->./madwifi/net80211/ieee80211_crypto_ccmp.c|
|-->./madwifi/net80211/ieee80211_crypto_none.c|
|-->./madwifi/net80211/ieee80211_crypto_tkip.c|
|-->./madwifi/net80211/ieee80211_crypto_wep.c|
|-->./madwifi/net80211/ieee80211_dot1x.c|
|-->./madwifi/net80211/ieee80211_input.c|
|-->./madwifi/net80211/ieee80211_linux.c|
|-->./madwifi/net80211/ieee80211_node.c|
|-->./madwifi/net80211/ieee80211_output.c|
|-->./madwifi/net80211/ieee80211_proto.c|
|-->./madwifi/net80211/ieee80211_radius.c|
|-->./madwifi/net80211/ieee80211_wireless.c|
|-->./madwifi/net80211/ieee80211_xauth.c|
|-->./madwifi/net80211/if_media.c|
|-->./madwifi/net80211/test_ccmp.c|
|-->./madwifi/net80211/test_tkip.c|
|-->./madwifi/net80211/test_wep.c|
|-->./madwifi/ath_rate/amrr/amrr.c|
|-->./madwifi/ath_rate/onoe/onoe.c|
----------------------------
./madwifi/net80211/if_wavelan_ieee.h
|-
----------------------------
./madwifi/net80211/md5.h
|./madwifi/net80211/md5.c|#include "md5.h"
----------------------------
./madwifi/net80211/rc4.h
|-->./madwifi/net80211/ieee80211_crypto_wep.c|#include "rc4.h"
|-->./madwifi/net80211/ieee80211_radius.c|#include "rc4.h"          /* XXX */
|-->./madwifi/net80211/rc4.c|#include "rc4.h"
----------------------------
./madwifi/net80211/version.h
|?
----------------------------
./madwifi/ath_rate/amrr/amrr.h
|-->./madwifi/ath_rate/amrr/amrr.c
----------------------------
./madwifi/ath_rate/onoe/onoe.h
|-->./madwifi/ath_rate/onoe/onoe.c|#include "onoe.h"
----------------------------
```

Cette liste de dépendances attirent notre attention sur les fichiers `if_athvar.h` et `ieee80211_var.h`. Nous verrons par la suite que ces fichiers définissent deux structures importantes, il s'agit des structures `ath_softc` et `ieee80211com`. Les autres fichiers .h n'ont pas une importance déterminante pour notre projet.

4.7 Analyse sémantique

4.7.1 Introduction

L'analyse sémantique est l'étude qui amènera à comprendre le fonctionnement interne du programme, la logique amenant une action provenant des hautes couches à agir sur les basses couches. Nous tenterons de mettre en parallèle l'architecture logique définie au niveau du modèle OSI[37], de la structure telle qu'elle est mise en place sous linux et le driver.

4.7.2 Modèle-OSI / structure sous linux

Lorsque l'on désire paramétrer la carte, on utilise des outils génériques (pour toute carte), tels que :

`iwconfig` qui a pour but de configurer tout ce qui concerne IEEE802.11, par exemple, le ESSID, la fréquence (canal), la clé (si l'on désire activer le WEP), etc.

`ifconfig` qui a pour but de configurer l'adresse IP (également l'adresse MAC) et bien d'autres paramètres concernant le réseau.

Le schéma 4 à la page 23 nous permet de comprendre à quel niveau intervient les diverses commandes.

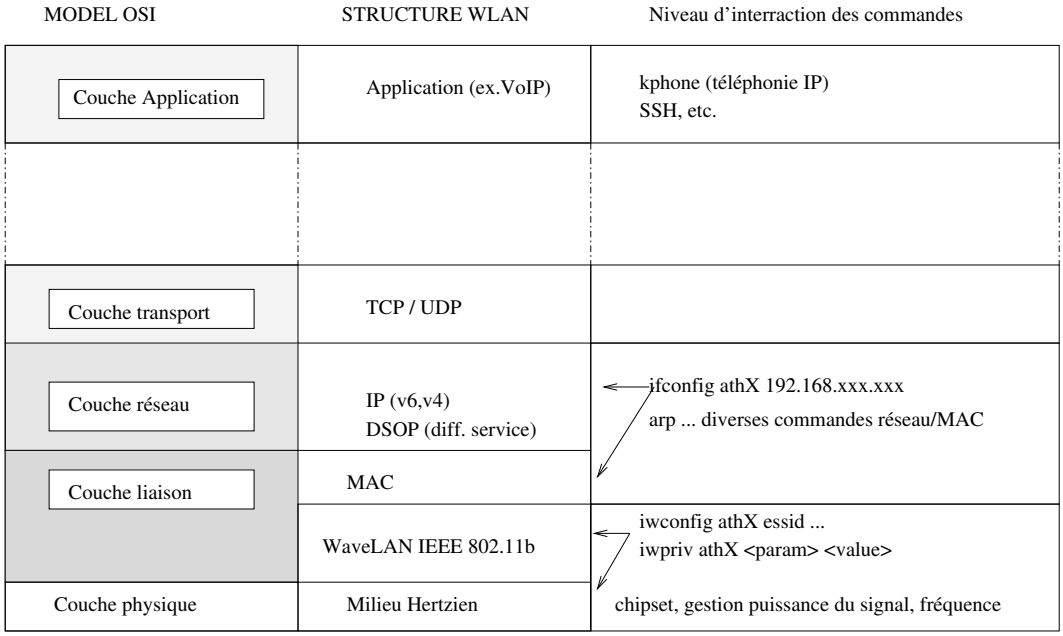


FIG. 4 – Correspondance entre le modèle OSI, notre structure (WLAN) et le niveau auquel intervient les commandes systèmes

Pour bien comprendre le fonctionnement du driver, il est nécessaire de voir quelle fonction (dans la source) est appelée lorsque l'on active ou modifie un paramètre. Pour cela, il faut tester, «débugger», en introduisant des «printk(...)»[38] dans les différentes fonctions. Par exemple, en analysant la fonction qui introduit la clé de cryptage WEP dans la carte, nous avons remarqué qu'il y a une première fonction `ath_set_key` qui envoie les paramètres à une fonction plus bas niveau `ath_setkey` et cette dernière

[37] Open Systems Interconnection

[38] L'usage de la fonction printk en c permet d'imprimer un commentaire dans les logs du noyau (/var/log/syslog)

va reformuler les paramètres de manière compréhensible pour la carte en l'envoyant à la fonction, interne au HAL (Hardware Access Layer). Ceci est possible lorsque `ath_setkey` introduit les paramètres dans hal.o par la fonction `ath_hal_setkey`. On descend d'étage en étage en retirant les couches d'abstraction. Ce qui demeure impossible à connaître, c'est le code exécuté par la dernière fonction, proche de la carte, `ath_hal_setkey` car le code est caché.

4.8 Analyse ciblée

4.8.1 Introduction

Actuellement, il serait difficile et inutile de comprendre l'intégralité du fonctionnement du driver. Nous allons d'abord cibler les fonctions qui nous intéressent :

1. Activation de WME (nous savons que le driver support EDCA, par extension uniquement DCF en CP)
2. Différenciation de service (Accès aux EDCA Paremeter Set Element)
3. Où sont définies les types de trame
4. Comment choisir le TBTT
5. Comment filtrer les files d'attentes (dans le but d'ajouter notre coordinateur)
6. Qu'est-ce qui définit la station en mode AP (le but sera de modifier le comportement de l'AP)

A priori, aucun "mode d'emploi" n'est fourni pour comprendre comment activer les commandes propres à notre pilote. En effet, les commandes de base tels que le choix du ESSID, la fréquence, etc sont accessibles via la commande iwconfig et ont le même fonctionnement qu'avec une autre carte wireless. Cependant, les autres propriétés n'étant accèssibles via la commande privée iwpriv nécessite une étude de la partie suivante du code, ceci dans le but de comprendre comment les utiliser.

4.8.2 Activation du WME

Au niveau du driver, dans le fichier `net80211/ieee80211_ioctl.h`, on trouve la structure de données suivante :

```
enum {
        IEEE80211_PARAM_TURBO           = 1,    /* turbo mode */
        IEEE80211_PARAM_MODE            = 2,    /* phy mode (11a, 11b, etc.) */
        IEEE80211_PARAM_AUTHMODE        = 3,    /* authentication mode */
        IEEE80211_PARAM_PROTMODE        = 4,    /* 802.11g protection */
        IEEE80211_PARAM_MCASTCIPHER     = 5,    /* multicast/default cipher */
        IEEE80211_PARAM_MCASTKEYLEN     = 6,    /* multicast key length */
        IEEE80211_PARAM_UCASTCIPHERS    = 7,    /* unicast cipher suites */
        IEEE80211_PARAM_UCASTCIPHER     = 8,    /* unicast cipher */
        IEEE80211_PARAM_UCASTKEYLEN     = 9,    /* unicast key length */
        IEEE80211_PARAM_WPA             = 10,   /* WPA mode (0,1,2) */
        IEEE80211_PARAM_ROAMING         = 12,   /* roaming mode */
        IEEE80211_PARAM_PRIVACY         = 13,   /* privacy invoked */
        IEEE80211_PARAM_COUNTERMEASURES = 14,   /* WPA/TKIP countermeasures */
        IEEE80211_PARAM_DROPUNENCRYPTED = 15,   /* discard unencrypted frames */
        IEEE80211_PARAM_DRIVER_CAPS     = 16,   /* driver capabilities */
        IEEE80211_PARAM_MACCMD          = 17,   /* MAC ACL operation */
        IEEE80211_PARAM_WME             = 18,   /* WME mode (on, off) */
        IEEE80211_PARAM_HIDESSID        = 19,   /* hide SSID mode (on, off) */
        IEEE80211_PARAM_APBRIDGE        = 20,   /* AP inter-sta bridging */
        IEEE80211_PARAM_KEYMGTALGS      = 21,   /* key management algorithms */
        IEEE80211_PARAM_RSNCAPS         = 22,   /* RSN capabilities */
        IEEE80211_PARAM_INACT           = 23,   /* station inactivity timeout */
        IEEE80211_PARAM_INACT_AUTH      = 24,   /* station auth inact timeout */
        IEEE80211_PARAM_INACT_INIT      = 25,   /* station init inact timeout */
};
```

On remarque ceci : `IEEE80211_PARAM_WME=18,/* WME mode (on, off) */`. Ce qui siginifie simplement que le 18ème flag correspond à l'activation/désactivation du WME. Si l'on analyse maintenant le fichier `net80211/ieee80211_wireless.c`, on remarque une fonction intéressante :

```
ieee80211_ioctl_setparam(struct ieee80211com *ic, struct iw_request_info *info, void *w, char *extra)
{
        struct ieee80211_rsnparms *rsn = &ic->ic_bss->ni_rsn;
        int *i = (int *) extra;
        int param = i[0];               /* parameter id is 1st */
        int value = i[1];               /* NB: most values are TYPE_INT */
        struct ifreq ifr;
        int retv = 0;
        int j, caps;
        const struct ieee80211_authenticator *auth;
        const struct ieee80211_aclator *acl;

        switch (param) {
```

Cette fonction permet de définir les paramètres :

ieee80211com *ic : *ic pointe sur une structure complexe contenant un grand nombre de paramètres dont les flags décrits plus loin dans le rapport (cf : ieee80211_var.h, ligne 229)
iw_request_info *info : (indéterminé)
void *w : (indéterminé)
char *extra : C'est le flag à modifier (position dans la structure) et sa valeur (1=vrai sinon faux). On convertit d'abord le pointeur char en pointeur int. Le char est un motif de 8 bits donc un nombre de 0 à 255 (s'il est unsigned). Ensuite, on remarque que l'on pointe sur le début *i (extra converti en int) on extrait le paramètre et ensuite sur le nombre suivant (la valeur associé au paramètre).

On remarque qu'en fonction de la valeur de `param`, on accède à un paramètre bien précis. Par exemple, si `param==18`, cela signifierait qu'il est égal à `IEEE80211_PARAM_WME` on répond donc à la condition suivante :

```
case IEEE80211_PARAM_WME:
                if (ic->ic_opmode != IEEE80211_M_STA)
                        return -EINVAL;
                if (value)
                        ic->ic_flags |= IEEE80211_F_WME;
                else
                        ic->ic_flags &= ~IEEE80211_F_WME;
                break;
```

Si la valeur est égale à '1', alors la condition est remplie, si elle est nulle ou égale à un autre nombre, elle est 'FAUX'. On active le flag si la condition est remplie, on le désactive dans le cas contraire. Une explication détaillée du mécanisme est donnée plus bas.

Il existe donc deux fonctions importantes, `ieee80211_ioctl_getparam` et `ieee80211_ioctl_setparam`. La fonction «setparam», requiert une structure nommée localement ic qui contient les divers paramètres. Quels sont les éléments qui appellent ces fonctions ? Si l'on observe le fichier `net80211/ieee80211_linux.h`, on remarque :

```
extern  int ieee80211_ioctl_setparam(struct ieee80211com *,
                struct iw_request_info *, void *, char *);
extern  int ieee80211_ioctl_getparam(struct ieee80211com *,
                struct iw_request_info *, void *, char *);
```

La structure `ieee80211com` est définie dans le fichier `ieee80211_var.h` dans lequel on trouve `ic_flags` à la ligne 229 :

```
u_int32_t               ic_flags;       /* state flags */
```

On en déduit donc que les flags sont contenus dans un type entier non-signé 32 bits. Si l'on analyse comment on active un flag, en prenant l'exemple de WME, on effectue un **ou-inclusif** «`|=`» entre `ic_flags` actuel et le motif `IEEE80211_F_WME` :

```
#define IEEE80211_F_WME         0x00002000      /* CONF: enable WME use */
```

$0x$ **:** signifie que l'on définit un nombre en hexadécimal. Ici 0x2000 en hexadécimal est égal à 8192 en decimal, ce qui correspond à 2^{13}, donc le 13ème bit est à 1. Voici un exemple :

```
ic_flags        :  0110 1010 0111 0010 1101 0110 0101 1101 1111 0011 1101 0100 0100 0000 0100 1110
|=
IEEE80211_F_WME :  0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0001 0000 0000 0000 0000
Resultat,
ic_flags        :  0110 1010 0111 0010 1101 0110 0101 1101 1111 0011 1101 0101 0100 0000 0100 1110
```

Pour désactiver le flag on effectue un **ET** logique entre `ic_flags` et le négatif du motif ce qui met le "bit" en question à **0**. Nous venons d'expliquer le mécanisme d'application des flags.

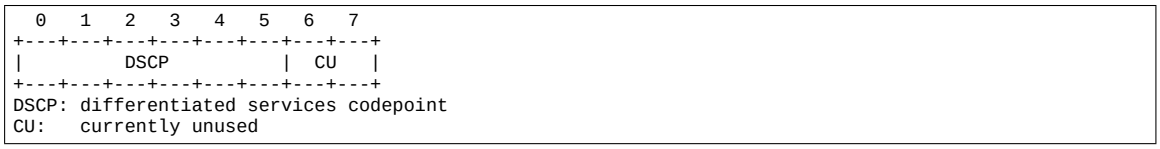
Il ne faut pas perdre de vue qu'à l'état actuel, nous n'avons pas affecté la modification du flag au sein de la carte mais seulement dans la mémoire de l'ordinateur, dans un buffer alloué par le kernel lorsque l'on a chargé le module, soit dans la structure `ieee80211com`. A la sortie de la fonction de mise à jour des paramètres, on effectue un `EXPORT_SYMBOL(ieee80211_ioctl_setparam)`. La section qui effectue la mise à jour des paramètres au sein de la carte se situe dans le binaire `hal.o`.

4.8.3 Différenciation des services

Après quelques recherches, la différenciation de service s'opère depuis les couche supérieures. Les paquets ip possède un champ (DSCP) permettant de définir la priorité. Il existe également un autre moyen de créer des réseaux prioritaires, et ceci à l'aide des VLAN, selon la norme IEEE802.1d. Plus loin dans le rapport, des tests seront effectués en utilisant l'un et l'autre de ces moyens de différenciation de service.

IEEE 802.1d : Si l'on se réfère au début du rapport, un tableau de correspondance entre les services définis selon la norme 802.1d et les files d'attentes selon WME est donné. Pour rappel, il s'agit du tableau 1 à la page 8. Nous verrons plus loin, comment nous créerons artificiellement des transmissions selon les divers ordres de priorité.

Champ DSCP : Les applications choisissent la qualité de service dont ils doivent disposer. Lorsqu'il s'agit de logiciel de téléphonie IP ou de streaming vidéo, le choix d'un protocole RTP ainsi que d'une différenciation des services sont nécessaires. La QoS est déjà garantie au niveau de la couche réseau par les routeur. Cependant, pour garantir notre priorité des transmissions sur le canal, l'information de la priorité des paquets au niveau IP doit être transmise dans la couche inférieure. L'entête ipv4 dispose d'un champ appellé «Differentiated Service Field». La RFC 2474 :



Le trafic ayant un traitement par défaut (sans priorité) a un codepoint de 000000. Ensuite, les autres types de service sont définis par classe.

D'après une documentation supplémentaire fournie à l'adresse `http://modular.fas.harvard.edu/t42/madwifi/README`, la correspondance entre les valeurs DSCP[39] et les quatres catégories d'accès [**AC**]définies par WME est la suivante :

DSCP	Catégorie d'accès
0x08, 0x20	Background
0x0, 0x18	Best Effort
0x28, 0xa0	Video
0x30, 0xe0	Voice

TAB. 2 – Correspondance DSCP-AC

Cette distribution de priorité s'effectue lorsque la couche IP transmet sont SDU à la couche MAC.

[39] Differentiated Service Codepoint

4.8.4 Traitement des quatres files d'attente

Il est important de déterminer la façon dont les files d'attente sont gérées. Pour rappel, nous avons quatre catégories d'accès, voici comment elles sont définies dans le fichier net80211/ieee80211.h :

```
/* WME stream classes */
#define WME_AC_BE       0               /* best effort */
#define WME_AC_BK       1               /* background */
#define WME_AC_VI       2               /* video */
#define WME_AC_VO       3               /* voice */
```

Lors de la détermination de la catégorie, les divers flux sont mis dans la queue correpondant à la bonne catégorie. On peut voir cela dans le fichier `if_ath.c`, lignes 384-391 :

```
        if (!ath_tx_setup(sc, WME_AC_BE, HAL_WME_AC_BE))
                goto bad2;
        if (!ath_tx_setup(sc, WME_AC_BK, HAL_WME_AC_BK))
                goto bad2;
        if (!ath_tx_setup(sc, WME_AC_VI, HAL_WME_AC_VI))
                goto bad2;
        if (!ath_tx_setup(sc, WME_AC_VO, HAL_WME_AC_VO))
                goto bad2;
```

On remarque que le principe est que si le le type de trafic ne correspond pas à la catégorie attendue, on renvoie le flux à la section bad2 :

```
bad2:
        if (sc->sc_txq[WME_AC_BK].axq_qnum != (u_int) -1)
                ATH_TXQ_LOCK_DESTROY(&sc->sc_txq[WME_AC_BK]);
        if (sc->sc_txq[WME_AC_BE].axq_qnum != (u_int) -1)
                ATH_TXQ_LOCK_DESTROY(&sc->sc_txq[WME_AC_BE]);
        if (sc->sc_txq[WME_AC_VI].axq_qnum != (u_int) -1)
                ATH_TXQ_LOCK_DESTROY(&sc->sc_txq[WME_AC_VI]);
        if (sc->sc_txq[WME_AC_VO].axq_qnum != (u_int) -1)
                ATH_TXQ_LOCK_DESTROY(&sc->sc_txq[WME_AC_VO]);
        ath_desc_free(sc);
```

Nous remarquons que nos files d'attente sont accessibles dans une structure nommée **sc** qui est définie dans le fichier `ath/if_athvar.h`[40], elle contient divers paramètres tels que `sc_txq`. Ce dernier n'est autre que la liste des queues prêtes à être transmises, selon la catégorie. Voici comment elle est définie :

```
struct ath_softc {
        struct net_device       sc_dev;         /* NB: must be first */
        struct semaphore        sc_lock;        /* dev-level lock */
        struct net_device_stats sc_devstats;    /* device statistics */
        struct ath_stats        sc_stats;       /* private statistics */
        struct ieee80211com     sc_ic;          /* IEEE 802.11 common */

.... divers paramètres définis ....

struct ath_txq          sc_txq[HAL_NUM_TX_QUEUES];   /* Pour rappel, il y a quatre
                                                        catégories, donc HAL_NUM_TX_QUEUES=4 */
.... divers paramètres définis ....
```

On notera également au passage que la structure `ath_softc` contient également **ic** qui n'est autre qu'un pointeur sur la structure `ieee80211com`. `sc_txq` est du type `ath_txq` qui est défini dans le même fichier, soit `ath_ifathvar.h`, lignes 171-178 :

```
struct ath_txq {
        u_int                   axq_qnum;       /* hardware q number */
        u_int32_t               *axq_link;      /* link ptr in last TX desc */
        STAILQ_HEAD(, ath_buf)  axq_q;          /* transmit queue */
        spinlock_t              axq_lock;       /* lock on q and link */
};
```

[40] Notons que la structure sur laquelle **sc** pointe s'appelle `ath_softc`

Buffer de données : Comme on le voit ci-dessus, la queue est insérée dans un buffer, voici le buffer en question (`ath/if_athvar.h` :

```
struct ath_buf {
        STAILQ_ENTRY(ath_buf)   bf_list;
        struct ath_desc         *bf_desc;       /* virtual addr of desc */
        dma_addr_t              bf_daddr;       /* physical addr of desc */
        struct sk_buff          *bf_skb;        /* skbuff for buf */
        dma_addr_t              bf_skbaddr;     /* physical addr of skb data */
        struct ieee80211_node   *bf_node;       /* pointer to the node */
};
```

Les données provenant des couches supérieures sont contenues dans la structure `bf_skb` qui est définie dans les includes de la source du kernel `${LINUXPATH}/include/linux/skbuff.h`, fichier inclu par `${LINUXPATH}/include/linux/netdevice.h`. Voici la description :

```
struct sk_buff {
        /* These two members must be first. */
        struct sk_buff          *next;
        struct sk_buff          *prev;

        struct sk_buff_head     *list;
        struct sock             *sk;
        struct timeval          stamp;
        struct net_device       *dev;
```

. . . , voir le fichier pour de plus amples détails, cependant, voici la description des champs :

```
/**
 *      struct sk_buff - socket buffer
 *      @next: Next buffer in list
 *      @prev: Previous buffer in list
 *      @list: List we are on
 *      @sk: Socket we are owned by
 *      @stamp: Time we arrived
 *      @dev: Device we arrived on/are leaving by
 *      @real_dev: The real device we are using
 *      @h: Transport layer header
 *      @nh: Network layer header
 *      @mac: Link layer header
 *      @dst: FIXME: Describe this field
 *      @cb: Control buffer. Free for use by every layer. Put private vars here
 *      @len: Length of actual data
 *      @data_len: Data length
 *      @mac_len: Length of link layer header
 *      @csum: Checksum
 *      @__unused: Dead field, may be reused
 *      @cloned: Head may be cloned (check refcnt to be sure)
 *      @pkt_type: Packet class
 *      @ip_summed: Driver fed us an IP checksum
 *      @priority: Packet queueing priority
 *      @users: User count - see {datagram,tcp}.c
 *      @protocol: Packet protocol from driver
 *      @security: Security level of packet
 *      @truesize: Buffer size
 *      @head: Head of buffer
 *      @data: Data head pointer
 *      @tail: Tail pointer
 *      @end: End pointer
 *      @destructor: Destruct function
 *      @nfmark: Can be used for communication between hooks
 *      @nfcache: Cache info
 *      @nfct: Associated connection, if any
 *      @nf_debug: Netfilter debugging
 *      @nf_bridge: Saved data about a bridged frame - see br_netfilter.c
 *      @private: Data which is private to the HIPPI implementation
 *      @tc_index: Traffic control index
 */
```

Nous le verrons plus tard, il n'y a pas de moyen d'intervenir directement au sein de la carte, il sera peut-être indispensable de disséquer le flux afin d'y redéfinir ou effectuer un tri supplémentaire au niveau

de la priorité. On remarque dans la description du buffer ci-dessus qu'il y a un champ "priority".. Normalement, le contenu des paquet n'a pas d'importance, seul leur type, les adresses et probablement quelques paramètres de configurations sont nécessaires.

Dès le moment où l'on doit traiter les quatre files d'attente, on utilise la méthode `ath_tx_tasklet_q0123(TQUEUE_ARG data)` (`if_ath.c`)

```
/*
 * Deferred processing of transmit interrupt; special-cased
 * for four hardware queues, 0-3 (e.g. 5212 w/ WME support).
 */
ath_tx_tasklet_q0123(TQUEUE_ARG data)
{
        struct net_device *dev = (struct net_device *)data;
        struct ath_softc *sc = dev->priv;
        struct ieee80211com *ic = &sc->sc_ic;

        /*
         * Process each active queue.
         */

        ath_tx_processq(sc, &sc->sc_txq[0]);
        ath_tx_processq(sc, &sc->sc_txq[1]);
        ath_tx_processq(sc, &sc->sc_txq[2]);
        ath_tx_processq(sc, &sc->sc_txq[3]);
        /*
         * Don't wakeup unless we're associated; this insures we don't
         * signal the upper layer it's ok to start sending data frames.
         */
        /* XXX use a low watermark to reduce wakeups */
        if (ic->ic_state == IEEE80211_S_RUN)
                netif_wake_queue(dev);
}
```

Cette méthode est exclusivement appelée lorsque l'on est en mode WME, dans le cas contraire, on a qu'une seule queue et c'est la méthode `ath_tx_tasklet_q0(TQUEUE_ARG data)`[41] qui est appelée.

On constate, d'après les commentaires, qu'on ne peut avoir quatre files d'attente au sein du hardware qu'avec le chipset 5212 (nos carte NetGEAR WAG511 et DLINK-DWL520 possède ce chipset). Dans le cas où l'on utilise qu'une seule file d'attente, on la traite avec la fonction `ath_tx_processq(...)` du même fichier.

[41] dans le fichier `if_ath.c`

Allocation du buffer : Afin d'allouer un nouveau buffer, on utilise la méthode `ath_alloc_skb` définie dans `if_ath.c`. Cette méthode va ensuite appeller une méthode qui va réellement effectuer l'allocation, il s'agit de `dev_alloc_skb` définie dans les sources du kernel :

```
ath_alloc_skb(u_int size, u_int align)
{
        struct sk_buff *skb;
        u_int off;

        skb = dev_alloc_skb(size + align-1);
        if (skb != NULL) {
                off = ((unsigned long) skb->data) % align;
                if (off != 0)
                        skb_reserve(skb, align - off);
        }
        return skb;
}
```

D'après la littérature [4], `dev_alloc_skb` initialise `skb->data`, `skb->tail` et `skb->head` avec une priorité `GFP_ATOMIC` et réseve également de l'espace utilisé pour des optimisations dans la couche réseau mais qui ne doit pas être touché par le driver. La méthode`dev_kfree_skb` est également utilisée dans le code source dans le but de libérer le tampon.

4.8.5 Mise à jour des flags

Nous avons explicité la manière dont les flags sont changés mais pour l'instant nous restons à un niveau purement software, il s'agit maintenant de comprendre comment ces flags sont appliqués au sein du hardware. Le fichier `if_ath.c` vérifie la valeur de `CONFIG_NET_WIRELESS` et si elle est définie (=1), tous les paramètres seront mis à jour successivement. La section qui remet le flag à 0 n'a pas été trouvée mais en toute logique elle doit se trouver dans le hal.o puisque c'est après avoir modifié les paramètres de la carte que l'on peut considérer que l'on a appliqué tous les nouveaux paramètres.

4.8.6 Format des trames

Dans le fichier `net80211/ieee80211.h` on découvre la définition de chaque type de trame. Intéressons-nous aux trames de contrôle de qos. Deux types ont été définis, la première sans l'adresse 4 et la seconde avec le champs d'adresse 4 (lignes 72-82 et 99-108 :

```
struct ieee80211_qosframe {
        u_int8_t        i_fc[2];
        u_int8_t        i_dur[2];
        u_int8_t        i_addr1[IEEE80211_ADDR_LEN];
        u_int8_t        i_addr2[IEEE80211_ADDR_LEN];
        u_int8_t        i_addr3[IEEE80211_ADDR_LEN];
        u_int8_t        i_seq[2];
        u_int8_t        i_qos[2];
        /* possibly followed by addr4[IEEE80211_ADDR_LEN]; */
        /* see below */
} __packed;
...
struct ieee80211_qosframe_addr4 {
        u_int8_t        i_fc[2];
        u_int8_t        i_dur[2];
        u_int8_t        i_addr1[IEEE80211_ADDR_LEN];
        u_int8_t        i_addr2[IEEE80211_ADDR_LEN];
        u_int8_t        i_addr3[IEEE80211_ADDR_LEN];
        u_int8_t        i_seq[2];
        u_int8_t        i_addr4[IEEE80211_ADDR_LEN];
        u_int8_t        i_qos[2];
} __packed;
```

Dans le draft, à la page 30, nous avons une définition d'une trame de QoS et on remarque que l'équipe de développement a suivit la norme. Voir le tableau tiré de la norme (TAB3 à la page 32).

Frame control	Dur /ID	Addr 1	Addr 2	Addr 3	Seq Ctrl	Addr 4	QoS Ctrl	Frame body	FCS
Octets : 2	2	6	6	6	2	6	2		4

TAB. 3 – Trame de QoS (avec champ addr. 4, tiré de la norme[3], à la page 30

Ensuite, observons les éléments d'information pour WME (fichier ieee8021.h, lignes 187-223) :

```
struct ieee80211_ie_wme {
        u_int8_t        wme_id;         /* IEEE80211_ELEMID_VENDOR */
        u_int8_t        wme_len;        /* length in bytes */
        u_int8_t        wme_oui[3];     /* 0x00, 0x50, 0xf2 */
        u_int8_t        wme_type;       /* OUI type */
        u_int8_t        wme_subtype;    /* OUI subtype */
        u_int8_t        wme_version;    /* spec revision */
        u_int8_t        wme_info;       /* AC info */
} __packed;

/*
 * WME/802.11e Tspec Element
 */
struct ieee80211_wme_tspec {
        u_int8_t        ts_id;
        u_int8_t        ts_len;
        u_int8_t        ts_oui[3];
        u_int8_t        ts_oui_type;
        u_int8_t        ts_oui_subtype;
        u_int8_t        ts_version;
        u_int8_t        ts_tsinfo[3];
        u_int8_t        ts_nom_msdu[2];
        u_int8_t        ts_max_msdu[2];
        u_int8_t        ts_min_svc[4];
        u_int8_t        ts_max_svc[4];
        u_int8_t        ts_inactv_intv[4];
        u_int8_t        ts_susp_intv[4];
        u_int8_t        ts_start_svc[4];
        u_int8_t        ts_min_rate[4];
        u_int8_t        ts_mean_rate[4];
        u_int8_t        ts_max_burst[4];
        u_int8_t        ts_min_phy[4];
        u_int8_t        ts_peak_rate[4];
        u_int8_t        ts_delay[4];
        u_int8_t        ts_surplus[2];
        u_int8_t        ts_medium_time[2];
} __packed;
```

Ces deux spécifications d'éléments WME n'ont pas été trouvées dans la norme. Aussi, les éléments EDCA, c'est-à-dire les éléments contenant les informations de gestion de la priorité et fournissant les opportunités d'émettre (TXOP) ne sont pas définis dans ce fichier. Ils le sont très probablement dans le HAL. Il est possible que les informations ci-dessus servent à construire à plus bas niveau les «EDCA Parameter Set Element». Nous avons vu que ces derniers doivent être insérés dans chaque beacon afin de gérer les transmissions prioritaires en mode EDCA, donc en période de contention (CP).

4.8.7 Gestion du beacon

Nous nous intéressons à la gestion beacon car selon HCF, nous devons pouvoir, à l'aide des beacon définir des temps précis durant lesquels nous avons une période de contention (CP) et une période libre de contention (CFP). Ces périodes démarrent juste après le beacon. Aussi, il faut s'assurer qu'aucune transmission n'entrave l'émission parfaitement régulière d'un beacon.

Dans le fichier `if_ath.c` est défini une fonction d'initialisation du beacon (lignes 1695-1706) :

```
static int ath_beacon_setup(struct ath_softc *sc, struct ath_buf *bf, struct sk_buff *skb)
{
#define USE_SHPREAMBLE(_ic) \
        (((_ic)->ic_flags & (IEEE80211_F_SHPREAMBLE | IEEE80211_F_USEBARKER))\
                == IEEE80211_F_SHPREAMBLE)
#define MIN(a,b)        ((a) < (b) ? (a) : (b))
        struct ieee80211com *ic = &sc->sc_ic;
        struct ieee80211_node *ni = bf->bf_node;
        struct ath_hal *ah = sc->sc_ah;
        struct ath_node *an = ATH_NODE(ni);
        struct ath_desc *ds;
        u_int8_t rate;
```

On remarque, ci-dessus, que pour construire notre beacon nous avons besoin des paramètres ieee80211com (comme nous l'avions décrit précédemment, les flags contiennent la configuration de la carte au niveau WLAN et le choix d'utiliser un préambule court ou long dépend du débit que l'on a choisi[42]

La fonction différencie les beacons avec ou sans «short preamble». Ceci est logique du fait que la structure même du beacon change (lignes 1708-1726) :

```
        bf->bf_skbaddr = bus_map_single(sc->sc_bdev,
                skb->data, skb->len, BUS_DMA_TODEVICE);
        DPRINTF(sc, ATH_DEBUG_BEACON,
                "%s: skb %p [data %p len %u] skbaddr %p\n",
                __func__, skb, skb->data, skb->len, (caddr_t) bf->bf_skbaddr);

        /* setup descriptors */
        ds = bf->bf_desc;

        ds->ds_link = 0;
        ds->ds_data = bf->bf_skbaddr;
        /*
         * Calculate rate code.
         * XXX everything at min xmit rate
         */
        if (USE_SHPREAMBLE(ic))
                rate = an->an_tx_mgtratesp;
        else
                rate = an->an_tx_mgtrate;
```

On observe ci-dessus que, si l'on utilise un short preamble (sp), ce sera la valeur du débit défini pour sp qui sera choisie, ici `rate=an->an_tx_mgtratesp` sinon, on opte pour l'autre valeur définie `an_tx_mgrate`. Ces valeurs proviennent de la "fonction" `ATH_NODE`, qui ne fait que retourner la structure `ath_node` définie dans dans le fichier `if_athvar.h` lignes 128-135 :

```
struct ath_node {
        struct ieee80211_node an_node;  /* base class */
        u_int8_t        an_tx_mgtrate;  /* h/w rate for management/ctl frames */
        u_int8_t        an_tx_mgtratesp;/* short preamble h/w rate for " " */
        u_int32_t       an_avgrssi;     /* average rssi over all rx frames */
        HAL_NODE_STATS  an_halstats;    /* rssi statistics used by hal */
        /* variable-length rate control state follows */
};
```

[42]La norme IEEE802.11b prévoit une optimisation de la gestion de la couche physique. Les trames contiennent un préambule permettant de synchroniser les stations qu'une trame est entrain d'être émise et qu'il ne s'agit pas de "bruit". Avec HR/DSSS on transmet les 9 premiers bytes à 1Mbits et les 6 suivants peuvent être transmit à 2Mbit/s

Donc si l'on observe la structure `ath_node` ci-dessus, elle contient simplement des champs de données concernant le débit à utiliser.

Dès que l'on a choisi le taux en fonction du type de préambule, on peut construire la trame de beacon (fichier `\if_ath.c` lignes 1727-1749) :

```
        ath_hal_setuptxdesc(ah, ds
                , skb->len + IEEE80211_CRC_LEN  /* frame length */
                , sizeof(struct ieee80211_frame)/* header length */
                , HAL_PKT_TYPE_BEACON           /* Atheros packet type */
                , MIN(ni->ni_txpower,60)        /* txpower XXX */
                , rate, 1                       /* series 0 rate/tries */
                , HAL_TXKEYIX_INVALID           /* no encryption */
                , 0                             /* antenna mode */
                , HAL_TXDESC_NOACK              /* no ack for beacons */
                , 0                             /* rts/cts rate */
                , 0                             /* rts/cts duration */
        );
        /* NB: beacon's BufLen must be a multiple of 4 bytes */
        ath_hal_filltxdesc(ah, ds
                , roundup(skb->len, 4)          /* buffer length */
                , AH_TRUE                       /* first segment */
                , AH_TRUE                       /* last segment */
        );
        return 0;
#undef MIN
#undef USE_SHPREAMBLE
}
```

Comme on le remarque, on descend d'un niveau d'abstraction pour atteindre la fonction `ath_hal_setuptxdesc` qui est définie par une macro dans `if_athvar.h` :

```
#define ath_hal_setuptxdesc(_ah, _ds, _plen, _hlen, _atype, _txpow, \
                _txr0, _txtr0, _keyix, _ant, _flags, \
                _rtsrate, _rtsdura) \
        ((*(_ah)->ah_setupTxDesc)((_ah), (_ds), (_plen), (_hlen), (_atype), \
                (_txpow), (_txr0), (_txtr0), (_keyix), (_ant), \
                (_flags), (_rtsrate), (_rtsdura)))
```

Comme on peut aisément le constater,il s'agit d'une macro qui ne fait que réécrire la fonction pour qu'elle soit interprétée par une autre fonction dont la syntaxe est différente. Pour finir, on se rend à nouveau compte que l'on a toujours pas à faire à la fonction finale qui va exécuter effectivement l'opération. `ath_hal_setuptxdesc(...)` ne fait que remplire la structure `ah_setupTxDesc` qui va être interprétée par la fonction de bas niveau (que l'on trouve dans le fichier `ah.h` lignes 477-482) :

```
HAL_BOOL  __ahdecl(*ah_setupTxDesc)(struct ath_hal *, struct ath_desc *,
                                u_int pktLen, u_int hdrLen,
                                HAL_PKT_TYPE type, u_int txPower,
                                u_int txRate0, u_int txTries0,
                                u_int keyIx, u_int antMode, u_int flags,
                                u_int rtsctsRate, u_int rtsctsDuration);
```

Il semblerait que l'on se trouve au plus bas niveau, le reste étant décrit dans la source que nous ne possédons pas. Cette découverte est une mauvaise nouvelle car, à priori, il n'y aurait pas moyen de contrôler directement le beacon. Ce n'est évidemment qu'un exemple parmi toutes les fonctions gérant la transmission des données au sein de la carte. La fonction que l'on vient d'expliciter fait partie de la section «Transmit function» du fichier `ah.h`.

Dans le fichier `ah.h`, lignes 388-409, on remarque également ceci :

```
/*
 * Per-station beacon timer state.  Note that the specified
 * beacon interval (given in TU's) can also include flags
 * to force a TSF reset and to enable the beacon xmit logic.
 * If bs_cfpmaxduration is non-zero the hardware is setup to
 * coexist with a PCF-capable AP.
 */
typedef struct {
        u_int32_t       bs_nexttbtt;            /* next beacon in TU */
        u_int32_t       bs_nextdtim;            /* next DTIM in TU */
        u_int32_t       bs_intval;              /* beacon interval+flags */
#define HAL_BEACON_PERIOD       0x0000ffff      /* beacon interval period */
#define HAL_BEACON_ENA          0x00800000      /* beacon xmit enable */
#define HAL_BEACON_RESET_TSF    0x01000000      /* clear TSF */
        u_int32_t       bs_dtimperiod;
        u_int16_t       bs_cfpperiod;           /* CFP period in TU */
        u_int16_t       bs_cfpmaxduration;      /* max CFP duration in TU */
        u_int32_t       bs_cfpnext;             /* next CFP in TU */
        u_int16_t       bs_timoffset;           /* byte offset to TIM bitmap */
        u_int16_t       bs_bmissthreshold;      /* beacon miss threshold */
        u_int32_t       bs_sleepduration;       /* max sleep duration */
} HAL_BEACON_STATE;
```

Il est important de comprendre comment définir le TBTT[43] afin de garantire les CFP / CP. Apparament ces variables existent mais elle sont à priori sous le contrôle du binaire `hal.o`. Ce manque de souplesse est très probablement dû à un risque que les développeurs ne voulaient pas prendre en matière de "sécurité". Si l'on peut modifier certains paramètres de bas niveau le risque de violer la norme est non négligeable et il est probable que les développeurs d'ATHEROS® ne veulent pas courir ce risque.

4.8.8 Interaction `if_ath.c hal.o`

Cette partie de l'analyse devrait nous permettre de comprendre comment est fait le pont entre la source que nous avons à disposition et le binaire (noyau de notre pilote). Si l'on prend un peu de recul, on remarque que tout ce qui provient des hautes couches est simplement remis en forme afin d'être injecté dans notre binaire. L'interface faisant le pont entre nos source - définissant les trames, le débit, les paramètres WLAN- et le binaire est identifié, il s'agit de `if_ath.c` complété par `if_athvar.h`. Dans ce dernier fichier, nous avons remarqué la liste des macros de redéfinition de fonction interminable. Comme nous n'avons pas accès à la source du hal.o, nous ne pouvons pas redéfinir les fonctions internes au binaire mais nous pourrions intercepter les données transitant des couches supérieures au hal et vice-versa. Le problème est qu'il faut pouvoir intercepter ces données. Ma première idée était de réécrire les macros pour qu'ils utilisent des fonctions que nous aurions redéfinies, cependant, les macros ne sont interprétées que par le compilateur lorsque l'on compile le projet. Aussi, nous n'aurions pas accès aux données traitées en interne. Après discussion avec M.Schaefer, il serait intéressant de "court-circuiter" ce qui s'appelle la table de saut.

En effet, en C, nous définissons des fonctions, comme en ada,en java, cependant, pour y accéder, nous le faisons à l'aide de pointeur. Lorsque l'on charge le module, de la mémoire est allouée pour chaque structure définie dans notre code source. Intéressons-nous donc à l'allocation en mémoire de notre binaire. Nous avons vu que ce qui nous permet d'utiliser les fonctions du dit binaire est donné dans le fichier à include ah.h. Dans ce fichier, plusieurs structures sont définies, notamment celles qui comportent l'accès aux principales fonctions, il s'agit de la structure `ath_hal`.

Intéressons-nous maintenant à la structure dont nous avons accès, qui inclut justement cette structure `ath_hal`. Nous la trouvons dans le fichier `if_athvar.h`, dans la structure `ath_softc` à la ligne 216 :

```
struct ath_hal          *sc_ah;         /* Atheros HAL */
```

[43] C'est le temps à attendre avant le prochain beacon, il est indispensable de le maîtriser lorsque l'on utilise HCF

Définition de structure : Actuellement, nous savons donc que c'est lorsque l'on allouera de la mémoire pour la structure `ath_softc` que, par conséquent, de la mémoire sera également allouée pour la structure `ath_hal`. Pour rappel, nous avions vu que le pointeur sur la structure `ath_softc` est ***sc**. Ici, nous remarquons que le pointeur sur la structure `ath_hal` est ***sc_ah** mais nous verrons que dans le fichier `if_ath.c`.

Allocation : Jusqu'à maintenant nous n'avons toujours pas vu où ces structure sont allouées, ni à quel moment. Ceci se fait lorsque la fonction `ath_attach` est appellée. Cette fonction est définie dans le fichier `if_ath.c` lignes 238-572 (source originale, sans modification, sinon, dans la source modifiée on la trouve de la ligne 248-589). Nous verrons plus loin dans le rapport, comment la fonction a été modifiée pour intercepter une fonction du hal en court-circuitant le pointeur.

A présent, le mécanisme d'allocation est explicité. On peut voir sur la figure 5 à la page 37.

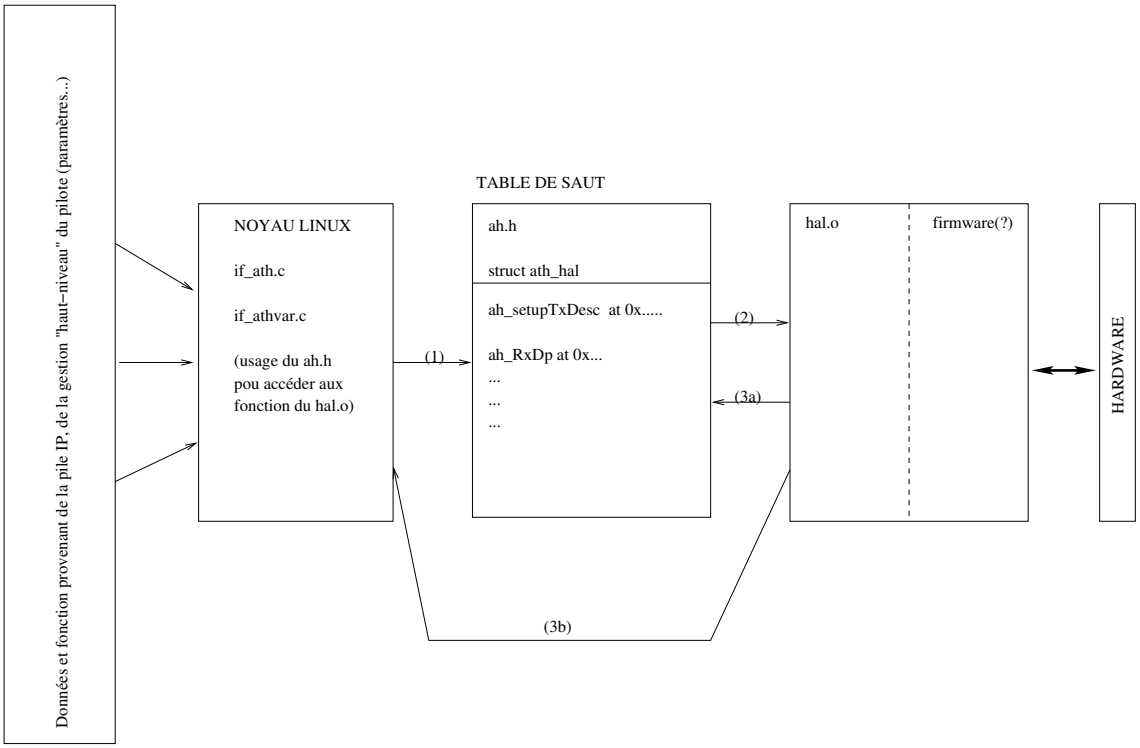


FIG. 5 – Schéma logique de l'interaction entre les sources visibles et le binaire hal.o

Les informations, les données provenant des couches supérieures sont traitées dans le `if_ath.c`. Elles sont ensuite modifiées de manière à être utilisées dans le binaire (uniquement à la compilation d'après les macros définies à la fin du fichier `if_athvar.h`). Elles sont ensuite envoyées aux fonctions situées au plus bas niveau dans le binaire hal.o via la table de saut[44](flèche 1 sur la figure 5). La flèche 2 symbolise le pointeur sur les fonctions définies dans la structure `ath_hal`.

[44]Cette table de saut est simplement une liste de pointeurs désignant une à une les fonctions allouées en mémoire par le binaire `hal.o`

Si l'on peut «court-circuiter» chaque pointeur et amener le flux de données dans un coordinateur que nous aurions conçu, nous pourrions filtrer les flux selon nos règles. Par exemple, n'autoriser qu'un nombre limité de stations à la fois d'user de la catégorie d'accès vocale (AC_VO). Si l'on observe la figure 6 à la page 38, il serait possible de définir une table de saut intermédiaire comportant des pointeurs désignant les fonctions de notre «coordinateur».

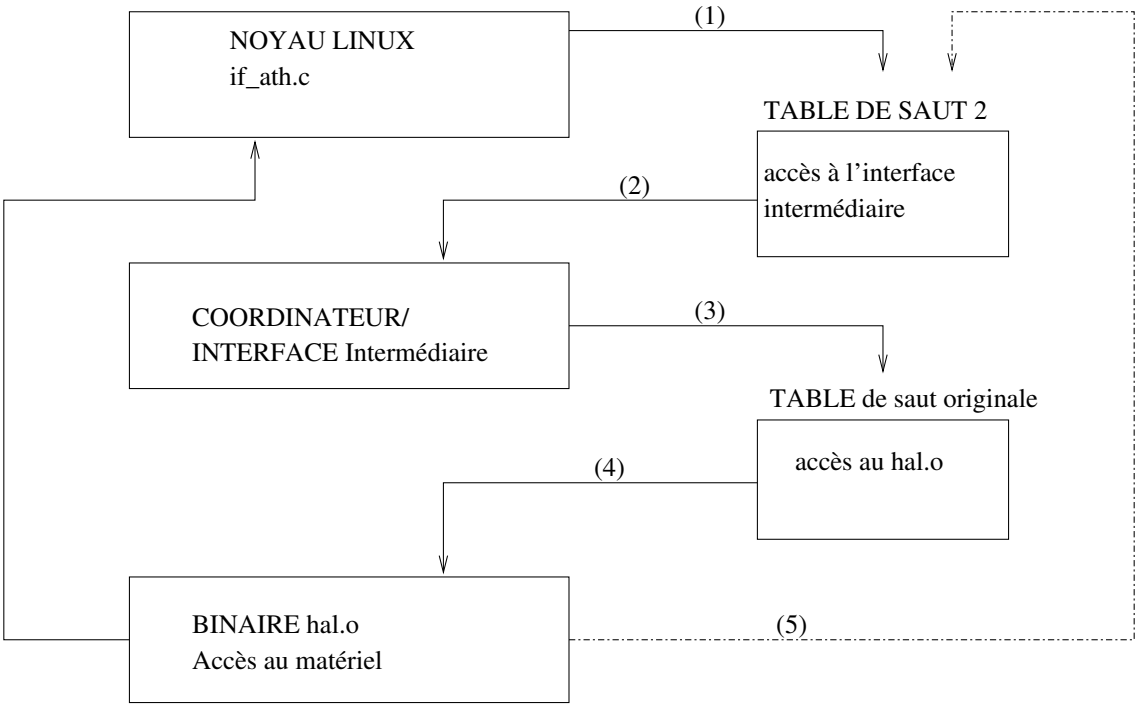


FIG. 6 – Interface intermédiaire conçu pour ajouter une coordinateur au pilote

Ainsi, selon la figure 6, les informations emprunteraient le chemin des flèches 1,2,3,4. Cependant, il n'y a, pour l'instant, aucun moyen de savoir ce qu'il se passe à l'intérieur du hal.o. M.Schaefer a émis l'hypothèse que le binaire fasse également appel à cette table de saut et nous pourrions, dans ce cas, intercepter le flux de données et les informations transitant de fonction en fonction en interne et cela via la flèche 5 (voir figure 6). On verra plus loin, à la section portant sur la modification du code source, comment intercepter ces pointeurs.

Une dernière remarque concernant le pointeur final défini dans le fichier `ah.h`, si l'on prend l'exemple de la fonction `get_RxDP`, on remarque qu'elle est traitée par une macro nommée `__ahdecl`() :

```
u_int32_t __ahdecl(*ah_getRxDP)(struct ath_hal*);
```

D'autre part, ici, on remarque que `get_RxDP` retourne un entier non signé 32 bit (ceci n'est qu'un exemple pour prendre conscience de la macro utilisée pour définir les fonctions). Très probablement cette macro (`__ahdecl`) reformate la syntaxe de la fonction pour qu'elle soit reconnue auprès du `hal.o`.

Remarque importante : Nous avons identifié la fonction qui effectue l'allocation en mémoire de nos structures. Cependant, il faut déterminer à quel moment cette allocation se produit, en d'autres termes, à quel niveau dans le code source la fonction `ath_attach` est appelée. Il s'agit de la ligne 273 du fichier

`if_ath.c` :

```
ah = _ath_hal_attach(devid, sc, 0, (void *) dev->mem_start, &status);
```

Cependant, on remarque ci-dessus la légère modification de syntaxe de la fonction `ath_attach`. C'est la fonction `_ath_hal_attach` qui est appelée, celle-ci est définie dans le fichier `ah_osdep.c`. Si l'on regarde la ligne 91 de ce fichier, on remarque que l'on appelle réellement la fonction définie dans notre fichier source principal `if_ath.c`. Voici l'extrait de cette ligne 91 :

```
struct ath_hal *ah = ath_hal_attach(devid, sc, t, h, &status);
```

4.8.9 Conclusion de l'analyse

Points positifs : Nous connaissons actuellement le fonctionnement du paramétrage de la carte et cela nous permet de rajouter de nouvelles fonctionnallités définissables depuis une commande système. Nous savons également comment modifier la source de telle sorte à intercepter les appels de fonction situés dans le binaire. Il est donc certain que l'on peut intercepter un flux allant du noyau linux au binaire `hal.o` (intermédiaire avec le matériel). Nous pourrions peut-être intercepter le flux interne au binaire à condition que ce dernier utilise notre table de saut.

Points négatifs : La gestion de la carte en mode master (access point) n'a pas été identifiée. Aussi, il n'est pas possible de modifier et de voir les «EDCA Parameter Set Element» comportant les informations de TXOP. La gestion se fait très propbablement en interne dans la `hal.o` ce qui expliquerait pourquoi l'on ne trouve pas les descriptions des éléments EDCA ni les algorithmes gérant réellement la carte en mode de fonctionnement. Nous n'avons pu observer que des adaptations de données pour les couches inférieures et non la gestion selon la norme IEEE802.11a,b,g.

5 Paramétrage de la carte

5.1 Introduction

Jusqu'à maintenant nous avons vu comment fonctionne le driver de manière interne. Cependant, il faut pouvoir le paramétrer depuis l'extérieur, lorsque le driver est chargé en tant que module au sein du système d'exploitation. Il s'agit ici d'utiliser le jeu d'outils wireless-tools[45]. Le kit fournit le jeu de commandes suivant :

iwconfig : configure une interface réseau sans-fil (wireless). Il s'agit des fonctionnalité générique d'une carte wireless

iwpriv : configure les paramètres optionnels (privés) d'une interface réseau sans fil. Il s'agit donc des fonctionnalités propres à une carte et c'est l'outil fondamental à connaître pour activer désactiver les diverses fonctions que nous avons avec Madwifi, par ex. WME, etc.

iwevent : Affiche les Événements Wireless (Wireless Events) générés par les pilotes et les changements de paramètres.

iwlist : Obtient plus d'informations wireless détaillées depuis une interface wireless telle que la liste des canaux, des AP, etc.

iwspy : Obtenir des statistiques wireless depuis des nœuds donnés.

Ces outils permettent de configurer l'interface wireless, il s'agit donc de configurer en partie la couche de liaison, le niveau 2. Par ces outils, on accède également aux réglages du support physique, de la couche PHY. Par exemple, on peut définir la fréquence, le débit, le mode (11a, b, g), etc. Mais on y définit également la couche intermédiaire WEP qui permet de crypter toute donnée issue des couches supérieures. Sans la clé, nous sommes juste en mesure de voir les trames de beacon et les informations définissant les normes utilisées et le ESSID. Les données sont cryptées.

5.2 Préparation de l'AP et activation du WME

Lorsque l'on désire rendre notre station AP, il suffit de tapers la commande :

```
iwconfig ath0 mode master
```

Il est évident qu'il faille définir un essid. Cependant, en lançant la commande `iwconfig ath0` sans rajouter de paramètres, on peut voir l'état actuel de la configuration telle qu'elle a été prise en compte par la carte. Le simple fait de lancer la commande ne garantit pas que la carte a pris sa nouvelle configuration en compte. En basculant dans le mode Master, on remarque que la carte se met automatiquement en mode 802.11a et utilise, par conséquent, une plage de fréquence interdite en Europe (5GHz). En voulant changer la fréquence manuellement avec `iwconfig ath0 channel 6` on remarque que la carte ne change pas sa fréquence. Pour éviter ce cas de figure, il faut accéder aux fonctionnalités "Bas-niveau" et propres à la carte, et pour ce faire on doit utiliser la commande `iwpriv`. En tapant `iwpriv ath0`, on affiche les paramètres possibles avec Madwifi :

[45] On peut télécharger ces outils sur le site `http://www.hpl.hp.com/personal/Jean_Tourrilhes/Linux/`

```
ath0      Available private ioctl :
          setoptie         (8BE8) : set 256 byte  & get   0
          getoptie         (8BE9) : set   0       & get 256 byte
          setkey           (8BE2) : set  60 byte  & get   0
          delkey           (8BE4) : set   7 byte  & get   0
          setmlme          (8BE6) : set  10 byte  & get   0
          addmac           (8BEA) : set   1 addr  & get   0
          delmac           (8BEC) : set   1 addr  & get   0
          chanlist         (8BEE) : set  32 byte  & get   0
          setparam         (8BE0) : set   2 int   & get   0
          getparam         (8BE1) : set   1 int   & get   1 int
          turbo            (0001) : set   1 int   & get   0
          get_turbo        (0001) : set   0       & get   1 int
          mode             (0002) : set   1 int   & get   0
          get_mode         (0002) : set   0       & get   1 int
          authmode         (0003) : set   1 int   & get   0
          get_authmode     (0003) : set   0       & get   1 int
          protmode         (0004) : set   1 int   & get   0
          get_protmode     (0004) : set   0       & get   1 int
          mcastcipher      (0005) : set   1 int   & get   0
          get_mcastcipher  (0005) : set   0       & get   1 int
          mcastkeylen      (0006) : set   1 int   & get   0
          get_mcastkeylen  (0006) : set   0       & get   1 int
          ucastciphers     (0007) : set   1 int   & get   0
          get_uciphers     (0007) : set   0       & get   1 int
          ucastcipher      (0008) : set   1 int   & get   0
          get_ucastcipher  (0008) : set   0       & get   1 int
          ucastkeylen      (0009) : set   1 int   & get   0
          get_ucastkeylen  (0009) : set   0       & get   1 int
          keymgtalgs       (0015) : set   1 int   & get   0
          get_keymgtalgs   (0015) : set   0       & get   1 int
          rsncaps          (0016) : set   1 int   & get   0
          get_rsncaps      (0016) : set   0       & get   1 int
          roaming          (000C) : set   1 int   & get   0
          get_roaming      (000C) : set   0       & get   1 int
          privacy          (000D) : set   1 int   & get   0
          get_privacy      (000D) : set   0       & get   1 int
          countermeasures  (000E) : set   1 int   & get   0
          get_countermeas  (000E) : set   0       & get   1 int
          dropunencrypted  (000F) : set   1 int   & get   0
          get_dropunencry  (000F) : set   0       & get   1 int
          wpa              (000A) : set   1 int   & get   0
          get_wpa          (000A) : set   0       & get   1 int
          driver_caps      (0010) : set   1 int   & get   0
          get_driver_caps  (0010) : set   0       & get   1 int
          maccmd           (0011) : set   1 int   & get   0
          wme              (0012) : set   1 int   & get   0
          get_wme          (0012) : set   0       & get   1 int
          hide_ssid        (0013) : set   1 int   & get   0
          get_hide_ssid    (0013) : set   0       & get   1 int
          ap_bridge        (0014) : set   1 int   & get   0
          get_ap_bridge    (0014) : set   0       & get   1 int
          inact            (0017) : set   1 int   & get   0
          get_inact        (0017) : set   0       & get   1 int
          inact_auth       (0018) : set   1 int   & get   0
          get_inact_auth   (0018) : set   0       & get   1 int
          inact_init       (0019) : set   1 int   & get   0
          get_inact_init   (0019) : set   0       & get   1 int
```

Par défaut, la carte choisit automatiquement la norme (802.11a,b ou g). Voici ce qui nous permet de commuter entre les divers modes :

```
iwpriv ath0 mode 1      #lock operation to 11a only
iwpriv ath0 mode 2      #lock operation to 11b only
iwpriv ath0 mode 3      #lock operation to 11g only
iwpriv ath0 mode 0      #autoselect from 11a/b/g (default)
```

Afin que notre AP ne soit pas automatiquement commuté en mode IEEE802.11a à 5GHz, nous allons par défaut le forcer à travailler en mode IEEE802.11b via la commande `iwpriv ath0 mode 2`.

Il s'agit maintenant d'attribuer une adresse ip à notre access point. Ce n'est pas aussi simple que lorsque l'on a configure une simple station dans le sens qu'il ne suffit pas simplement d'utiliser la commande `ifconfig <athX><ip>`. En effet, le but premier de l'AP est de relier le réseau WLAN au LAN et il fait donc le pont entre les deux interfaces (ethernet-wireless). Les AP ont en général la même adresse IP pour l'interface WLAN que pour l'interface Ethernet. Sous linux, il faut simuler ce lien. Nous devons pour ce faire utiliser une fonctionnalité du kernel. Il s'agit de la fonction de pont (bridge). Il est donc nécessaire, lors de la compilation du kernel (2.6.x), de mettre `CONFIG_BRIDGE_NETFILTER=y` dans le fichier de configuration des sources du noyau linux (généralement /usr/src/linux-<version>/.config). Ensuite, il faut installer les outils de configuration (paquetage «bridge-utils» sous debian). Lorsque l'on a l'interface wireless configuré en AP, il faut définir le pont :

1) **ifconfig athX 0.0.0.0** *On attribut pas d'adresse à l'interface WLAN*
2) **ifconfig ethX 0.0.0.0** *Il s'agit ici de l'interface Ethernet faisant le pont*
3) **brctl addbr br0** *On crée une interface Ethernet* ***virtuelle***
4) **brctl addif br0 athX** *On établit un lien entre l'interface virtuelle et l'interface WLAN*
5) **brctl addif br0 ethX** *On établit un lien entre l'interface virtuelle et l'interface de sortie Ethernet*

Vu de l'*extérieur*, br0 fait office d'une seule et unique interface. Si l'on désire effectuer un "ping" depuis le réseau filaire via l'interface réelle Ethernet, c'est br0 qui répondra au ping. Le même comportement s'effectuera depuis le réseau sans fil. On peut voir une illustration de ce pont à la figure 7 à la page 42.

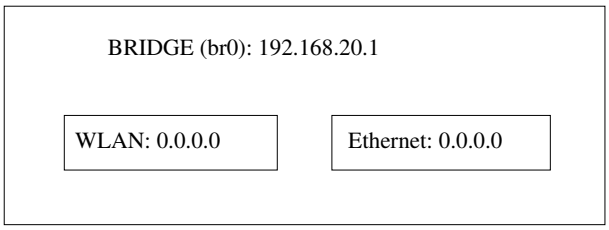


FIG. 7 – Illustation du pont entre la carte réseau Ethernet et la carte wireless

Il est possible que l'on ait pas besoin de faire le pont directement avec une interface Ethernet et que l'on désire faire un AP ayant une adresse IP au niveau du WLAN et une autre adresse IP sur son interface Ethernet. Cela est possible mais il est nécessaire, pour l'interface WLAN de créer malgré cela une interface virtuelle, cette-fois ci liée uniquement avec la carte wireless. On opère de la manière suivante :

1) **ifconfig athX 0.0.0.0**
2) **brctl addbr br0**
3) **brctl addif br0 athX**

La suite des opérations nécessiterait un routage vers le réseau Ethernet, ayant probablement une autre classe d'adresse. Dans le cadre de ce projet, cela a été nécessaire et un exemple concret est donné plus loin.

Le mode wme n'est pas activé par défaut, on peut vérifier l'état et l'activer de la manière suivante :

```
-- Vérification de l'état
blues:# iwpriv ath0 get_wme
ath0      get_wme:0

-- Il n'est pas activé car il retourne '0'
-- On l'active:
blues:# iwpriv ath0 wme 1

-- On vérifie
blues:# iwpriv ath0 get_wme
ath0      get_wme:1
```

Pour effectuer les tests, nous avons choisi le mode 802.11b. Ensuite avec le second poste (moniteur), nous avons capturé un beacon pour vérifier si notre station s'anonce convenablement.

5.3 Banc de test

5.3.1 Introduction

Pour effectuer les tests de performance de l'état actuel du driver, avec le mode WME prédéfini (selon la méthode de coordination EDCA), nous avons à disposition une station fixe (DELL 4550) ayant une carte DLINK disposant du chipset ATHEROS® AR5212, deux stations mobiles (laptop) disposant chacune d'une carte NetGEAR WAG511(également un chipset ATHEROS® AR5212.

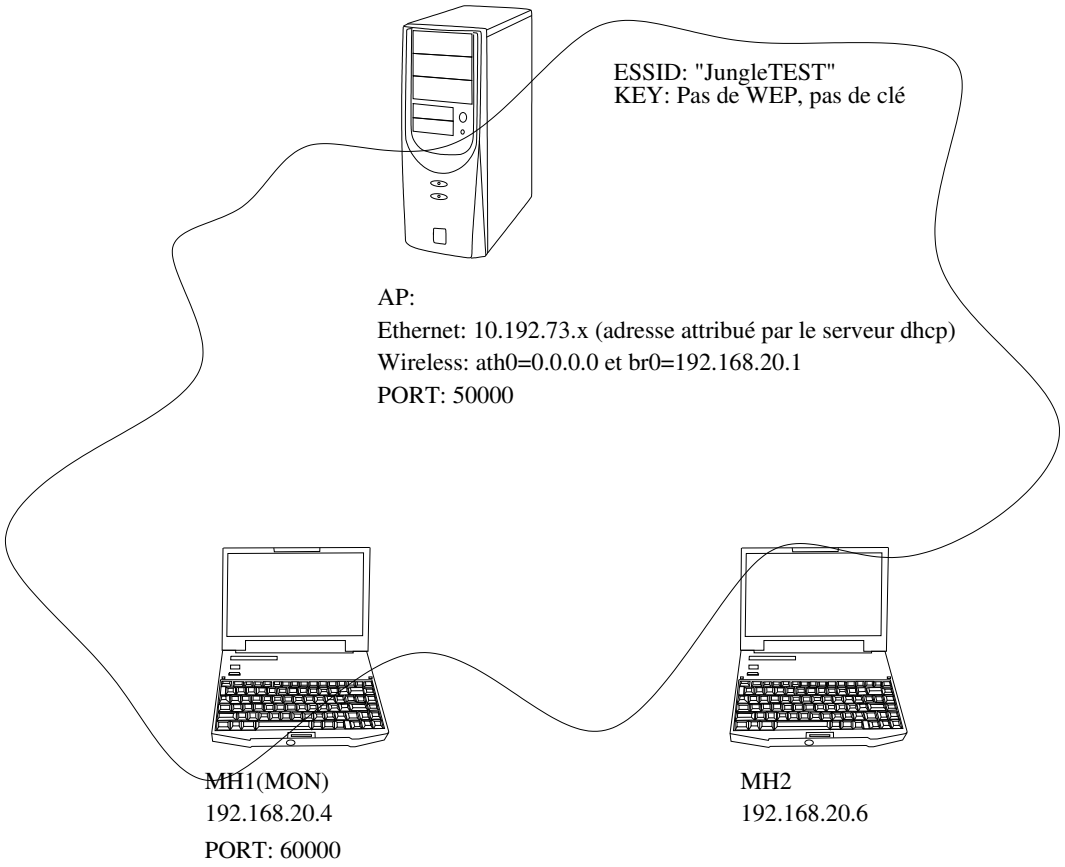


FIG. 8 – Illustration du banc de test

La figure 8 à la page 44 nous permet de voir comment est défini notre réseau wireless. La passerelle avec le réseau Ethernet faisant office d'AP possède l'adresse IP 192.168.20.1. La station mobile (MH1) qui dispose à la fois d'une carte ATHEROS® pour faire les tests et d'une carte supplémentaire passive uniquement dédiée au monitoring (utilisée avec Ethereal) dispose de l'adresse IP 192.168.20.4. Finalement, le laptop qui servira essentiellement à générer un trafic sans priorité (MH2) dispose de l'adresse IP 192.168.20.6. Les ports 50000 et 60000 spécifiés sur la figure serviront lorsque nous ferons des mesures avec un générateur de paquet (packETH) afin de tester la différenciation de service à l'aide de la norme IEEE802.1d. L'usage de ces ports permet d'identifier immédiatement les paquets qui nous intéressent et également voir s'ils ont subi un traitement.

Aussi, lors des tests, l'idéal aurait été d'installer un serveur SIP, mais il y a eu quelques ennuis de dernières minutes qui ont empêchés le bon fonctionnement de SER[46]. Pour éviter de perdre du temps,

[46] SER : Service Express Routeur. Il s'agit d'un serveur SIP simple à mettre en route

une passerelle vers le réseau Ethernet a été mise en place mais pour ce faire il est indispensable d'activer le système "ip_forward" dans le kernel linux ainsi que le masquerading. Un scipt a été écrit pour activer le masquerading mais, en parallèle, j'ai découvert un script nettement plus performant qui analyse les interfaces montées, les règles de routage et qui active automatiquement le masquerading[47], il s'agit de la commande ipmasq. Lors de son exécution elle active également "ip_forward" dans le kernel[48]

5.3.2 Configuration de l'AP

Voici la procédure à suivre :

1) **modprobe ath_pci|** *Chargement du pilote, modprobe se charge de gérer les dépendances et charge les modules manquants*
2) **iwpriv ath0 mode 2** *mode IEEE802.11b*
3) **iwpriv ath0 wme 1** *Activation du mode wme*
4) **iwconfig ath0 essid "JungleTEST"** *Nous donnons un nom à notre ESSID*
5) **iwconfig ath0 channel 3** *Nous choisissons un canal libre, dans la salle où les mesures ont été faites, le canal 3 était libre*
6) **iwconfig athX key <key>** *optionnel, le WEP n'a pas été activé dans le cadre des tests*
7) **ifconfig ath0 0.0.0.0** *On active la carte en y attribuant une adresse IP passive*
8) **brctl addbr br0** *Création de l'interface virtuelle*
9) **brctl addif br0 ath0** *Etablissement du pont entre la carte wireless et l'interface Ethernet virtuelle*
10) **ifconfig br0 192.168.20.1** *Attribution de l'adresse à l'interface virtuelle*
11) **ipmasq -v** *Activation du masquerading*

L'AP est prêt à fonctionner.

5.3.3 Configuration des stations mobiles

La configuration des stations mobiles est plus simple. Elle ont chacune comme route par défaut la passerelle, c'est-à-dire l'adresse de l'interface br0 de l'AP. Il s'agit dans notre exemple de 192.168.20.1. Aussi, si l'on veut accéder au réseau externe (internet), il faut copier le resolv.conf de l'AP, car il comporte la liste des serveurs DNS. Pour ce faire, dès que le réseau wireless est configuré, la commande : `scp <user>@192.168.20.1:/etc/resolv.conf /etc/resolv.conf`. Voici la procédure à suivre pour chaque station :

1) **modprobe ath_pci** *Dans le cas où l'insertion de la carte PCMCIA ne suffit pas pour charger automatiquement le module, on le charge manuellement*
2) **iwpriv ath0 wme 1** *Activation de WME*
3) **iwconfig ath0 essid "JungleTEST"** *Choix du ESSID*
4) **iwconfig ath0 key <key>** *Optionnel, nous ne passons pas par cette étape car le WEP n'a pas été utilisé pour effectuer les tests*
5) **ifconfig ath0 192.168.20.x** *Ici x=4 pour MH1 et 6 pour MH2*
6) **route add default gw 192.168.20.1** *On définit la passerelle comme étant l'AP*
7) **scp <user>@192.168.20.1 :/etc/resolv.conf /etc/resolv.conf** *Cette étape n'est pas obligatoire si les DNS sont déjà configurés ou si l'on ne fait que du trafic local*

Les stations mobiles sont opérationnelles.

[47] Le masquerading consiste simplement à modifier l'adresse IP d'un paquet du réseau local et lui attribuer l'adresse IP de l'interface de sortie. Ici, notre réseau local est de type 192.168.20.x alors que l'interface de sortie possède une adresse de type 10.192.73.x

[48] Pour activer manuellement ip_forward dans le noyau, il suffit d'executer : `echo "1" >/proc/sys/net/ipv4/ip_forward`

5.3.4 Configuration de la station de monitoring

Selon la figure 8 à la page 44, c'est MH1 qui possède la carte supplémentaire. Pour la configurer nous opérons de la manière suivante :

1) **iwconfig eth2 mode mon** *Mettre la seconde carte wireless en mode monitor*[49]
2) **iwconfig eth2 essid "JungleTEST"** *Cette phase n'est pas forcément obligatoire, cependant, il faut spécifier le canal utilisé par JungleTEST*
3) **iwconfig eth2 channel 3** *On se connecte sur le bon canal*
4) **iwconfig eth2 key <key>** *Au besoin on spécifie la clé, mais cette étape n'est pas nécessaire pour notre test*
5) **ifconfig eth2 0.0.0.0** *On active la carte, elle est utilisable depuis Ethereal*

A ce stade il est possible d'observer les trames de beacon, voir l'exemple à la figure 9 à la page 47

[49] Dans ce mode elle est capable de voir les trame de contrôle IEEE802.11. Sans ce mode, seules les trames au niveau "Ethernet" sont visible. En effet, le but d'une carte wireless est qu'elle se comporte, vu des couche supérieures, comme une carte Ethernet.

```
No.     Time        Source                Destination           Protocol Info
     11 0.000165    D-Link_86:b6:4a       Broadcast             IEEE 802.11 Beacon frame

Frame 11 (63 bytes on wire, 63 bytes captured)
    Arrival Time: Dec  2, 2004 14:42:10.748003000
    Time delta from previous packet: 0.000007000 seconds
    Time since reference or first frame: 0.000165000 seconds
    Frame Number: 11
    Packet Length: 63 bytes
    Capture Length: 63 bytes
IEEE 802.11
    Type/Subtype: Beacon frame (8)
    Frame Control: 0x0080 (Normal)
        Version: 0
        Type: Management frame (0)
        Subtype: 8
        Flags: 0x0
            DS status: Not leaving DS or network is operating in AD-HOC mode (To DS: 0  From DS: 0) (0x00)
            .... .0.. = More Fragments: This is the last fragment
            .... 0... = Retry: Frame is not being retransmitted
            ...0 .... = PWR MGT: STA will stay up
            ..0. .... = More Data: No data buffered
            .0.. .... = WEP flag: WEP is disabled
            0... .... = Order flag: Not strictly ordered
    Duration: 0
    Destination address: ff:ff:ff:ff:ff:ff (Broadcast)
    Source address: 00:0f:3d:86:b6:4a (D-Link_86:b6:4a)
    BSS Id: 00:0f:3d:86:b6:4a (D-Link_86:b6:4a)
    Fragment number: 0
    Sequence number: 3282
IEEE 802.11 wireless LAN management frame
    Fixed parameters (12 bytes)
        Timestamp: 0x000000000E006243
        Beacon Interval: 0,102400 [Seconds]
        Capability Information: 0x0021
            .... .... .... ...1 = ESS capabilities: Transmitter is an AP
            .... .... .... ..0. = IBSS status: Transmitter belongs to a BSS
            .... .... .... 00.. = CFP participation capabilities: No point coordinator at AP (0x0000)
            .... .... ...0 .... = Privacy: AP/STA cannot support WEP
            .... .... ..1. .... = Short Preamble: Short preamble allowed
            .... .... .0.. .... = PBCC: PBCC modulation not allowed
            .... .... 0... .... = Channel Agility: Channel agility not in use
            .... .0.. .... .... = Short Slot Time: Short slot time not in use
            ..0. .... .... .... = DSSS-OFDM: DSSS-OFDM modulation not allowed
    Tagged parameters (27 bytes)
        Tag Number: 0 (SSID parameter set)
        Tag length: 10
        Tag interpretation: JungleTEST
        Tag Number: 1 (Supported Rates)
        Tag length: 4
        Tag interpretation: Supported rates: 1,0(B) 2,0(B) 5,5 11,0 [Mbit/sec]
        Tag Number: 3 (DS Parameter set)
        Tag length: 1
        Tag interpretation: Current Channel: 3
        Tag Number: 5 ((TIM) Traffic Indication Map)
        Tag length: 4
        Tag interpretation: DTIM count 0, DTIM period 1, Bitmap control 0x0, (Bitmap suppressed)

0000  80 00 00 00 ff ff ff ff ff ff 00 0f 3d 86 b6 4a   ............=..J
0010  00 0f 3d 86 b6 4a 20 cd 43 62 00 0e 00 00 00 00   ..=..J .Cb......
0020  64 00 21 00 00 0a 4a 75 6e 67 6c 65 54 45 53 54   d.!...JungleTEST
0030  01 04 82 84 0b 16 03 01 03 05 04 00 01 00 00      ...............
```

FIG. 9 – Un exemple de beacon disséqué par Ethereal

Cette capture a été faite lorsque l'on faisait fonctionner notre réseau avec le support WME. Ce beacon provient de notre AP. On remarque qu'il est impossible de voir les éléments EDCA, indispensable pour savoir si les TXOP sont distribués correctement. Cependant, soulignons que l'on a une information de temps d'apparition de la trame, ainsi que la différence de temps passé entre la trame courante et la précédente trame (Time delta from previous packet).

5.3.5 Conception de script

Comme les procédures de mise en place des stations sont relativement longues à mettre en place, il a fallut concevoir des scripts pour automatiser ces processus. Dans un but d'optimisation, le script est capable de mettre en route une station mobile ou un AP. Il est capable d'utiliser les paramètres par défaut (défini dans le script), d'utiliser un fichier de configuration (spécifier dans la ligne de commande ou pris par défaut dans le répertoire /usr/local/etc. Il est capable de fonctionner en spécifiant directement les paramètres d'exécution en ligne (choix AP, MH ou moniteur, activer ou désactiver) ou sinon, fonctionner de manière interactive (via des menus). Il est écrit en bash (langage shell). Voici la commande en question :

```
boston@blues:~/testbed/captures/Test8021q/sansWME/MH1$ wlanConfig --help
Syntax: wlanTest <ap|mh> <up|down>
or: wlanTest <--config> <file>
+------+
| MH1  |----+          ____________________________
+------+    |        | +---------+         +--------+ |
             ===-----| | AP-ATH0 |---BR0---|AP-ETHX | |
+------+    |        | +---------+    |    +--------+ |
| MH2  |----+        +----------------|---------------+
+------+                          BR_IF=AP_IP
```

Pour utiliser le script en mode interactif, il ne faut pas définir de paramètres et taper simplement "wlanConfig" dans la ligne de commande. D'après le fichier de configuration, il est défini si la station courante est l'AP, MH1 ou MH2[50], le menu adéquat au type de station apparaît (comme nous l'avons vu, la procédure pour définir un AP est différente de celle pour définir une simple station mobile). De manière interactive, on peut redéfinir l'adresse IP, l'activation / désactivation du WME et finalement activer/désactiver la carte. Si l'on désire approfondir la configuration, il faut simplement éditer le fichier de configuration. Voici un exemple du fichier de configuration :

[50] pour notre test, les dénominations MH1 et MH2 on été insérées dans le script afin d'éviter toute confusion dans la hâte de mettre en place le banc de test, mais il est possible de définir une station mobile générique

```
#!/bin/bash
#Script de mise en place du test pour l'AP
#
########################################
# Préparation du MH                    #
#                                      #
########################################

  # This is a wlan config file (test of conformity)
  wlanConfigFile="YES"
  # Wich station you ARE (AP|MH1|MH2|MH)
  THISISAN=MH1
  # PATH to IFCONFIG
  IFCONFIG=/sbin/ifconfig
  # PATH to IWTOOLS
  IWTOOLS=/sbin
  # PATH to Bridge-utils
  BRPATH=/usr/sbin
  # MONITOR IF
  MON_IF=eth2
  # ATHEROS IF
  ATH_IF=ath0
  # Ethernet Interface
  ETH_IF=eth0
  # Second Interface
  ETH2_IF=eth1
  # Branche Interface
  BR_IF=br0
  # NAME OF THE WLAN ID
  ESSID="JungleTEST"
  # CHANNEL 1-13
  CHANNEL=3
  # LEGACY: 802.11a=1 802.11b=2 802.11g=3 auto=0
  MODE=2
  # WME Activation (no=0, yes=1)
  WME=0
  # IP of the AP
  IP_AP=192.168.20.1
  # IP of the mobile host
  IP_MH=192.168.20.4
  # Debug Window
  DEBUGWINDOW="YES"
```

Dans le fichier de configuration, il est possible de définir deux interfacse Ethernet (la première est l'interface de sortie sur le réseau externe et la seconde celle du réseau local), le ESSID, le canal utilisé, le mode utilisé (802.11a,b,g), l'adresse IP de la passerelle (si l'on est MH), l'adresse IP du MH et finalement il est possible de demander l'exécution automatique des fenêtres de monitoring lors de la mise en place du banc de test, cela a pour effet d'ouvrir un terminal affichant les accès au essid (iwevent), un terminal affichant l'état de l'interface "Ethernet" de la carte wireless (ifconfig ath0) et l'état de la carte wireless (iwconfig ath0). Ceci permet de gagner un temps énorme pour mettre en place l'installation. On pourrait étendre les fonctionnalités, malheureusement le temps manque pour le faire.

5.3.6 Logiciels utilisés

Analyseur Pour effectuer nos mesures, il est nécessaire d'avoir un analyseur capable de voir les trames au plus bas niveau (IEEE802.11). Ethereal [1] est l'analyseur le plus approprié (sans compter qu'il est libre d'utilisation). Il est disponible sur linux et sur windows. Il est conseillé d'installer la dernière version (actuellement 0.10.7) car le support des trames IEEE802.11 est très récent. Aussi, il est possible, depuis l'analyseur, d'extraire les stream VoIP ce qui nous permettrait d'apprécier après avoir terminé les tests, la dégradation éventuelle au niveau purement qualitatif (une perte de paquet implique une perte de données probablement appréciable au niveau auditif).

Générateur de trafic Dans un premier temps, afin de voir si le champ "DSCP" est effectivement utilisé pour déterminer la priorité au sein des files d'attentes WME, il suffit d'initier une communication VoIP (selon SIP), entre un des deux mobiles (nous utiliserons MH1). Le logiciel utilisé est kphone (sous linux). Dans un deuxième temps, il faut générer des paquets utilisant les propriétés de IEEE802.1d qui permet de faire des réservation de priorité sur 8 niveau. Pour faire le bon choix de priorité au sein de IEEE802.1d,

il faut consulter la table des correspondances priorité-catégorie1 à la page 8. Aussi, au cours du projet, la recherche d'un générateur de paquets était indispensable. Après quelques entrées dans les moteurs de recherche, un outil s'est présenté, il s'agit de packETH[2]. Il est possible de générer des paquets en choisissant toute la séquence, le rythme, le contenu ainsi que la priorité des paquets selon IEEE802.1d.
Pour générer du trafic non prioritaire avec la seconde station mobile, il suffit simplement d'ouvrir une connexion TCP et d'envoyer un flux de données continu. Pour ce faire, il suffit simplement de lancer la commande suivante :

```
tar cf - * | ssh -e none <user>@192.168.20.1 "tar tf -"
```

Cette commande ne fait qu'archiver le répertoire courant et les sous-répertoires, envoyer au travers d'une connexion ssh les données à la station partenaire (ici l'AP est également un PC fonctionnant sous linux). Dans les "guillemets" on exécute la commande tar pour désarchiver les données provenant de la station mobile, mais avec l'argument "t" cela se fait dans le "vide". Ainsi nous générons du trafic artificiellement.

Paramétrage de Ethereal Après avoir commencé les tests, le nombre de trames atteignant facilement 100'000 nécessitait une configuration d'Ethereal de telle sorte que seuls les paquets nous intéressant soient mis en valeur. En effet, il est possible de filtrer et de n'afficher que les trames répondant à nos attentes, cependant, dans le cadre de ces tests, il est utile de voir également d'autres trames (notamment les beacons). Pour rendre la visualisation plus agréable, j'ai redéfini la coloration des trames lors du test avec VoIP. On peut voir les règles de coloration à la figure 10 à la page 51.

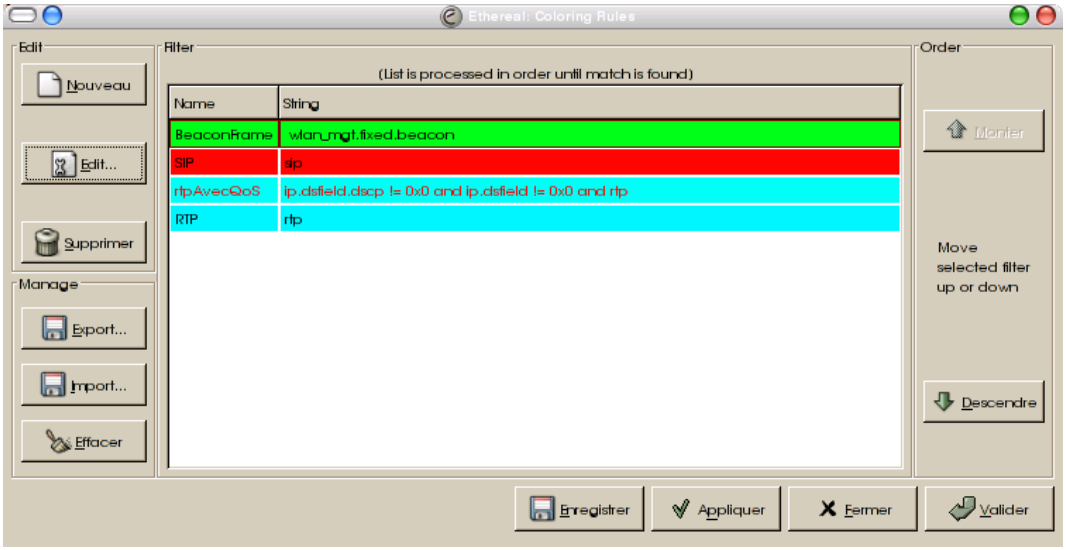


FIG. 10 – Définition des règles de coloration d'Ethereal

Les trames de beacon sont vertes, les trames rouges définissent la signalisation SIP, les trames bleues sont celles transportant le protocol RTP (ayant par extention un stream voice), mais si la couleur du texte n'est plus noir mais rouge, cela signifie que les paquets au niveau IP (champ DSCP) sont marqués d'une priorité pour le vocal. Aussi, la fenêtre de détail a été disposée sur la droite et prenant le maximum de place afin de visualiser le maximum d'informations contenues dans la trame selectionnée. Cette redéfinition de l'agencement d'Ethereal paraît être un «caprice», cependant elle a permis d'être plus efficace pour parcourir les multitudes de mesures. On peut voir un exemple d'une capture faite avec Ethereal à la figure 11 à la page 52.

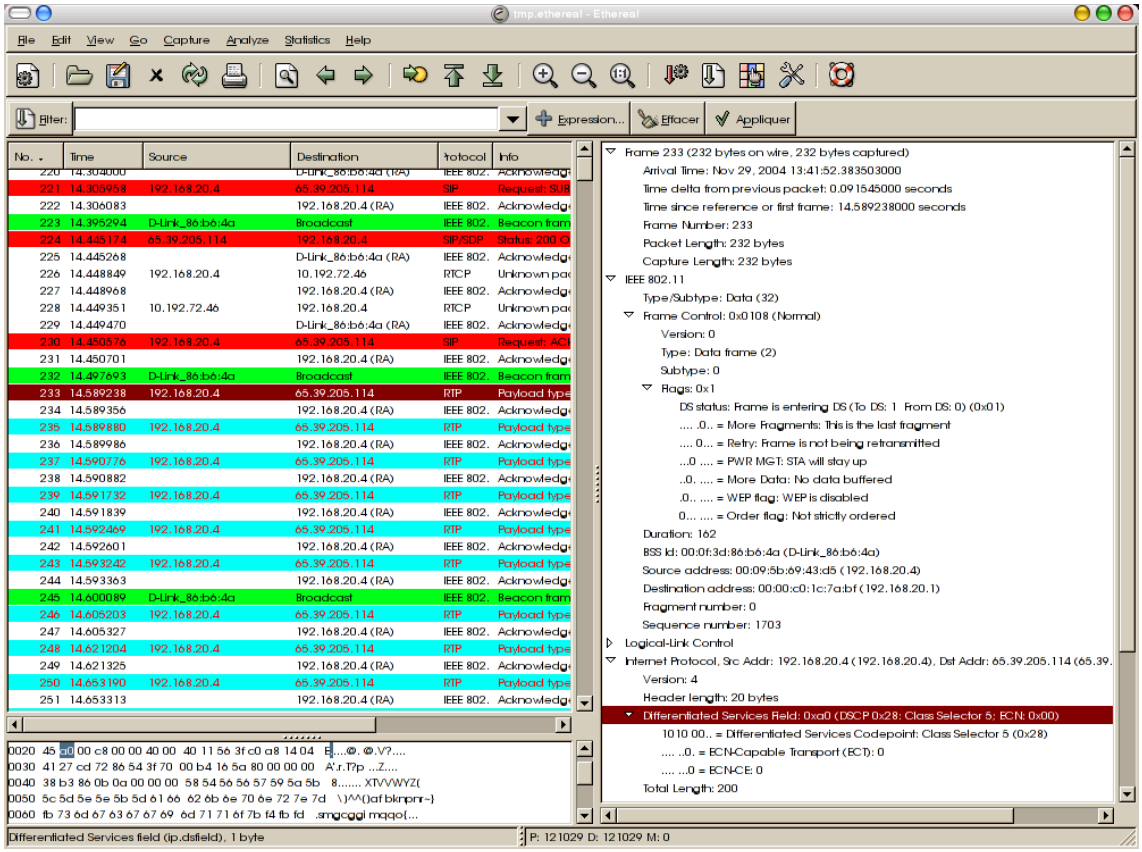


FIG. 11 – Agencement et configuration sur mesure d'Ethereal

Paramétrage que packETH Il s'agit maintenant de préparer notre générateur de paquets. Le premier défaut à souligner lorsque l'on utilise un générateur de paquet est qu'il ne faut pas s'attendre à obtenir une réponse en retour. C'est pourquoi nous allons, pour tester la QoS de l'AP au MH et tu MH à l'AP, lancer le générateur sur l'AP et sur un des deux MH (au choix MH1). Pour ce faire, selon les règles de routage, pour être sûr que notre paquet est ouvert des deux côtés (du côté de l'AP et de celui du MH), nous devons spécifier l'adresse source et destination hardware, et pour nous simplifier la lecture des trames dans l'analyseur, nous définirons également les bonnes adresses de source et de destination au niveau IP. Finalement, pour simplifier le tri des paquets provenant de l'AP et ceux provenant du MH, nous définirons que les ports de source et de destination de l'AP soient égaux à 50000 et ceux du MH soient égaux à 60000.

paquETH est capable de lire un fichier (base de données) comportant une liste d'adresse, voici les adresse du banc de test :

```
192.168.20.1,00:09:5b:69:43:d5,APblues
192.168.20.1,00:00:c0:1c:7a:bf,APDell
192.168.20.4,00:09:5B:69:43:D5,MHBlues
192.168.20.6,00:09:5b:69:43:da,MHJazz
```

Notre ap est "APDell" et notre mobile host (station mobile) est MHBlues. MHJazz est la station MH2. Maintenant il s'agit de définir un paquet UDP, la priorité (1-6), on peut voir une capture à la figure 12 à la page 54

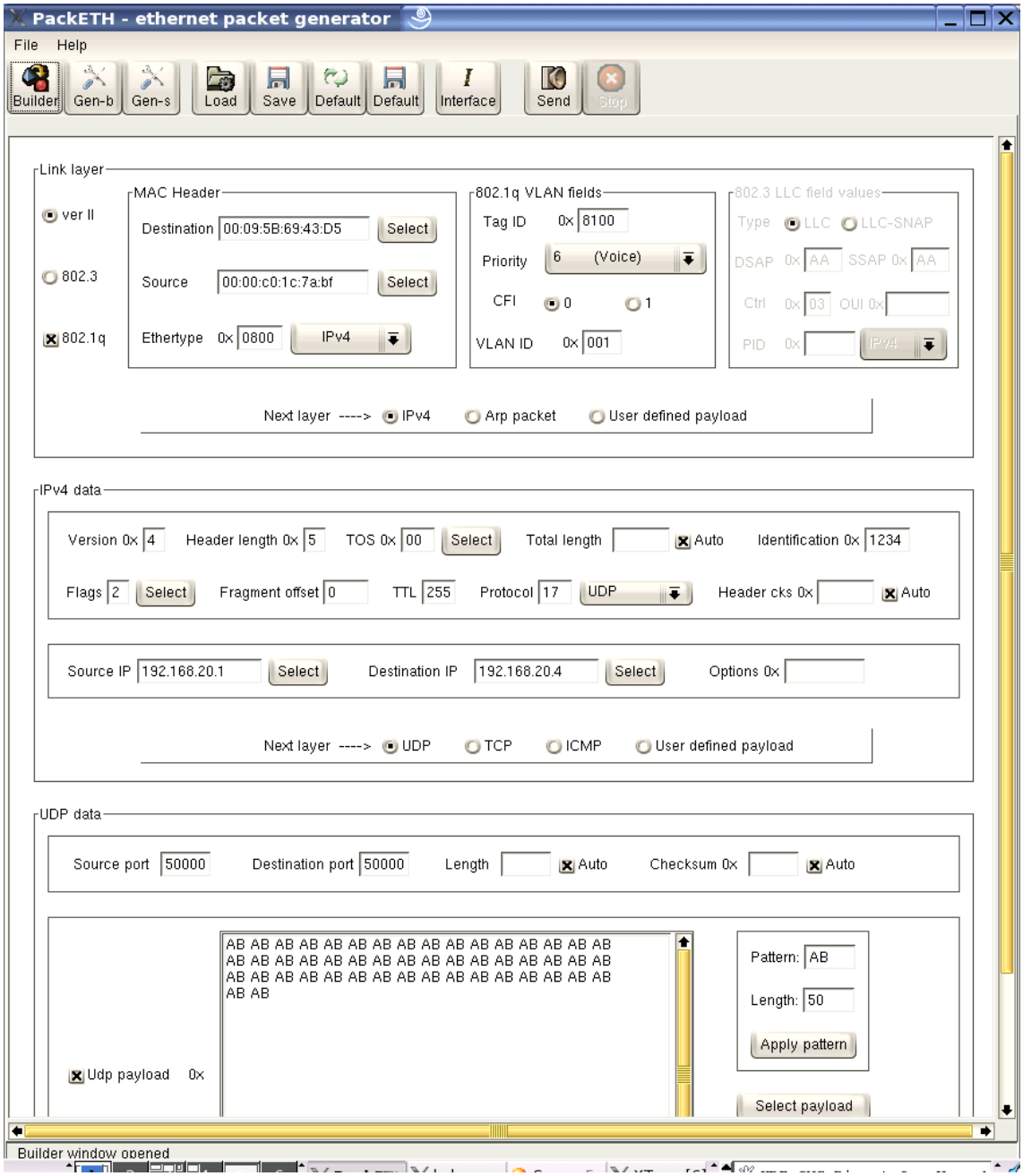


FIG. 12 – Capture de la fenêtre de configuration principale de packETH [2]

Ensuite on définit la séquence d'envoi, le rythme (toutes les 5ms). On définit également un payload (il est possible d'appliquer un pattern).

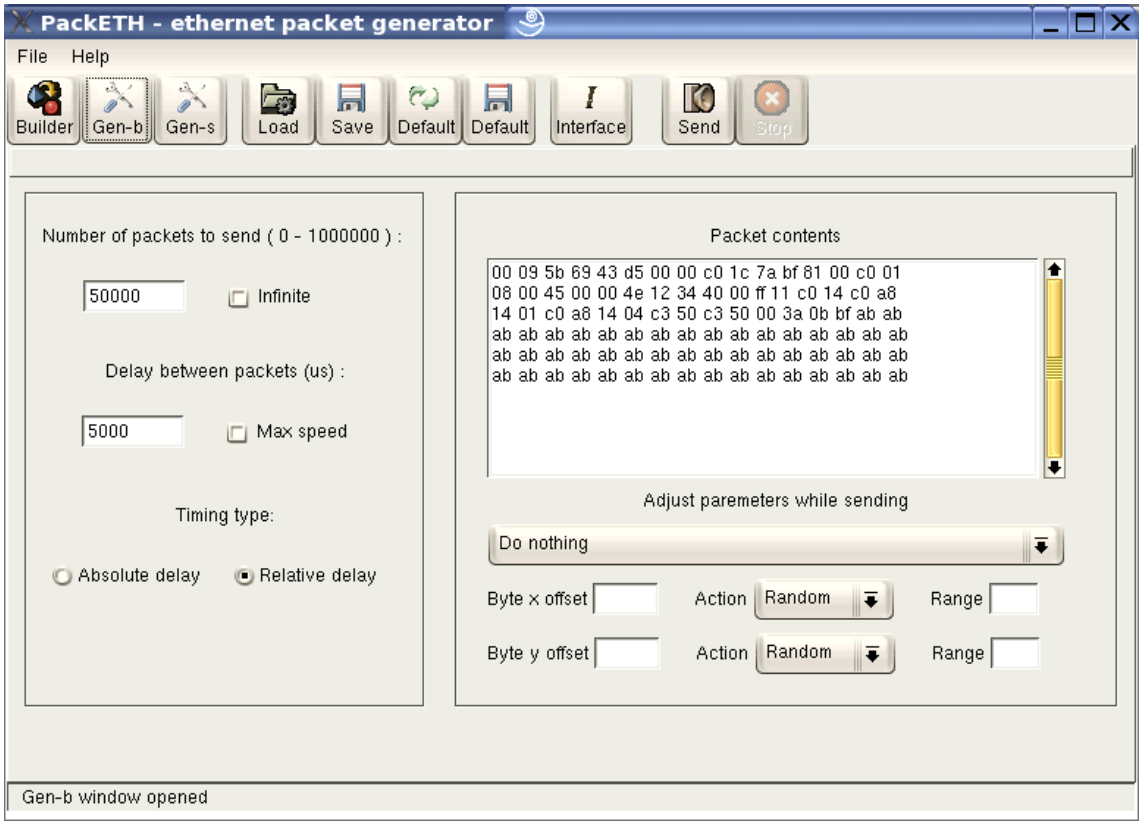


FIG. 13 – Capture de la fenêtre de configuration des séquences de packETH [2]

Avant de lancer le processus, il faut se mettre dans la deuxième fenêtre (Gen-b), choisir l'interface de sortie (pour le MH, il s'agit de ath0, pour l'AP il s'agit de br0). La séquence est envoyée au rythme programmé, en parallèle il suffira de saturer la bande passante avec la seconde station mobile (MH2) ceci en trafic non prioritiaire. Les détails concernant la méthode de mesure sont explicités plus loin, à la section 6.2.

6 Test des performances

6.1 Introduction

Les outils utilisés pour préparer le banc de test ont étés données dans la section 5.3.6. D'après l'analyse du driver, il n'y a pas moyen de voir ce qui s'opère dans le noyau du pilote `hal.o`. Il n'a été découvert que quelques fonctions faisant référence aux quatre catégories d'accès ainsi que les trames de "QoS". Le but premier de ces tests est de voir si, lorsque l'on active le mode WME sur toutes les cartes de notre réseau, il y a effectivement une différence de comportement et une amélioration de la gestion des trafics prioritaires.

Cependant, après avoir bien étudié l'analyseur Ethereal, il n'y a aucun moyen de voir les éléments de paramètre EDCA contenu logiquement dans les beacons. Il n'est non plus pas possible de voir les fameuses trames de QoS dont nous avions pu voir la description (voir section 4.8.6).

6.2 Méthode de mesure

Le seul moyen qui se présente est d'effectuer des mesures basées sur les statistiques. La méthode de coordination utilisée est EDCA, cela signifie que la différenciation de traitement entre les trames prioritaires et les trames moins prioritaires se fait au travers d'un temps d'attente proportionel à la catégorie pour accéder au canal. Selon les temps AIFSn, dès le moment où le canal est libre, les trames AC_VO auront un temps d'attente plus court à respecter avant de pouvoir accéder au canal par rapport au autres catégories. Par le même principe, AC_VI, moins prioritaire qu'AC_VO verra son temps d'attente légèrement augmenter, AC_BE encore plus, pour finir la catégorie AC_BK verra sa priorité réduite au minimum, donc son temps d'attente augmenté au maximum.

Sans la gestion EDCA, les temps AIFS n'existent pas, on peut voir une illustration à la figure 14 à la page 55

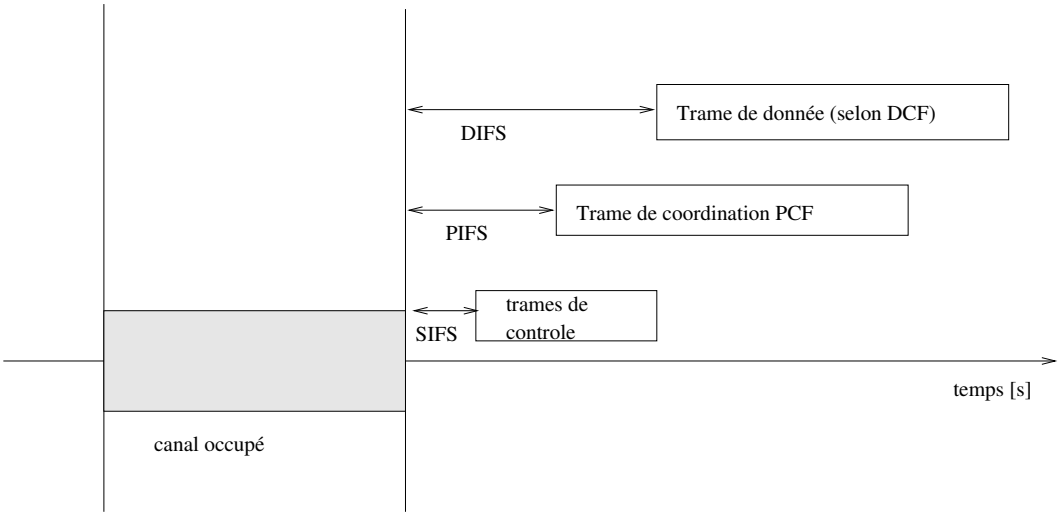


FIG. 14 – Temps d'attente après libération du canal, sans WME (Chronogramme inspiré de l'article[5])

Lorsque l'on active la gestion WME, les temps d'attente sont gérés par priorité, voir une illustration à la figure 15 à la page 56. Les temps d'attente AIFS sont définis de la manière suivante : $AIFS =$

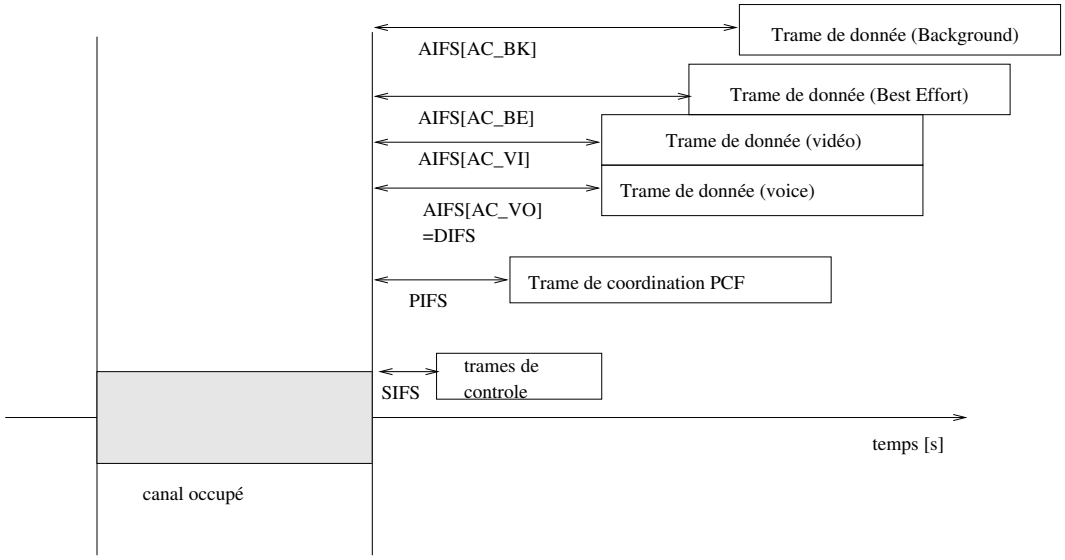


FIG. 15 – Temps d'attente après libération du canal, sans WME (Chronogramme inspiré de l'article[5])

$SIFS + aSlotTime \cdot AIFSN$. Pour citer un ordre de grandeur, dans le mode IEEE802.11a, un slot de temps dure $9\mu s$.

Heureusement, l'analyseur nous offre un moyen de mesurer les temps séparant les trames (le temps delta, définissant le temps écoulé entre la trame courante et la précédente). Si l'on mesure la moyenne des temps "delta" pour chaque trame par type de priorité, dans un premier temps sans activer le WME au sein des cartes, dans un deuxième temps en l'activant et que l'on constate que la moyenne est sensiblement réduite pour les trames prioritaires par rapport aux trames ayant une priorité réduite, il serait possible de démontrer que EDCA est bien supporté par MadWiFi et que ce mode améliore les performances pour les trafic nécessitant de la QoS.

Afin de bien clarifier la précédente explication, voici la démarche à utiliser pour vérifier que les trames AC_VO ou AC_VI sont traités en priorité avec WME par rapport au mode sans WME :

1) Desactiver le mode WME
2) Lancer Ethereal (sur les 3 postes)
3) Lancer une conversation VoIP (nécessitant la priorité AC_VI) entre MH1 et AP
4) Générer du trafic non prioritaire avec MH2
5) A l'aide de Ethereal, filtrer pour ne voir que les paquets RTP, dérouler les champs pour que l'on voit les temps (temps deltas), les champs de priorité au niveau IP
6) Séparer les flux provenant de l'AP des flux provenant du MH (pour voir s'il y a une différence de comportement et vérifier que la QoS est valable dans les deux sens.
7) Filtrer les mesures dépassant déraisonnablement les temps d'attente réglementaires (il y en a une infime partie, ce sont des accidents), Calculer, à l'aide d'un tableur, la moyenne des temps delta. Retenir le résultat (éventuellement générer un graphique.
8) Réitérer toute l'opération (points 1-7)mais en activant le WME sur toutes les cartes
9) Comparer les deux moyennes.

Il est possible qu'il n'y ait pas une grande différence entre les deux moyennes car nous avons choisi de comparer les trafics AC_VO et AC_VI, mais d'après l'article de Stefan Mangold[5], le temps d'attente AC_VO et AC_VI sont équivalent à DIFS. Il faudrait comparer un trafic AC_VO avec WME avec un trafic AC_BE sans WME et AC_BE avec WME. On remarquerait donc, logiquement, si l'on se réfère à la figure 15, qu'il y a une différence de comportement.

6.3 Traitement des données

Malheureusement, Ethereal n'exporte pas les données sous forme de tableau. Malgré sa capacité à exporter les mesures sous forme XML, le grand volume de trame rend Ethereal inutilisable. Il a fallut concevoir deux scripts pour extraire les données, les rendre utilisables pour les importer dans un tableur :

1) Depuis Ethereal, filtrer pour obtenir que les trames nous intéressant
2) Ouvrir les couches suivantes : Niveau trames (pour voir les temps), niveau IP ainsi que le champ DSCP ou 802.1d selon la mesure
3) Aller dans le menu d'impression en "cliquant sur l'imprimante" et choisir le format text
4) Sélectionner "as displayed"
5) Exporter dans un fichier (texte)
6) Exécuter le script adéquoit à la mesure (pckgen si il s'agit de mesure avec une priorité définie par IEEE 802.1d ou kphone si la mesure a été faite avec une communication VoIP)

Ce jeu de script est conçu de la manière suivante :

- Un script bash simple permettant de choisir directement le fichier d'entrée, de sortie, le type de mesure, la priorité et la source (car nous séparons les deux flux)
- Un script perl exécuté par le premier script bash. Il a pour but de convertir notre capture (VoIP) Ethereal sous format texte en fichier au format CSV (champs séparé par des virgules)
- Un second script perl également exécuté par le premier script bash convertissant la seconde mesure (IEEE802.1d) Ethereal sous format texte en fichier au format CSV

Voici un exemple de conversion d'une capture faite lors d'une mesure (type voip avec kphone) :
Avant traitement

```
No.     Time        Source                Destination           Protocol Info
    106 13.209605   65.39.205.114         192.168.20.4          RTP      Payload
type=ITU-T G.711 PCMU, SSRC=167772160, Seq=52, Time=990955718

Frame 106 (232 bytes on wire, 232 bytes captured)
    Arrival Time: Nov 29, 2004 13:41:53.285296000
    Time delta from previous packet: 0.029835000 seconds
    Time since reference or first frame: 13.209605000 seconds
    Frame Number: 106
    Packet Length: 232 bytes
    Capture Length: 232 bytes
IEEE 802.11
Logical-Link Control
Internet Protocol, Src Addr: 65.39.205.114 (65.39.205.114), Dst Addr: 192.168.20.
4 (192.168.20.4)
    Version: 4
    Header length: 20 bytes
    Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)
    Total Length: 200
    Identification: 0x0000 (0)
    Flags: 0x04 (Don't Fragment)
    Fragment offset: 0
    Time to live: 51
    Protocol: UDP (0x11)
    Header checksum: 0x63df (correct)
    Source: 65.39.205.114 (65.39.205.114)
    Destination: 192.168.20.4 (192.168.20.4)
User Datagram Protocol, Src Port: 16240 (16240), Dst Port: 34388 (34388)
Real-Time Transport Protocol

No.     Time        Source                Destination           Protocol Info
    107 13.211764   192.168.20.4          65.39.205.114         RTP      Payload
type=ITU-T G.711 PCMU, SSRC=167772160, Seq=50, Time=951297355
```

Après traitement :

```
106,0.029835000,13.209605000,0x00 ;
```

Et le script permettant de convertir une mesure faite lors de l'utilisation du générateur de trame packETH permet de transformer un fichier comportant des trames tel que :

```
No.     Time        Source                Destination           Protocol Info
   2094 25.179522   192.168.20.1          192.168.20.4          UDP      Source p
ort: 50000  Destination port: 50000

Frame 2094 (114 bytes on wire, 114 bytes captured)
    Arrival Time: Dec  6, 2004 10:36:31.837068000
    Time delta from previous packet: 0.000385000 seconds
    Time since reference or first frame: 25.179522000 seconds
    Frame Number: 2094
    Packet Length: 114 bytes
    Capture Length: 114 bytes
IEEE 802.11
Logical-Link Control
802.1q Virtual LAN
    110. .... .... .... = Priority: 6
    ...0 .... .... .... = CFI: 0
    .... 0000 0000 0001 = ID: 1
    Type: IP (0x0800)
Internet Protocol, Src Addr: 192.168.20.1 (192.168.20.1), Dst Addr: 192.168.20.4
(192.168.20.4)
User Datagram Protocol, Src Port: 50000 (50000), Dst Port: 50000 (50000)
Data (50 bytes)

0000  ab ab ab ab ab ab ab ab ab ab ab ab ab ab ab ab   ................
0010  ab ab ab ab ab ab ab ab ab ab ab ab ab ab ab ab   ................
0020  ab ab ab ab ab ab ab ab ab ab ab ab ab ab ab ab   ................
0030  ab ab                                           ..
```

En données formatées comme suit :

```
2094,0.000385000,25.179522000,6,50000;
```

Les champs 4 et 5 ont été conservés car ils donnent encore l'information de la priorité et du sens du flux (Le nombre 50000 signifie le sens AP à MH1, 60000 signifie MH1 à AP).

Le script est nommé `traitcapteth.sh` et donne l'aide suivante lorsqu'aucun paramètre n'est pas donné :

```
SRC=
usage: traitcapeth.sh <kphone|pckgen> <priority (0-6)> <ap|mh><in> <out>
```

L'affichage "SRC" était simplement là pour confirmer que si l'on choisit comme source "ap", nous aurons SRC=50000 et lorsque l'on choisit "mh" ce serait 60000. Cependant, ce n'est affiché que dans un but de "debug" car le script a été conçu dans la hâte pour obtenir au plus vite les résultats des tests. Une version finale ne devrait pas être combinée de deux scripts perl et un script bash mais uniquement d'un unique script perl.

6.4 Test sans WME

6.4.1 Test avec kphone

Voici les résultats vus de l'analyseur MH1 (carte configurée en mode monitor) :

Priorité Best Effort, DSCP=0x00 temps moyen mesuré : $11.49ms$ (Annexe : Graphique 1)
Priorité Vidéo, DSCP=0xa0 temps moyen mesuré : $10.53ms$ (Annexe : Graphique 2)

Remarque : il y a peu de différence de traitement entre les deux types de priorité. On remarque par ailleurs que les trames Best Effort ont bénéficiées d'un meilleur traitement.

6.4.2 Test avec pckETH

Voici les résultats vus de l'analyseur MH1 (carte configurée en mode monitor) :

Priorité Voice temps moyen mesuré : $3.57ms$ (Annexe : Graphique 3)

Il y a probablement une erreur au niveau de cette dernière valeur car elle est trop proche à celle donnée en mode WME.

6.5 Test avec WME

6.5.1 Test avec kphone

Voici les résultats vu de l'analyseur MH1(carte configurée en mode monitor) :

Priorité Best Effort, DSCP=0x00 temps moyen mesuré : $10.3ms$ (Annexe : Graphique 4)
Priorité Vidéo, DSCP=0xa0 temps moyen mesuré : $10ms$ (Annexe : Graphique 5)

On une légère différence (avantageuse) dans le traitement des trames lorsque WME est activé, cependant, le résultat n'est pas significatif.

6.5.2 Test avec pckETH

Voici les résultats vus de l'analyseur MH1 (AP>MH)
Priorité Voice (AC_VO) temps moyen mesuré : $3.57ms$ (Annexe : Graphique 6)
Priorité Vidéo (AC_VI) temps moyen mesuré : $3.63ms$ (Annexe : Graphique 7)
Priorité Background (AC_BK) temps moyen mesuré : $4.00ms$ (Annexe : Graphique 8)

Ces résultats ci-dessus démontrent qu'il y a effectivement une différenciation de service et ceci dans le bon sens. Les trames dont la priorité d'importance apparaissent plus vite sur le canal que les trames de moindre importance, ce qui correspond à EDCA. Dans le cas d'un trafic du MH à l'AP, la mesure est légèrement plus favorable pour les trames AC_VO : $3.05ms$ (Annexe : Graphique 9)

6.6 Conclusion

Comme dit précédement le comportement change lorsque l'on active le mode WME, et l'on peut observer une gestion des trames selon EDCA et ceci de l'AP au MH ainsi que du MH à l'AP. Cependant, il n'y a pas d'explication concernant la différence de temps d'attente entre les mesures faites avec une conversation VoIP (kphone) et celles faites avec le générateur de paquets car la différence est d'environ $7ms$. Les deux tests ont étés effectués à 4 jours d'intervalle. Il est probable qu'il y a eu plus de perturbations dues à d'autres ESSID sur des canaux proches lors du test car aucun trafic autre que celui généré sur le ESSID "JungleTEST" n'a été observé. Aussi, à priori les paramètres sont identiques car le script d'initialisation et de mise en place des installations (wlanConfig) n'a pas été changé.

Les graphiques de ces mesures sont données en annexe.

7 Modification du driver

7.1 Introduction

Ce chapitre porte sur des modifications du code source faites durant le projet. Elles ont essentiellement un but démonstratif, pour bien comprendre à quel niveau on intervient pour :

- Rajouter des paramètre accessibles via les commandes systèmes (iwpriv)
- Rajouter des fonctionnalités internes au pilote (par exemple dans un but de concevoir un coordinateur)

7.2 Rajout de nouveaux paramètres

Les premiers ajouts nécessaires ne sont autres que des outils permettant à modifier le driver sans dégrader son efficacité et sa stabilité actuelle. Pour ce faire, nous intégrons un système de débuggage ciblé. Il permettra d'analyser le fonctionnement interne, en direct du driver. Il s'agit de créer des zones de débug séléctives. On pourra paramétrer le choix de la zone à l'aide de l'outil de paramétrage de la carte `iwpriv`. Pour que le driver prenne en considération en "temps réel" choix de l'utilisateur, il faut que la valeur de débug définie soit accessible partout dans le driver. Nous allons, dans un premier temps rajouter cette valeur dans la structure `ieee80211com`. Voici la démarche :

1) Ajouter un pointeur[51] sur une variable stockant l'état du paramètre, il s'agit de `DEBUGLEVEL` dans le fichier `net80211/ieee80211_var.h`, dans la structure ieee80211com. Voici un extrait de la définition de ieee80211com, ligne 310-324 :

```
/* RAJOUTE PAR F.Miremad, DEBUG */
/* Cette valeur correspond au niveau de debug. Voici les niveaux:
 * 0  : Pas de debug
 * 1  :
 * 2  :
 * 3  :
 * 4  :
 * 5  :
 * 6  :
 * 7  :
 * 8  :
 * 9  :
 * 10 : Surveillance de la gestion des files d'attente (QUEUES)
*/
int                     *DEBUGLEVEL;
/* Fin du RAJOUT par F.Miremad */
```

   On remarque dans l'extrait ci-dessus, qu'un seul niveau de débug a été défini, le but était surtout de préparer une structure, le temps imparti pour le projet ne permettait pas de rajouter d'autres règles. Le système choisi est simple et ne permet d'activer qu'un niveau à la fois. Si l'on voulait un système plus li bre, il aurait fallu s'inspirer du système qui a été utilisé pour activer/désactiver les flags des paramètres de la carte, en d'autres termes, instaurer un système de flags sur un entier 32 bits avec des masques pour activer/desactiver le flag correspondant au niveau de débug à activer/désactiver. L'idée première était de s'assurer que l'on puisse obtenir des informations provenant du driver lorsqu'il est en fonction, ceci en consultant les logs du kernel linux. Le résultat est positif, car cela prouve bien qu'en ajoutant une commande «printk("message") ;», on écrit bel et bien dans les logs du kernel.

2) Ajouter l'accès au paramètre dans `net80211/ieee80211_var.h` dans la liste énumérée des paramètres. Par exemple, j'ai rajouté :
   - `IEEE80211_PARAM_DEBUG=50`
   - `IEEE80211_PARAM_HCF=51`

3) Modifier les fonctions `ieee80211_ioctl_setparam` et `ieee80211_ioctl_getparam` dans le fichier `net80211/ieee80211_wireless.c`. Voici comment nous avons défini le debug dans la commande setparam, lignes 1513-1517 :

[51] Lors de l'analyse, je me suis rendu compte que les développeurs initialisent systématiquement des pointeurs, c'est pourquoi j'ai créé un pointeur sur DEBUGLEVEL.

```
        /* RAJOUT par F.Miremad */
        case IEEE80211_PARAM_DEBUG:
                ic->DEBUGLEVEL = value;
                printk ("La valeur au mode debug a été affectée!");
                break;
```

4) On rajoute une fonction à la structure `iw_priv_args` dans le fichier `ieee80211_wireless.c` afin que l'on puisse séparément atteindre la fonction qui nous intéresse, cependant, il est possible d'y accéder avec la fonction générique setparam et getparam en spécifiant le numéro correspondant à nos paramètres `IEEE80211_PARAM_HCF` et `IEEE80211_PARAM_DEBUG`.

Remarque sur le point 1 : Après réflexion, j'ai effectivement construit mon nouveau paramètre selon la logique des développeurs du pilote mais je modifie la valeur du pointeur et non la valeur de la variable pointée. Malgré mon erreur, cela fonctionne. Le risque est de ne plus pouvoir déréférencer mon pointeur.

Remarque sur le point 3 et 4 : Le point 3 nous permettra de modifier la valeur de `DEBUGLEVEL` via la commande `iwpriv ath0 setparam 50 <valeur>`. Cependant, à l'aide du point 4, nous avons rajouté une commande qui le fait directement via la commande `iwpriv ath0 set_debug <valeur>`.

Par exemple, en lançant la commande `iwpriv ath0 set_debug 10`, commande que nous avons rajoutée, nous affectons 10 à la variable $*DEBUGLEVEL$ (cette variable a été rajoutée dans la structure ieee80211com). Ceci a pour effet d'activer toutes les fonctions supplémentaires de debug que nous avons rajoutées, qui répondent au niveau de debug 10, exemple dans le fichier `if_ath.c` :

```
ath_tx_tasklet_q0123(TQUEUE_ARG data)
{
        struct net_device *dev = (struct net_device *)data;
        struct ath_softc *sc = dev->priv;
        struct ieee80211com *ic = &sc->sc_ic;

        /*
         * Process each active queue.
         */
        /* RAJOUTE PAR FMiremad */
        if (ic->DEBUGLEVEL==10){
                printk("if_ath: On vide la queue 0\n");}
        /* FIN RAJOUT */
        ath_tx_processq(sc, &sc->sc_txq[0]);
        /* RAJOUTE PAR FMiremad */
        if (ic->DEBUGLEVEL==10){
                printk("if_ath: On vide la queue 1\n");}
        /* FIN RAJOUT */
        ath_tx_processq(sc, &sc->sc_txq[1]);
        /* RAJOUTE PAR FMiremad */
        if (ic->DEBUGLEVEL==10){
                printk("if_ath: On vide la queue 2\n");}
        /* FIN RAJOUT */
        ath_tx_processq(sc, &sc->sc_txq[2]);
        /* RAJOUTE PAR FMiremad */
        if (ic->DEBUGLEVEL==10){
                printk("if_ath: On vide la queue 3\n");}
        /* FIN RAJOUT */
```

On remarque ci-dessus, que l'on pourra observer avec le niveau de debug 10, les moments où les files d'attente sont envoyées dans les buffer de la carte via une fonction de bas-niveau (au sein de `hal.o`).

Il faut toutefois souligner que l'on utilisera cette méthode pour nous permettre d'activer / désactiver le mode HCF[52].

[52] Au fur et à mesure du projet, le mode HCF proprement dit devient impossible à mettre en place pour des raisons de limitation du pilote (code source pas entièrement libre), cependant, il serait possible d'ajouter un contrôleur filtrant les priorités d'accès au mode VOICE et éviter que trop de stations utilisent la plus haute priorité. Cependant, pour activer / désactiver notre contrôleur, cette méthode nous permet de le faire lorsque le driver est chargé.

7.3 Test des paramètres rajoutés

Nous avons donc ajouté une variable définissant le niveau de DEBUG. Voici la démarche à appliquer pour tester notre modification :

1) On décharge les modules gérant la carte de la mémoire. Au préalable tout voici la liste des modules chargés lorsque l'on utilise la carte :

```
blues:/home/boston# lsmod
Module                  Size    Used    by
ath_pci                 52580   0
ath_rate_onoe   6792    1               ath_pci
wlan                            107228  3               ath_pci,ath_rate_onoe
ath_hal                         131152          2               ath_pci
```

   Pour le décharger, il suffit donc de lancer la commande `rmmd ath_pci ath_rate_onoe wlan ath_hal`.

2) Le pilote doit être donc compilé après la modification en tapant la commande `make KERNELPATH=/usr/sr/c/linux clean ; make` dans le réperertoire de madwifi. Ici, `KERNELPATH` n'est autre que le chemin des sources du kernel pour lequel le pilote doit être compilé. Lors de nos essai, il s'agissait du kernel 2.6.9 et le chemin était `/usr/src/linux-2.6.9`.

3) Ensuite, on installe le driver dans le répertoire réservé au modules du kernel linux pour lequel le pilote a été compilé et il se situe dans `/lib/modules/<kernelversion>`. Du simple fait que la version du kernel pour lequel nous avons compilé notre pilote était 2.6.9, le répertoire réservé pour ses modules n'est autre que `/lib/modules/2.6.9`. Il est ensuite conseillé de lancer la commande `/bin/sync` par précaution, elle permet de forcer l'écriture des données qui sont encore dans le cache du kernel et ne sont pas encore effectivement écrites sur le disque[53]. L'installation des modules du pilote se fait simplement en exécutant `make install`[54]. Le pilote sera installé dans `/lib/modules/<kernelversion>/net/`. Lorsque l'on installe un nouveau module, il est nécessaire de lancer la commande `/sbin/depmod -a`. Si le module n'est pas compatible avec le kernel, il est signalé. Cependant, dans notre cas, cette étape a déjà été effectuée lorsque l'on a lancé l'installation (make install)

4) Il s'agit maintenant de vérifier que notre nouvelle version du pilote fonctionne. On lance la commande
   `modprobe ath_pci`[55], on vérifie avec `lsmod` que tous les modules soient bien chargés, soit, dans notre cas :
   `ath_pci, ath_hal, ath_rate_onoe, wlan`.

5) Au stade actuel, notre carte est visible en tant que périphérique réseau «wireless» :

```
blues:/home/boston# iwconfig ath0
ath0      IEEE 802.11  ESSID:""
          Mode:Managed  Frequency:5.32 GHz  Access Point: FF:FF:FF:FF:FF:FF
          Bit Rate:6 Mb/s   Tx-Power:50 dBm   Sensitivity=0/3
          Retry:off   RTS thr:off   Fragment thr:off
          Encryption key:off
          Power Management:off
          Link Quality=38/94  Signal level=-57 dBm  Noise level=-95 dBm
          Rx invalid nwid:0  Rx invalid crypt:0  Rx invalid frag:0
          Tx excessive retries:0  Invalid misc:0   Missed beacon:0
```

   Maintenant, il s'agit de voir si notre rajout de paramètre fonctionne bien. On observe la liste des paramètres privés avec la commande `iwpriv ath0`, la liste des paramètres privés apparaît :

[53] Sur linux, la gestion du filesystem ext2 est faite de façon à écrire les données de manière contigües sur le disque. Cela évite des problèmes de fragmentation des fichiers au sein du filesystem

[54] Par convention, les développeurs écrivent un paramètre appelé `install` dans le `Makefile` où la séquence d'installation est décrite

[55] modprobe a pour effet de charger le module et les module dépendants. Il est possible d'utiliser `insmod` mais il faudra dans ce cas charger séparément et dans l'ordre des dépendance

```
blues:/home/boston# iwpriv ath0
ath0      Available private ioctl :
        setoptie        (8BE8) : set    256 byte        & get   0
        getoptie        (8BE9) : set    0               & get   256 byte
        setkey          (8BE2) : set    60 byte         & get   0
        delkey          (8BE4) : set    7 byte          & get   0
        setmlme         (8BE6) : set    0 byte          & get   0
        addmac          (8BEA) : set    1 addr          & get   0
        delmac          (8BEC) : set    1 addr          & get   0
        chanlist        (8BEE) : set    32 byte         & get   0
        setparam        (8BE0) : set    2 int           & get   0
        getparam        (8BE1) : set    1 int           & get   1 int
...
...
        set_debug       (0032) : set    1 int           & get   0
        get_debug       (0032) : set    0               & get   1 int
        hcf             (0033) : set    1 int           & get   0
        get_hcf         (0033) : set    0               & get   1 int
```

On remarque pour notre debug nos quatre paramètres **set_debug**, **get_debug**, **set_hcf** et **get_hcf**. Pour vérifier la valeur actuelle de la «variable» `DEBULEVEL`, on exécute `iwpriv ath0 get_debug`. A l'initialisation de la carte, la valeur est 0.

On observe en parallèle les logs écris par le kernel dans /var/log/message, /var/log/syslog. La carte wireless est passive tant que l'interface «Ethernet» n'est pas activée. L'activation se fait par le biais de la commande `ifconfig ath0 up` ou en affectant une adresse IP par la commande `ifconfig ath0 <ip>`. Pour "démonter" la carte et la rendre à nouveau passive, il suffit simplement de lancer la commande `ifconfig ath0 down`.

Pour vérifier que les messages que nous avions ajoutés lors du traitement des quatres files d'attentes s'inscrivent dans les logs du kernel, nous affectons la valeur "10" telle que nous l'avions définie dans les clauses de test
(`iwpriv ath0 set_debug 10`. Ensuite, il suffit de réactiver la carte et on observe les logs. Voici un extrait des logs (`/var/log/syslog` ou via la commande `dmesg`) :

```
La valeur au mode debug a été affectée!if_ath: On vide la queue 0
if_ath: On vide la queue 1
if_ath: On vide la queue 2
if_ath: On vide la queue 3
if_ath: On vide la queue 0
if_ath: On vide la queue 1
if_ath: On vide la queue 2
if_ath: On vide la queue 3
```

On remarque qu'au moment où l'on a affecté 10 à notre DEBUGLEVEL, le commentaire d'affectation au mode debug s'est bien inscrit dans les logs. Ensuite, nous pouvons observer les quatre files d'attente se vider successivement.

Remarque : J'ai choisi de créer mes variables supplémentaires dans la structure `ieee80211com` car elle est accessible au niveau du fichier `if_ath.c`, fichier dans lequel on peut intervenir au plus bas niveau avant d'atteindre le "firmware" géré par le hal.o.

7.4 Interception des fonctions de bas niveau

7.4.1 Introduction

Nous avions vu l'interaction entre le noyau linux et notre binaire hal.o à la section 4.8.8 à la page 36. Il est effectivement possible de modifier le code de telle manière à intercepter une fonction du hal. Lors d'une entrevue avec M.Schaefer, nous avions pris au hasard une commande :

- Au sein de `if_athvar.h` : `ath_hal_getrxbuf`
- Fonction redéfinie pour atteindre la fonction dans le fichier `ah.h` : `ah_getRxDP`
- Fonction réellement appliquée dans le ah.h : `*ah_getRxDP`

7.4.2 Méthodologie et interception de la fonction

Il s'agit ici de définir ce qu'il faut modifier et à quel endroit :

1) Au dessus de la fonction d'allocation mémoire (`ath_attach`), on déclare un pointeur pour mémoriser l'adresse originale de la fonction du `hal.o` (ici un exemple avec `get_RxDP` :

```
u_int32_t __ahdecl(*get_RxDP_Original)(struct ath_hal*);
```

L'usage de `__ahdecl` est inspiré des déclarations des pointeurs de fonctions faites dans le fichier `ah.h`

2) Déclarer et définir notre fonction de remplacement (intermédiaire) :

```
u_int32_t __ahdecl get_RxDP_Modifie (struct ath_hal *a)
{
        printk("Fonction RxDP intercepté\n");
        /* Dans notre exemple, on renvoie simplement les données à
         * la fonction orginal dont le pointeur est get_RxDP_Original
        */
        return get_RxDP_Original(a);
}
```

Dans cet exemple, nous avons simplement rajouté un "printk" pour savoir quand la fonction RxDP est utilisée et nous renvoyons le flux de données à la fonction originale via le pointeur que nous avons défini plus haut (ici `get_RxDP_Original`).

3) Après l'allocation mémoire (l'allocation de la structure `ah_hal` s'effectue à la ligne 273 dans le fichier original `if_ath.c` ou sinon, à la ligne 286 dans le fichier modifié) , afin de vérifier que nous avons bien intercepté le pointeur, nous affichons l'adresse de la fonction originale puis ensuite l'adresse de la fonction modifiée :

```
        /* RAJOUTE par F.Miremad */
        printk("Rajoute par F.Miremad: ah_getRxDP: %lx\n", ah->ah_getRxDP);
        get_RxDP_Original = ah->ah_getRxDP; //On sauvegarder le pointeur original
        ah->ah_getRxDP = &get_RxDP_Modifie; //On force l'accès à notre fonction
        printk("Rajoute par F.Miremad: ah_getRxDP2: %lx\n", ah->ah_getRxDP);
        /* FIN de RAJOUT */
```

Ces lignes sont donc rajoutées dans la définition de la fonction `ath_attach`.

Nous pouvons voir le résultat de la modification dans la capture des logs ci-dessous :

```
ath_hal: 0.9.12.14 (AR5210, AR5211, AR5212)
wlan: 0.8.4.4 (EXPERIMENTAL)
ath_rate_onoe: 1.0
ath_pci: 0.9.4.11 (EXPERIMENTAL)
Rajoute par F.Miremad: ah_getRxDP: f1bb3e84
Rajoute par F.Miremad: ah_getRxDP2: f1b92000
ath0: 11a rates: 6Mbps 9Mbps 12Mbps 18Mbps 24Mbps 36Mbps 48Mbps 54Mbps
ath0: 11b rates: 1Mbps 2Mbps 5.5Mbps 11Mbps
ath0: 11g rates: 1Mbps 2Mbps 5.5Mbps 11Mbps 6Mbps 9Mbps 12Mbps 18Mbps 24Mbps 36Mbps 48Mbps 54Mbps
ath0: mac 5.6 phy 4.1 5ghz radio 1.7 2ghz radio 2.3
ath0: 802.11 address: 00:09:5b:69:43:d5
ath0: Use hw queue 0 for WME_AC_BE traffic
ath0: Use hw queue 1 for WME_AC_BK traffic
ath0: Use hw queue 2 for WME_AC_VI traffic
ath0: Use hw queue 3 for WME_AC_VO traffic
ath0: Atheros 5212: mem=0x30800000, irq=11
```

Cela montre que l'adresse a bel et bien été modifiée. Il manque encore le test en cours de fonctionnement. C'est-à-dire lorsque la fonction est réellement utilisée.

8 Conclusion générale

8.1 Introduction

MadWiFi n'est malheureusement pas intégralement libre. Ceci a pour effet de ne pas permettre de gérer le chipset directement, au plus bas niveau . Le risque est d'obtenir des stations qui émettent malencontreusement durant les CFP. Il nous faut donc, en attendant d'avoir un accès plus bas niveau, contrer ces manques. Cette section a pour but d'émettre des idées afin de contourner les limitations dües au chipset et à la source verrouillée du pilote. Aussi, ces idées peuvent permettre de poursuivre le projet basé sur MadWiFi.

8.2 Nouvelles stations sur le ESSID

Afin d'empêcher dans tous les cas une station d'émettre, malgré que l'on ait pas accès aux basses couches, on peut néanmoins commuter la carte en mode Monitor. Dans ce mode, la carte peut tout recevoir mais a l'interdiction d'émettre. Cela peut nous permettre de synchroniser la station au ESSID avant de tenter de s'associer. On pourrait la forcer à mémoriser le contenu des beacons, car ceux-ci, en plus d'offrir une horloge locale, délivrent tous les paramètres de contrôle (EDCA). Il est donc possible par ce biais, de garantir une CFP protégée contre les tentatives d'association.

Cependant, ce n'est probablement pas une méthode fiable car le fait de changer de mode de fonctionnement, au stade actuel, a montré un instabilité nécessitant le déchargement puis chargement du pilote (ceci provoque une réinitialisation complète de la carte).

8.3 Filtrage des flux prioritaires

Si l'on ne peut pas agir directement dans le fonctionnement interne du binaire du fait que la source est indisponible, il est possible de faire un filtrage supplémentaire. C'est-à-dire limiter le nombre de flux ayant une haute priorité[56]. Pour ce faire il faut agir au sein de l'AP. L'AP pourrait refuser certains flux, ou, simplement les considérer comme du trafic "Best Effort" de moins grande priorité. D'après une idée de M.Schaefer, on pourrait concevoir un coordinateur qui stocke tous les flux prioritaires mais qui retarde certaines transactions pour garantir la priorité aux premières stations ayant demandé un flux de haute priorité (ceci est possible lorsqu'une station mobile communique avec une autre station mobile car tout trafic transite par l'AP). Cependant, il est nécessaire de court-circuiter la fonction d'AP défini dans le pilote. Lors du projet, je n'ai malheureusement pas identifié la section gérant la carte en tant qu'AP, il est fort probable est même logique que cela soit défini dans le hal.o.

En effet, l'AP est un élément qui fait office de "switch" wireless, il se doit d'être très efficace afin de ne pas perdre de trame reçue et d'émettre (réémettre) dans les temps. Si, pour chaque trame l'on doit :

1) Faire remonter le flux des buffer de la carte wireless au sein du kernel linux
2) Traiter le flux en mémoire (donc allouer de la mémoire dans le PC)
3) Renvoyer le flux au sein de la carte (dans les buffer de la carte)

alors les cartes atheros pourvus du pilote MadWiFi fonctionneraient selon la figure 16 à la page 68.

Pour commenter la figure 16, la carte serait autonome pour la signalisation et la gestion en tant qu'AP et limiterait les transferts avec le kernel. Il serait donc logique que le binaire contienne lui-même un partie de code faisant office de firmware[57]. Aussi, lors de l'analyse, il a été découvert que la carte possède réellement quatre buffers hardware pour stocker les flux des quatre catégories d'accès ce qui confirmerait donc l'hypothèse de la gestion interne à la carte nécessaire pour éviter d'être perturbé par les latences d'un kernel linux trop chargé par des interruptions système externes. Ainsi la carte bénéficie d'une autonomie

[56] Suggestion de M.Robert lors d'un entretien
[57] Cette réflexion est basée sur une discussion que j'ai eu avec M.Schaefer

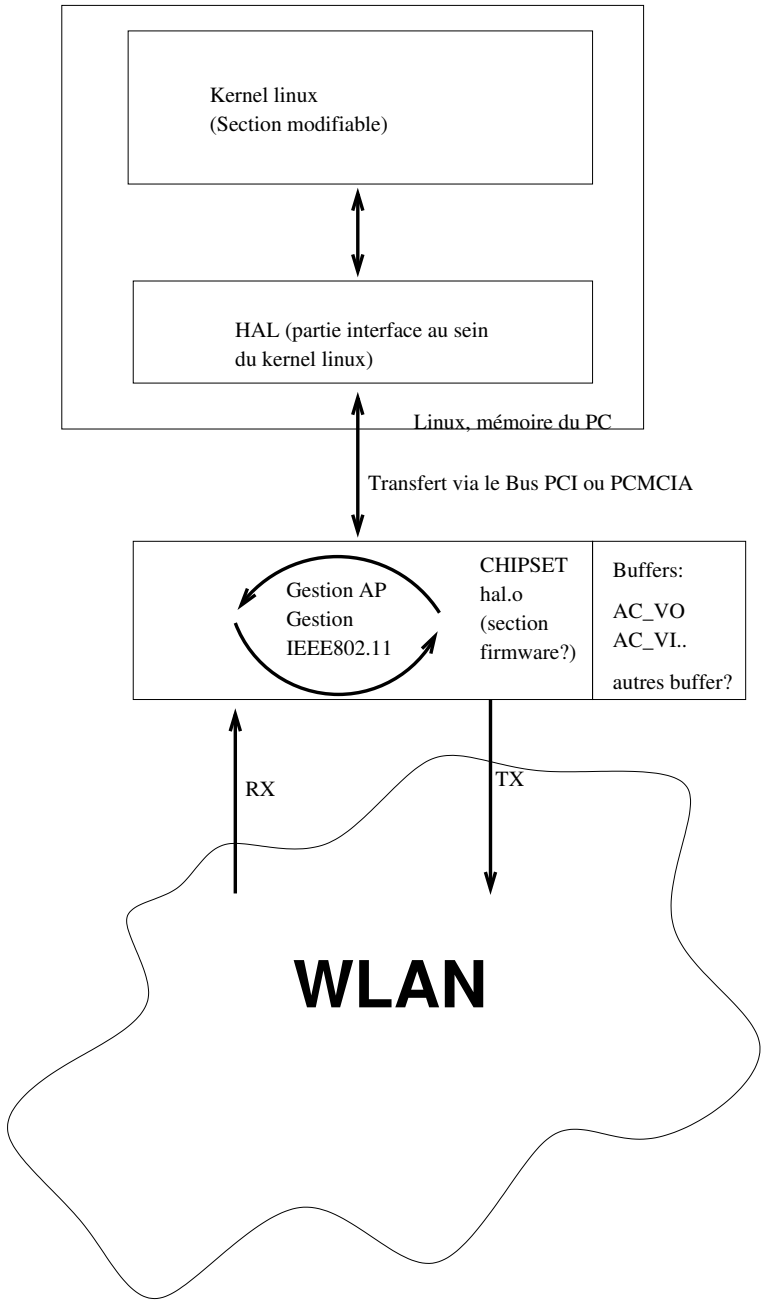


FIG. 16 – Gestion de bas niveau faite par le `hal.o`

complète et peut, en cas de latence du kernel linux, gérer le trafic (en tant qu'AP), stocker dans un buffer les données lui parvenant et envoyer ce qu'il n'a pu encore émettre.

Si l'on se base sur la procédure préalablement décrite, l'AP aurait très probablement déjà consommé trop de temps pour émettre de manière synchronisé. Aussi, si l'on désire par la suite gérer HCF, on doit concevoir des TXOPs définis dans les beacons en fonction des demandes. Le va-et-vient des données entre la carte et le kernel linux est coûteux et le risque que linux ne réponde pas à temps pour des raisons de latence due à divers appels systèmes indépendants de notre carte wireless risque fortement de compromettre

la fiabilité de la carte. Elle risque donc de perdre des données ou de ne pas répondre assez vite.

8.4 Limite des chipsets en général

Malgré les limitations que nous avons rencontrées durant l'étude du driver, il est possible que l'implémentation rigoureuse d'HCF ne soit pas encore possible si l'on se réfère à l'article de Stefan Mangold[5], section 2.5.3. En effet, on ne pourrait pas garantir des temps CP et CFP rigoureusement synchronisés du fait que si une station a l'opportunité d'émettre près de la fin d'une période (CP ou CFP), elle risque de dépasser le temps imparti car on ne peut prévoir précisément, selon le mode de modulation choisi, la fin de la transmission et, par conséquent, retarder l'émission du beacon signalant la fin d'une période. Cependant, malgré ce risque, l'implémentation de HCF pourrait statistiquement améliorer les performances bien qu'il n'y ait pas une application rigoureuse de la norme. Il faut souligner que le milieu du "wireless" n'est de loin pas un milieu propre et que l'on est malgré tout confronté à des perturbations externes qui peuvent également couvrir l'émission d'un beacon. Ceci démontre qu'il ne sera "jamais" possible d'appliquer parfaitemet les règles défini par la gestion des CP et CFP

8.5 Appréciation de EDCA (sans HCCA)

D'après les tests, les trames prioritaire ont bel et bien été traitées en priorité par rapport au trafic de fond. Dans le cadre de notre banc de test il aurait fallu une dizaine de stations indépendantes pour vraiment voir que cette gestion priorité est efficace. Dans un premier temps on peut affirmer que EDCA fonctionne. Il serait intéressant de se demander si c'est un véritable gain de gérer la QoS au sein du réseau WLAN. Sachant que WLAN signifie "Wireless Local Area Network" nous aurons d'une manière générale un nombre limité de stations au sein du même ESSID, mais si l'on considère l'évolution de la communication actuelle, on remarque que des téléphones mobiles disposent également d'une gestion de IEEE802.11x, ceci dans le but de commuter entre la téléphonie IP et le réseau GSM. Aussi, avec l'UMTS certains téléphones mobiles disposent d'un système permettant de voir des émissions TV, de faire de la vidéo phonie. Si l'on considère la venue de la vidéo phonie on peut s'imaginer que même dans un environnement restreint comme celui des ESSID, il peut y avoir le trafic des ordinateurs (téléchargement de fichier, mail, etc), et le trafic des téléphones nécessitant de la QoS pour effectuer des transferts vidéos, vocaux. Dans ce cas, l'optimisation à tous les niveaux est utile.

8.6 Remarque concernant la documentation

La documentation fournie pour comprendre la norme est complète. Une série d'articles ainsi que le draft no8 de IEEE802.11e ont été la base d'étude pour bien comprendre le fonctionnement des mécanismes de coordination d'une manière purement théorique. De plus, le format des trames, les algorithmes de coordinations, les temps et délais sont clairement spécifiés dans la norme.

La documentation liée à MadWiFi est essentiellement destinée aux utilisateurs, mais il n'y a aucune explication concernant la structure du code source, ce qui a passablement ralenti l'analyse. Aussi, pour bien comprendre l'interaction entre le kernel linux, la gestion des flux, la définition d'une carte wireless générique ainsi que d'une carte Ethernet, l'usage du livre sur les périphériques [4] est indispensable. De plus l'aide de M. Marc Schaefer et son expérience en qualité de développeur de pilote sous linux a été précieuse pour comprendre comment lire le code source ainsi que les subtilités qui y sont inscrites.

Par contre, la documentation permettant l'usage spécialisé de MadWiFi (activation du WME, etc.) est inexistante. Pour ce faire, une lecture du code source, des connaissances personnelles en matière d'utilisateur de linux ont été indispensables pour comprendre comment définir les bonnes procédures à suivre afin de monter le réseau de test. Par exemple, le fait de devoir forcer la carte à rester en mode 802.11b n'est expliqué sur aucun forum ni aucune documentation.

8.7 Outils manquants et diverses remarques

Analyseur Lors de l'élaboration du banc de test pour vérifier le fonctionnement de WME déjà implémenté au sein de MadWiFi, la nécessité d'un analyseur capable de lire le trame de QoS, les paramètres EDCA (compris dans les beacon), était indispensable. Malheureusement, la norme IEEE802.11e n'est pas encore établie et, d'après des commentaires des auteurs découverts dans le forum d'Ethereal (`http://www.ethereal.com/`) le support d'IEEE802.11e ne se fera pas tant que la norme n'est pas établie et qu'il n'existe pas de système l'utilisant. La réaction des développeurs est logique, mais, dans notre cas ce serait très utile. Il est donc possible de développer un plugins pour Ethereal, mais le temps manquait durant le projet et l'écriture d'un plugins est un projet à part entière.

Moteur de recherche au sein de l'école La recherche d'outils permettant de générer du trafic (le générateur de paquet packETH) a pris du temps. Il serait pratique, qu'au sein de l'école, il y ait un site local faisant office de boîte à outils, proposant des outils d'analyse, des outils permettant de générer du trafic à la demande, etc. Si ces outils avaient été directement mis à disposition et leur recherche ne nécessitait pas de temps, le gain de temps serait évident.

Conseil concernant la modification de code Le projet actuel nécessitait la modification d'un code source existant. Le problème principal est que pour assurer la pérénité des modifications faites sur le code source, il faut concevoir un système de patch permettant d'appliquer les modification sur les nouvelles versions du code source. Aussi, l'idéal serait de se baser sur des "realease" du code et non l'évolution continuelle que l'on observe à chaque mise-à-jour cvs (celle-ci découvre un changement par jour). Aussi, il serait nettement plus intelligent de produire des fichiers à inclure (.h) ou des fichiers source (.c) externe. Actuellement, les modifications faites sont directement situées dans le code fourni ce qui implique qu'une modification structurelle empêcherait notre patch de fonctionner. Par manque de temps, je n'ai pas fait de système de patch et j'ai travaillé sur une version statique.

Conseil concernant l'interception des fonctions L'idée finale a été d'intercepter les fonctions du hal.o en présumant qu'il utilise notre table de saut. Cependant c'est un système qui n'est pas viable si l'on considère l'évolution rapide des chipsets. Pour intercepter les fonctions nous aurions dû analyser la manière avec laquelle elles sont utilisées en interne[58], faire une analyse approfondi des données transitant d'une fonction à l'autre. Avec cette méthode il nous serait impossible de suivre l'évolution car pour chaque nouveau chipset, un jeu de fonctions pourrait changer, apparaître ou disparaître. Nous n'avons pas les spécifications du chipset. Il est indispensable d'obtenir ces spécifications afin de concevoir nous-mêmes le firmware et de gérer entièrement les algorithmes régissant le trafic (selon IEEE802.11). Pour conclure, l'usage de MadWiFi n'est malheureusement pas approprié tant qu'il n'existe pas de version libre du hal.o et que l'on ne dispose pas des spécifications du chipset.

8.8 Conclusion finale et personnelle

L'issu du projet est difficile à définir clairement. Il serait raisonnable de choisir un système entièrement ouvert et permettant de contrôler le système sur tout les plans. Cependant, sachant qu'actuellement il est difficile d'obtenir un chipset conçu selon les dernière technologies tout en ayant sa spécification, si l'on désire tester les principes de HCF dans des condition réelles, ou, concevoir un autre type de coordinateur utilisant EDCA, nous sommes obligé de travailler avec les contraintes rencontrées.

Le projet a permis de mettre en lumière une partie des mécanismes utilisés lorsque l'on conçoit un pilote réseau sous linux, et ce, avec un kernel récent.

Il a également permis de soulever les manques actuels en matière d'outils d'analyse (conception d'un plugins pour Ethreal)

[58] En analysant le contenu de la mémoire, en analysant les données transitant en interne dans le kernel linux car le module `ath_hal.o` (binaire hal.o) tourne en partie sous linux

Des tests de EDCA en conditions réelles ont pu être menés, il serait pratique de les refaire avec des outils d'analyse capable de repérer les informations de QoS.

Finalement à titre personnel, ce projet m'a permis de faire un cours accéléré du langage C, de comprendre une partie de la logique régissant les périphériques réseaux au noyau linux. Il n'était pas aisé de prendre en main les 30'000 lignes de code composant le pilote, sans compter les inclusions de sources du noyau linux. Cette étude m'a donc appris à disséquer un grand volume d'information sans me perdre dans les détails.

J'ai également pu approfondir mes connaissances en matière de réseau wireless, de paramétrage de carte. J'ai également pu concevoir des scripts, en perl, en bash. Aussi, d'un point de vue purement théorique, l'étude de normes telles qu'elles sont produites au sein de l'IEEE est très instructif et nécessite beaucoup de rigueur.

FRÉDÉRIC MIREMAD

9 Remerciements

Je remercie les personnes qui m'ont soutenu pour ce projet :
– Dr. Stephan Robert : Professeur au sein l'Ecole d'Ingénieurs du Canton de Vaud (HES-SO)
– Dr. Daniel Rodellar : Ingénieur auprès de Swisscom Innovation
– Marc Schaefer : Ingénieur informatique auprès de CRIL (www.cril.ch)

10 Annexes

Les annexes papier attaché au projet sont les suivantes :
- Un extrait du code source modifié (la version originale est fournie sour format électronique)
- Les scripts de conversion des logs ethereal-csv (traitcapteth.sh, traitement3b.pl et traitement3c.pl)
- Le script de mise en route du banc de test et le fichier de configuration (wlanConfig, wlanConfigrc)
- Les graphiques des tests numéroté de 1 à 9
- Le journal de travail
- Après la page de garde, le cahier des charges

11 Glossaire

11.1 Abrévations

TAB. 4 – Abréviations

Acronyme	Signification	Commentaire
AC	Access Category	Catégorie d'accès pour la gestion de la QoS
ACK	Acknowledgment	
AIFS	Arbitration Interframe Space	Utilisé dans la gestion QoS (EDCA)
AIFSN	Arbitration Interframe Space Number	
AMRR	Adaptive Multi Rate Retry	
AR	Access Router	Routeur permettant l'accès à un réseau étranger
CSMA/CA	Carrier Sense Multiple Access/Collision Avoidance	système de prévention des collisions par insertion de temps d'attente incompressibles
CSMA	Carrier Sense Multiple Access	
CN	Correspondant Node	Noeud correspondant (avec notre MN)
CA	Collision Avoidance	
CW	Contention Window	
DCF	Distributed Coordination Function (MAC)	
DIFS	Distributed Coordination Function (DCF) Interframe Space	
DNS	Domain Name System	Service d'anuaire faisant le lien entre l'adresse IP et le nom de domaine
DSCP	Differentiated Services Code Point	Champ définissant la priorité au niveau de la pile IP
DSSS	Direct Sequence Spread Spectrum	
EDCA	Enhanced Distributed Channel Access	Mécanisme de différenciation de service (QoS)
EIFS	Extended Interframe Space	
ESSID	Extended Service Set Identifier	Nom du réseau WLAN au sein de l'AP
HC	Hybrid Coordinator	Le coordinateur d'HCF au sein de l'AP
HCCA	HCF Controlled Channel Access	Méthode de coordination durant les CFP (également CFP)
HCF	Hybrid Coordination Function	Mécanisme de gestion de QoS utilisant à la fois EDCA et HCCA
MH	Mobile Host	Station mobile
MN	Mobile Node	Noeud Mobile (par exemple un PC portable)
RTS/CTS	Request to Send / Clear to Send	Système de réservation du canal pour éviter les collisions
ISM	Industrial Scientific and Medical	On cite souvent les fréquences libérées ISM (2.4GHz)
MadWiFi	Multiband Atheros Driver for WiFi	Pilote des chipset WiFi Atheros

TAB. 5 – Suite des abréviations

MSDU	MAC Service Data Unit	
MPDU	MAC Protocol Data Unit	
NAV	Network Allocation Vector	Défini le temps de réservation du canal
OFDM	Orthogonal Frequency Division Multiplexing	
OSI	Open Systems Interconnection	Modèle de réseau
OUI Type	Organizationally Unique Identifier	
PCF	Point Coordination Function	Mécanisme de gestion de QoS
PIFS	PCF Interframe space	
PC	Point Coordinator	Entité de gestion de la QoS selon PCF
QBSS	QoS supporting BSS	Réseau WLAN doté d'une gestion de la QoS
QoS	Quality Of Service	
SIFS	Short Interframe Space	La plus petite période séparant deux trames
TBTT	Target Beacon Transmission Time[5]	
WiFi	Wireless Fidelity	Label définissant les cartes WLAN compatibles selon la norme IEEE802.11
WLAN	Wireless Local Area Network	Réseau local sans fil (généralement sous la norme IEEE802.11)

Références

[1] Ethereal, a network protocol analyser. Logiciel disponible à l'url `http://www.ethereal.com`, license GPL.

[2] packeth, ethernet packet generator. Générateur de packet sous license GPL, disponible à l'adresse : `http://packeth.sourceforge.net/`.

[3] IEEE 802.11e team. Draft Amendment to STANDARD [for] Information Technology - Telecommunications and Information Exchange Between Systems - LAN/MAN Specific Requirement. Part 11 - Wireless Medium Access Control (MAC) and Physical Layer (PHY) specifications : Medium Access Control (MAC) Quality of Service (QoS) Enhancements, Février 2004. IEEE P802.11e/D8.9.

[4] Alessandro Rubini and Jonathan Corbet. *Linux, pilotes de périphériques*. O'Reilly, 2002.

[5] Guido R.Hiertz Stephan Mangold, Sunghyun Choi and Bernard Walke. Analysis of IEEE 802.11e for QoS support in wireless LANs. Sous forme PDF.

[6] Lassaâd Gannoune Stephane Robert. Intermediate report for HCF-lite (concerned work packages wp1 and wp2, Juin 2004. Rapport interne EIVD/TCOM.