

Estimation de la qualité des réseaux Wi-Fi

Mémoire présenté par
Sébastien DUVAL , Jérémy VANHEE

en vue de l'obtention du grade de Master
Sciences informatiques & Ingénieur civil en informatique

Promoteur(s)
Olivier BONAVENTURE

Lecteur(s)
Charles PECHEUR, Quentin DE CONINCK

Année académique 2015-2016

Résumé

Les réseaux Wi-fi font partie intégrante du quotidien de ses utilisateurs. Des entreprises comme *Proximus* proposent des hotspots afin de permettre à ses utilisateurs de profiter de leurs services où qu'ils aillent.

Ce travail s'oriente sur l'estimation du meilleur réseau Wi-Fi de la manière la moins intrusive possible. Pour ce faire, deux approches ont été réalisées : une analyse des éléments présents sur la couche Liaison de données afin de déterminer au moyen de filtres quel peut être le meilleur réseau Wi-Fi disponible dans une certaine zone géographique. La deuxième approche consiste à estimer les informations qui peuvent passer au travers des hotspots de *Proximus* sans être authentifiés pour venir accentuer les résultats trouvés lors de la première approche. Finalement, nous proposons également une solution supplémentaire due aux résultats rencontrés durant les deux premières approches.

Remerciements

Tout d'abord, nous souhaitons remercier notre promoteur, le Professeur Olivier Bonaventure, pour sa disponibilité et ses conseils qui nous ont permis de nous faire avancer le plus loin possible dans ce travail.

Nous souhaitons aussi remercier Quentin De Coninck pour son support tout au long de l'année.

Nous remercions également Olivier Tilmans pour son aide avec l'outil *Tracebox*.

Nous remercions toute l'équipe des Sysadmin qui nous ont aidé à mettre en place les différents serveurs que nous avons utilisés.

Nous souhaitons remercier nos familles et amis pour leur support et leurs conseils et plus particulièrement Philippe Mathues pour sa relecture attentive.

Table des matières

Résumé	i
Remerciements	iii
1 Introduction	1
1.1 Travaux apparentés	2
2 Concepts théoriques	5
2.1 Infrastructure du 802.11	5
2.1.1 Composant de l'architecture	5
2.1.2 Modes d'opération	7
2.2 Les trames	7
2.2.1 Management	8
2.2.2 Data	9
2.2.3 Contrôle	9
2.3 Vue d'ensemble du IEEE 802.11	10
2.3.1 Standards	10
2.3.2 Canaux et fréquences utilisables	11
2.3.2.1 2.4 GHz	12
2.3.2.2 5 GHz	12
2.3.3 Couche PHY	13
2.3.4 Contrôle d'accès	14
2.3.5 Méthodes d'accès au média	16
2.3.5.1 Inter Frame Spaces	16
2.3.5.2 Accès normal	17
2.3.5.3 Accès via RTS/CTS	18
3 Méthodologie	21
3.1 Comment déterminer le meilleur Wi-Fi ?	21
3.1.1 Définition du problème	22
3.1.2 Idée générale du fonctionnement	22
3.2 Choix du matériel	23

3.2.1	Raspberry Pi 2	23
3.2.2	Antenne Wi-Fi	23
3.2.3	Serveurs	24
3.2.3.1	UCL	24
3.2.3.2	OVH	24
3.2.4	Routeurs	24
3.3	Hotspots	24
3.3.1	Hotspot ouvert	25
3.3.2	Hotspot fermé	25
3.4	Observations	25
3.4.1	Couche 2	26
3.4.1.1	Structure générale d'une trame	26
3.4.1.2	Informations importantes des trames	27
3.4.2	Couche 3	36
3.4.2.1	Caractéristiques des hotspots ouverts	36
4	Implémentation	39
4.1	Vue générale de l'algorithme	39
4.1.1	Partie 1 - Scan	40
4.1.2	Partie 2 - Analyse des trames	41
4.1.3	Partie 3 - Filtres	42
4.2	Description de l'implémentation des différents filtres	42
4.2.1	Couche 2	43
4.2.1.1	Puissance du signal reçu	43
4.2.1.2	Pourcentage de <i>Beacon</i> capturées	44
4.2.1.3	Perte des trames <i>Beacon</i>	45
4.2.1.4	Consommation de l' <i>airtime</i>	49
4.2.1.5	Délai des trames <i>Beacon</i>	54
4.2.1.6	Estimation de la bande passante	55
4.2.1.7	Estimation de l'influence des interférences des canaux adjacents	57
4.2.1.8	Retransmission des trames de données	59
4.2.1.9	Nombre de stations connectées	59
4.2.1.10	RTS/CTS	59
4.2.2	Couche 3	61
4.2.2.1	Protocole DHCP	62
4.2.2.2	Évaluation de la bande passante	62
4.2.2.3	Protocole DNS	65
4.3	Définition de l'influence de chaque filtre	66
5	Validation et résultats	69
5.1	Environnements de tests	69
5.2	Validation	70

5.2.1	Trafic généré	71
5.2.1.1	Déterminer les limites	71
5.2.1.2	Génération du trafic via DITG	72
5.2.2	Couche 2	74
5.2.2.1	Consommation de l' <i>airtime</i> : trames de données . . .	75
5.2.2.2	Délai des trames <i>Beacon</i>	76
5.2.2.3	Estimation de la bande passante	77
5.2.2.4	Pertes des trames <i>Beacon</i>	78
5.2.2.5	Retransmissions des trames de données	79
5.2.2.6	Consommation de l' <i>airtime</i> : trames <i>Beacon</i>	80
5.2.2.7	Estimation de l'influence des interférences des canaux adjacents	83
5.2.3	Récapitulatif de l'impact des filtres sur la couche 2	84
5.2.4	Couche 3	85
5.2.4.1	Validation avec OVH	86
5.3	Conclusion	90
6	Solution alternative	93
6.1	Informations récupérées	93
6.2	Implémentation	94
6.2.1	Récolte des données	94
6.2.2	Développement sur OpenWrt	96
6.3	Améliorations possibles	97
7	Conclusion	99
7.1	Réalisations futures	100
A	Outils utilisés pour les mesures	103
A.1	iw	103
A.2	TCPdump	103
A.3	Wireshark	103
A.4	DITG	104
A.5	DNSdiag	104
A.6	Traceroute	104
A.7	Tracebox	104
A.8	iPerf	104
A.9	DITG	105
B	Antennes compatibles mode <i>monitor</i>	107
C	Valeurs des données utilisées	109
D	Pseudo code pour le calcul des pertes	111

E Résultats utilisés pour la validation	113
F Problèmes rencontrés avec les smartphones	117
G Compiler et installation pour OpenWrt	119
G.1 Compiler pour OpenWrt	119
G.2 Installation des packages	120
G.2.1 speedtest	120
G.2.2 scanning et evaluate	120
Table des figures	121
Liste des tableaux	123
Bibliographie	127

List of Abbreviations

AIFS	Arbitration Inter Frame Space
AP	Access Point
ARP	Address Resolution Protocol
BSS	Basic Service Set
BSSID	Basic Service Set Identifier
CCA	Clear Channel Assessment
CCK	Complementary Code Keying
CSMA/CA	Carrier Sense Multiple Access with Collision Avoidance
CTS	Clear To Send
DCF	Distributed Coordination Function
DHCP	Dynamic Host Configuration Protocol
DIFS	DCF Inter Frame Space
DNS	Domain Name System
DSSS	Direct Sequence Spread Spectrum
EDCA	Enhanced Distributed Channel Access
EIFS	Extended Inter Frame Space
ESS	Extended Service Set
FCS	Frame Check Sequence
HCCA	HCF Controlled Channel Access
HCF	Hybrid Coordination Function
HR/DSSS	High Rate Direct Sequence Spread Spectrum
MBSS	Mesh Basic Service Set
MCF	Mesh Coordination Function
NAV	Network Allocation Vector
OFDM	Orthogonal Frequency-Division Multiplexing
PCF	Point Coordination Function
PIFS	PCF Inter Frame Space
PLCP	Protocol Layer Convergence Procedure
PMD	Physical Medium Dependent
QBSS	QoS enhanced Basic Service Set
QoS	Quality of Service

RIFS	Reduced Inter Frame Space
RSSI	Received Signal Strength Indication
RTS	Request To Send
RTT	Round Trip Time
SIFS	Short Inter Frame Space
SSID	Service Set Identifier
TBTT	Target Beacon Transmission Time
TU	Time Unit
VAP	Virtual Access Point
WLAN	Wireless Local Area Network
WMM	Wi-Fi Multi-Media

Chapitre 1

Introduction

Ces dernières années, nous avons pu remarquer que le nombre de hotspots disponibles a fortement augmenté. Il a augmenté grâce aux opérateurs comme *Proximus* et *Voo* qui permettent aux routeurs installés chez leurs particuliers de diffuser un hotspot sur lequel n'importe qui peut se connecter sans altérer la vitesse de connexion du réseau privé. *Proximus* et *Fon* ont trouvé une collaboration pour offrir un accès gratuit à internet via la plus grande communauté Wi-Fi au monde. Cela représente plus de 1 million Wi-Fi hotspots en Belgique et plus de 19 millions de Wi-Fi hotspots dans les pays partenaires de *Fon* [50]. De plus, nous avons aussi pu remarquer que l'utilisation de smartphones et autres appareils mobiles connectés est très présente et est en forte augmentation [8].

Ce grand nombre de hotspots nous laisse penser que nous serons donc confrontés assez rapidement, dans les grandes villes par exemple, à plusieurs points d'accès proposant le même hotspot d'un même opérateur. Nous savons aussi que les performances d'une station connectée en Wi-Fi dépendra fortement de la sélection du point d'accès. L'approche conventionnelle pour la sélection d'un point d'accès est basée sur la puissance de son signal [32]. Cette approche peut amener à préférer un point d'accès qui ne donnera pas forcément les meilleures performances en termes de débit. Illustrons ce fait par un exemple afin de mieux cerner le problème : supposons qu'on dispose de plusieurs points d'accès dans une même zone géographique mais avec des puissances de signal reçues différentes à cause de l'environnement dans lequel ils sont positionnés. Si on prend la méthode conventionnelle, celui avec la meilleure puissance de signal reçue sera sélectionné par un grand nombre d'appareil, altérant obligatoirement ses performances délivrées. Les autres points d'accès se trouvent donc délaissés alors qu'ils sont susceptibles de proposer de meilleures performances que leur homologue.

Dans ce travail de fin d'étude, nous avons recherché un maximum d'informations

qui pourraient nous être utiles pour estimer la qualité d'un réseau Wi-Fi dans le contexte d'une utilisation d'un appareil mobile. Cette estimation a été voulue comme la moins intrusive possible et la plus rapide possible. Dans la section suivante, nous discuterons des travaux existants dans ce domaine. Le chapitre 2 aura pour but de donner des informations sur les concepts théoriques sur les réseaux sans fil, nécessaires à la compréhension de ce travail. Dans le chapitre 3, on s'intéressera plus en profondeur à la définition du problème de ce mémoire, la méthodologie appliquée afin d'estimer le meilleur Wi-Fi, le matériel utilisé et les observations réalisées sur les réseaux sans fil. Le chapitre 4, quant à lui, parlera de l'implémentation de notre algorithme, comment elle a été réalisée et sur quelles bases elle se fonde. Le chapitre 5 se concentrera exclusivement sur la validation de notre implémentation ainsi que sur les problèmes rencontrés durant cette dernière. Dans le chapitre 6, un autre point de vue sera développé afin d'estimer le meilleur Wi-Fi. Finalement, le chapitre 7 conclura ce travail sur les leçons apprises et abordera les réalisations futures.

1.1 Travaux apparentés

L'article, "How's My Network ? A Java Approach to Home Network Measurement" [54], se concentre sur l'exécution d'un applet Java de mesure de réseau domestique. Cet applet a été développé pour comprendre les capacités d'un tel outil pour mesurer les caractéristiques d'un utilisateur du réseau. Ce travail contribue à l'évaluation d'un réseau domestique.

L'applet est composé de plusieurs modules qui vont obtenir différents types d'information sur la machine et le réseau sur lesquels l'utilisateur va faire les tests.

Le premier module va récupérer les informations de configuration de la machine faisant le test. Mais aussi, des informations sur le système local comme la version de Java, le type de CPU, etc.

Un module est créé pour faire des mesures de débit montant et descendant grâce à une connexion TCP entre le client et leur serveur. Ce module va être utilisé pour pouvoir faire des comparaisons avec les outils déjà existants.

Pour tester les performances DNS, les auteurs ont créé un outil en Java, JDIG. Il va faire une série de tests pour connaître la moyenne du RTT DNS pour un ensemble de serveurs aléatoires.

Le dernier module détermine le nombre et les types d'appareils connectés au réseau local de la machine réalisant les tests. Cela est fait grâce aux requêtes ICMP et ARP.

La principale différence avec notre travail est que les informations récupérées sont obtenues de manière intrusive. L'utilisateur décide de faire le test et doit avoir un accès direct au réseau. Notre but est de récupérer un maximum d'information sur la qualité du réseau de manière non-intrusive.

L'article, "Improved access point selection" [43], présente *Virgil*, un programme qui va trouver et sélectionner automatiquement un point d'accès. *Virgil* va scanner pour avoir tous les AP qui l'entoure, se connecter sur chaque AP et lancer une batterie de tests pour estimer la qualité de chaque connexion à internet. Selon leurs tests, leur application trouverait un meilleur AP de 22% à 100% plus souvent que la méthode de sélection selon la puissance du signal. Il va récupérer premièrement différentes informations sur l'AP et ensuite essayer d'avoir une adresse DHCP. S'il reçoit une adresse DHCP, il va faire différents tests pour pouvoir comparer les résultats avec les autres AP et sélectionner le meilleur. La qualité de la connexion sera évaluée sur la bande passante disponible, les ports qui sont bloqués ou redirigés et le RTT du client vers des serveurs distants.

La différence avec notre travail est, une fois de plus, que les données sur la qualité de la connexion sont récupérées de manière intrusive et qu'il faut faire une connexion avec tous les AP qui nous entoure pour faire les différents tests mis en place.

L'article, "Measurement-based characterization of 802.11 in a hotspot setting" [55], analyse des mesures sur le Wi-Fi réalisées durant la SIGCOMM 2004 pour comprendre comment le 802.11 opère dans des environnements réels. Leurs analyses ont montré que 40% du temps total de transmission est utilisé pour envoyer de nouveaux paquets de données. Le reste du temps est utilisé pour les retransmissions (35%), par les ACK (15%) et par le trafic de management (10%). Ils ont remarqué aussi que les retransmissions sont fréquentes, pour 28% de toutes les données échangées, 46% du temps est utilisé pour les données. Les pertes sont dues aux collisions et aux erreurs de transmission. Les résultats ont montré que la plupart des transmissions sont faites au taux d'envoi le plus bas. Cela va donc diminuer les performances du réseau car il va être occupé plus longtemps.

Cet article ne parle pas de sélectionner un meilleur AP dans un environnement mais il nous servira de base pour évaluer un réseau Wi-Fi selon le nombre de retransmissions ou encore du pourcentage de temps utilisé pour envoyer différents paquets.

Chapitre 2

Concepts théoriques

Dans ce chapitre, les concepts théoriques utiles à la compréhension des différents points abordés dans ce travail de fin d'études seront développés. Ces concepts seront variés et se focaliseront exclusivement sur la couche 2 (Liaison de données), plus précisément sur la communication sans-fil. Il sera fait état en section 2.1 de l'infrastructure qui compose les réseaux 802.11. La section 2.2 aura ensuite pour but de détailler les types de trames rencontrés dans ces réseaux sans-fil au niveau de la couche 2. Enfin, la section 2.3 donnera une vue d'ensemble de l'IEEE 802.11. Les standards actuellement utilisés dans les réseaux publics seront développés ainsi que le fonctionnement général du 802.11.

Ce chapitre est principalement basé sur "802.11 wireless networks : the definitive guide" [18] et sur "IEEE Std 802.11-2012 (Revision of IEEE Std 802.11-2007)" [25].

2.1 Infrastructure du 802.11

Avant de parler des concepts qui se cachent derrière le 802.11, prenons d'abord le temps de parcourir brièvement l'architecture des réseaux 802.11. Cette section mentionnera rapidement les composants de l'architecture ainsi que les modes d'opérations possibles dans ces réseaux. Tout ceci étant nécessaire pour assimiler le vocabulaire requis pour ce travail.

2.1.1 Composant de l'architecture

Prenons la figure 2.1 suivante. On y retrouve :

- Station Terminal possédant une carte réseau sans-fil lui permettant d'envoyer et recevoir des données sur les réseaux 802.11. On compte parmi les terminaux : les pc portables, les tablettes, smartphones, imprimantes, etc.
- AP Les points d'accès représentent des dispositifs qui font le lien entre le réseau sans-fil et le réseau filaire. Ils permettent la connexion des stations

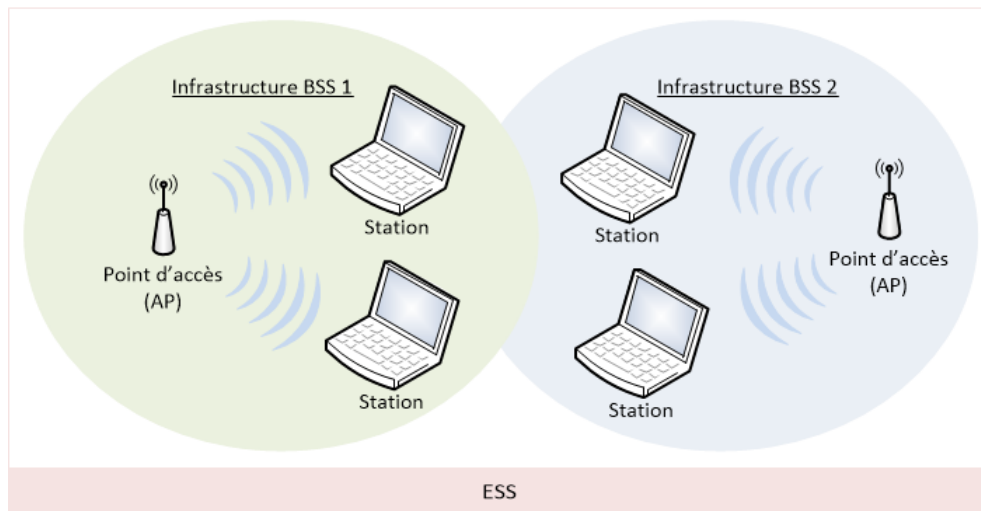


FIGURE 2.1 – Composant de l'architecture

dans le but d'envoyer ou recevoir des informations via le réseau sans-fil. En fonction du point d'accès, la zone couverte par le signal peut être plus ou moins importante.

BSS Acronyme signifiant *Basic Service Set*, il s'agit d'un groupe de stations qui communiquent entre elles. Toute station dans la zone du BSS peut transmettre aux autres stations. Il existe deux types de BSS :

- IBSS (*Independent BSS*) dans lequel les stations peuvent communiquer directement avec les autres. Il peut également se nommer *Ad hoc*.
- Infrastructure BSS est celle représentée sur la figure 2.1 et est probablement l'une des plus utilisées sur les réseaux sans-fil [5]. Chacun utilise un point d'accès identifié par une adresse unique (BSSID) qui servira lors des communications. Ces dernières, lorsqu'elles seront établies par les stations, passeront obligatoirement par ces points d'accès. Les stations devront alors s'associer auprès des points d'accès dans le but de communiquer.

ESS Signifiant *Extended Service Set*, il est créé en chaînant les BSS entre eux avec un backbone. Seules les BSS possédant le même SSID (nom diffusé par un réseau sans-fil) pourront être chaînés. Ainsi une station s'associant avec un BSS dans un ESS peut communiquer avec une autre station située sur un BSS distinct dans le même ESS. Par l'intermédiaire de ce chaînage, les stations peuvent profiter de la même connectivité tout en changeant de BSS. On peut préciser qu'il faut que les zones des BSS se chevauchent pour pouvoir changer de BSS comme le montre la figure 2.1.

2.1.2 Modes d'opération

Lorsqu'une station ou un point d'accès (AP) veut émettre sur un réseau sans-fil, la carte réseau de ces dispositifs se doit d'utiliser un certain mode d'opération en fonction du réseau ou de l'action qu'il souhaite réaliser. Il existe actuellement 6 modes d'opération [11], étant : *Monitor*, *Master*, *Managed*, *Ad-hoc*, *Mesh* et *Repeater*, dont voici un descriptif rapide :

- Monitor** Dans ce mode, la carte ne se connecte à aucun réseau sans-fil. Elle agit de manière passive, c'est-à-dire qu'aucun paquet ou trame n'est envoyé par la carte. Elle réceptionne, sans aucun filtre, toutes les trames qui circulent sur le réseau dans lequel elle est présente. Ce mode n'est pas applicable à toutes les cartes réseau, car il est dépendant du driver utilisé et des caractéristiques du chipset.
- Master** Il s'agit du mode qui est principalement utilisé par les points d'accès.
- Managed** Aussi appelé mode infrastructure, il est utilisé par les stations qui se connectent auprès d'un AP.
- Ad-hoc** Ce mode permet aux stations de communiquer les unes avec les autres sans l'intervention d'un point d'accès. C'est ce mode qui est utilisé dans les IBSS.
- Mesh** Permet à des dispositifs d'établir une communication entre eux en établissant des routes.
- Repeater** La carte réseau peut alors être configurée pour se connecter à un réseau sans-fil et étendre son signal et donc la zone couverte par le réseau d'un point d'accès.

Lors d'une recherche de Wi-Fi, le but est de s'associer au point d'accès le plus approprié. Il y a deux façons pour trouver les APs qui nous entourent : de manière active ou passive. En mode *Monitor*, la station trouve les APs en écoutant les trames *Beacon* pour le mode passif. Pour le mode actif, la station diffuse des trames *probe requests* sur chaque canal et elle peut recevoir plusieurs trames *probe response* des APs se trouvant sur ce même canal [32].

2.2 Les trames

Sur la couche liaison de données, de nombreuses trames peuvent être rencontrées ayant chacune leur propre but. Cette section se charge d'en faire rapidement le tour.

Les trames sont classées en trois catégories : *management*, *data* et *control*.

Les trames de *management* agissent comme des fonctions de surveillance permettant aux stations de voir les réseaux sans-fil, de s'y associer, de les quitter, etc. Les trames de données (*data*) transportent les données d'une station à une autre. Finalement, les trames de contrôles avertissent, en combinaison avec les trames des

données, la réception des données, la disponibilité d'un média sans-fil, etc.

Les sections suivantes reprennent brièvement les différents types de trames pouvant être rencontrés au niveau de la couche 2. Toutes les trames ne seront pas mentionnées, car leur nombre est assez conséquent. Ainsi seules les trames que nous pensons être les plus utiles seront expliquées. Tout ce qui concerne le format des ces trames ne sera pas abordé dans ces sections, mais fera partie de la section 3.4.1.1.

2.2.1 Management

Comme indiqué, les trames de management vont permettre d'établir l'identité d'un réseau, offrir la possibilité aux stations de s'y connecter, etc. Ce type représente une grande partie des trames du 802.11 et les trames envoyées renferment beaucoup d'informations concernant les points d'accès (nous pouvons citer rapidement les SSID et BSSID). Différents types existent dont voici les plus utiles :

- | | |
|-----------------------|--|
| <i>Association</i> | Ces requêtes permettent aux stations d'essayer de rejoindre un réseau. Elles vont de pair avec les réponses d'association qui représentent les réponses du point d'accès. D'autres types sont dérivés à partir de ces derniers. On notera la requête de ré-association (et sa réponse) qui sera envoyée par une station se déplaçant dans un ESS et quittant la zone couverte par un BSS pour atteindre un autre BSS. D'un autre côté quand une station souhaite quitter un AP, elle émet une requête de dissociation. |
| <i>Authentication</i> | Elles précèdent les requêtes d'association. Leur but est d'authentifier la station qui essaye de se connecter au point d'accès. Quatre méthodes d'authentification existent pour vérifier l'identité d'une station : <i>Open System</i> , <i>Shared key</i> , <i>FT</i> et <i>Simultaneous Authentication of Equals</i> . Ces requêtes vont de pair avec les dé-authentifications qui sont employées dans le but de terminer une relation d'authentification. |
| <i>Probe</i> | Les probes peuvent être décrites comme étant des requêtes formulées dans le but de demander/recevoir des informations sur le réseau (habituellement donné de manière passive par les trames <i>Beacon</i>). Deux types se distinguent : les <i>probes request</i> et les <i>probes response</i> . Les premières sont initiées par les stations souhaitant scanner les réseaux sans-fil et demandant des informations. Les deuxièmes sont les réponses de l'AP aux premières et contiennent les informations précédemment demandées. Si une <i>probe request</i> rencontre un réseau, alors ce dernier renvoie une <i>probe response</i> contenant toutes les informations qu'on peut retrouver dans une trame <i>Beacon</i> . |

Les trames d'action Ces trames servent à initier une communication entre deux appareils dans le but d'envoyer, par la suite, des blocs de *QoS data*. Via ce mécanisme, plusieurs trames de données *QoS data* peuvent être envoyées sans envoyer de trame de *ACK* [41].

Beacon Ces trames sont importantes pour la couche 2, leur but premier est d'annoncer la présence de points d'accès sur le réseau et plus précisément de *SSID*. Ce rôle doit être fait par le point d'accès et, pour pouvoir entendre ces trames, les stations doivent s'en trouver suffisamment proche. Ces trames sont transmises à un intervalle régulier appelé *Beacon Interval*. De nombreux champs composent ces trames *Beacon* mais ne seront abordés que dans la suite de ce travail, en section 3.4.1.

2.2.2 Data

Cette section englobe également les types utiles qui existent au niveau des trames de données. Certaines d'entre elles, même si faisant partie des trames de données, ne transportent aucune donnée entre les stations (ex. : *Null function*). La liste ci-dessous résume les trames de données les plus utiles :

- | | |
|----------|---|
| Data | Simple trames de données entre deux ou plusieurs stations. |
| QoS data | Toujours des trames de données qui sont transmises entre 2 ou plusieurs stations. La différence majeure réside dans le fait que ces trames sont envoyées par des points d'accès qui supportent et utilisent la qualité de service (<i>QoS</i>) dans leurs transmissions. |
| Null | Comme précisé précédemment, il s'agit d'une trame de données ne transportant aucune donnée. Leur but est uniquement d'informer un point d'accès que la station change de mode d'énergie (économie d'énergie). Ainsi quand la station est en mode économie d'énergie, l'AP doit garder dans un buffer les trames destinées à cette station. Ces données seront récupérées par la suite, quand la station sortira de son mode économique. |

2.2.3 Contrôle

Ce dernier type de trames sert principalement à administrer les accès au média sans-fil. On retrouve dans cette catégorie des trames connues comme :

- | | |
|-----------------|---|
| Request-to-Send | Sera discuté en section 2.3.5.3, elles sont requises pour réserver le média pendant une certaine période pendant laquelle les données pourront être envoyées. |
| Clear-to-Send | Sera également mentionné en section 2.3.5.3, elles sont utilisées en réponse aux trames RTS pour valider la réservation du média pendant une période. |

Acknowledgment Présentes sur la couche 2, elles sont envoyées pour confirmer la bonne réception d'autres trames précédemment envoyées. Pour que ces trames puissent être envoyées, les données réceptionnées ne doivent pas avoir été modifiées lors de la communication. Si tel est le cas, aucune trame d'*acknowledgment* ne sera transmise.

2.3 Vue d'ensemble du IEEE 802.11

L'IEEE 802.11 regroupe les normes utilisées pour les communications sur les réseaux sans-fil locaux (WLAN). Un groupe de travail est responsable de la création de standards et il opère sous la supervision du comité de normalisation LAN/-MAN de l'IEEE, de l'IEEE Computer Society et de l'IEEE-SA Standards Board [24]. L'IEEE 802.11 existe dans la plupart des appareils connectés aux réseaux sans-fil tel que les smartphones, tablette, pc portables, infrastructure réseau, etc. Le standard 802.11 définit les couches basses du modèle OSI pour une liaison sans fil utilisant des ondes électromagnétiques : la couche physique et la couche de liaison de données.

Dans cette section, nous allons nous intéresser brièvement aux différents types de standards de communication utilisés et définis par IEEE 802.11. Ensuite, il sera fait mention des canaux qu'on peut retrouver sous le 802.11, d'une explication de certains éléments de la couche physique, du contrôle d'accès du média et enfin des méthodes d'accès à ce même média.

Cette vue d'ensemble de l'IEEE 802.11 est une vue superficielle, toutes les informations détaillées ne seront pas révélées étant donné qu'elles font partie d'une matière bien plus vaste que le sujet de ce travail de fin d'études. Seules les idées utiles à ce mémoire sont synthétisées dans cette section dans le but de pouvoir avoir toutes les cartes en main pour la compréhension des concepts développés.

Pour plus d'informations détaillées concernant le IEEE 802.11, [18] et [25] sont des supports parfait.

2.3.1 Standards

Le standard 802.11 est la norme initiale offrant des débits de 1 ou 2 Mbps. Des révisions ont été apportées à cette norme pour optimiser les communications. Actuellement, de nombreux standards de communication ont été créés par l'IEEE 802.11, mais seulement une fraction d'entre eux bénéficie d'une utilisation courante. Il s'agit principalement des protocoles 802.11a/b/g/n que l'on peut facilement retrouver dans la vie quotidienne. Voici une brève description de ces standards [34, 45].

802.11a : Développé en 1999 et publié en 2000, il opère dans la bande des fréquences des 5 GHz. Ses taux de transmissions peuvent osciller entre 6 et 54 Mbit/s (plus précisément 6, 9, 12, 24, 36, 48 ou 54 Mbit/s).

802.11b : Développé en 1999, il opère entre les 2.4 et 2.5 GHz. Ses taux de transmissions peuvent être les suivants : 1, 2, 5.5 ou 11 Mbit/s.

Le 802.11b utilise la bande de fréquences des 2.4 GHz lui permettant de couvrir une plus grande zone géographique que le 802.11a mais le 802.11b adopte des taux de transmissions inférieurs à ceux du 802.11a.

802.11g : Développé en 2003, il opère entre les 2.4 et 2.5 GHz. Il reprend les meilleurs aspects du 802.11a et du 802.11b, à savoir, les taux de transmissions du 802.11a tout en gardant la compatibilité avec le 802.11b et donc sa bande de fréquences.

802.11n : Développé en 2009, il apparaît comme une amélioration des standards précédents. Pouvant opérer sur les 2 bandes de fréquences, il offre des taux de transmissions assez importants pouvant atteindre les 150 Mbits/s. Il est probablement le standard récent le plus déployé sur les réseaux sans-fil [5].

Tous ces standards peuvent être et sont actuellement utilisés dans les réseaux sans-fil. Pour permettre aux terminaux de communiquer au moyen de ces standards, leur hardware doit pouvoir le supporter et doit opérer sur la même bande de fréquences. De plus, une compatibilité entre les standards du 2.4 GHz est disponible (*legacy mode*). Ce mode permet de faire cohabiter d'anciens et de nouveaux standards. Cependant, le fait d'utiliser ce mode de compatibilité peut desservir la vitesse de transmission qui sera réduite à la vitesse maximale pouvant être offerte par le standard disposant des taux de transmissions les plus faibles. L'exemple le plus courant est l'utilisation du 802.11g (*legacy mode*) qui accepte pour des raisons de compatibilité toute transmission faite en 802.11b et en 802.11g.

On peut aussi noter que les réseaux sur les 5GHz sont moins enclins à souffrir d'interférences que les réseaux sur les 2.4 GHz. La raison est simple, de nos jours, beaucoup de dispositifs émettent sur différentes bandes de fréquences. Beaucoup d'entre eux emploient la bande des 2.4 GHz comme les fours à micro-ondes, les appareils utilisant la technologie Bluetooth, les téléphones portables, etc. [48] À l'opposé, les 5 GHz auront une saturation beaucoup moins importante vu que les interférences provenant de l'environnement seront minimisées [48].

2.3.2 Canaux et fréquences utilisables

Pour les fréquences disponibles, nous analyserons exclusivement celles utilisées en Europe. Nous omettons par défaut les autres bandes de fréquences utilisées. Ainsi les 3.6 GHz (utilisées avec le standard 802.11y, équivalent au 802.11a) qui sont en vigueur aux États-Unis ne seront pas mentionnées.

Nous nous intéresserons donc uniquement aux 2.4 et 5 GHz.

2.3.2.1 2.4 GHz

Pour la bande de fréquences des 2.4 GHz, un canal est défini tous les 5Mhz et chaque canal possède une largeur de 20Mhz (22Mhz pour le 802.11g). Le nombre de canaux obtenus varie en fonction du pays : en Europe, 13 canaux seront disponibles, 14 au Japon et 11 pour l'Amérique du Nord. Il faut noter que dans le cas du Japon l'espacement entre les 2 derniers canaux passe de 5 MHz à 12 MHz.

Comme les canaux sont séparés de 5 MHz et que chaque canal dispose d'une largeur de bande de 20 ou 22 MHz, des interférences vont obligatoirement se produire entre certains d'entre eux. Ces interférences peuvent alors perturber les opérations du système, à savoir : corruptions des trames, augmentation du délai entre les trames, etc.

La figure 2.2 présente ci-dessous nous renseigne sur les canaux qui se chevauchent causant donc ces interférences. Dans cette figure, une largeur de bande de 20 MHz a été choisie. Dans le cas où on a une largeur de bande de 22Mhz, les canaux 1 et 5 se chevauchent légèrement contrairement à une largeur de bande de 20Mhz. On remarque assez facilement qu'il existe un maximum de 3 canaux (1, 6 et 11) qui ne se chevauchent pas quelle que soit la largeur de bande, ne causant donc aucune interférence entre eux. Ces 3 canaux sont les choix idéaux lors de l'installation d'une borne d'accès sans-fil et sont les plus utilisés [5].

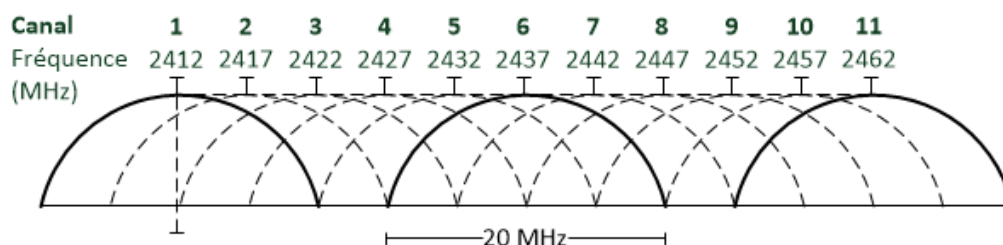


FIGURE 2.2 – canaux sur le 2.4 GHz (20 MHz largeur de bande)

2.3.2.2 5 GHz

Pour les 5 GHz, l'espacement entre les bandes de fréquences est toujours similaire, soit 5 MHz. Si le 5GHz décidait d'élire un canal à chaque intervalle, il pourrait théoriquement atteindre un nombre de 200 canaux. Cependant, cela causerait un nombre important d'interférences, car chaque canal possède une largeur de bande de 20 MHz (ou 40 MHz optionnel avec le 802.11n). Ainsi pour éviter toute interférence, un canal est élu tous les $4 * 5$ MHz.

Le 5 GHz étant plus large que le 2.4 GHz, il possède bien évidemment un nombre plus important de canaux. Tout comme le 2.4 GHz, son nombre dépend du pays/

continent dans lequel la bande de fréquences est déployée. En Europe, 19 canaux sont disponibles alors qu'en Amérique du Nord, 24 canaux existent.

La figure 2.3 nous montre seulement quelques canaux pouvant être utilisés sans risquer des interférences avec d'autres. Il faut également noter que cette figure 2.3 nous montre des canaux employant une bande de fréquences d'une largeur de 20 MHz. Si la largeur de la bande avait été de 40 MHz, le résultat aurait été différent.

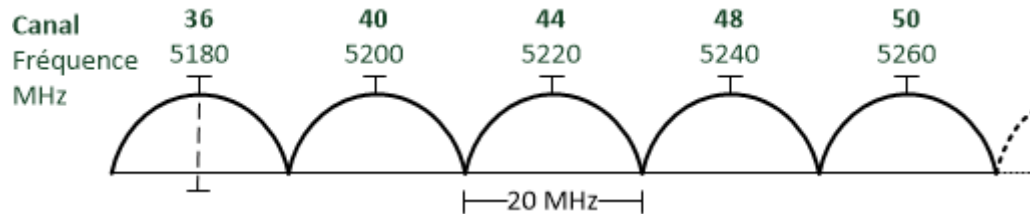


FIGURE 2.3 – Canaux sur le 5 GHz (20 MHz largeur de bande)

2.3.3 Couche PHY

La couche physique se doit d'interagir avec la couche liaison de données, qui régit le 802.11, dans le but de pouvoir encoder/décoder et envoyer/recevoir les informations lors d'une transmission. Cette partie étant fort large, nous nous limiterons à une compréhension globale de la couche physique qui est utilisée pour le 802.11.

La figure 2.4 décrit de manière succincte les éléments qui composent la couche liaison de données et physique.

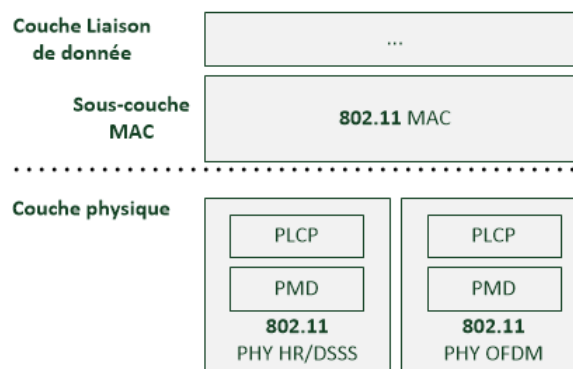


FIGURE 2.4 – Couche liaison de données et physique 802.11

La couche physique de la norme 802.11 définit plusieurs techniques de transmission afin de limiter les problèmes dus aux interférences. Ainsi, on y trouve deux techniques de modulation : HR/DSSS (*High Rate Direct Sequence Spread Spectrum*) et OFDM (*Orthogonal Frequency-Division Multiplexing*). Par simplicité, seulement ces deux-là sont citées car elles sont utilisées respectivement par les standards 802.11b/g(*legacy*) et 802.11a/g. La technique DSSS consiste à transmettre une séquence de bits pour chaque bit à envoyer. Donc chaque bit valant 1 sera remplacé

par la séquence de bits et chaque bit valant 0 sera remplacé par son complément. Le HR/DSSS est une extension du DSSS qui permet d'utiliser les taux de transmission 5.5 et 11 Mbit/s. La technique OFDM consiste à diviser la bande de fréquence en des multiples sous-porteuses orthogonales. Le fonctionnement du HR/DSSS et de l'OFDM ne faisant pas partie du sujet de ce mémoire, ils ne seront pas détaillés de manière plus précise. Leur existence doit cependant être connue, car elles auront leur utilité par la suite.

Pour amener plus de précisions sur le standard 802.11g, nous devons mentionner qu'il utilise le ERP (*Extended Rate PHY*) afin de fournir la capacité maximale tout en maintenant la compatibilité ascendante. Cette technique de modulation peut employer quatre modes d'opérations différents (ERP-DSSS, ERP-OFDM, ERP-PBCC, DSSS-OFDM). Le 802.11g (ou pure-g) fera alors usage du ERP-OFDM qui se comporte comme le OFDM mais en utilisant la bande de fréquences des 2.4 GHz. Pour le cas du 802.11g (legacy mode), celui-ci doit pouvoir être compatible avec le 802.11b (voir section 2.3.1). Pour réaliser cela, il a l'obligation d'utiliser le HR/DSSS et donc d'utiliser le mode d'opération ERP-DSSS.

La seconde partie de la figure 2.4 nous montre l'utilisation des sous-couches *Protocol Layer Convergence Procedure* et *Physical Medium Dependent* respectivement nommés PLCP et PMD. La première se chargera de préparer les données pour la communication : ajouter le PLCP preamble permettant au récepteur d'acquérir un signal OFDM entrant [27] ainsi que l'en-tête contenant les informations requises par les transmetteurs/receveurs physiques. Le deuxième va réaliser l'interface avec le support sans-fil dans le but de permettre la transmission et la réception des unités de données de la couche physique. Il se chargera également de fournir une modulation (ou démodulation) des trames en fonction du taux de transmission désiré. Il va donc être responsable de la transmission de tous les bits provenant de PLCP. Cette transmission se fera en symboles par seconde et non en bit par seconde. Le PMD devra opérer une conversion des bits en symboles avant de les transmettre. Le nombre de bits par symbole dépendra du standard et du taux de transmission.

2.3.4 Contrôle d'accès

La sous-couche MAC est la moitié basse de la couche de liaison de donnée. Elle est l'interface entre la partie logicielle contrôlant la liaison d'un nœud et la couche physique. Elle permet le contrôle d'accès au support, l'adressage et le formatage des trames, le contrôle d'erreur, la fragmentation et le réassemblage, etc. Différents types de fonctions existent pour s'adapter aux exigences des utilisateurs. La figure 2.5 liste les différentes fonctions actuellement existantes. Notons également que la figure offre une représentation simplifiée, à savoir que seules les fonctions les plus répandues sont mentionnées. Ainsi le mécanisme MCF (*Mesh Coordination Function*) n'est pas pris en compte, car cette technique de contrôle n'est utilisée que dans

les MBSS (*Mesh Basic Service Set*) où toutes les stations forment un lien avec leurs stations voisines dans le but d'échanger des messages.

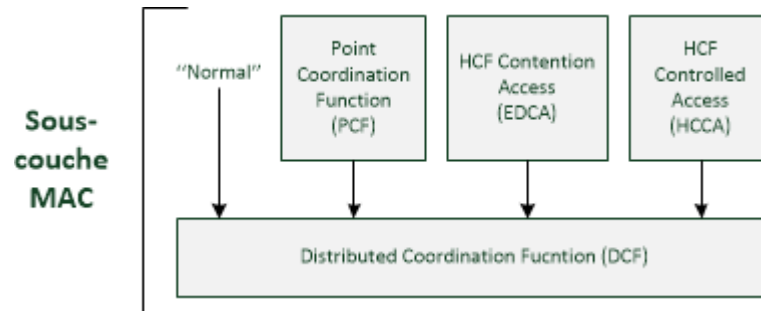


FIGURE 2.5 – Contrôle d'accès de la sous-couche MAC

On peut dès lors trouver :

- DCF** Base du mécanisme de contrôle CSMA/CA qui va s'assurer que le support radio est libre avant d'envoyer des données (ce mécanisme est décrit de manière condensée en section 2.3.5). Il s'agit probablement du mode le plus déployé dans les terminaux existants.
- PCF** Cette méthode est optionnelle. Elle permet de fournir des services sans contention en s'assurant qu'il n'y ait pas de saturation sur le média. Pour ce faire, il requiert des stations spéciales nommées *Point Coordinator* qui, localisées dans les points d'accès, se chargeront d'établir quelles stations ont le droit de transmettre. Ce service est fourni en prenant comme base le DCF. Cependant, cette méthode n'est pas fortement déployée, car il faut que les terminaux soient compatibles PCF.
- HCF** Principalement utilisé pour tout ce qui se rapporte à la qualité de service (QoS) sans devoir avoir la précision du PCF. Dès lors, cette méthode emploie plusieurs files et l'accès entre ces files est déterminé en fonction de l'application qui requiert la meilleure qualité de service. Le service qui est fourni par le HCF se base sur le DCF. De plus, le HCF peut utiliser 2 types de mécanismes : le EDCA (*Enhanced Distributed Channel Access*) et le HCCA (*HCF Controlled Channel Access*). Le premier autorise la présence de contention tandis que le deuxième l'interdit. Dans ce travail, nous laissons de côté le HCCA pour uniquement nous concentrer sur le EDCA étant donné que le HCCA requiert l'utilisation d'un coordinateur centralisé que nous ne retrouvons pas dans les réseaux publics.
- EDCA** Ce mécanisme permet de fournir un accès distribué pour tout ce qui rapporte au WMM (Wi-Fi Multi-Media). Le WMM est basé sur le standard 802.11e et est une extension se chargeant de la qualité de service [2]. Ainsi dans l'EDCA, nous retrouvons quatre catégories d'accès qui détermineront la nature du trafic transporté : *Background* (AC_BK), *Best Effort* (AC_BE), *Video* (AC_VI) et *Voice* (AC_VO). En fonction de ces catégories, une priorité sera établie sur le trafic qui atteindra la file du HCF : *Background* et

Best Effort qui se partageront les priorités les plus faibles et *Voice* qui aura la priorité la plus importante. D'autres informations concernant les paramètres du EDCA seront communiquées avec plus de détails dans la partie "Gestion de la qualité de service" de la section 3.4.1.2.

2.3.5 Méthodes d'accès au média

Quand une station présente sur un réseau sans-fil souhaite émettre des données, elle doit avant toute chose vérifier qu'elle a le droit d'envoyer ses données. C'est à dire qu'il faut qu'elle teste si ce média est déjà utilisé par une autre station. Si c'est le cas, elle ne pourra alors pas envoyer ses données au risque de créer une collision entre les données envoyées et de les corrompre ou d'engendrer une perte. Comme précisé dans le point 2.3.4, le DCF se charge d'être la base du mécanisme de contrôle CSMA/CA qui s'assurera que le média n'est pas occupé avant d'émettre des données.

Ainsi pour vérifier que le média n'est pas utilisé, deux types de détection existent : le premier se situe au niveau de la couche physique et le deuxième, au niveau de la couche liaison de données, nommés respectivement CCA (*Clear Channel Assessment*) et NAV (*Network Allocation Vector*). Le CCA va uniquement indiquer à la sous-couche MAC quand un signal est détecté sur le support sans-fil. Le NAV quant à lui est une période pendant laquelle le média est supposé être occupé (en microsecondes). Cette durée est récupérée dans les trames 802.11 (voir figure 2.6) et décrémentée jusqu'à atteindre zéro. Zéro signifiant, pour la station qui souhaite émettre, que le média est supposé être libre. Le NAV est mis à jour si un NAV, reçu depuis une trame, est supérieur au NAV actuel.

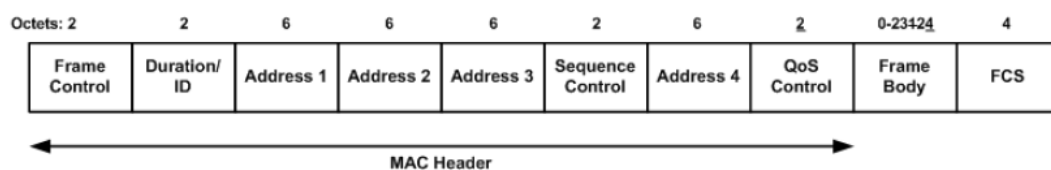


FIGURE 2.6 – Format de la trame MAC [25]

Nous allons commencer par aborder les différents intervalles de temps entre trames (ou IFS) (section 2.3.5.1). Ceux-ci seront nécessaires pour la compréhension des deux types d'accès au média existants : le mode normal et le mode RTS/CTS (section 2.3.5.2 et 2.3.5.3).

2.3.5.1 Inter Frame Spaces

L'*inter Frame Space* (ou IFS) représente l'intervalle de temps entre l'échange des trames. Il en existe six différents :

SIFS Cet intervalle est utilisé pour les transmissions de haute priorité (ex. : RTS, CTS, ACK). Les trames envoyées après le SIFS ont une plus grande priorité que les trames envoyées après un plus long intervalle. Cet intervalle représente le plus court des intervalles IFS pour les transmissions entre stations. À noter qu'il existe aussi le **RIFS** qui est un délai plus court, mais uniquement utilisé lors de multiples transmissions depuis un simple transmetteur (principalement utilisé avec le 802.11n).

PIFS Utilisé durant le PCF mode.

DIFS Intervalle minimal utilisé par les stations sous le mode d'accès DCF. Cette durée est plus importante que le temps requis par le SIFS.

EIFS Cet intervalle est seulement utilisé quand les trames réceptionnées contiennent des erreurs. La station attendra le temps EIFS avant d'entamer une nouvelle transmission.

AIFS Intervalle utilisé avec le mécanisme EDCA, il servira quand les stations, utilisant la qualité de service, veulent accéder au média pour transmettre. Comme précisé dans le point 2.3.4, différentes catégories d'accès sont disponibles. L'intervalle AIFS dépendra de la catégorie associée (4 différents types d'intervalles sont alors possibles).

2.3.5.2 Accès normal

Ici nous nous focaliserons exclusivement sur le DCF et le ECDA. Le premier étant utilisé par un trafic sans priorité (voir figure 2.7) et le second s'applique à un trafic avec priorité. Décrivons rapidement ce qui se passe lorsqu'une station souhaite émettre des trames :

1. Le média est libre : une station peut alors transmettre directement une trame en attente si le temps depuis que le média est libre est supérieur ou égal à un certain IFS. Si la trame précédente a été reçue sans erreur, cet IFS sera le DIFS. Dans le cas où la trame précédente contenait des erreurs, alors ce temps sera le EIFS.
2. Le média est occupé : attendre que ce dernier soit libre. Pour ce faire, deux mécanismes de détection de la disponibilité du média sont requis : CCA et NAV.
3. Dès lors que le média est détecté libre via CCA et NAV, la station doit attendre un intervalle IFS par rapport au standard défini par le mécanisme (DCF ou EDCA).
4. Si le média est toujours libre après l'intervalle, la station devra attendre un temps additionnel minimisant par la même occasion les collisions si plusieurs stations décident de reporter la transmission au même moment. Ce temps est communément appelé *backoff time* ou *contention window*. Il se calcule de la manière suivante : $Random() * aSlotTime$ où

- *Random()* est un nombre pseudorandom entre $[0, CW]$ où CW est la taille de la fenêtre de contention. Cette fenêtre s'agrandit de manière exponentielle à chaque retransmission de trames jusqu'à atteindre son maximum qui est lié au standard utilisé (par exemple : pour 802.11b DCF (non-QoS), la fenêtre maximale est de 1023 [40]).
 - *aSlotTime* est une constante liée au PHY utilisé, à savoir HR/DSSS, OFDM, etc. Cette constante peut être égale à $9\mu s$ ou $20\mu s$.
5. Durant chaque slot du *backoff timer*, la station doit vérifier l'état du média. Si aucune activité n'est détectée, alors le *backoff timer* est décrémenté d'un *aSlotTime*. Si le média est occupé, alors le *backoff timer* est suspendu et la station devra recommencer les étapes 2, 3 et enfin reprendre avec l'étape 5 avec le précédent *backoff timer*.
 6. Quand le timer atteint 0, et que le média est toujours libre, alors la transmission pourra se réaliser.

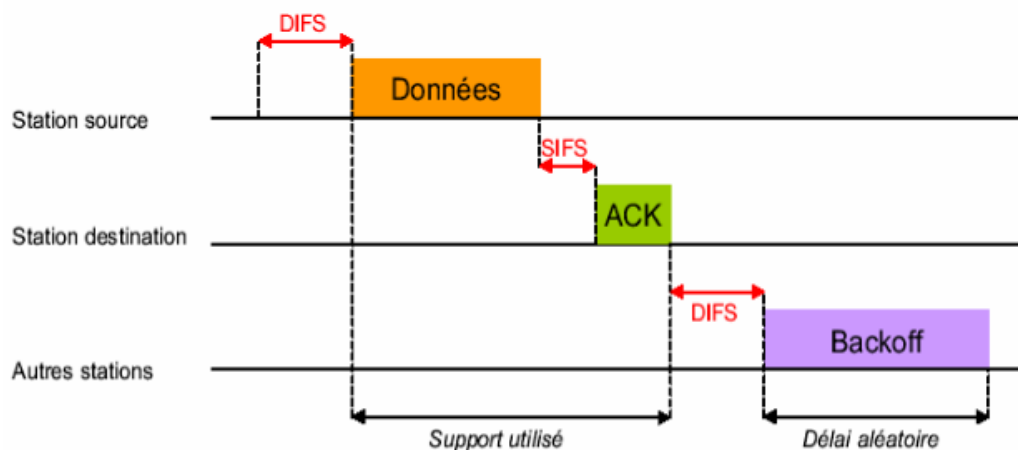


FIGURE 2.7 – Algorithme de la méthode d'accès DCF [62]

Précisons que dans le cas du EDCA, l'IFS sera dépendant du trafic qui doit être transmis, de même que le *backoff time*. Plus de précisions seront apportées dans la section 4.2.1.4.

2.3.5.3 Accès via RTS/CTS

L'utilisation du RTS/CTS est requise dans la situation du problème du nœud caché (*hidden node problem*) (il s'agit du problème le plus connu, mais d'autres existent également). Ce problème du nœud caché se produit quand une station ne détecte pas qu'une autre station est sur le même réseau qu'elle. À savoir, si trois stations nommées respectivement A, B et C existent (voir figure 2.8). Soit A communique avec B et B communique avec C mais A ne peut communiquer avec C. Dès lors, si A et C souhaitent transmettre des données vers B, ni l'une ni l'autre ne peut savoir qu'une communication est en train de se dérouler. Les deux vont alors envoyer

leurs données vers B, résultant en une collision. Ainsi du point de vue de A, C est le noeud caché (et inversement).

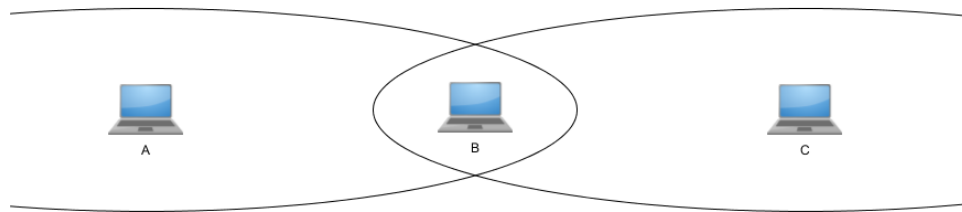


FIGURE 2.8 – Hidden node problem

Les trames RTS/CTS contiennent un champ *Duration* (voir figure 2.9 et 2.10) permettant d'indiquer, aux autres stations recevant ces trames, le temps qu'une station souhaite réserver dans le but d'envoyer sa trame de données.

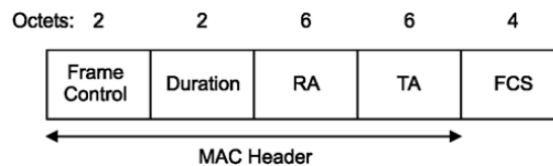


FIGURE 2.9 – Trame RTS [25]

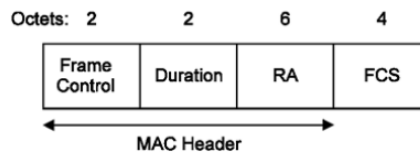


FIGURE 2.10 – Trame CTS [25]

1. Une station souhaitant envoyer doit d'abord respecter les points 2. et 3. présents lors d'un accès normal (voir section 2.3.5.2).
2. Quand le média est libre, la station émettrice doit alors avertir son voisinage qu'elle désire envoyer une trame. Pour ce faire, elle émet une trame RTS (*Request To Send*) à destination d'une autre station. Toute autre station écoutant le support reçoit cette trame et met à jour son NAV (via le champ *Duration*).
3. La station destinataire, ayant reçu le RTS, va attendre un court intervalle SIFS avant d'envoyer en retour une trame CTS indiquant que la requête de communication a été acceptée. Ce CTS va également être reçu par toute autre station se trouvant à proximité et permet, pour les stations n'ayant pas reçu le RTS précédent, de mettre à jour leur NAV avec les informations de durée fournie par le champ *Duration*.
4. La station émettrice, après réception du CTS, attend un SIFS et envoie sa trame de données.

5. Du côté de la réception, la donnée est reçue et un temps SIFS est maintenu avant de répondre par un ACK. La trame ACK est uniquement envoyée à condition que la donnée soit reçue ou que son intégrité n'ait pas été modifiée. Cela est vérifié grâce au *frame check sequence* (FCS) qui est le code de détection d'erreurs ajouté à la fin d'une trame.
6. Le ACK réceptionné, le NAV mis en place par les autres stations touche à sa fin. Le média peut alors être utilisé par les autres stations après avoir attendu un temps IFS et un *backoff time* (pour éviter que toutes les stations envoient leurs trames RTS au même moment).

Le temps IFS ainsi que le *backoff time* dépendront du mode d'accès utilisé : DCF ou EDCA.

Chapitre 3

Méthodologie

Dans ce chapitre, nous allons détailler l’approche que nous avons prise. Nous allons dans un premier temps décrire la méthodologie appliquée afin d’établir, en section 3.1.1, ce que nous considérons comme le meilleur Wi-Fi. Par la suite nous développerons en section 3.2 les composants physiques que nous avons utilisés pour réaliser nos différentes observations ainsi que les tests de validation. Ces derniers seront détaillés dans le chapitre 5.

La section 3.3 va reprendre quelques points théoriques sur le fonctionnement des Hotspots pour ensuite amener quelques points précis sur le fonctionnement de ceux de *Proximus* et *Voo*. La dernière section 3.4, quant à elle, sera responsable d’expliquer en détail les différentes observations sur lesquelles l’analyse se portera.

3.1 Comment déterminer le meilleur Wi-Fi ?

Dans le chapitre 1, nous précisions que le but de notre travail était de pouvoir se connecter au point d’accès proposant le hotspot ayant les meilleures performances dans le cas où plusieurs AP proposent un hotspot d’un même opérateur, tel que *Proximus* ou *Voo* par exemple.

Nous nous sommes permis de généraliser cette orientation. Comme il s’agit, dans ce cas, de réseaux distribuant une connexion sans fil auprès de terminaux mobiles, nous avons pris le problème dans son entièreté, à savoir : déterminer le meilleur Wi-Fi de manière générale et non spécifique à un réseau en particulier. Pour enfin uniquement récupérer les informations spécifiques aux deux fournisseurs précédents.

Pour ce faire, il nous faut une méthodologie à suivre. C’est cette dernière qui sera développée dans cette section.

3.1.1 Définition du problème

Avant toute chose, donnons précisément la signification quant à la détermination du meilleur Wi-Fi. En langage plus élaboré et technique, cela signifie :

Déterminer, dans une zone géographique spécifique, quel est le meilleur point d'accès auquel un terminal client peut se connecter. Le meilleur point d'accès étant celui qui offre les meilleures performances possibles en fonction de son environnement et de son utilisation.

Pour ce faire, ce point d'accès devra :

- Tout d'abord, être déterminé de la manière la moins intrusive possible, sans utiliser l'estimation via le rapport signal/bruit qui se réalise dans la plupart des cas sur les terminaux [32]. Le fait d'être non intrusif signifie que l'appareil réalisant cette détermination peut uniquement analyser ce qu'il voit avec une carte réseau en mode monitor (section 2.1.2). Cette détermination est uniquement basée sur la couche 2 (liaison de données) et ce qu'elle transporte.
- Dans un second temps, prendre les n premiers éléments établis lors de la première phase et faire des tests basés sur la couche 3 (réseau). Cette seconde phase se fera uniquement sur les réseaux publics de *Proximus* et *Voo* afin de venir raffiner les résultats obtenus précédemment.

3.1.2 Idée générale du fonctionnement

Pour pouvoir atteindre notre but, un algorithme a été créé avec les informations récupérées depuis les phases d'observation réalisées en section 3.4.

Pour que l'algorithme puisse exécuter son travail, nous devons nous installer dans une zone géographique possédant un certain nombre de réseau sans fil. Nous lançons le programme sur un terminal doté d'une carte réseau sans fil. Ce programme se charge de scanner toutes les fréquences disponibles selon l'ETSI sur les 2.4 Ghz et 5 GHz. Pour ce faire, il place sa carte réseau en mode *monitor* et écoute ce qui circule sur chaque canal (13 canaux pour le 2.4 GHz et 19 canaux pour le 5 GHz) durant une période de temps déterminée. Le choix de la durée de cette période dans le cadre de notre algorithme est détaillé dans la section 4.1.1. Par la suite, les informations récupérées seront parcourues et filtrées au moyen de différents critères afin de déterminer les meilleurs réseaux Wi-Fi auxquels se connecter.

L'implémentation de l'algorithme ainsi qu'une description plus approfondie de son fonctionnement et de sa composition sera faite dans le chapitre 4.

3.2 Choix du matériel

Pour réaliser nos observations et nos tests, nous avons utilisé différents composants physiques.

3.2.1 Raspberry Pi 2

Nous avons utilisé un Raspberry Pi 2 pour pouvoir se rapprocher le plus possible de la mobilité et la rapidité d'un smartphone ou d'une tablette vu les difficultés rencontrées avec ceux-ci (détaillées en annexe F). Le système d'exploitation installé sur le nano-ordinateur est la version 7 (wheezy) de Raspbian. Ce dernier étant un système d'exploitation basé sur Debian mais optimisé pour l'architecture du Raspberry Pi [37]. Ceci nous a permis de faire un maximum de développements sur ordinateur portable pour ensuite passer en phase de test sur le Raspberry.

D'un point de vue plus technique, il est équipé d'un processeur Broadcom BCM2836, quatre coeurs ARMv7 à 900 MHz et de 1 Go de RAM [52]. Il contient aussi un port Ethernet et quatre ports USB. Par contre, il ne contient pas d'antenne Wi-Fi interne. Plus d'informations quant aux spécifications sont disponible sur [51].

3.2.2 Antenne Wi-Fi

Vu que le Raspberry Pi 2 ne contient pas d'antenne Wi-Fi interne, il a fallu que nous trouvions une antenne capable de réaliser nos tests. Nous tenons à attirer votre attention sur le fait que toutes les antennes ne peuvent pas se mettre en mode *monitor* et donc, elles ne pourront pas réaliser les tests que nous avons effectués. L'antenne se doit de pouvoir supporter un pilote de carte réseau compatible avec le mode *monitor* ou plus généralement avec la suite *Aircrack-ng*. De plus elle doit pouvoir faire des mesures en 2.4 Ghz et 5 Ghz.

Si les cartes réseaux intégrées se voient octroyer plus de chances d'avoir une compatibilité avec le mode *monitor*, ceci est bien plus rare pour les adaptateurs réseaux sans fil.

Nous avons réussi à lister un certain nombre d'antennes pouvant réaliser cette tâche mais avec la restriction de se limiter au 2.4 GHz ou encore de limiter les standards pris en charge. De telles antennes peuvent fortement réduire les fréquences scannées. Nous n'avons dès lors trouvé qu'une seule antenne apte à scanner les 2 fréquences que sont les 2.4GHz et 5GHz : l'*Alfa AWUS051NH* possédant le pilote *RT2870STA* du fabricant de chipset Wi-Fi *Ralink*. Cette antenne est compatible avec les standards 802.11 a/b/g et n. Pour les débits maximums suivant le standard, veuillez vous référer à la section 2.3.1. Elle sera connectée au Raspberry par un port USB 2.0.

En annexe B sont listés les antennes et pilotes pouvant être utilisés pour faire du mode monitor.

3.2.3 Serveurs

Nous avons utilisé les serveurs uniquement pour la seconde phase qui a pour but de réaliser des tests via la couche 3. Nous listons ici les deux types de serveur que nous avons utilisés. Leur exploitation est décrite dans le chapitre 4 en section 4.2.2.

3.2.3.1 UCL

Durant une grande partie de la phase de tests, nous avons utilisé un serveur physique situé à l'Université Catholique de Louvain-la-Neuve. Le système d'exploitation est la version *Jessie 8.3* de Debian en 64 bits. Il est installé sur un *Intel(R) Pentium(R) 4 CPU 3.20GHz* avec 2GB de mémoire RAM. La carte réseau est une *NetXtreme BCM5751 Gigabit Ethernet PCI Express* ayant une vitesse de 100Mbit/s et une capacité de 1Gbit/s.

3.2.3.2 OVH

Suite à différentes observations décrites dans le chapitre 4, nous avons utilisé un serveur loué chez OvH [47]. Le système d'exploitation est un Centos [10] en 64 bits. Il est installé sur un *Intel(R) Atom(TM) CPU N2800 1.86GHz* avec 4GB de mémoire RAM. La carte réseau est une *Intel 82574L Gigabit Network Connection* ayant une vitesse de 100Mbit/s et une capacité de 1Gbit/s.

3.2.4 Routeurs

Pour valider notre algorithme, nous avons utilisé deux routeurs pour que le Raspberry capte des points d'accès avec des charges d'utilisation que nous pouvions contrôler. Nous avons utilisé deux *TP-LINK TL-WR741ND*, ceux-ci ont été flashés sur *OpenWrt* avec la version *Attitude Adjustment 12.09*. *OpenWrt* est une distribution basée sur GNU/Linux pour du matériel embarqué comme des routeurs. Ces deux routeurs sont capables d'opérer seulement en 2.4 GHz et ont le pilote "ath9k". Pour la phase de validation, veuillez vous référer au chapitre 5.

3.3 Hotspots

Les Hotspots sont des points d'accès mis à disposition des utilisateurs mobiles pour se connecter à internet. Cet accès peut être gratuit ou payant. Dans le cas de *Proximus* et *Voo*, l'accès est gratuit seulement pour leurs abonnés et payant pour toutes autres personnes. Il existe deux types de Hotspot : ouvert ou fermé. Nous pouvons généraliser ces accès à tous types de réseau (sans-fil ou filaire) nécessitant un contrôle d'accès (universités, hôtels, ...).

3.3.1 Hotspot ouvert

Lorsque le hotspot est de type ouvert, il s'agit d'un portail captif. Cette technique va forcer l'utilisateur qui désire accéder à un service HTTP ou HTTPS à s'authentifier sur le réseau. Pour cela, l'utilisateur sera redirigé vers une page web et devra entrer ses données ou payer pour s'authentifier. Il existe deux techniques.

La première intercepte les paquets liés aux protocoles HTTP ou HTTPS et crée la redirection. Cela peut être vu comme une attaque "*Man-in-the-middle*". Plus de détails sur cette technique seront apportés à la section 3.4.2.1.

La deuxième prévient le client qu'un portail captif est présent grâce au serveur DHCP ou grâce au routeur [15].

3.3.2 Hotspot fermé

Lorsque le hotspot est de type fermé, il utilise le protocole de communication extensible, *Extensible Authentication Protocol (EAP)*. Ce protocole permet différentes méthodes d'authentification et est utilisé pour fournir un mécanisme d'authentification pour les liaisons PPP afin d'autoriser l'établissement de la couche réseau [7].

Dans le cas d'un hotspot fermé *Proximus*, les protocoles EAP-TTLS et EAP-SIM sont utilisés [60] et, en général, le serveur est de type RADIUS [36]. D'autres hotspots fermés utilisent le protocole EAP-MSCHAPv2 (PEAP) par exemple [44].

Le protocole PEAP et le protocole EAP-TTLS sont assez proches, la différence est que TTLS supporte toutes les méthodes d'authentification de EAP (CHAP, PAP, MS-CHAP et MS-CHAPv2) alors que PEAP utilise un tunnel TLS pour sécuriser un second échange EAP [19]. PEAP utilise deux certificats pour la création d'un tunnel sécurisé, un du côté serveur et un du côté client [1].

Le protocole EAP-SIM permet une authentification des clients des réseaux Wi-Fi et des réseaux de téléphonie mobile grâce à leur carte SIM [16].

3.4 Observations

Maintenant que les bases ont été décrites, intéressons-nous, dans cette section, aux différentes observations que nous avons réalisées lors de l'analyse des paquets échangés entre les stations et le point d'accès.

Les réseaux que nous analyserons sont des *infrastructure BSS* (voir section 2.1.1). Ainsi, toutes les données échangées entre deux terminaux se font obligatoirement via un point d'accès.

Nous avons séparé les observations en deux sous-sections : la couche 2 et la couche 3. Nous avons réalisé différents scans du trafic avec la carte réseau mise en mode *Monitor* pour la couche 2 et en mode *Managed* pour la couche 3.

3.4.1 Couche 2

Pour atteindre les buts de notre travail, il nous faut absolument trouver les APs qui nous entourent afin de déterminer parmi ces derniers quel est le meilleur. Comme précisé en section 2.1.2, il existe deux façons de faire : actif et passif. Les deux peuvent être utilisées sans grande différence dans les résultats retournés. Cependant, comme le principal but de notre implémentation est d'être destiné à être employé sur des dispositifs mobiles, la question de la consommation énergétique est à prendre en compte. En sachant que les scans actifs sont plus gourmands en terme d'énergie que les scans passifs, nous avons décidé de nous concentrer sur ce dernier type [23].

Lors d'un *sniffing* en mode *Monitor*, différents types de trames peuvent être découverts : des trames de management, de contrôle et de données. La section 2.2 contient un rapide descriptif de chacune de ces différentes trames. Toutes ces trames peuvent être observées dans les réseaux sans fil à condition qu'elles soient diffusées par le point d'accès. Un seul point peut cependant restreindre le nombre de trames capturées : le fait que l'interface en mode *Monitor* ne soit pas capable de capturer toutes les trames envoyées [55]. Ceci étant dit, les traces récupérées offrent toutefois un contenu suffisamment riche qui pourra être exploité.

Dans les sous-sections suivantes, il sera fait état de la structure générale d'une trame et des informations considérées comme importantes dans les trames *Beacon*.

3.4.1.1 Structure générale d'une trame

Avant de décrire les informations importantes relevées dans les trames *Beacon*, il faut d'abord comprendre comment sont structurées les trames reçues. La figure 3.1 illustre cette structure.

```

Frame 18: 243 bytes on wire (1944 bits), 243 bytes captured (1944 bits)
Radiotap Header v0, Length 18
IEEE 802.11 Beacon frame, Flags: .....C
IEEE 802.11 wireless LAN management frame
+ Fixed parameters (12 bytes)
+ Tagged parameters (185 bytes)

```

FIGURE 3.1 – Structure d'une trame capturée

Radiotap Header

C'est le standard pour la réception et l'injection de trame 802.11 [3]. Il définit des informations supplémentaires par rapport à la trame originale comme la fréquence de réception, la puissance du signal, le débit de données, etc. Ces dernières sont ajoutées par le pilote de la carte réseau. Donc le *Radiotap Header* n'est pas assuré d'être présent dans une capture en mode *monitor* car il dépend entièrement de

la carte réseau qui est utilisée. Cependant, d'après les différents tests réalisés, il semble être présent sur la plupart des terminaux.

Type de trames

Il représente la seconde partie des trames 802.11 (ou la première si le *Radiotap Header* n'est pas disponible) et permet de définir le type de trame qui est réceptionné (trame *Beacon*, *probe request*, trame de Ack, trame de données, etc). Elle contient également des informations quant à la retransmission d'une trame et les adresses MAC correspondant au receveur, au transmetteur, au BSSID, à la source et à la destination. Notons qu'en fonction du type de trame envoyé, certaines de ces adresses peuvent être similaires. Ainsi dans le cas d'une trame *Beacon*, les adresses du receveur et de la destination sont l'adresse de *broadcast*. De plus, cette structure contient également le numéro de fréquence de la trame.

Contenu de la trame

C'est la partie qui contient les données de la trame, à savoir : les informations que la trame véhicule. Le contenu dépend du type de trame qui a été envoyé. Pour les trames de données, le contenu sera uniquement des données. Pour les trames de management (*Beacon*, *probes response*, etc.), le contenu est uniquement de type informatif. Seules les trames de contrôle ne contiennent pas cette organisation car toutes leurs informations sont disponibles dans la partie référant le type de la trame.

3.4.1.2 Informations importantes des trames

Dans cette section, nous détaillons les informations que nous considérons comme importantes dans les trames que pouvons capturer. Ces informations s'orientent principalement sur les trames *Beacon* car elles contiennent beaucoup d'éléments qui méritent qu'on y jette un coup d'œil. Nous nous intéresserons également aux autres trames dans le chapitre 4.

Le tableau 3.1 reprend les informations importantes avec une brève explication. Elles seront détaillées plus précisément dans la suite de cette section.

SSID et BSSID

Décrits très brièvement en 2.1.1, le SSID est présent dans tous les *Beacon* mais aussi dans les *probes requests*, *probes responses*, les requêtes d'association et les requêtes de ré-association. Signifiant *Service Set Identifier*, son rôle est de renseigner le nom qu'un point d'accès veut diffuser dans un réseau sans fil dans le but de permettre aux utilisateurs de s'y connecter. Ce nom n'est pas unique et plusieurs points d'accès peuvent diffuser le même.

Champs	Explications
BSSID	Permet de différencier les mêmes SSID.
N° de séquence	Compteur associé à différentes trames.
Timestamp	Temps depuis lequel l'antenne de l'AP est active.
Intervalle de <i>Beacon</i>	Intervalle d'envoi entre deux trames <i>Beacon</i> .
Data Rate	Taux d'envoi des trames.
Channel	N° de canal de l'AP.
SSI	Rapport signal bruit.
PHY type	Standard utilisé par l'AP.

TABLE 3.1 – Tableau reprenant les champs importants rencontrés dans les trames *Beacon*

Un point d'accès n'est pas limité à la diffusion d'un seul SSID. Il peut donc en diffuser plusieurs dans un réseau sans fil et ce, sans limite théorique. Le nombre de SSID diffusé aura des conséquences sur la performance du réseau.

Pour différencier les mêmes SSID entre eux ou pour connaître le nombre de SSID qu'un point d'accès diffuse, le BSSID joue un rôle primordial. Signifiant *Basic Service Set Identifier*, il identifie de manière unique et au moyen d'une adresse MAC un point d'accès virtuel (VAP). Ces points d'accès virtuels sont considérés comme des entités dérivées des points d'accès physiques. Pour les clients, ils apparaissent comme des points d'accès avec leur propre SSID [18, 31]. Ainsi quand un point d'accès physique crée un nouveau SSID, il crée en fait un nouveau VAP avec son propre BSSID et sa propre sécurité. Pour ce qui est de la création du BSSID, ce dernier sera dérivé de l'adresse MAC du point d'accès physique.

La figure 3.2 permet d'illustrer ces propos de manière plus compréhensible et de montrer que plusieurs AP peuvent utiliser le même SSID tout en étant différenciables les uns des autres en utilisant leur BSSID.

56:33:8e:53:60:84	Beacon frame, SN=564, FN=0, SSID=PROXIMUS_AUTO_FON
52:33:8e:53:60:84	Beacon frame, SN=565, FN=0, SSID=PROXIMUS_FON
9e:97:26:12:63:ff	Beacon frame, SN=4001, FN=0, SSID=PROXIMUS_FON
9e:97:26:12:63:f1	Beacon frame, SN=4002, FN=0, SSID=PROXIMUS_AUTO_FON
Technico_12:63:fe	Beacon frame, SN=4003, FN=0, SSID=wifi-2.4-63FE

FIGURE 3.2 – SSID et BSSID chez des VAP de *Proximus*¹

On y retrouve en rouge deux VAP possédant le même SSID mais bien différenciable vis à vis de leur adresses MAC. L'encadré bleu sert à illustrer le fait que plusieurs VAP proviennent d'un même point d'accès.

1. Afin de mettre en évidence les éléments voulus, certaines informations de la capture ont été volontairement omises. De plus, un filtre a été appliqué sur la capture pour ne récupérer que ces éléments.

N° de séquence

C'est une valeur qui est associée aux trames et qui sera incrémentée à chaque envoi d'une nouvelle trame ou des trames fragmentées. Ces valeurs sont comprises entre 0 et 4095 (selon un compteur modulo 4096 [18, 25]) et quand la valeur atteint son maximum, elle est réinitialisée.

Notons également que le compteur utilisé pour déterminer le numéro de séquence des trames *Beacon* est le même que le compteur utilisé pour déterminer le numéro de séquence des trames de données et de la plupart de trames de management (*probe response*, trame d'authentification, trame d'association, trames d'action, ...). A l'inverse, certaines trames n'utilisent pas ce compteur ou n'en n'ont tout simplement pas :

- *Probe request* : elles sont envoyées par la station et non par l'AP.
- *Les trames de contrôle* : RTS, CTS, ACK.
- *Block ACK* : utilisées avec les trames d'action.
- *Block ACK request* : utilisées avec les trames d'action.
- *EAPOL (key message)* : message utilisé pour le 4-way handshake dans le but de crypter la communication entre une station et un point d'accès [42].
- *NULL function (no data)* : trames s'occupant de la gestion de l'économie d'énergie des stations associées à un points d'accès [25].
- *QoS data* : trames de données possédant une qualité de service.

La figure 3.3 reflète un exemple d'attribution des numéros de séquences parmi les trames de management. On peut observer que ces numéros se suivent de façon incrémentale.

Time	Info
13.637743	Probe Request, SN=526, FN=0, Flags=....., SSID=AP1
13.640838	Probe Response, SN=1059, FN=0, Flags=....., BI=100, SSID=AP1
13.656889	Authentication, SN=527, FN=0, Flags=.....C
13.664922	Authentication, SN=1060, FN=0, Flags=.....C
13.666685	Association Request, SN=528, FN=0, Flags=....., SSID=AP1
13.668725	Association Response, SN=1061, FN=0, Flags=.....C
13.721626	Beacon frame, SN=1062, FN=0, Flags=....., BI=100, SSID=AP1
13.827073	Beacon frame, SN=1063, FN=0, Flags=....., BI=100, SSID=AP1
13.926489	Beacon frame, SN=1064, FN=0, Flags=....., BI=100, SSID=AP1

FIGURE 3.3 – Numéro de séquence pour les trames de management²

La seconde figure 3.4 sert à démontrer que le compteur de numéro de séquence utilisé par les trames de management est également celui utilisé par les trames de données. On peut aussi remarquer d'après cette figure que les trames de données

2. Un filtre a été appliqué sur la capture afin de ne montrer que les trames de management et de données.

s'assurant d'une qualité de service (QoS) possèdent un compteur indépendant des autres trames. La raison est la suivante : les stations qui utilisent la qualité de services auront différentes files de transmission et donc différents compteurs en fonction de ces files [18].

Time	Info
16.588917	Beacon frame, SN=1096, FN=0, Flags=.....C, BI=100, SSID=AP1
16.620410	QoS Data, SN=4, FN=0, Flags=.p....TC
16.623978	Data, SN=1097, FN=0, Flags=.p....F.C
16.624835	Data, SN=1098, FN=0, Flags=.p....F.C
16.628175	Null function (No data), SN=536, FN=0, Flags=.....TC
16.691290	Beacon frame, SN=1099, FN=0, Flags=.....C, BI=100, SSID=AP1
16.793683	Beacon frame, SN=1100, FN=0, Flags=.....C, BI=100, SSID=AP1

FIGURE 3.4 – Numéro de séquence pour les trames de données ²

Timestamp

Cette valeur représente le temps (en microseconde) depuis lequel l'antenne du point d'accès est active. Cette valeur peut être aussi retrouvée dans les *probes response*. Le *timestamp* qui est inséré dans chaque trame *Beacon* (ou *probe response*) correspond au temps d'activité de l'antenne, le moment où le symbole de donnée contenant le premier bit est passé à la couche PHY pour la transmission (il comprend aussi le délai requis pour traverser les interfaces entre la couche MAC et PHY [25]).

Pour prouver que le *timestamp* est lié à l'antenne, nous avons réalisé un scan sur le canal de l'AP. Pendant cette écoute, nous avons désactivé l'antenne pendant quelques secondes pour ensuite la réactiver. On peut voir à la figure 3.5 que le *timestamp* est remis à 0 entre les trames 3150 et 3742. Le temps écoulé entre ces deux trames correspond au temps pendant lequel l'antenne est restée désactivée.

Time	Info
34.076113	Beacon frame, SN=3094, FN=0, Flags=.....C, BI=100, SSID=AP3
Frame 3150: 153 bytes on wire (1224 bits), 153 bytes captured (1224 bits)	
Radiotap Header v0, Length 24	
IEEE 802.11 Beacon frame, Flags: C	
IEEE 802.11 wireless LAN management frame	
[-] Fixed parameters (12 bytes)	
Timestamp: 0x00000000001482180	
Frame 3742: 153 bytes on wire (1224 bits), 153 bytes captured (1224 bits)	
Radiotap Header v0, Length 24	
IEEE 802.11 Beacon frame, Flags: C	
IEEE 802.11 wireless LAN management frame	
[-] Fixed parameters (12 bytes)	
Timestamp: 0x000000000019181	

FIGURE 3.5 – Timestamp envoyé dans les trames *Beacon*

Intervalle des *Beacon*

Cette valeur représente le nombre d'unité de temps (TU) obligatoire entre les temps de transmission de trames *Beacon*. Par défaut, cette valeur est égale à 100 TU et comme 1 TU représente $1024 \mu s$, on obtient un intervalle pour la génération des *Beacon* de 102.4 ms.

On parle alors de TBTT (Target Beacon Transmission Time) comme étant le moment où les *Beacon* sont censées être transmises [18]. Pour être plus précis, une trame *Beacon* est envoyée au *scheduler* à chaque TBTT et considérée comme la trame suivante à être transmise selon les règles de transmission définies en 2.3.5.

Tous les TBTT sont alors espacés d'un certain intervalle qui par défaut devrait être de 100 TU. La première trame *Beacon* est transmise au moment appelé *Time 0* et correspond au premier TBTT [25]. Il s'agit du moment où l'antenne s'active pour la première fois depuis que le point d'accès est alimenté. À partir de cet instant, une trame *Beacon* sera générée à chaque TBTT.

Concernant la gestion de la génération de ces trames *Beacon*, elle est assurée par le pilote de la carte réseau sans fil. N'ayant pas accès aux contenus des routeurs disponibles de Proximus ou Voo, nous nous sommes uniquement concentrés sur les pilotes fournis avec *OpenWrt*. Les routeurs utilisés pour la validation de l'algorithme (section 3.2.4) possèdent tous deux des pilotes *ath9k*. Pour comprendre globalement les mécanismes, [38] décrit les fonctions, structures et autres composants utilisés dans le but de réceptionner et transmettre des données. Ainsi on peut se rendre compte que le pilote est responsable de la création des *Beacon* en fonction d'un modèle statique, de leur mettre le timestamp approprié et d'appeler les interfaces de configuration pour récupérer les paramètres (longueur trames, intervalles, etc) dans le but de générer les *Beacon*. Son rôle est aussi d'insérer les trames *Beacon* dans les files hardware quand une interruption hardware a été enclenchée (en fonction du TBTT).

Nous avons remarqué que *Fon* utilise un firmware dérivé de *OpenWrt* donc nous pouvons supposer que *Proximus* utilise le même type d'architecture pour pouvoir avoir le même comportement que *Fon*.

La figure 3.6 montre la variation de l'intervalle de temps entre la génération des trames *Beacons*. On peut facilement voir que lorsqu'une trame a un intervalle plus long que 102.4ms, comme par exemple entre la première et la deuxième trame, on aura un intervalle plus court pour la trame suivante, comme pour la troisième trame, afin de grader une moyenne des intervalles entre toutes les trames égales à 102.4ms (en pointillé sur le graphe). Ceci montrant que les *Beacon* sont générés en fonction d'un intervalle (avec une gigue).

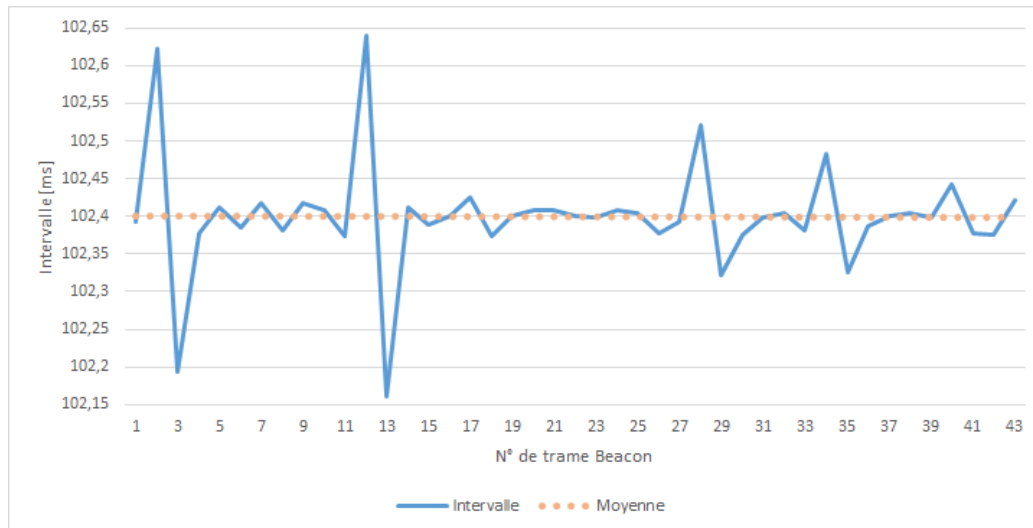


FIGURE 3.6 – Graphe représentant l'intervalle entre la génération des trames *Beacon*

Data Rate & Supported Rates

Le premier indique le débit de données en réception et le second les différents taux supportés par l'antenne du point d'accès. Chaque taux supporté est représenté par un octet, le premier bit indique si c'est un taux basic ou si c'est un taux supporté et les 7 bits suivants spécifient la valeur du taux en unité de 500kbps (ex : un taux basic de 6Mbps sera représenté par "10001100"). Ces taux peuvent être aussi récupérés dans les requêtes et les réponses des trames de *probe*, d'*Association* et de *reassociation*.

Ces taux représentent souvent tous les taux de transmission possibles du standard utilisé. S'il s'agit du 802.11g, les taux possibles seront 6, 9, 12, 18, 24, 36, 48 et 54 Mbit/s. Notons que pour le débit de données en réception des trames *Beacon*, il correspond par défaut au débit minimal supporté. Ainsi pour le 802.11g, il sera de 6Mbit/s. Cela devient plus critique quand le standard utilisé se trouve être le 802.11b où le débit minimal est de 1Mbit/s. Plus critique car cela signifie que les trames *Beacon* prendront plus de temps à être envoyées que si le 802.11g avait été utilisé. On parlera alors de consommation d'*airtime* plus grande pour les trames *Beacon*. Cette notion étant importante dans les réseaux sans fil, elle sera développée plus en profondeur dans la section 4.2.1.4.

Channel

Le champ *DS parameter* n'est pas obligatoire mais nous en avons besoin pour réaliser nos analyses. Pour pouvoir récupérer le numéro de canal, il y a trois possibilités : soit il se trouve dans la trame de management (deux endroits possibles), soit il faut récupérer la fréquence contenue dans le *radiotap header*.

Channel flags

Ce paramètre nous permet de connaître la technique de diffusion du signal. Le IEEE 802.11 en définit plusieurs, *Proximus* utilise la "*Complementary Code Keying (CCK)*" qui est une technique décrite dans le standard IEEE 802.11b et qui permet d'obtenir des débits de 5.5Mbps et 11Mbps en 2.4GHz [25]. Les techniques "*Direct Sequence Spread Spectrum (DSSS)*" et "*Orthogonal Frequency Division Multiplexing (OFDM)*" sont largement utilisées dans les réseaux d'aujourd'hui (voir section 2.3.3).

SSI signal

Le SSI, ou RSSI, correspond à la puissance du signal reçu et est exprimé en *dBm*. Cette valeur fait partie intégrante du *Radiotap Header* et ne prend pas en compte le rapport signal/bruit. Elle ne prend donc pas en compte le bruit environnant. Il faut également préciser que ces valeurs sont toujours des nombres entiers négatifs car le signal provient d'un point d'accès et a tendance à se dégrader en fonction de la distance parcourue jusqu'à une station.

Ce signal peut être influencé par différents facteurs environnementaux tels que l'architecture d'un bâtiment, les matériaux présents dans les murs et plafond d'une infrastructure ou encore par d'autres fréquences présentes dans la même zone géographique. Ces facteurs peuvent réduire le signal voir le détériorer au point de rendre la superficie couverte par le signal radio assez faible.

Concernant les cartes réseaux, ces dernières ont des limites quant à la détection des signaux envoyés par les points d'accès. En lisant leur *datasheet*, on remarque assez rapidement que ces limites sont aussi variées que les techniques de modulation et les standards utilisés. Par exemple, l'antenne Wi-Fi utilisée dans ce travail (l'*Alfa AWUS051NH*) présente différentes limites de détection [26]. Cette observation permet d'introduire une nouvelle question : Si la puissance maximale du signal capturée par une antenne lambda est différente en fonction du standard, il est fort probable que ces limites soient différentes en fonction de l'antenne ou de son fabricant. Par conséquent, les trames capturées par deux antennes différentes auront-elles la même valeur RSSI ?

Afin de répondre correctement à cette question, nous avons conduit des tests sur deux terminaux dotés de cartes réseaux sans fil différentes. La figure 3.7 et 3.8 nous montrent les résultats de ces tests. Il apparait nettement que pour une même trame *Beacon* capturée, la puissance du signal n'est pas similaire.

3. La capture ne montre que les informations utiles afin d'appuyer les propos. Tout autre type d'information a été volontairement omis.

Time	Info
1.418428	Beacon frame, SN=2142, FN=0, Flags=....., SSID=PROXIMUS_FON
Frame 230:	309 bytes on wire (2472 bits), 309 bytes captured (2472 bits)
Radiotap Header v0,	Length 18
SSI Signal:	-58 dBm

FIGURE 3.7 – Signal capturé par une carte réseau *Intel WiFi Link 5100* AGN³

Time	Info
0.128996	Beacon frame, SN=2142, FN=0, Flags=....., SSID=PROXIMUS_FON
Frame 10:	305 bytes on wire (2440 bits), 305 bytes captured (2440 bits)
Radiotap Header v0,	Length 18
SSI Signal:	-37 dBm

FIGURE 3.8 – Signal capturé par une carte réseau *Alfa AWUS051NH*³

D'après cette observation, on peut retirer le fait que le signal capturé dépend bien de la carte réseau qui est en activité. Par conséquent, alors que certaines antennes seront capables de détecter un grand nombre de réseau sans fil, d'autres seront limitées à un nombre plus restreint.

PHY type

Ce champ indique le standard utilisé par le point d'accès.

Gestion de la qualité de service

En dehors de trames *Beacon*, on voit aussi apparaître des trames de données utilisant une qualité de service. En plus de posséder les informations de bases comme définit dans le point 3.4.1.1, elles possèdent un champ nommé *Qos Control* qui s'occupe de la gestion de la qualité de service. Dans ce champ est spécifiée la priorité qu'une trame peut avoir sur les autres et donc la nature du trafic qui circule.

En 2.3.4, nous expliquions que la gestion de la qualité de service était traitée par le 802.11e définissant le mécanisme de contrôle d'accès EDCA. Ce mécanisme définit quatre catégories d'accès : AC_BK, AC_BE, AC_VO, AC_VI. Une d'entre elle est alors communiquée dans la trame de données QoS capturée. Leur but étant de définir la priorité du trafic. Cependant à quoi correspondent-elles ?

Il suffit d'analyser les trames *Beacon* en provenance d'un point d'accès supportant la qualité de services pour en déterminer leur portée. Tout point d'accès permettant l'envoi de trames utilisant la qualité de service le mentionne dans ses trames *Beacon*. Il réalise ceci en insérant un paramètre nommé WMM. C'est cette partie qui nous permet de comprendre plus en profondeur l'utilisation des priorités dans les réseaux sans fil. À la figure 3.9 se trouve les paramètres nommés AC définissant les valeurs des différents éléments dépendant des priorités.

```

AC Parameters ACI 0 (Best Effort), ACM no , AIFSN 3, ECWmin 4 ,ECWmax 10, TXOP 0
AC Parameters ACI 1 (Background), ACM no , AIFSN 7, ECWmin 4 ,ECWmax 10, TXOP 0
AC Parameters ACI 2 (Video), ACM no , AIFSN 2, ECWmin 3 ,ECWmax 4, TXOP 94
AC Parameters ACI 3 (Voice), ACM no , AIFSN 2, ECWmin 2 ,ECWmax 3, TXOP 47

```

FIGURE 3.9 – Information sur les priorités depuis les trames *Beacon*

Les valeurs des paramètres présentés dans la figure correspondent à celles d’une configuration par défaut. Cependant étant configurable, elles dépendent entièrement de l’administrateur en charge du réseau.

Dans les valeurs transportées, on trouve l’ACI, l’AIFSN, l’ECWmin, l’ECWmax, l’ACM et le TXOP[25] :

- La première, signifiant *Access Category Index*, désigne simplement la catégorie d’accès auquel le paramètre fait référence (AC_BK, AC_BE, etc).
- La deuxième valeur, de son nom complet *Arbitration Interframe Space Number*, permet de définir le nombre de *slots* qui devront être utilisés après un temps *SIFS* afin de transmettre une trame ou avant d’attendre un temps additionnel (*backof time*) comme mentionné dans la section 2.3.5.2. Plus ce nombre sera important, plus longtemps il faudra attendre. Cette valeur rentre directement en compte dans le calcul de l’AIFS défini en section 2.3.5.1.
- l’ECWmin et l’ECWmax vont de pair. Elles correspondent aux valeurs minimales et maximales des exposants à prendre en compte dans le calcul de la *contention window*. A l’inverse du mécanisme du contrôle du DCF où les valeurs de la fenêtre de contention sont dépendantes du standard utilisé, dans l’EDCA, cette fenêtre est calculée de la manière suivante :

$$aCW_{min} = 2^{ECW_{min}} - 1 \text{ ou } aCW_{max} = 2^{ECW_{max}} - 1$$

- L’ACM sert à indiquer si le contrôle d’admission est d’actualité. En d’autres mots, cela indique si une politique pour administrer et contrôler les ressources de bande passante est activée.
- La dernière valeur TXOP sert à indiquer l’intervalle de temps pendant lequel une station a le droit d’échanger des séquences de trames. Cette dernière n’est pas utilisée dans le cas des *Beacon* et prend beaucoup plus d’importance pour la transmission de la voix et vidéos.

Les deux dernières valeurs sont uniquement mentionnées à titre d’information et n’entre pas dans les calculs qui suivront dans le chapitre 4. Les autres valeurs, annoncées par les trames *Beacon*, seront d’une grande aide dans la section 4.2.1.4.

Dans cette section les trames *Beacon* ont été analysées afin de comprendre les valeurs relatives à la gestion de la qualité de service. Cependant, si nous parlons de la qualité de service délivrée aux trames de données, qu’en est-il des trames *Beacon* ? Est-ce qu’elles disposent d’une QoS ? Une chose est sûre, aucune information

quant à une appartenance à une classe de priorité n'est disponible dans les trames *Beacon*. [25] nous donne un complément d'information sur ce point, précisant que les trames de données vont disposer une qualité de service si celle-ci est activée et utilisée. Cependant, aucune indication spécifique n'indique quelle est cette classe de priorité.

3.4.2 Couche 3

D'après nos recherches, *Voo* ne propose qu'un hotspot ouvert alors que *Proximus* offre un hotspot ouvert, *PROXIMUS_FON*, et un hotspot fermé, *PROXIMUS_auto_FON*.

Dans cette section, nous ne nous intéresserons uniquement qu'à ce qui est observable du côté des hotspots ouverts car les hotspots fermés requièrent une authentification afin de s'y connecter. Sans cette authentification, aucune information ne peut être récupérée depuis la couche 3 et, une fois authentifié, plus aucune restriction quant à la disponibilité des services ne se fait ressentir.

3.4.2.1 Caractéristiques des hotspots ouverts

Nous avons observé que *PROXIMUS_FON* et *Voo* utilisent la même technique pour rediriger leurs utilisateurs pour le hotspot ouvert.

Un utilisateur va se connecter au réseau Wi-Fi via l'intermédiaire d'un point d'accès. Un serveur DHCP lui assigne alors une adresse IP avec un *lease time* (temps de réservation de l'adresse). D'après les observations, il s'avère que ce *lease time* n'est que de 30 secondes sur le hotspot ouvert de *Proximus*. Donc un client va faire une requête DHCP toutes les 15 secondes, créant ainsi beaucoup d'échanges en peu de temps. Ce choix appartient à *Proximus* et nous ne pouvons que nous y soumettre. Cependant, on pourrait améliorer cela en augmentant le temps à 2 minutes par exemple (c'est ce que fait déjà *Voo*).

Dès que l'adresse a été récupérée, le client va essayer d'accéder à une page web mais le point d'accès va rediriger le client vers sa page d'authentification grâce à une réponse *HTTP 302 found* qui est une redirection temporaire (comme démontré dans la figure 3.10).

192.168.182.3	192.168.182.1	DNS	standard query	0xe704	A google.com
192.168.182.1	192.168.182.3	DNS	standard query response	0xe704	
192.168.182.3	194.78.99.217	HTTP	GET /	HTTP/1.1	
194.78.99.217	192.168.182.3	HTTP	HTTP/1.0 302 Moved Temporarily	(text/html)	

FIGURE 3.10 – Redirection vers hotspot ouvert⁴

Une fois que le client se sera authentifié, il pourra surfer normalement sur le réseau. L'authentification sur un hotspot *Proximus* se fait grâce à une requête *http* :

4. L'adresse 194.78.99.217 étant l'adresse récupérée via la requête DNS précédemment effectuée

192.168.182.1:3990/logon?username=xxxxxxxx%40belgacomfon.be_guest&password=xxxxxxx. Dans cette requête, l'identifiant est envoyé en clair et le mot de passe a été crypté. La figure 3.11 montre la réponse à cette authentification. On peut remarquer que l'identifiant (1) est toujours en clair et que l'on peut retrouver aussi l'adresse mac (2), l'adresse id (3) de l'utilisateur et aussi l'adresse url (4) de la page que l'utilisateur souhaite visiter.

FIGURE 3.11 – Réponse à la demande d'authentification de *Proximus*

En plus de l'analyse de l'authentification auprès des portails captifs de *Proximus*, différents tests ont été réalisés afin de vérifier l'exploitation de divers protocoles à partir du hotspot. Nous avons commencé par les requêtes de type DNS, étant les prémisses des requêtes HTTP/HTTPS.

Nous avons remarqué que les requêtes DNS sont traitées normalement et ne renvoient donc pas l'adresse du serveur de l'opérateur (voir figure 3.12). Nous avons regardé pour différents sites internet connus tels que *Facebook*, *Google*, etc.

FIGURE 3.12 – Capture de packets DNS sur le hotspot de *Voo*⁵

Concernant les autres tests réalisés sur *Proximus* seulement, il s'avère que la connexion vers un serveur grâce à FileZilla, faire un ping vers un site connu (google.be par exemple), la connexion en ssh vers un serveur ou encore l'accès à différentes boîtes mail (grâce à Outlook par exemple) ne fonctionnent pas. Nous pouvons donc imaginer que l'opérateur bloque tous les ports dont il n'a pas besoin pour afficher uniquement la page d'authentification. Le tableau 5.7 reprend les différents protocoles testés et s'ils sont exploitables.

Protocole	Ouvert
http	X
https	X
dhcp	✓
dns	✓
ftp	X
icmp	X
ssh	X
imap	X
smtp	X

TABLE 3.2 – Tableau reprenant les différents protocoles testés

Pour évaluer le réseau Wi-Fi sur les hotspots, nous pourrions tenter d'exploiter deux protocoles différents. Avec le DHCP et le DNS, nous allons comparer le temps moyen de réponse des requêtes DHCP et DNS. De plus, nous remarquons que les requêtes DNS ne sont pas bloquées, quelle que soit la demande effectuée. Le but est dès lors d'utiliser un programme qui évalue la bande passante grâce à des requêtes DNS. Cette partie sera plus amplement développée dans la section 4.2.2.

5. Ce comportement est similaire sur les hotspots de *Proximus*

Chapitre 4

Implémentation

Dans ce chapitre, nous expliquons premièrement le fonctionnement général de notre algorithme. Ensuite, nous décrivons comment nous avons exploité les différents points observés dans le chapitre précédent (chapitre 3) et comment nous avons implémenté les filtres qui évaluent ces critères.

En section 4.1, la vue générale de l'algorithme sera détaillée. Le point 3.1.2 avait déjà réalisé une esquisse du fonctionnement de l'algorithme. Ici la description se fera de manière plus complète afin comprendre les différentes étapes d'exécution du programme. Par la suite, il sera expliqué, en section 4.2, le cœur de ce chapitre reprenant le fonctionnement des différentes parties de l'implémentation pour les différentes couches réseaux que l'on désire analyser et comment ces dernières ont été réalisées. La dernière section, quant à elle, aura le rôle de définir l'influence de chaque partie de l'implémentation.

4.1 Vue générale de l'algorithme

L'approche conventionnelle pour déterminer la sélection d'un point d'accès est basée sur la puissance du signal qui est envoyé par ce point d'accès. Il a été démontré que cette méthode de sélection peut mener à de mauvaises performances pour la station vu que la puissance du signal n'est pas influencée par la consommation réelle du réseau, par exemple lorsqu'il y a de la saturation ou que le point d'accès est chargé [32].

Comme précisé dans la section 3.1.1, déterminer le meilleur Wi-Fi signifie réellement de déterminer quel point d'accès, auquel un client peut se connecter, offre les meilleures performances. À partir de ce ou ces points d'accès, différentes actions, séparées en trois parties distinctes, doivent être réalisées par notre algorithme.

Sa conception a été faite en langage C. Ce choix a été réalisé pour deux raisons. La première est que le langage C fait partie intégrante des systèmes de type Unix. La deuxième raison est que, en comparaison avec des langages interprétés comme Python, le langage C offre de meilleurs résultats pour tout ce qui concerne des mesures faites sur les réseaux.

La figure 4.1 ci-dessous montre les différentes parties de l'algorithme qui seront définies dans les trois sections suivantes.

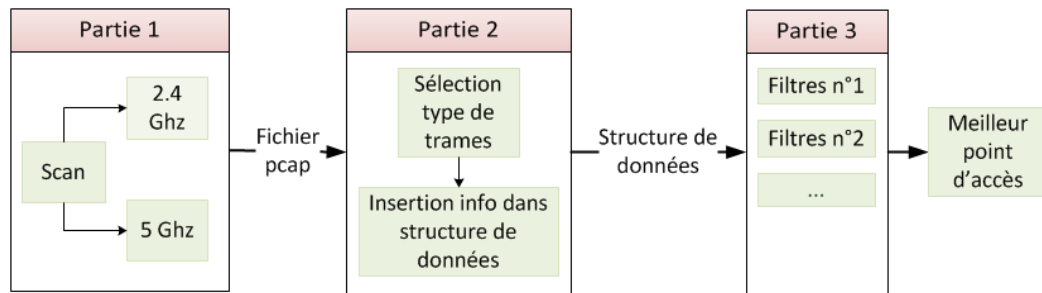


FIGURE 4.1 – Vue générale de l'algorithme

4.1.1 Partie 1 - Scan

La première partie de l'algorithme consistera à récupérer les trames disponibles sur la couche 2. Pour réaliser cette étape, il est nécessaire de disposer d'une carte réseau capable de se mettre en mode *Monitor* comme précisé dans le point 3.2.2.

Il a fallu déterminer le temps d'écoute que l'on va passer sur chaque canal. Nous avons tenté de trouver un temps raisonnable pour une utilisation mobile et qui permet de récolter assez d'informations pour réaliser l'analyse. Pour cela, nous avons réalisé des scans dans un environnement contenant un nombre important de BS-SID différents et de comparer le nombre de trames *Beacon* capturées en fonction du temps d'écoute. Nous avons remarqué que, en nombre de trames reçues par seconde, le scan de 3 secondes reçoit moins de trames que celui de 5 secondes. Par contre, le scan de 10 secondes en reçoit autant que celui de 15 secondes. Nous pouvons donc éliminer ces deux fenêtres de temps extérieures. Ensuite, nous avons choisi de prendre une fenêtre de 5 secondes car nous avons remarqué qu'elle capturait un peu moins de trames mais en restant proche de celle de 10 secondes. Nous avons donc décidé qu'il n'était pas nécessaire d'utiliser une fenêtre plus grande.

Afin de pouvoir réaliser le changement de canal, l'outil *iw* A.1 fut utilisé, permettant entre autres de mettre une carte réseau en mode *Monitor*, d'en modifier la fréquence ou d'en récupérer toutes les informations comme les fréquences disponibles ou vérifier le fait qu'elle supporte le mode *Monitor*. En parallèle à cette écoute, un analyseur de paquet (*TCPdump* A.2) est utilisé afin de capturer les trames circulant

sur le réseau. Le résultat sera un fichier au format PCAP contenant les trames capturées sur chaque canal (soit 13 canaux pour le 2.4 GHz et 19 canaux pour le 5 GHz) pendant une période de 5 secondes.

Ce fichier est ensuite passé à la partie 2 pour être parcouru.

4.1.2 Partie 2 - Analyse des trames

Cette seconde partie se chargera du parcours du fichier de capture. Ce fichier est ouvert en utilisant en parallèle un certain filtre fourni par la librairie *pcap.h* permettant de sélectionner certains types de trames. Ce filtre d'entrée est très bénéfique car il permet de simplifier le travail en ne prenant que les trames qui sont intéressantes pour la réalisation de ce travail. Ces dernières ont été sélectionnées au fur et à mesure de l'avancement de ce mémoire. Ainsi les seules trames que nous décidons d'analyser sont :

- l'entièreté des trames de management
- la plupart des trames de données. Seules les trames s'occupant de la gestion de l'économie d'énergie des stations associées à un point d'accès sont omises (nommées *NULL FUNCTION*).
- seules les trames de contrôle correspondant aux ACK, RTS et CTS.

Même si beaucoup de trames sont autorisées, elles ne seront pas toutes utiles. Cependant, certaines de leurs informations peuvent s'avérer utiles dans la partie 3, comme leur numéro de séquence.

Par la suite, grâce à l'utilisation de la librairie *pcap.h*, chaque trame autorisée peut être analysée individuellement. En fonction de leur type (*Beacon*, association, donnée, ACK, etc), la façon de parcourir le contenu de la trame différera. Cet aspect ne sera pas mentionné dans ce travail car il relève plus d'une façon de programmer. Afin d'aider à comprendre le schéma de pensée de ce travail, voici une explication des structures employées dans notre algorithme :

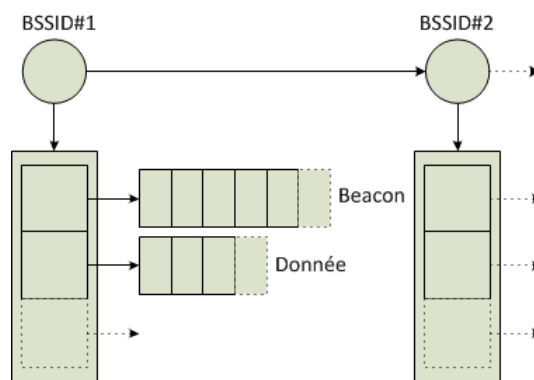


FIGURE 4.2 – Structure de données afin de mémoriser les trames

La structure des données présente en 4.2 est basée sur une structure en liste chaînée dans laquelle chaque nœud représente un BSSID et contient plusieurs structures représentant les trames *Beacon*, les trames de données, les trames de données utilisant de la qualité de services, etc. Ainsi, dès qu'une trame est parcourue, en fonction de son type et de l'adresse du BSSID, elle est placée dans le nœud approprié et dans la structure appropriée. Notons également que les informations retenues pour chaque type de trame ne sont pas identiques et dépendent entièrement de nos besoins. Donc toutes les informations disponibles ne sont pas enregistrées.

Dès que toutes les trames du fichier capturé ont été parcourues et leurs informations récupérées, l'estimation du meilleur Wi-Fi peut se dérouler.

4.1.3 Partie 3 - Filtres

On arrive au cœur de l'algorithme. C'est ce dernier qui contiendra les implémentations requises afin de déterminer quel est le meilleur Wi-Fi comme définit en 3.1.1. Afin de réaliser cette estimation, nous avons décidé d'utiliser ce que nous appelons des *filtres*. Nous les avons nommés de cette manière car nous considérons que le résultat de chaque filtre va venir raffiner le résultat de l'algorithme et, par conséquent, mettre en évidence les points d'accès offrant de meilleures performances par rapport à d'autres.

Chaque filtre évaluera un critère déterminé, en partie, en fonction des observations réalisées dans le chapitre 3. Pour ce faire, chacun d'entre eux devra alors parcourir la structure définie précédemment afin de récolter les informations nécessaires. Leur résultat qui dépendra bien évidemment du critère évalué sera finalement transformé en un indice de priorité. Tous les indices combinés formeront alors la priorité qu'aura un point d'accès. Tout ceci permettant au final de déterminer quel AP est considéré comme le meilleur, disposant de la somme des indices la plus petite. La validation de ces filtres sera réalisée dans le chapitre 5.

Leur développement ainsi que leurs objectifs individuels sont définis dans la section suivante.

4.2 Description de l'implémentation des différents filtres

Cette section reprend le développement et les objectifs de chaque filtre mis en place dans ce travail. On y trouve deux sous-sections s'occupant de compartimenter les filtres en fonction de la couche réseau dans laquelle ils opèrent : en 4.2.1 la couche liaison de données et en 4.2.2 la couche réseau.

4.2.1 Couche 2

Dans cette section, nous allons décrire les différents filtres qui ont été créés pour la couche 2. Le tableau 4.1 reprend l'ensemble des filtres développés dans cette section.

Nom du filtre	But recherché
Puissance du signal reçu	Supprimer les trames reçues avec un signal trop faible.
Nombre de <i>Beacon</i> capturées	Supprimer les points d'accès où trop peu de trames <i>Beacon</i> ont été reçues.
Pertes des trames <i>Beacon</i>	Déterminer les pertes des trames <i>Beacon</i> dans une fenêtre de temps.
Consommation de l' <i>airtime</i>	Calculer le temps consommé par les <i>Beacon</i> et les trames de données.
Interférences des canaux adjacents	Déterminer l'impact des canaux environnant sur le canal d'un point d'accès.
Délai des trames <i>Beacon</i>	Calculer le délai entre l'envoi des <i>Beacon</i> .
Estimation de la bande passante	Estimer la bande passante potentielle de manière non intrusive.
Retransmissions des données	Calculer le pourcentage de retransmissions des trames de données.
Stations connectées	Compter le nombre de stations connectées sur l'AP
RTS/CTS	Filtre envisagé mais non implémenté.

TABLE 4.1 – Tableau reprenant les filtres développés et envisagés

4.2.1.1 Puissance du signal reçu

Ce filtre est le premier à intervenir et se réalise lors de l'enregistrement des informations en provenance des trames. Il nous permet de mettre un seuil minimal quant à la puissance du signal reçu. Il va éliminer les BSSID qui seront en dessous de ce seuil. Ce seuil a été mis en place pour la raison suivante : d'après nos observations, il ressort qu'à partir d'une certaine puissance de signal reçu, le nombre de trames *Beacon* capturées diminue fortement démontrant la faible qualité du signal. Ceci causant un problème dans la suite des calculs réalisés. En effet, si une trame *Beacon* est envoyée suivant un intervalle de 102.4ms et que seulement 3 trames sont reçues durant une période de 5 secondes, cela déteint fortement sur la suite du travail.

Il s'avère alors que, selon [18], un seuil minimal de sensibilité doit être respecté pour les différentes techniques de modulation. Pour le *High Rate Direct Sequence Spread Spectrum (HR/DSSS)*, le seuil minimal est de -76dBm et pour l'OFDM, ce même seuil prend la valeur de -74dBm. Or, comme constaté en section 3.4.1.2, le

signal reçu dépend exclusivement de la carte réseau utilisée. Donc ces seuils théoriques ne peuvent être pris au pied de la lettre et appliqués sans vérification.

Afin de déterminer quelle devrait être la sensibilité minimale, nous sommes partis d'une capture de trames prise dans un milieu hétérogène où un grand nombre de réseaux Wi-Fi existent, ceci afin de déterminer à partir de quel signal la réception était altérée. Le tableau 4.2 nous montre le résultat de cette capture en comparant le signal reçu avec le nombre de trames *Beacon* capturées.

Puissance du signal	Nbr. de <i>Beacon</i>
-73 <i>dBm</i>	7
-69 <i>dBm</i>	11
-81 <i>dBm</i>	3
-78 <i>dBm</i>	4
-84 <i>dBm</i>	1

TABLE 4.2 – Comparaison entre la puissance du signal reçu et le nombre de trames *Beacon* capturées¹

Le résultat étant fort aléatoire, nous avons décidé de mettre la valeur à -80dBm (cette valeur étant uniquement valable pour l'antenne Wi-Fi utilisée dans ce travail). Cette valeur étant très subjective et sujette à discussion, nous avons été forcés d'établir un nouveau filtre. En effet, après mise en place de ce filtre, il s'avère que les captures arrivaient encore à déceler des BSSID pour lesquelles uniquement 1 ou 2 trames *Beacon* étaient capturées endéans une période de scan de 5 secondes. Réduire la valeur de -80dBm à -75dBm (se rapprochant ainsi de la valeur théorique) fut aussi pris en compte mais avec pour résultat de supprimer certains BSSID offrant un nombre suffisant de trames *Beacon*.

Le filtre additionnel est décrit ci-dessous.

4.2.1.2 Pourcentage de *Beacon* capturées

Ce filtre vient en supplément par rapport au filtre sur le signal reçu. Il intervient une fois que toutes les trames ont été parcourues car il prend en compte le nombre de trames *Beacon* capturées, valeur qu'on ne sait uniquement vérifier une fois que toutes les trames ont été analysées.

Son rôle est de mettre en place un seuil minimal pour le nombre de trames *Beacon* reçues provenant d'un BSSID. Cela nous permet de pouvoir éliminer les BSSID qui n'envoient pas assez de trames *Beacon* car, si on a un faible taux de réception, on

1. Toutes les données n'ont pas été reprises afin de ne montrer uniquement que la partie où le nombre de trames *Beacon* capturées est fortement faible.

peut supposer que ce sera le cas pour tous les types de trames et, par conséquent, les performances de ce réseau ne seront pas optimales.

Afin de réaliser ce filtre, nous avons simplement compté le nombre de trames *Beacon* reçues sur toute la fenêtre du temps de sniffing. Étant donné que les trames *Beacon* ont été capturées, on connaît désormais l'intervalle des trames *Beacon*. Avec ce nombre, on peut supposer le nombre de *Beacon* qui auraient dû être capturées et calculer ainsi le pourcentage de trames capturées par rapport à la valeur théorique maximale. Par exemple, si l'intervalle se trouve être de 102.4 ms, alors cela correspond à un nombre de ± 10 trames *Beacon* reçues par seconde.

La valeur du pourcentage pour ce filtre se trouve être de l'ordre de 10%. Cela correspond à un ratio de 1 trame *Beacon* par seconde dans le cas d'un scan d'une durée de 5 secondes (comme défini dans la section 4.1.1) avec un intervalle entre les *Beacon* de l'ordre de 102.4 ms. Cette valeur de 10% peut sembler faible mais elle a été choisie car elle permet de facilement supprimer les BSSID indésirables qui seraient passés au travers du filtre relatif à la puissance du signal.

Tout comme le filtre précédent, celui-ci va permettre de raffiner les données à parcourir et donc de limiter le nombre de BSSID à analyser et supprimant directement ceux qui sont considérés comme médiocre ou trop éloigné.

4.2.1.3 Perte des trames *Beacon*

Avec ce nouveau filtre, on souhaite connaître le pourcentage de perte des trames *Beacon* afin de pouvoir donner une plus grande importance aux BSSID avec un faible taux de perte. Pourquoi uniquement ce pourcentage de perte et pourquoi ne pas le généraliser à d'autres types de trames ?

La réponse est simple : les trames *Beacon* sont les seules trames dont la présence est assurée et régulière sur la couche 2 (toutes les 102.4ms dans un environnement idéal). Les autres trames telles que les trames de données dépendent uniquement du trafic présent.

Les trames peuvent être perdues pour différentes raisons [29, 64, 65] : les interférences avec d'autres points d'accès et bruits environnant, les collisions de trames, les variations dans la puissance du signal, la congestion d'un réseau sans fil, ..., peuvent toutes venir altérer la réception ou transmission des trames et par conséquent augmenter le pourcentage de pertes.

Comme précisé dans [55] et [64], il est possible de détecter les pertes en s'intéressant au décalage dans les numéros de séquence présents dans les trames. La difficulté dans cet exercice, relevée dans nos observations en section 3.4.1.2, est que ce numéro de séquence n'est pas uniquement lié aux trames *Beacon* mais peut aussi être

utilisé par d'autres trames de management ou encore de données. Il faut savoir que ce numéro est lié au point d'accès et si il y a, par exemple, 3 VAP (avec 3 BSSID différents) sur cet AP, ils utiliseront le même compteur pour les numéros de séquences et ainsi incrémenterons ce compteur chacun à leur tour.

Afin de calculer ce pourcentage de perte, nous avons décidé de focaliser nos calculs sur la période de temps entre le moment où la première trame *Beacon* a été reçue et le moment où la dernière trame *Beacon* a été reçue durant la fenêtre de capture. Ceci afin de se concentrer uniquement sur les trames et non le temps de la capture. Dans l'implémentation, il y a deux parties distinctes : la première consiste à déterminer quel sera l'incrément entre les numéros de séquence pour un BSSID dans le cas où plusieurs VAP sont émis par un point d'accès et la deuxième partie aura comme but final de déterminer la perte.

Ce filtre peut sembler similaire au précédent car il s'occupe également de calculer un pourcentage pour les trames *Beacon*. Précisons simplement que le premier a été mis en place afin de raffiner les données à analyser, tandis que ce dernier va plus loin dans l'analyse des trames *Beacon* en s'assurant que le calcul des pertes ne dépende pas d'une simple comparaison avec une valeur théorique mais plutôt d'une vérification précise basée sur le numéro de séquence présent dans chacune des trames.

Déterminer l'incrément pour le numéro de séquence

Si n VAP sont présents sur un point d'accès, ils utiliseront le même compteur pour leur numéro de séquence. Afin de déterminer le pourcentage de perte des trames *Beacon* par BSSID, il faut déterminer combien de VAP sont présents sur un AP afin de connaître l'incrément qui doit être appliqué sur chaque trame *Beacon* en provenance d'un BSSID spécifique. Cet incrément sera utile afin de savoir si une trame *Beacon* est perdue ou non. La figure 4.3 décrit le rôle de cet incrément. On remarque clairement qu'un point d'accès *Proximus* possède 3 VAP (et donc 3 BSSID). Tous utilisent le même compteur pour leur numéro de séquence. Cependant si on s'intéresse uniquement à un seul BSSID, il s'avère que le numéro de séquence est incrémenté de 3 à chaque transmission de trames *Beacon* (par exemple les numéros de séquence 4002 et 4005 correspondent à deux trames successives non perdues).

```

9e:97:26:12:63:ff Beacon frame, SN=4001, FN=0, SSID=PROXIMUS_FON
9e:97:26:12:63:f1 Beacon frame, SN=4002, FN=0, SSID=PROXIMUS_AUTO_FON
Technico_12:63:fe Beacon frame, SN=4003, FN=0, SSID=wifi-2.4-63FE
9e:97:26:12:63:ff Beacon frame, SN=4004, FN=0, SSID=PROXIMUS_FON
9e:97:26:12:63:f1 Beacon frame, SN=4005, FN=0, SSID=PROXIMUS_AUTO_FON
Technico_12:63:fe Beacon frame, SN=4006, FN=0, SSID=wifi-2.4-63FE

```

FIGURE 4.3 – Incrément de 3 pour le numéro de séquence d'un BSSID

Avant de plonger dans cet algorithme, il faut avoir quelques informations complémentaires. En section 4.1.2, nous présentons la structure que nous avons utilisée

afin d'enregistrer les informations jugées utiles à propos des trames.

Cependant, en plus de cette structure, nous avons dû créer une structure secondaire pour ce filtre. Cette dernière (fig. 4.4) enregistre pour chaque point d'accès toutes les informations essentielles des trames capturées (principalement les trames de données, management, etc). Cette structure existe pour deux raisons :

1. Le fait qu'on ait besoin de connaître le numéro de séquence des autres trames capturées afin de voir si le saut de numéro de séquence est dû à une perte ou à l'envoi d'autres trames utilisant le même compteur.
2. Cette nouvelle structure va également mémoriser dans chaque nœud tous les BSSID dérivés du point d'accès étudié dans ce nœud.

Toute la phase d'insertion se déroule dans la "Partie 2 - Analyse des trames" (4.1.2).

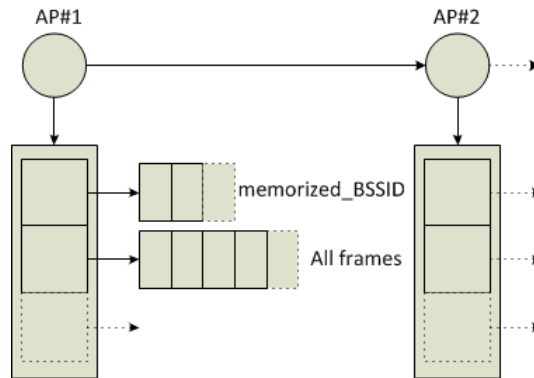


FIGURE 4.4 – Structure secondaire

Algorithme 4.1 Calcul de l'incrément du numéro de séquence des trames *Beacon*

```

1: for each node_frames in all_frame do
2:   for each memorized_BSSID in node_frames do
3:     for each node in list_BSSID do
4:       if node.BSSID[0].mac equals
5:         memorized_BSSID.mac then
6:         inc ← memorized_BSSID.nbr_BSSID
7:         break
8:       end if
9:     end for
10:  end for
11: end for

```

Afin de réaliser ceci, le pseudo code 4.1 a été implémenté. Dans ce dernier, la structure secondaire répondra au nom de "all_frame" et la structure principale au nom de "list_BSSID". Dans cet algorithme, on regarde si les BSSID enregistrés dans la structure principale ont une correspondance avec les BSSID enregistrés dans la liste *memorized_BSSID* définie dans la structure secondaire. Si c'est le cas alors l'incrément entre les numéros de séquence des trames *Beacon* sera égale au nombre de

BSSID présents dans la structure secondaire (égale au nombre de VAP diffusés par un point d'accès).

Calcul de la perte

Une fois l'incrément déterminé, il ne reste plus qu'à rechercher le pourcentage de perte. Pour cela, un pseudo code est proposé mais, vu sa longueur, nous vous invitons à consulter l'annexe D qui en contient l'intégralité. Celui-ci se réalise comme suit : pour chaque trame *Beacon*, on regarde si la prochaine trame mémorisée respecte bien l'incrément (*inc*) calculé précédemment.

1. Si tel est le cas, alors il n'y a pas de perte.
2. Si ce n'est pas le cas, on va parcourir les trames enregistrées dans la structure secondaire. Pour ce faire, on recherche la prochaine trame *Beacon* dont le numéro de séquence est plus grand que celui de la trame actuelle et dont le *timestamp* est plus récent (ce dernier point est important car il est possible de rencontrer plusieurs fois le même numéro de séquence au vu de la limite est fixée à 4095). Pour chaque autre type de trame rencontré, on incrémente un compteur.
3. La trame *Beacon* suivante trouvée, on dispose maintenant du nombre de trames entre ces deux *Beacon*. Cependant ce nombre n'est souvent pas correct car entre les deux *Beacon*, un certain nombre de cas particuliers existent :
 - Certaines trames ne faisant pas partie du même BSSID ne possèdent pas le même compteur pour le numéro de séquence.
 - Si beaucoup de trames de données circulent, même si leur numéro de séquence se situe après celui d'une trame *Beacon*, elles peuvent apparaître avant la trame *Beacon* dans la capture.
 - On peut rencontrer des pertes (nommées "fausses" pertes) qui ne font pas partie des trames *Beacon* car le moment supposé de la perte n'est pas possible par rapport à l'intervalle de génération des trames *Beacon*.
 - etc.

Tous ces cas particuliers pris en compte, il faut alors revoir à la baisse le nombre de trames entre les deux trames *Beacon*.

4. Le nombre de pertes se calcule comme étant la différence entre le numéro de séquence de la trame *Beacon* rencontrée par rapport à la trame *Beacon* initiale. On y soustrait également le nombre de trames rencontrées et les "fausses" pertes. Si le résultat est différent de l'addition entre le numéro de séquence initiale et l'incrément, alors il s'agit d'une perte.
5. Finalement, pour calculer le pourcentage, il faut d'abord connaître le nombre total de trames qui auraient dû être reçues. Pour cela, on additionne le nombre de trames réellement reçues avec les nombre de trames perdues. Une simple

division entre le nombre de trames perdues et le nombre total de trames permet de donner le pourcentage de perte.

4.2.1.4 Consommation de l'*airtime*

L'*airtime* peut se définir comme étant le temps utilisé pour envoyer des données sur le média sans fil. De manière plus précise, il s'agit du temps que les données à transmettre passent dans la file de transmission ainsi que le temps de leur transmission entre un point d'accès et une station (ou inversement). L'estimation de l'*airtime* peut se faire sur tout type de données étant transmises sur le réseau sans fil. La décision de créer ce type de filtre nous vient directement de [39] et de [55], l'un comme l'autre s'intéressent à la consommation des données dans un réseau sans fil. Dans ce travail, nous nous concentrons uniquement sur les trames *Beacon* et les trames de données car elles représentent le plus gros pourcentage d'*airtime* consommé [55].

S'intéresser au pourcentage d'*airtime* consommé par les trames *Beacon* est assez important. Bien qu'elles soient d'une taille faible (environ 200 octets), elles sont envoyées assez régulièrement (comme définit dans la partie "Intervalle des *Beacon*" du point 3.4.1.2). L'intervalle des trames *Beacon* est défini par défaut à une valeur de 102.4 ms, équivalent à un envoi de ± 10 trames par seconde. Chacune de ces trames consomme un pourcentage de l'*airtime* disponible et ce par point d'accès virtuel. Ainsi plus le nombre de BSSID présents sur un canal est important, plus la consommation de l'*airtime* va se faire ressentir sur les autres types de données à transmettre. De plus, il faut également prendre le compte que cette variation dépend du standard de communication utilisé pour diffuser les trames *Beacon*, comme observé dans le point 3.4.1.2 : si le 802.11b est utilisé, l'antenne transmettra par défaut au taux le plus faible, 1 Mbit/s, et a pour conséquence que le temps de transmission sera plus long comparé aux standards 802.11g et 802.11a (qui ont un taux de transmission de 6Mbit/s).

En plus des trames *Beacon*, on décide de s'intéresser aussi à l'*airtime* consommé par les trames de données. Bien qu'elles ne soient pas constamment présentes sur un réseau, l'*airtime* qu'elles consomment peut permettre d'indiquer le pourcentage du temps de transmission qui est occupé par ces trames et ainsi montrer qu'un point d'accès est plus enclin à recevoir des données qu'un autre. Plus les données consomment d'*airtime* sur un VAP, moins d'*airtime* sera à la disposition des autres stations souhaitant s'y associer.

Ce filtre est donc divisé en deux parties distinctes : les *Beacon* et les données.

- Le but du premier est de trouver le canal qui montre le plus faible pourcentage d'utilisation d'*airtime*. Ce filtre prend également en compte que plus le nombre de points d'accès augmente, plus l'*airtime* augmentera. C'est pour

cette raison qu'il est souvent conseillé de limiter le nombre de SSID diffusés sur un point d'accès et, de manière plus générale, sur un canal [31].

- Le second aura comme finalité de montrer le pourcentage d'*airtime* consommé par chaque VAP au niveau des trames de données qui circulent par ce dernier.

Afin de réaliser le calcul de l'*airtime*, plusieurs étapes sont à respecter : il faut tout d'abord calculer le temps pris pour transmettre le *preamble* de la trame correspondant à la sous-couche physique du PLCP et ensuite calculer le temps mis pour envoyer les données au niveau du PMD (2.3.3).

Calcul de l'*airtime* de la couche Physique

La raison de ce calcul est que pour envoyer une trame, quel que soit son type, il faut passer par la couche physique afin de préparer les données pour la transmission. Cette partie de l'implémentation est identique pour les deux filtres (*Beacon* et données).

Dans cette partie, on s'intéresse au temps requis pour la transmission du *preamble* et de son en-tête sur un réseau sans fil. Différentes valeurs doivent être connues pour pouvoir réaliser cette opération :

- Le temps requis pour l'envoi du *preamble* et son en-tête.
- Le temps *Inter Frame* du SIFS (2.3.5.1)
- Le temps correspondant au *slot time*. Cette variable est définie dans le point 2.3.5.2 sous la dénomination *aSlotTime*.

Ces valeurs sont toutes liées au standard utilisé pour la transmission.

Elle sont toutes récupérables depuis [25] mais par facilité elles peuvent être récupérées directement dans l'annexe C.

Concernant le calcul proprement dit, il faut également prendre en compte la notion de priorité. Comme nous pouvons voir dans le point 3.4.1.2, les *Beacon*, tout comme les trames de données, peuvent posséder une priorité afin d'obtenir une qualité de service. Cependant, il n'est pas possible de déterminer si les trames *Beacon* sont sous l'emprise d'une priorité à partir des trames capturées. Nous supposons donc que si un point d'accès mentionne dans ses trames *Beacon* qu'il peut faire de la QoS, alors les trames *Beacon* ont une priorité. Nous avons dû faire de même pour la classe de priorité : elle a été mise au *Best Effort* (AC_BE).

Si une priorité existe, elle vient alors modifier les valeurs de temps de certaines variables requises pour le calcul. Les valeurs de ces variables se trouvent également altérées par le standard en vigueur (il peut ne pas être le même pour les trames *Beacon* et les trames de données). Ces deux variables sont les suivantes :

- Le temps pour la *contention window*. Cette valeur représente le temps additionnel (*backoff time*) qu'il faudra attendre après un IFS. Ce temps peut ne pas être présent dans la réalité, comme mentionné en section 2.3.5.2, mais est pris en compte dans ce calcul car on suppose qu'un réseau aura d'autres transmissions en cours et que le média ne pourra pas être directement accessible. Sans connaître la valeur exacte de ce temps, nous considérons la valeur minimale possible de la fenêtre de contention dans nos calculs, nommée aCW_{min} .
- Le temps *Inter Frame* requis en fonction du standard et/ou de l'utilisation de la qualité de service.

Si la priorité n'est pas présente, on peut évaluer ces valeurs comme suit :

$$contention_window = aCW_{min} * slot_time \quad (4.1)$$

$$IFS = SIFS + 2 * slot_time \quad (4.2)$$

Pour rappel, les informations sur la fenêtre de contention (aCW_{min} et $slot_time$) sont disponibles en section 2.3.5.2. L'IFS, quant à lui, correspond alors au temps DIFS qui est le temps d'attente par défaut lors d'une utilisation du mécanisme DCF.

Dans le cas où la qualité de service est présente, ces mêmes données sont calculées de manières différentes :

$$contention_window = (2^{ECW_{min}} - 1) * slot_time \quad (4.3)$$

$$IFS = SIFS + (AIFSN * slot_time) \quad (4.4)$$

Les informations quant à ces nouvelles variables ont déjà été observées et analysées dans la partie "Gestion de la qualité de service" de la section 3.4.1.2.

Par la suite, il ne reste que la détermination du temps requis pour le *preamble* et son en-tête. Ces derniers sont des temps constants définis par les techniques de modulation (HR/DSSS ou OFDM) décrites en 2.3.3. Comme les autres valeurs, décrites plus haut, elles sont disponibles dans [25] mais aussi dans l'annexe C par soucis de facilité.

Au final, on obtient la formule suivante, en additionnant toutes les valeurs calculées ci-dessus :

$$\begin{aligned} airtime_PHY = IFS + preamble_time \\ + header_preamble_time + contention_window \end{aligned} \quad (4.5)$$

Calcul du temps pour envoyer les données

Cette deuxième partie s'intéresse au temps nécessaire afin d'envoyer les données

sur le média. Ce temps dépend grandement de la taille des données à envoyer, ainsi que du taux de transmission utilisé. Ici encore, l'implémentation est identique pour les deux filtres (*Beacon* et données).

Avant d'entamer ce calcul, il faut savoir que les données qui doivent être envoyées sont transmises en terme de symboles par seconde et non en bits par seconde. Ces symboles dépendent entièrement des techniques d'encodage utilisées. Ainsi avant tout envoi de données, les bits sont encodés en symboles. Le nombre de bits par symbole dépend de la technique d'encodage. En fonction de la technique utilisée et du taux de transmission (et par conséquent du standard utilisé), un symbole peut correspondre à un nombre varié de bits (2, 4, 48 ou encore 216 bits) [18, 25]. Par exemple, un symbole encodé en 802.11b avec un taux de 1Mbit/s correspondra à 1 bit, alors que ce même symbole encodé en 802.11g avec un taux de 54 Mbit/s correspondra à 192 bits.

Ce concept étant assimilé, il faut aussi connaître le nombre de symboles à envoyer par seconde. Cette valeur dépendra de la technique de modulation. Par exemple, si l'OFDM est utilisé, chaque symbole nécessitera une durée de 4 μ s peu importe le taux de transmission, ce qui correspond à 250.000 symboles par seconde.

Passons maintenant au calcul proprement dit. Les équations ci-dessous montrent le calcul réalisé pour le temps d'envoi des données :

$$time_data = \frac{frame_size * 8}{data_bit_symbol} * time_send_symbol \quad (4.6)$$

$$time_send_symbol = \frac{1000000}{symbol_per_sec} \quad (4.7)$$

Le *time_data* (eq. 4.6) donne le temps qui est utilisé afin d'envoyer une trame de taille *frame_size* en fonction du nombre de bits par symbole et du temps requis par symbole (en μ). Sachant que le nombre de bits par symbole change également en fonction du taux de transmission, il est facile de le retrouver en regardant le taux affiché par le *Radiotap Header* de la trame capturée (voir annexe C). Le *time_send_symbol* (eq. 4.7) quant à lui correspond à la durée par symbole en microseconde : connaissant le nombre de symboles par seconde pour les différentes techniques de modulation (et donc pour les différents standards 802.11), il s'agit simplement d'une division de 1.000.000 (étant la valeur en microseconde d'une seconde) par le nombre de symboles par seconde.

Pourcentage total d'*airtime* utilisé

On peut maintenant trouver le pourcentage d'*airtime* utilisé pour un BSSID sur un canal. Il faut additionner le temps de transmission de l'*airtime_PHY* avec celui requis pour l'envoi des données (*time_data*) et diviser par 1000 afin de trouver un temps en millisecondes. A partir de ce point, la suite du calcul diffère un peu en

fonction du type de trames (*Beacon* ou données).

Si ce sont des trames *Beacon*, on sait que ces dernières sont envoyées selon un intervalle déterminé. On a donc besoin de connaître le nombre de trames *Beacon* par seconde afin de savoir quel sera l'*airtime* consommé par seconde pour un BSSID.

$$airtime_Beacon = \frac{airtime_PHY + time_data}{1000} * \frac{1}{beacon_interval} \quad (4.8)$$

Pour obtenir le pourcentage total d'utilisation, il faut additionner les temps *airtime_Beacon* de chaque BSSID sur un même canal.

S'il s'agit de trames de données, on a besoin de calculer l'*airtime* consommé par chaque donnée transmise via un point d'accès ou VAP durant la période d'écoute du média.

$$airtime_data = \frac{airtime_PHY + time_data}{1000} \quad (4.9)$$

Pour connaître le pourcentage total d'utilisation, il est nécessaire d'additionner les temps *airtime_data* requis pour chaque donnée appartenant à un BSSID.

Afin de réaliser ceci pour tous les BSSID enregistrés dans la structure définie en 4.1.2, les pseudo-codes 4.2 et 4.3 ont été mis en place pour, respectivement, les *Beacon* et les données.

Le premier insérera (ligne 1) dans une nouvelle liste tous les BSSID qui se trouvent sur un même canal. Ainsi, chaque nœud de cette nouvelle structure contiendra tous les BSSID d'un seul et même canal. Pour chaque BSSID présent dans ces nœuds, on calcule l'*airtime* consommé par les trames *Beacon* (eq. 4.8) pour finalement connaître l'*airtime* total consommé par ces trames sur un canal spécifique (ligne 9).

Algorithme 4.2 Calcul de l'*airtime* pour les trames Beacon

```

1: new_list ← Set together all BSSID belonging to the same channel
2: for each node in new_list do
3:   for each BSSID in node do
4:     Compute airtime_PHY
5:     Compute time_data
6:     Compute airtime_Beacon
7:     total_airtime += airtime_Beacon
8:   end for
9:   percentage_airtime = total_airtime * 100
10:  print "Total percentage for channel" node.channel_nbr ":" percentage_airtime
11: end for

```

Le deuxième pseudo-code n'aura pas besoin d'une nouvelle structure de données. Toutes les données nécessaires ont été enregistrées au préalable dans la structure principale du 4.1.2. Pour chaque trame de données, on calcule l'*airtime* consommé

(eq. 4.9) pour finalement connaître l'*airtime* total consommé par ces données sur un BSSID pendant la fenêtre d'écoute (ligne 8).

Algorithme 4.3 Calcul de l'*airtime* pour les trames de données

```

1: for each node in list_BSSID do
2:   for each data in node do
3:     Compute airtime_PHY
4:     Compute time_data
5:     Compute airtime_data
6:     total_airtime += airtime_data
7:   end for
8:   percentage_airtime = (total_airtime TIME_SCAN) * 100
9:   print "Total percentage for channel" node.channel_nbr "   :"  
      percentage_airtime
10: end for
  
```

Les résultats de ses deux algorithmes seront, par la suite, utilisés afin de définir un indice de priorité comme expliqué dans le point 4.1.3.

4.2.1.5 Délai des trames *Beacon*

Ce filtre a pour but de déterminer la médiane du délai que les trames *Beacon* subissent avant d'être transmises sur le média sans fil. Mais que représente ce délai ? L'idée de ce filtre provient de [64] qui utilise cette notion afin de calculer la bande passante potentielle délivrée par un point d'accès (voir section 4.2.1.6). La notion de délai des trames *Beacon* peut se définir comme étant le temps que les trames *Beacon* vont passer dans la file d'un point d'accès avant d'être envoyées.

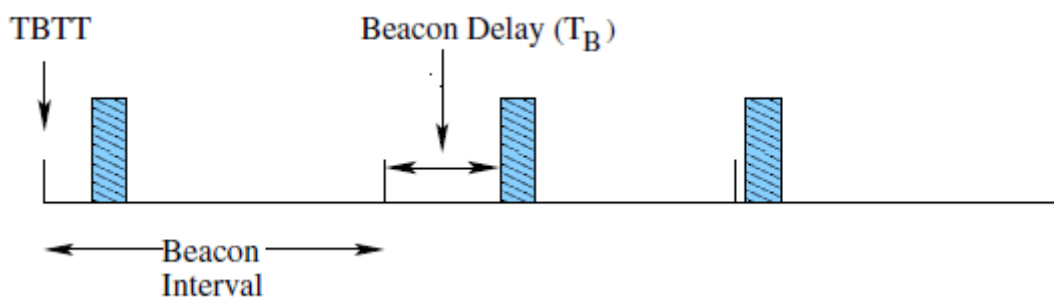


FIGURE 4.5 – Transmission de *Beacon* d'un point d'accès [64]

La figure 4.5 permet d'illustrer comment la transmission des trames *Beacon* est assurée par l'AP. On y retrouve le TBTT, *Beacon interval* et *Beacon delay*. Pour rappel du point 3.4.1.2, le TBTT correspond au moment (d'un point de vue AP) où les trames *Beacon* sont censées être transmises, à savoir le moment où elles sont mises dans la file.

Connaissant le TBTT ainsi que l'intervalle des *Beacon*, une station peut connaître avec précision le moment où le point d'accès prévoit d'envoyer des trames *Beacon*. Une fois que la trame est préparée pour l'envoi, elle est transmise selon les règles de transmission du point d'accès (2.3.5.2 ou 2.3.5.3). La différence entre le moment de transmission et le TBTT conduit à l'estimation du délai des *Beacon*, T_B , qui correspond au temps qu'une trame *Beacon* passe à attendre avant d'être transmise. Si on considère que les trames ne sont pas prioritaires par rapport aux autres, T_B donne une estimation du temps total passé dans la file.

Afin d'obtenir ces deux temps que sont le TBTT et le délai, il faut récupérer le *timestamp* contenu dans la trame *Beacon*. Vu que le temps est propre à l'AP et que l'intervalle entre la génération de chaque *Beacon* est connu, nous pouvons calculer le TBTT (4.12) et le délai (4.13) comme suit :

$$ap_time_send = timestamp/1000 \quad (4.10)$$

$$nbr_beacon_time0 = E(ap_time_send/interval) \quad (4.11)$$

$$TBTT = nbr_beacon_time0 * interval \quad (4.12)$$

$$T_B = ap_time_send - TBTT \quad (4.13)$$

où (4.10) est le temps (en ms) du point d'accès auquel la trame a été envoyée et (4.11) donne une valeur (arrondie vers le bas) permettant de connaître le nombre de trames *Beacon* envoyées depuis que le *Time 0*.

Le résultat de ce filtre est de donner pour chaque BSSID, la médiane du délai afin de pouvoir la comparer avec celle calculées pour les autres BSSID capturés et ainsi voir, en fonction du trafic, l'évolution de ce délai.

4.2.1.6 Estimation de la bande passante

Pour réaliser ce filtre, nous nous sommes basés principalement sur l'article "Facilitating Access Point Selection in IEEE 802.11 Wireless Networks" [64].

Le but est de déterminer la bande passante qu'une station est susceptible de recevoir si elle était connectée à un point d'accès donné, la bande passante potentielle. Il ne faut pas la confondre avec la bande passante disponible.

Afin de respecter la contrainte définie dans 3.1.1, l'algorithme a besoin d'être non-intrusif, il ne faut donc pas qu'il crée de trafic supplémentaire pour le point d'accès ou pour les autres utilisateurs. Il faut aussi que l'algorithme n'ait pas besoin d'appliquer des changements sur le point d'accès et que l'approche d'estimation de la bande passante se base sur des mesures passives des temps d'envoi des trames *Beacon* d'un AP. Dans cette optique, à l'instar de l'approche utilisée, une autre méthode existe. On retrouve en [32] une approche semblable à celle décrite ici mais

plus orientée sur les probabilités de collisions afin de déterminer le niveau de saturation.

Dès lors comment une station peut-elle estimer la bande passante potentielle entre l'AP et elle-même ?

Pour estimer la bande passante descendante du point d'accès vers la station, la technique proposée utilise le délai des trames *Beacon*, envoyées périodiquement par le point d'accès. Ce délai a déjà fait partie d'une explication approfondie dans la section précédente (4.2.1.5). Les équations proposées seront réutilisées dans ce filtre.

Afin de réaliser cette estimation, il est nécessaire de trouver le délai total requis par une trame de donnée. Ce délai en l'absence de *RTS/CTS* est donné par : les temps de contention et de transmission d'une trame de données et des délais respectifs de *Ack*.

$$T = T_D + T_A \quad (4.14)$$

où T_D peut être estimé à partir du délai de *Beacon*, T_B , et du temps de transmission de la trame :

$$T_D = T_B + \frac{DATA}{R} \quad (4.15)$$

avec $DATA$, étant la taille moyenne des données et R , le taux d'envoi. Après avoir reçu la trame, le receveur envoie une trame de *Ack* après un délai *SIFS* (délai est lié au standard utilisé [25]). Ces trames de contrôle ont une taille fixe et un taux de transmission qui leur est propre. On peut dès lors facilement calculer T_A :

$$T_A = SIFS + \frac{ACK}{R} \quad (4.16)$$

La bande passante potentielle, B , du point d'accès vers la station est ainsi donnée par :

$$B = \frac{DATA}{T} \quad (4.17)$$

En présence de l'échange de *RTS/CTS*, chaque transmission de trames de données prend un temps, T , qui est la somme des temps de transmission pris par les trames *RTS*, *CTS*, de données et de *Ack* :

$$T = T_R + T_C + T_D + T_A \quad (4.18)$$

Vu que les règles de transmission sont les mêmes pour les *Beacon* que pour les *RTS*, on peut estimer le temps de transmission d'une trame *RTS* (toutes les trames de contrôle sont envoyées au taux de base, R_b) :

$$T_R = T_B + \frac{RTS}{R_b} \quad (4.19)$$

Une fois que la trame RTS est reçue, la station va attendre un temps $SIFS$ et puis transmettre une trame CTS , on aura donc un délai :

$$T_C = SIFS + \frac{CTS}{R_b} \quad (4.20)$$

Et ensuite, le délai créé par la transmission de la trame de données est égal à :

$$T_D = SIFS + \frac{DATA}{R} \quad (4.21)$$

Le calcul de T_A est le même que l'équation 4.16 et la bande passante potentielle est obtenue de la même manière que l'équation 4.17.

Notons que, dans le cadre de notre algorithme, la récupération des taux de transmission et des tailles des trames (de management, de données et de contrôle) peuvent se faire facilement en parcourant la structure définie en 4.1.2.

4.2.1.7 Estimation de l'influence des interférences des canaux adjacents

Grâce à ce filtre, nous privilégions les canaux qui ont le risque le plus faible de subir des interférences créées par les points d'accès sur des canaux adjacents. Pour réaliser ce filtre, nous nous sommes basés sur l'article "*Effect of adjacent-channel interference in IEEE 802.11 WLANs*" [65].

Il existe deux types d'interférences : un premier type, *co-channel interference*, qui est causé par des transmissions non voulues sur le même canal de fréquence et le second type, l'interférence de canaux adjacents, qui est produite par des transmissions sur un canal adjacent (qui chevauche le canal de la station). La présence d'interférences sur les canaux adjacents réduit le rapport signal bruit et, le nombre d'erreurs de réception est augmenté.

Les réseaux Wi-Fi qui opèrent sur la bande de fréquence des 2.4 GHz (802.11b/g) doivent implémenter des techniques de propagation de spectre et limiter la puissance de transmission pour minimiser l'impact des interférences sur les autres stations. Comme on peut le voir dans la section 2.3.2.1, il n'y a qu'un maximum de 3 canaux qui ne se chevauchent pas en 2.4 GHz.

Dans l'article, les auteurs ont mesuré l'impact des interférences des canaux adjacents. Pour réaliser cela, ils ont mis une station (A) ayant un débit constant en présence d'une autre station (B) qui fait évoluer son débit. En plus de cela, la station A reste sur le même canal tandis que la station B change. Ce qui a été observé, en figure 4.6, est que l'impact des interférences provenant des canaux d'une distance de 1 ou de 2 sera plus importante que l'impact des interférences du même canal, le débit de A sera respectivement divisé par 4 (1), par 2.1 (2) et par 1.9 (même canal)

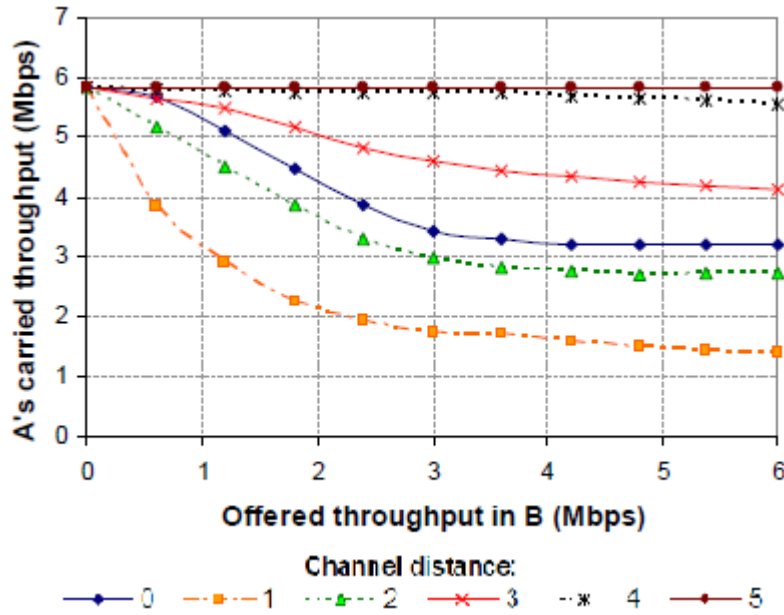


FIGURE 4.6 – Variation du débit en fonction des interférences des canaux adjacents[65]

par rapport à son débit en l'absence de B. L'impact des interférences des canaux d'une distance de 3 sera moins important, le débit sera divisé par 1.5 et, pour une distance plus grande, l'impact sera très faible. Nous avons donc pris la décision de les négliger.

Pour réaliser ce filtre, nous avons décidé de nous inspirer de l'impact d'une station sur le débit d'une autre en fonction de la distance entre deux canaux (en termes de canal). Grâce aux observations faites ci-dessus, nous avons calculé un poids pour chaque canal et le canal qui aura le poids le plus faible contiendra les points d'accès qui subiront le moins d'interférences.

Afin de calculer ce poids, nous avons compté, pour chaque canal, le nombre de BSSID présents sur ce canal et sur les canaux adjacents (limité à une distance de 3). Chacun de ses nombres va être multiplié par le facteur affectant le débit de la station A (voir équation 4.22). Ensuite, toutes ces valeurs doivent être additionnées pour donner comme résultat le poids associé au canal. Ce calcul doit donc être fait pour tous les canaux sur lesquels des VAP sont présents. Précisons également que nous n'avons pas pris en compte les canaux qui sont à une distance de 4 et de 5 car, comme décrit dans l'observation précédente, ils ne présentent que très peu d'impact par rapport à une distance de 3.

$$\begin{aligned}
 poids = & nbr_chan0 * 1.9 + nbr_chan1 * 4 + \\
 & nbr_chan2 * 2.1 + nbr_chan3 * 1.5
 \end{aligned}
 \tag{4.22}$$

Dans le cas des points d'accès proposant du 5 GHz, ils ne subiront pas d'interférence due aux canaux adjacents car les canaux utilisés ne se chevauchent pas (voir section 2.3.2.2). Dès lors, pour calculer le poids correspondant au canal, nous repreneons simplement le nombre de BSSID présents sur celui-ci.

En reprenant tous ces poids séparés selon si c'est du 2.4 GHz ou du 5 GHz, on pourra avoir deux classements qui reprendront tous les canaux du moins susceptibles de subir des interférences au plus exposé.

4.2.1.8 Retransmission des trames de données

Nous avons réalisé ce filtre dans le but de pouvoir avoir le pourcentage de retransmissions, c'est-à-dire : le nombre de trames de données retransmises sur le nombre total de trames de données. Les trames ont un *flag* égal à 1 lorsqu'elles sont retransmises, ce qui nous permet de les retrouver facilement.

Nous n'avons pas mis d'indice de priorité pour le moment sur ce filtre car nous n'avons aucune échelle pour pouvoir dire qu'un certain pourcentage de retransmissions est bon ou mauvais. Le but sera de voir si on peut le corrélérer avec la charge du trafic et ensuite déterminer l'indice au moment de la validation.

4.2.1.9 Nombre de stations connectées

Nous avons réalisé ce filtre dans le but de connaître le nombre de stations connectées par point d'accès. Pour réaliser cela, nous avons analysé les trames de données. Nous avons compté le nombre de stations qui envoient des données grâce à leur adresse MAC. Nous avons donc compté le nombre d'adresses MAC différentes qui envoient vers une même adresse correspondant à un point d'accès.

Le débit maximum réalisable étant fortement influencé par le nombre de stations connectées [4, 61], nous pouvons facilement imaginer que plus il y a de stations connectées, plus le trafic sur le point d'accès sera grand. Par contre, ce nombre de stations ne nous permet pas de connaître la charge réelle consommée par station. Ce filtre n'aura pas besoin de faire l'objet d'une validation car le résultat ne sera pas influencé par le trafic généré.

4.2.1.10 RTS/CTS

Le filtre expliqué dans cette section n'a pas été implémenté pour diverses raisons expliquées ci-dessous.

Les trames RTS/CTS sont utilisées pour contrer, entre autres, le problème du nœud caché présent dans certaines architectures (pour plus d'explications, voir section 2.3.5.3). Dès lors, ce type de trames n'est pas utilisé par tous les points d'accès. Si elles sont activées sur un point d'accès, on peut en observer malgré le fait qu'il n'y ait aucun nœud caché dans le réseau. Ces trames ne représentent pas une solution

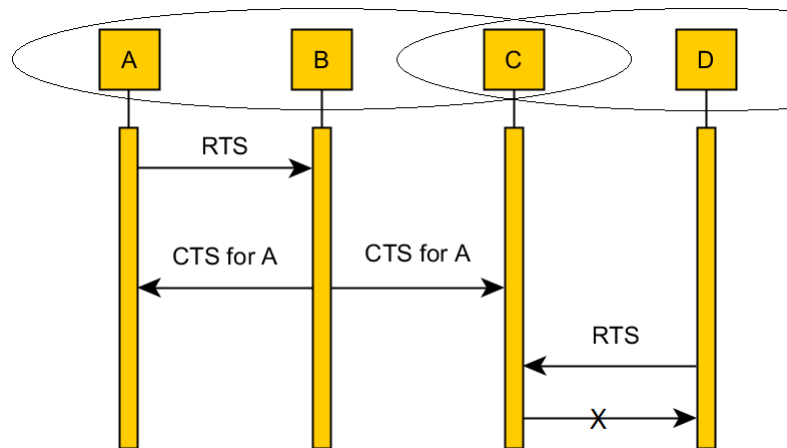
complète car elles peuvent, dans certains cas, faire diminuer le débit.

Pour l'argumentation de ce filtre, nous nous sommes basés sur l'article "*Performance Analysis of the IEEE 802.11 Distributed Coordination Function*" [4]. Premièrement, il est découvert que le mécanisme d'accès normal atteint un débit maximal très proche du débit atteint avec le mécanisme RTS/CTS. En plus de cette constatation, il apparaît que le débit dépend fortement du nombre de stations lors d'un accès normal, et que, plus la taille du réseau est grande, plus le débit est faible. À l'inverse, avec le mécanisme RTS/CTS, le débit n'est pas fortement influencé par le nombre de stations connectées, il reste pratiquement constant. Le RTS/CTS, grâce à sa technique de réservation du média, va réduire le nombre de collisions par rapport au mécanisme d'accès normal [67]. Il faut cependant noter que cette réduction est uniquement efficace si la taille du réseau et la taille de la fenêtre de congestion conduisent à une forte probabilité de collision.

Alors que dans certains cas l'utilisation du RTS/CTS est bénéfique, il apparaît que ce bénéfice dépend grandement de la taille des trames échangées sur le réseau. En effet, le RTS/CTS a un impact négatif quand il s'agit de trames de petite taille car il rajoute davantage d'*overhead* sur le réseau [4, 67]. Par défaut, une limite assez grande est mise en place pour la taille des trames qui peuvent utiliser le RTS/CTS [25]. Le problème est qu'aucune information concernant cette limite n'est communiquée dans les trames. Il est donc impossible de la connaître et de savoir si elle a été modifiée ou si elle est restée à sa valeur par défaut. Pour résumer cette partie, le mécanisme RTS/CTS aura un meilleur avantage dans de larges réseaux que le mécanisme d'accès normal.

Même si l'insertion du RTS/CTS vient régler le problème de nœud caché, il ne reste pas forcément la solution idéale, comme mentionné dans l'article "*Why RTS-CTS is not your ideal wireless LAN multiple access protocol*" [58]. L'analyse montre qu'en plus de ne pas résoudre complètement ce problème connu - il subsiste des collisions entre les nœuds cachés -, il ajoute un autre type de problèmes que l'on ne retrouve pas avec le mécanisme d'accès normal : le problème des "stations bâillonnées", "*the gagged stations problem*". Nous expliquons ce problème grâce à l'exemple illustré à la figure 4.7. Supposons que lors d'une transmission de *A* vers *B*, *D* souhaite transmettre vers *C*. Lorsque *C* va recevoir le paquet RTS provenant de *D*, *C* ne pourra pas répondre à la requête car le canal est réservé par *A*.

Après avoir pris en compte ces différentes informations, nous avons pris la décision de ne pas créer de filtre qui permettrait de mettre une priorité sur les réseaux utilisant le protocole RTS/CTS. La raison est que les points positifs et négatifs à propos de l'utilisation des trames RTS/CTS sont assez variés : elles peuvent soit

FIGURE 4.7 – *The gagged stations problem*

offrir un débit similaire à un mécanisme d'accès normale, soit avoir un meilleur débit car elles permettent de réduire le nombre de collisions (dans de large réseau) ; cependant le bénéfice qu'elles peuvent apporter dépend de la limite de la taille des trames qu'elles vont véhiculer - limite qui nous est inconnue -, et autre point négatif, les trames RTS/CTS introduisent un nouveau type de problème qui risque de diminuer les performances du réseau (l'impact étant non quantifié).

Toutes les informations nécessaires afin d'implémenter le filtre sont disponibles dans notre programme, cela pourra faciliter l'implémentation dans le cas où il serait intéressant de développer ce filtre ou si les variables sont quantifiées.

4.2.2 Couche 3

Pour faire des analyses au niveau de la couche 3, l'écoute des paquets échangés entre les points d'accès et les stations connectées ne suffit plus. Il faut maintenant avoir une connexion entre l'AP et la station de la personne qui souhaite analyser cet AP.

Pour cette implémentation, nous suivons les observations réalisées dans la section 3.4.2. Ainsi, nous nous concentrons exclusivement sur les hotspots de type ouvert car ce sont les seuls qui permettent une connexion sans authentification. Comme observé, peu de protocoles sont autorisés à passer. Nos implémentations se focalisent donc uniquement sur deux protocoles : DNS et DHCP.

Pour réaliser les tests au niveau de la couche 3, nous reprenons les 5 BSSID qui seront considérés comme les meilleurs par les filtres de la couche 2. Le but premier des filtres de la couche 3 sera de pouvoir comparer entre eux les différents BSSID repris et de pouvoir vérifier si les filtres de la couche 2 donnent le même résultat.

4.2.2.1 Protocole DHCP

Nous avons, grâce à ce filtre, récupéré le temps moyen de réponse du routeur aux requêtes DHCP. Si nous avons un temps de réponse long, nous pouvons croire que le réseau local est chargé. Nous parlons bien ici de réseau local, entre la station et le point d'accès, car c'est l'AP qui va donner l'adresse IP à la station. Nous avons assez simplement implémenté cela en analysant les paquets échangés lors de la connexion.

Ces paquets ont été récupérés grâce à TCPdump (A.2). Nous avons aussi provoqué l'échange de requêtes DHCP en utilisant la commande *dhclient \$interface* : la station envoie une requête DHCP en *broadcast* et l'AP répond avec un ACK. Nous avons donc calculé la moyenne et elle pourra être comparée aux autres points d'accès qui sont testés.

4.2.2.2 Évaluation de la bande passante

Nous avons vu, dans le chapitre 3, que les paquets DNS n'étaient pas modifiés ni bloqués, nous avons donc déduit que le port 53 d'un hotspot ouvert laissait passer les paquets normalement. À partir de ce constat, nous en avons déduit que l'on pouvait exploiter ce port 53 pour pouvoir faire des tests pour évaluer bande passante grâce à des échanges de paquets UDP. Il existe différentes techniques open source qui utilisent des trains de paquets pour pouvoir évaluer la bande passante.

L'article "End-to-End Available Bandwidth Estimation Tools, an Experimental Comparison" [21] compare les techniques d'évaluation de bande passante les plus utilisées et disponibles gratuitement. Ils ont comparé Abing, ASSOLO, DietTopp, IGI, pathChirp, Pathload [28], PTR, Spruce et Yaz [59] dans un environnement contrôlé. Les performances de chaque programme ont été analysées suivant la précision, le temps d'exécution et le trafic injecté dans le réseau pour réaliser l'estimation.

Dans notre cas, nous recherchons une technique qui ne surévalue pas la bande passante, qui est la plus proche de la réalité et qui nous permet de la mettre en place sans devoir avoir accès au point d'accès sur lequel la station se connecte. Au niveau de la précision, nous pouvons voir à la figure 4.8 les différentes techniques qui sont les plus proches de ce que l'on recherche avec une rapide description ci-dessous.

Pathload est un outil de mesure actif qui estime la bande passante disponible sur un chemin dans le réseau. Il peut mesurer la bande passante de manière "non-intrusive" car il n'affecte pas le débit des autres connections, il injecte un trafic non-significatif et sans causer une augmentation des délais des files ou des pertes sur le chemin [28].

Yaz se base sur les mêmes concepts que Pathload. Les auteurs ont démontré que

Yaz est plus précis et donne des résultats plus rapidement comparés à Spruce et Pathload [59].

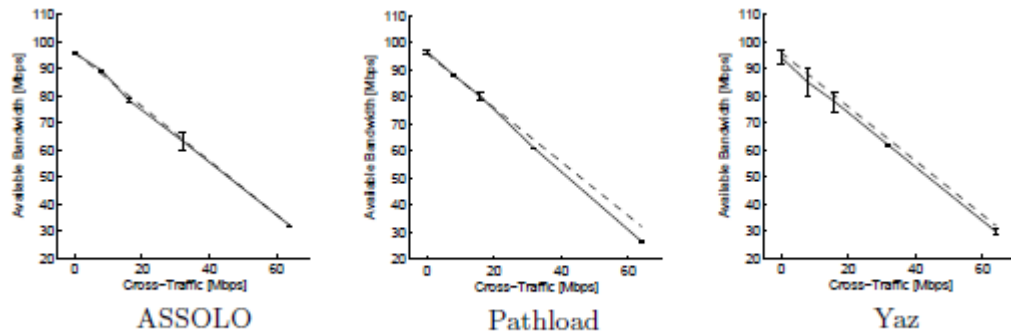


FIGURE 4.8 – Résultats expérimentaux obtenus en présence d'un trafic à un taux constant avec une variation des taux de 0 à 64 Mbps [21]

Nous avons fait donc le choix de prendre la technique ASSOLO car c'est la seule qui permet à la fois d'avoir une bonne précision avec un trafic généré et un temps faibles.

La technique ASSOLO [20], "Available-bandwidth Smart Sampling On-Line Tool", est basée sur le concept de la "congestion auto-induite" ("*self-induced congestion*"), et comporte un mécanisme, appelé REACH ("*Reflected ExponentiAl Chirp*"), qui teste un large spectre de taux d'envoi dans un seul flot de paquets et qui est plus précis au centre de l'intervalle des taux.

Cet outil injecte une quantité de trafic négligeable et minimise l'impact de différentes sources d'erreurs. ASSOLO utilise donc une technique de mesure active, c'est-à-dire que des paquets vont être envoyés à différents taux pour mesurer les délais que le trafic normal d'utilisation provoque. Cette méthode demande d'avoir un accès aux deux extrémités du réseau.

ASSOLO a deux avantages : il n'a pas besoin d'avoir une synchronisation des horloges ni de connaître l'offset de l'horloge entre les stations qui évaluent la bande passante. Il est important de considérer que le premier paquet du train n'est pas associé à un taux d'envoi, il est utilisé pour calculer tous les délais relatifs dans un flot de paquets.

Le flot REACH de paquets utilisé par ASSOLO teste des taux croissants de la limite inférieure L jusqu'à la limite supérieure U . L'augmentation du taux n'étant pas linéaire, la densité du flot augmente lorsqu'il se rapproche du centre ($H = \frac{U+L}{2}$) et donc on peut dire la même chose pour la précision de l'estimation, si elle est proportionnelle à la densité du trafic.

La précision relative maximale est déterminée par le paramètre σ . Si on connaît l'écart entre le taux minimal et le maximal, on peut calculer l'erreur absolue S

proche du centre de l'intervalle :

$$S = \sigma \left(\frac{U - L}{2} \right) \quad (4.23)$$

L'algorithme utilise un coefficient γ pour contrôler la vitesse de changement de la densité des flots. ASSOLO utilise par défaut $\sigma = 5\%$ et $\gamma = 1.2$. Ces choix sont arbitraires, mais le fait de diminuer ces paramètres augmente la quantité de paquets échangés et augmente aussi la précision de l'estimation. Au contraire, les augmenter résulte par une estimation moins précise mais avec moins de paquets échangés.

Un autre paramètre Δ_x est important pour décrire le flot de REACH créé par ASSOLO. Il décrit l'écart entre deux paquets consécutifs du flot :

$$\Delta_x = S * \gamma^{|x-1|} = \sigma \frac{U - L}{2} \gamma^{|x-1|} \quad (4.24)$$

En commençant par le centre H jusque la limite maximale, on aura comme taux instantanés : $H, H + \Delta_1, H + \Delta_1 + \Delta_2, \dots$ Et commençant par le centre vers la limite minimale, on aura : $H, H - \Delta_1, H - \Delta_1 - \Delta_2, \dots$ La figure 4.9 montre la distribution des taux, on peut voir que le profil de la fonction est symétrique et qu'il représente deux fonctions exponentielles.

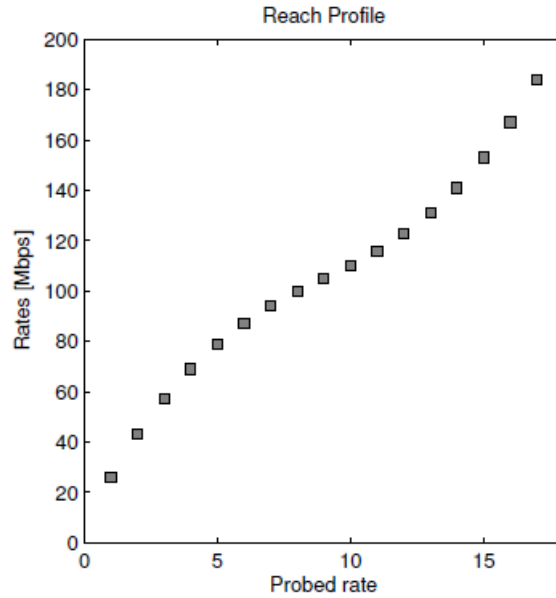


FIGURE 4.9 – Distribution des paquets dans un flot REACH [20]

REACH envoie à $2k + 1$ différents taux, il aura donc besoin d'envoyer $N = 2k + 2$ paquets : les k premiers sont pour les taux plus petits que H , les k derniers pour les taux plus grands que H , un paquet pour le taux H et un pour le premier paquet qui est envoyé. A partir de l'équation $U = R_k = H + \sum_{i=1}^k \Delta_i$, il est possible de

trouver la valeur de k suivant les paramètres choisis :

$$k = \left\lceil \log_{\gamma} \left(\frac{\gamma - 1}{\sigma} + 1 \right) - 1 \right\rceil \quad (4.25)$$

Vu que l'on connaît la taille de REACH, on peut combiner les équations des taux instantanés développées ci-dessus pour obtenir le débit produit :

$$R_j = H + \text{sign}(j - \frac{N}{2}) * S * \frac{\gamma^{|j - \frac{N}{2}| - 1}}{\gamma - 1} \quad (4.26)$$

Afin de pouvoir utiliser le programmes selon nos besoins, nous avons dû l'adapter. Le programme était organisé de manière à avoir un programme qui envoie les trains de paquets et un qui les réceptionne. Ces deux programmes écoutaient sur un port défini le temps qu'un programme, dit Master, demande le lancement du test. La première modification a été de rassembler le Master et l'envoyeur pour ne faire qu'un programme car nous voulons que ce soit la station elle-même qui décide de lancer l'estimation de bande passante. Ensuite, le programme n'autorisait pas l'utilisation du port 53, ce que nous avons changé.

D'autres modifications seront apportées et expliquées dans la section 5.2.4 suite aux observations que nous avons faites lors de la validation.

4.2.2.3 Protocole DNS

Le but de ce filtre est de pouvoir estimer la charge du trafic grâce au protocole DNS. Pour cela, notre but était de créer des requêtes DNS pour analyser ensuite le temps de réponse à ces requêtes. Nous voulions utiliser l'outil *DNSdiag* (A.5) et, plus particulièrement pour ce filtre, le programme *DNSeval* (A.5). Ce programme permet de comparer plusieurs temps de réponse d'une multitude de serveurs DNS en une fois. Le but était de récupérer la moyenne de ces réponses pour pouvoir les comparer avec les différents points d'accès testés mais le programme envoie directement des requêtes pour un même domaine vers différents serveurs. Comme nous le verrons dans la section 5.2.4, il n'est pas possible d'envoyer des requêtes vers un serveur directement, il faut passer par le *DNS resolver*.

Nous avons donc décidé de faire quelques requêtes vers différents domaines connus (*google, facebook, uclouvain, ...*) et vers notre domaine (*tests.dnstfe.be*) pour être certain que les réponses ne viennent pas de la cache. Nous avons pris le temps moyen de réponse et nous l'avons comparé entre les différents BSSID testés pour pouvoir comparer le classement avec les filtres de la couche 2. Dans ce cas-ci, nous n'évaluons plus la qualité du réseau local mais la qualité de la connexion à internet.

4.3 Définition de l'influence de chaque filtre

Comme expliqué au début de ce chapitre, des indices de priorité ont été mis en place afin de permettre d'évaluer un point d'accès en fonction des résultats qu'il fournit par l'intermédiaire de nos filtres. Ces indices sont déterminés en fonction des résultats du filtre et peuvent prendre une valeur minimale de 1 (la plus haute priorité). La valeur maximale (la plus faible priorité) n'est pas définie car elle dépend du résultat du filtre. La mise en place de ces indices ainsi que leur importance est faite de manière subjective. Étant donné la nature de nos différents filtres, aucun article ne définit précisément ce qui est considéré comme un bon ou un mauvais résultat. Nous avons, par conséquent, dû définir quel était l'impact des résultats sur les performances d'un point d'accès.

Notons avant tout que les deux premiers filtres (la puissance du signal reçu et le nombre de *Beacon* capturés) ne se verront pas attribués d'indice de priorité étant donné qu'ils servent uniquement à réduire les données à analyser.

Pour la consommation de l'*airtime*, nous considérons que si les trames utilisent 50% de l'*airtime*, l'impact sur les performances du réseau est très important. Il aura alors la plus faible des priorités que nous définissons pour ce filtre (à savoir 5). À l'inverse, s'il y a moins de 10% qui est utilisé, alors la priorité sera la plus haute possible (à savoir 1). Nous reprendrons ces mêmes valeurs pour le filtre s'occupant du pourcentage de pertes des trames *Beacon*.

Concernant l'*airtime* consommé par les trames de données, nous revoyons ces valeurs à la hausse. Selon [55], il apparaît que les trames de données utilisent en général plus d'*airtime* que les trames *Beacon*. La plus faible des priorités est alors mise si les données consomment plus de 85% de l'*airtime*. À l'inverse, la plus grande des priorités sera mise si la consommation est inférieure à 25%.

Le filtre qui permet d'analyser le délai des *Beacon*, quant à lui, aura besoin d'une donnée supplémentaire afin de pouvoir mettre une priorité sur ses résultats. Cette donnée est le temps minimum que peut prendre une trame *Beacon* avant la transmission. Celui-ci est récupéré en calculant les temps IFS et le temps minimal de la fenêtre de contention en fonction de la présence de la qualité de service et du standard de communication utilisé. Quand les résultats du filtre sont fortement similaires au temps minimum, nous lui donnerons la priorité la plus haute (à savoir 1). À chaque fois que le résultat sera plus grand que la n ème multiplication de la valeur minimal théorique, la priorité sera augmenté de n .

En ce qui concerne la bande passante potentielle, il s'agit également d'une valeur subjective. Si le résultat est inférieur à 1 Mbit/s, la priorité sera la plus faible. La priorité sera attribuée en fonction d'un résultat inférieur à 2, 4, 8, 16 etc. La plus

grande bande passante potentielle aura la priorité la plus importante.

Notre filtre - les canaux adjacents - utilisera une autre méthode afin d'établir les indices de priorité. Pour rappel, ce filtre nous donne déjà, en fonction du poids des canaux environnants, une mesure précise pour les canaux. L'unique tâche est donc de trier ces résultats par ordre croissant afin de leur donner un indice : les résultats avec le poids le plus faible auront la priorité la plus haute, indiquant que le canal en question n'offre que peu d'interférence avec ses voisins.

Pour le nombre de stations connectées, nous nous sommes inspirés de différents articles [4, 61], qui nous ont permis de mettre la priorité la plus haute (1) pour un AP qui a moins de 5 stations connectées. Ensuite, nous avons mis un indice de 2 pour moins de 10 stations connectées, de 3 pour moins de 20 et de 4 pour plus de 20.

Chapitre 5

Validation et résultats

Dans ce chapitre, nous nous chargerons de la validation de notre algorithme ainsi que des résultats qu'il nous donne. En section 5.1, nous détaillons les différents lieux et environnements où les tests de notre algorithme ont été conduits. Ensuite, en section 5.2, nous expliquons la démarche que nous avons entreprise afin de valider notre programme. Finalement, nous concluons ce chapitre dans la section 5.3.

5.1 Environnements de tests

Notre validation se concentre fortement sur la couche 2 car elle représente une partie majoritaire de notre algorithme. Pour cette dernière, nous avons recherché différents endroits pour avoir des situations dans lesquelles nous aurons des degrés de complexité variés. Nous pourrions réaliser des mesures à deux endroits :

1. Nous avons tout d'abord voulu avoir un environnement le plus sain possible pour pouvoir avoir un contrôle complet sur ce dernier. Nous avons donc recherché un lieu qui avait le moins de réseaux Wi-Fi disponibles possible. Nous nous sommes donc placés dans un endroit qui ne comptait aucun point d'accès sur les canaux autres que le 1 pour le 2.4 GHz. Pour réaliser nos tests, nous avons mis en place les 2 routeurs *TP-LINK* que nous possédons (voir section 3.2.4). Dans cet environnement (voir figure 5.1), nous avons deux points d'accès (AP2 et AP3) et, sur chacun de ces points d'accès, deux ordinateurs portables y sont connectés (nommés respectivement station *A*, station *B* et station *C*, station *D*). Ces derniers généreront du trafic afin de voir si cela impacte la prise de décision de notre algorithme. Le *Raspberry*, quant à lui, se placera dans cet environnement et prendra des mesures avec son antenne (voir section 3.2) qui seront par la suite analysées par notre programme.
2. Pour avoir plus de BSSID proches, nous nous sommes placés dans un kot étudiant à Louvain-la-Neuve. Nous avons comptabilisé un total de 50 BSSID sur le 2.4GHZ.

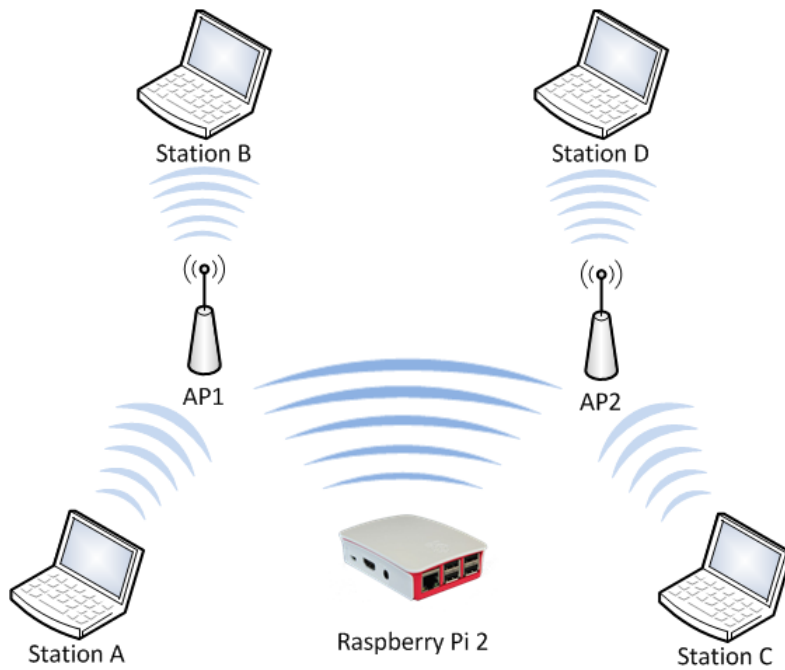


FIGURE 5.1 – Organisation de l'environnement de test sain

L'environnement décrit à la figure 5.1 n'a qu'un seul but : vérifier si l'algorithme peut déterminer le meilleur point d'accès s'il n'y a aucun chevauchement de canal, que le nombre de stations connectées est similaire et que le nombre de points d'accès sur un canal se limite à une unité. Cette validation veut se concentrer uniquement sur la réaction du point d'accès en fonction de la charge de trafic qu'il recevra des stations qui y sont connectées.

Nous savons déjà que si l'environnement a un trop gros impact sur le point d'accès, par exemple beaucoup de points d'accès sur les canaux adjacents, le point d'accès aura très peu de chances d'être choisi. Donc si l'environnement est similaire, la seule façon de différencier deux points d'accès est de vérifier leur comportement par rapport au trafic qu'ils subiront.

5.2 Validation

Dans cette section, nous avons utilisé l'environnement de test décrit dans la section précédente afin de valider les différents filtres au niveau de la couche 2. Les tests réalisés au niveau de la couche 3 sont réalisés dans un environnement réel, comme n'importe quel réseau fourni par *Proximus* par exemple.

En 5.2.1, le trafic déployé afin de valider notre algorithme sera détaillé. Par la suite en section 5.2.2, il sera fait état de la validation opérée sur la couche 2 ainsi que le résultat observé via les différents filtres implémentés dans le chapitre 4. Un récapitulatif quant aux résultats des filtres sera donné en section 5.2.3. Pour la couche 3,

la section 5.2.4 se chargera de vérifier si l'évaluation de la bande passante grâce à l'échange de paquets UDP est possible.

5.2.1 Trafic généré

Pour l'environnement sain (voir figure 5.1), nous avons voulu avoir le moins d'interactions possibles entre les points d'accès pour pouvoir analyser le comportement des AP selon la charge de trafic qu'ils subissent. Nous avons donc décidé de mettre un routeur (AP2) sur le canal 6 et un autre routeur (AP3) sur le canal 11 comme préconisé en section 2.3.2.1. Dans cette configuration, nous n'aurons qu'un point d'accès par canal et aucune interférence entre eux. Le but de cette configuration est de pouvoir analyser le nombre de trames *Beacon* reçues et perdues, leur délai, l'*airtime* consommé par les trames de données et l'évaluation de la bande passante selon la charge créée sur l'AP. Cela aura donc pour but de pouvoir différencier deux points d'accès selon la charge de trafic qu'ils doivent gérer.

Dans cette optique, les deux sections suivantes vont s'intéresser plus particulièrement au trafic qui a été généré dans notre architecture de validation.

5.2.1.1 Déterminer les limites

Avant toute chose, il est nécessaire de connaître la quantité de paquets qui peuvent être envoyés avant d'atteindre la bande passante maximale. Pour ce faire, l'outil *iPerf* (A.8) a été utilisé. Via ce dernier, nous avons pu générer un trafic de paquets d'une taille variable sur l'AP2. La variation de la taille des paquets nous a permis de connaître la bande passante maximale supportée par les AP en notre possession. Le tableau 5.1 nous donne le résultat de ces tests.

Taille des segments [octets]	Bande passante [Mbits/s]
200	5,3
500	5,55
1000	8,9
1500	11,3
2000	11,1

TABLE 5.1 – Add caption

La bande passante maximale supportée par notre point d'accès (AP2), pour une taille de 1500 octets, est dès lors connue. La partie qui nous intéresse concerne plus particulièrement est le nombre de paquets envoyés par seconde. La figure 5.2 nous renseigne un peu plus sur ce point.

On observe clairement qu'il y a environ 1500 paquets qui sont envoyés par seconde afin d'atteindre la bande passante maximale.

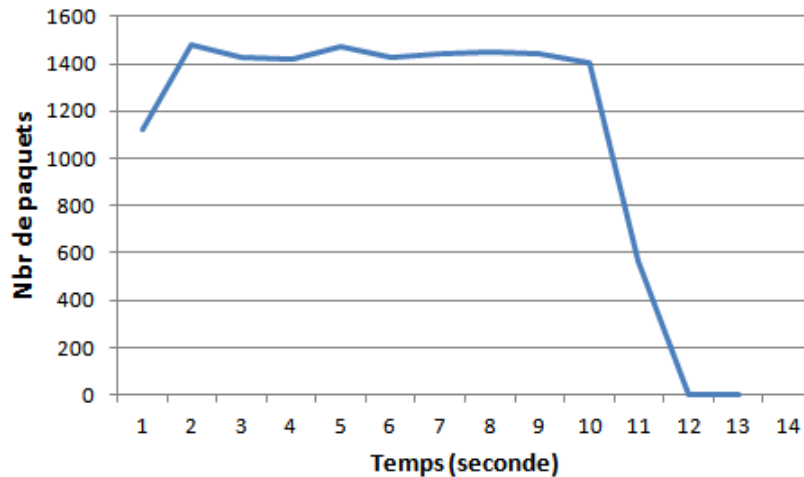


FIGURE 5.2 – Nombre de paquets envoyés par seconde

5.2.1.2 Génération du trafic via DITG

Le nombre de paquets maximal connu, nous pouvons nous intéresser plus intensément au trafic à générer sur nos points d'accès.

Nous avons décidé de simplifier au maximum le trafic généré sur les points d'accès pour avoir un contrôle maximal. Pour cela, nous avons créé un *burst* de données constant (*background*) entre les deux ordinateurs d'un même AP grâce à l'outil DITG (A.9). Nous utilisons un flux constant de paquets UDP entre deux ordinateurs connectés sur un même AP. Nous sommes conscients que le trafic généré ne se rapproche pas d'une utilisation normale d'un réseau internet mais nous avons fait ce choix pour avoir un environnement le plus simple possible. Pour avoir une utilisation plus proche de la réalité, on devrait utiliser le protocole TCP et de ne pas avoir un flux constant mais par exemple utiliser une distribution de Poisson pour la génération du trafic.

Pour garder un témoin et pour pouvoir comparer nos résultats, nous avons généré, sur l'AP3, un trafic faible et identique pour tous les tests réalisés, 10 paquets par seconde de 50 octets. Le but de ce trafic est d'assurer une charge minimale au niveau de l'AP3, facilitant ainsi les calculs de l'algorithme. Concernant la charge sur l'AP2, celle-ci varie en fonction des tests réalisées. Nous avons donc contrôlé le nombre de paquets par seconde et la taille de ces paquets. Le but est de faire varier ces paramètres jusqu'à surcharger complètement le point d'accès pour observer ou non une corrélation entre la charge du point d'accès et les données observées.

Nous avons principalement fait varier le nombre de paquets par seconde. En analysant le trafic disponible sous les AP de *Proximus*, nous avons décidé d'utiliser une taille de 1000 octets bien que *iPerf* nous renseigne une taille approximative de

1500 octets pour obtenir la bande passante maximale. Nous avons alors fait varier le nombre de paquets par seconde entre 200 et 4000 afin de tester différentes charges. Même si la section précédente mentionne qu'il faudrait environ 1500 paquets de 1500 octets pour obtenir la bande passante maximale, nous avons décidé d'augmenter le nombre de trames de données générées afin de voir comment les résultats de notre algorithme sont altérés quand la saturation de l'AP est plus élevée.

Le tableau E.1 reprend les valeurs de configuration de ces tests. Chaque configuration a été réitérée 6 fois afin d'éliminer les cas anormaux qui pourraient survenir. Par la suite, la médiane des valeurs trouvées a été calculée pour chaque test.

N° test	Paquets/sec	Octets
1	200	1000
2	500	1000
3	1000	1000
4	2000	1000
5	4000	1000

TABLE 5.2 – Tableau reprenant les valeurs de la charge produite sur l'AP2

La section suivante décrit de manière succincte les variations en fonction du trafic.

Observation du trafic généré via DITG

Dans cette partie, nous évoquerons brièvement les résultats obtenus directement depuis DITG et montrerons comment ces derniers peuvent influencer le résultat de notre algorithme. Toutes les données utiles obtenues via DITG sont disponibles dans l'annexe E.

Si le nombre de paquets par seconde est inférieur au taux déterminé dans la section 5.2.1.1, on remarque que le nombre de pertes de paquets est, pour chaque test réalisé, fortement proche de 0%. À l'inverse avec des taux plus élevés, par exemple, de 2000 ou 4000 paquets par seconde, le pourcentage de pertes peut facilement atteindre les 25%.

Concernant les débits observés, il montre une saturation dès que le nombre de paquets par seconde dépasse les 1500 paquets comme défini précédemment. A savoir, quand on monte à 2000 ou 4000 paquets par seconde, les débits stagnent aux alentours de 11 Mbit/s en transmission et de 9 Mbit/s en réception.

Le fait de voir que le trafic généré par DITG influence les débits et les pertes de paquets va nous permettre alors de vérifier si notre algorithme peut déterminer qu'un point d'accès subit une saturation.

5.2.2 Couche 2

Dans les sections précédentes, nous spécifions l'environnement dans lequel notre validation allait se conduire ainsi que le trafic qui était généré afin de réaliser cette validation. Dans cette section, nous nous intéresserons plus aux résultats de nos différents filtres lors de la validation.

Comme précisé en section 5.1, étant donné que l'environnement de validation principal est axé majoritairement sur la réaction d'un point d'accès en fonction du trafic qu'il rencontre et non des interférences environnantes, certains des nos filtres n'ont aucune utilité dans cette validation. Afin de quand même montrer des résultats pour ces derniers, nous avons utilisé le deuxième environnement de test. Cela nous permet d'avoir un réseau avec pléthore de réseaux sans fil, de canaux se chevauchant et d'interférences entre ces canaux. Le tableau 5.3 reprend les différents filtres ainsi que l'environnement dans lequel ils ont été testés.

Nom du filtre	Environnement
Consommation de l' <i>airtime</i> : trames de données	contrôlé
Perte des trames <i>Beacon</i>	contrôlé
Délai des trames <i>Beacon</i>	contrôlé
Estimation de la bande passante	contrôlé
Retransmission des données	contrôlé
Consommation de l' <i>airtime</i> : trame <i>Beacon</i>	secondaire
Interférence des canaux adjacents	secondaire
Nombre de stations connectées	secondaire

TABLE 5.3 – Tableau reprenant les filtres développés et envisagés

Notre programme va alors écouter 5 secondes sur chaque canal en mode monitor pour pouvoir récupérer les paquets dont on a besoin comme décrit dans la section 4.1.1. Afin d'éviter des pertes de données lors de l'écriture vers la carte SD de notre *Raspberry*, l'écriture se réalise d'abord dans un dossier monté en mémoire RAM.

Les sections suivantes reprennent les différents résultats obtenus par les filtres. Toutes les données brutes peuvent être retrouvées à l'annexe E. Avant de décrire les résultats rencontrés, il faut noter le fait que les données sont envoyées avec une qualité de service, comme celles présentes sur les hotspots de *Proximus*.

5.2.2.1 Consommation de l'airtime : trames de données

Via ce premier filtre, nous essayerons de voir si, en fonction du trafic généré, l'airtime consommé par les trames de données varie au point de permettre d'élire un point d'accès plutôt qu'un autre. Cela permettrait aussi de voir si un point d'accès ressent une surcharge selon le trafic entrant et sortant de ce dernier.

La figure 5.3 nous montre, sous forme de graphe, la variation de cet airtime en fonction du trafic généré. Elle affiche cette variation en fonction d'une augmentation croissante du nombre de paquets créés par seconde. Pour rappel, quand nous parlons d'airtime par rapport aux trames de données, cela correspond au temps que les données consomment pendant notre fenêtre d'écoute, soit 5 secondes.

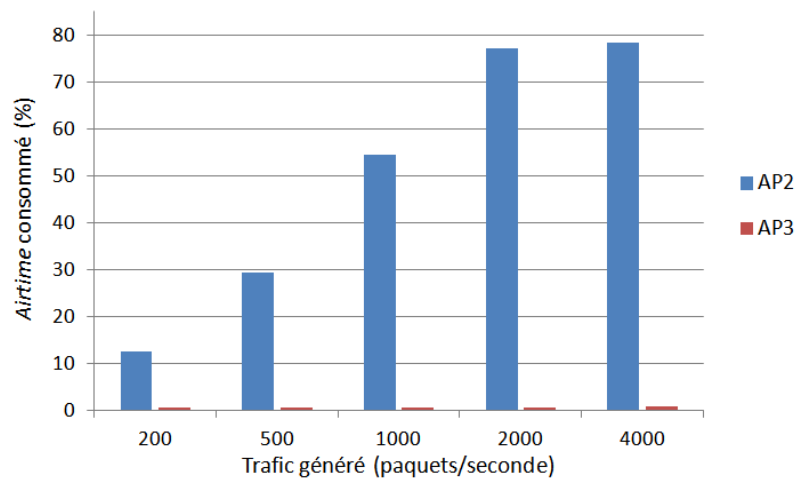


FIGURE 5.3 – Variation de la consommation de l'airtime des trames de données pour différent trafic

Il apparaît clairement que l'AP3 n'est uniquement présent que pour assurer le rôle de témoin pour un trafic très faible et pour permettre une comparaison avec les autres résultats. Concernant l'AP2, on remarque que l'airtime consommé est fortement influencé par le taux d'envois par seconde.

Intéressons-nous maintenant à l'impact de ce trafic. Comme précisé en section 4.3, si l'airtime se trouve être inférieur à 25%, la priorité donnée au résultat sera nettement plus grande que si l'airtime dépasse les 85%. Ainsi quand le trafic (2000 ou 4000 paquets/sec) dépasse le trafic maximal possible afin d'obtenir une bande passante maximale, l'airtime consomme la quasi totalité du temps disponible (soit aux alentours des 80%). Ce pourcentage est considéré comme le maximum que les trames de données peuvent consommer sur nos routeurs car il faut également prendre en compte l'envoi des trames retransmises, trames de ACK, etc.

Dans l'optique de trouver un meilleur point d'accès par rapport au trafic des trames de données, le graphe montre nettement que l'airtime consommé peut apporter une

différence quant à l'élection d'un point d'accès. Cependant, ceci est uniquement intéressant à condition que le trafic se trouve au-dessus du seuil minimal des 25% d'*airtime*. Si ce n'est pas le cas et qu'il n'y a pas de trafic de données ou très peu, alors l'impact de ce filtre s'en trouvera nul.

5.2.2.2 Délai des trames *Beacon*

Nous allons voir si on peut valider le fait que le délai des trames *Beacon* est altéré par la charge subie par l'AP et si on peut y voir un rapport proportionnel.

Grâce à la figure 5.4, on peut remarquer que les médianes des délais observés pour l'AP3 sont égaux quel que soit le test réalisé étant donné que le trafic est faible et constant. Pour l'AP2, on peut observer que les médianes obtenues sur l'entièreté des tests sont fortement similaires. Quand le nombre de paquets par seconde se trouve en dessous des 2000, le délai tourne constamment aux alentours des 383 μ s. Cette valeur correspond à l'addition des l'IFS et *backoff time* qu'une trame doit attendre avant d'être transmise. Comme expliqué en section 4.2.1.4, ces valeurs dépendent de l'utilisation de la qualité de service et du standard utilisé pour la communication. En observant plus attentivement le graphique, on peut voir que, quand le taux de paquets par seconde dépasse les 2000, le délai augmente légèrement de ± 100 μ s. Cette augmentation correspond au moment où le point d'accès reçoit plus de données qu'il peut gérer comme observé en section 5.2.1.2.

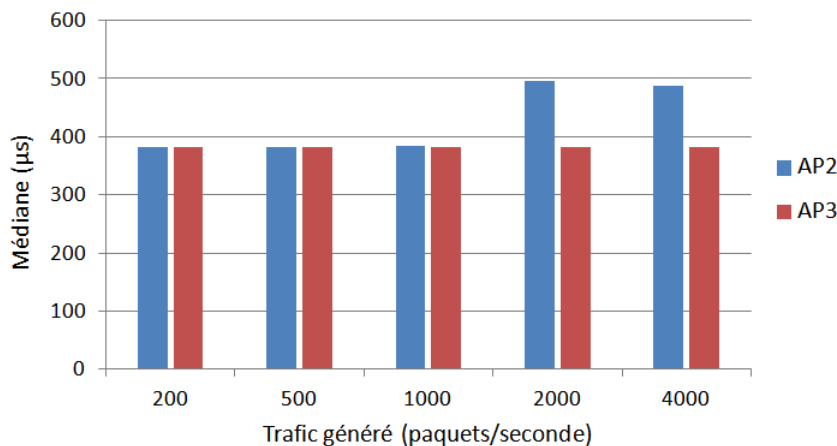


FIGURE 5.4 – Médiane des délais observés pour les tests allant de 1 à 5 (voir tableau E.1)

La première conclusion que nous pouvons en tirer est que le délai des trames *Beacon* n'est pas proportionnel à la charge subie par le point d'accès. On peut certes voir une légère hausse mais rien qui ne démontre la variation du trafic généré.

Ceci nous amène vers une seconde conclusion. Elle nous permet de dire que la priorité des trames *Beacon* en présence du QoS supposée en section 4.2.1.4 n'est pas correcte. Nous avons pris comme hypothèse que les trames *Beacon* utilisaient comme priorité du *Best Effort* tout comme les trames de données. La figure ci-dessus nous démontre le contraire. En effet, étant donné que le délai des trames *Beacon* ne se trouve presque pas altéré quel que soit le trafic généré, cela nous démontre qu'elles ne font pas partie de la même file que les trames de données. Cela suppose qu'elles possèdent une priorité plus élevée que les trames de données.

Le fait que la variation entre les différents délais est très petite et que la diffusion des trames *Beacon* ne semble pas être altérée par le trafic ne rend donc pas possible l'utilisation de ce filtre pour comparer les différents BSSID.

5.2.2.3 Estimation de la bande passante

Pour valider la technique d'évaluation de la bande passante décrite à la section 4.2.1.6, nous avons tenté de nous rapprocher le plus possible des conditions dans lesquelles les auteurs ont réalisé leurs expériences. La différence non négligeable est qu'ils ont utilisé une chambre anéchoïque. De plus, ils ont désactivé l'utilisation de RTS/CTS et fixé la carte à un rate constant de 11 Mbps (pour le 802.11b). Ce que nous n'avons pas fait pour nous rapprocher aussi des conditions que nous rencontrons dans des environnements réels.

Nous avons observé que, quel que soit le test utilisé (voir tableau E.1), l'AP2 a toujours une bande passante potentielle plus élevée que celle de l'AP3. Nous avons, par exemple pour le test n° 4, la médiane de la bande potentielle pour l'AP2 est égale à 13.27 Mbps et nous avons 2.54 Mbps pour l'AP3. Or, nous savons que ce dernier est le moins chargé et probablement meilleur que l'autre.

Nous pouvons donc constater que ce filtre ne donne pas les résultats attendus. Nous pensons que le problème peut venir de différents endroits. Premièrement, l'évaluation est faite grâce à la variation du délai des trames *Beacon*, cette variation étant mesurée en microsecondes, il se peut qu'elle soit trop faible pour pouvoir avoir une nette distinction. Ensuite, si les trames de données ont une priorité moins élevée que les trames *Beacon* (voir section 3.4.1.2), l'envoi de ces dernières ne sera pas altéré par la charge subie par le point d'accès. Et finalement, le calcul de la bande passante potentielle utilise principalement la taille des données pour calculer le délai total d'une trame.

La première et deuxième supposition trouvent leur réponse dans la section 5.2.2.2 précédente. Celle-ci nous informe donc que les délais s'appliquant à la transmission des trames *Beacon* ne semblent pas être altérés par le trafic généré. Pour la troisième supposition, d'après le descriptif de nos tests réalisés en section 5.2.1.2, nous savons

que des trames de 100 octets seront générées pour l'AP2, alors qu'il s'agit de trames de 1000 octets pour le point d'accès chargé, l'AP3. Si on reprend les grandes lignes de l'algorithme proposé pour ce filtre, la bande passante potentielle est déterminée en divisant la taille d'une trame de données par le délai des *Beacon* et du temps de transmission de la trame. Si le délai ne varie pas fortement mais bien la taille de la trame, le résultat sera alors en faveur de l'AP2, le point d'accès le plus chargé.

Tout ceci pris en compte, nous ne pourrions donc pas utiliser ce filtre pour pouvoir comparer les différents BSSID.

Notons, que si les points d'accès n'autorisaient pas la qualité de service, les trames de données et les trames *Beacon* auraient utilisé la même file de transmission. Dans ce cas, des résultats positifs auraient pu être disponibles car le délai ressenti pour les *Beacon* aurait été influencé par le trafic des trames de données. Cependant, comme le montre nos observations du chapitre 3, les hotspots sous *Proximus* utilisent la qualité de service.

5.2.2.4 Pertes des trames *Beacon*

Lors de cette validation, nous nous sommes également intéressés à l'évolution des pertes des trames *Beacon*. Ce pourcentage a été calculé au moyen du filtre défini en section 4.2.1.3 vérifiant les pertes, principalement via le numéro de séquence des trames *Beacon*. La figure 5.5 nous montre les médianes des pertes observées pour l'ensemble des tests réalisés.

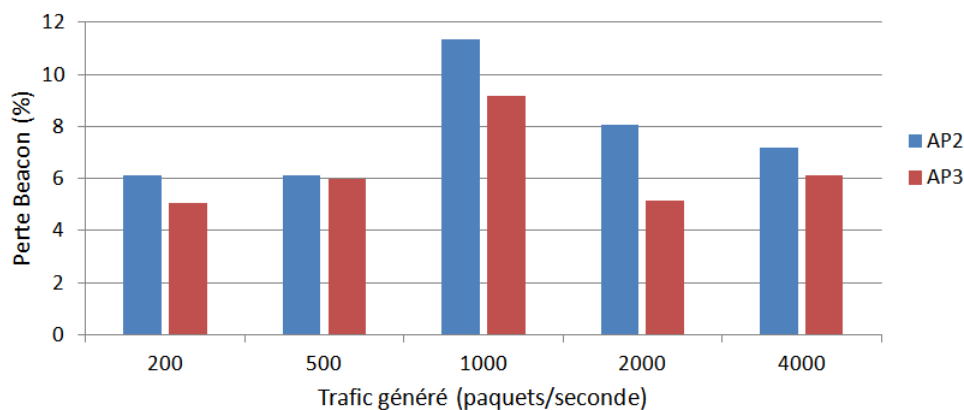


FIGURE 5.5 – Pourcentage de perte des trames *Beacon* les tests allant de 1 à 5 (voir tableau E.1)

Les variations sont clairement minimales entre les deux points d'accès. En général, les pertes sont inférieures à 10% montrant un faible taux de perte au niveau des trames *Beacon*. Pour rappel, l'environnement de validation est un environnement sous contrôle où aucune interférence avec les canaux voisins ou avec d'autres points d'accès ne peut se dérouler. Les trames ne peuvent donc entrer en collision

avec aucune autre diffusée sur le réseau.

De plus, comme démontré dans la section 5.2.2.2, il apparaît que les trames *Beacon* n'utilisent pas la même file que les trames de données lors de l'utilisation d'une qualité de service, ce qui prouve qu'elles détiennent une priorité plus élevée. Elles ne sont alors pas affectées par le trafic généré et ont donc une transmission qui est toujours assurée.

La conclusion à laquelle nous sommes arrivés est que, comme une interface en mode monitor n'est pas capable de capturer toutes les trames envoyées (voir section 3.4.1), il est possible que ces pertes de trames *Beacon* soient en fait des trames qui n'ont pas pu être capturées par l'interface.

Ces différents points peuvent expliquer le fait que, dans un environnement sous contrôle, le nombre de pertes des trames *Beacon* soit faible.

Cependant, si ce filtre vient à être mis en place dans un environnement hétérogène, la diffusion des trames pourrait alors être altérée par les canaux adjacents, les points d'accès abondamment présents sur un canal ou encore l'architecture de l'environnement (type matériaux, murs, etc). Ce filtre refléterait alors l'état de l'environnement.

Au moment de l'écriture de cette analyse, nous ne pouvons confirmer que l'utilisation d'un tel filtre aura un impact bénéfique ou non sur le résultat final de notre algorithme.

5.2.2.5 Retransmissions des trames de données

Nous pouvons voir à la figure 5.6 que le pourcentage de retransmissions des trames n'est pas toujours proportionnel à la quantité de données échangées. L'AP3 voit très peu de données passer, seule une quantité constante de données pour tous les tests. Par contre, on peut voir que le pourcentage de retransmissions est très variable. Quand on ne regarde que l'AP2, le pourcentage de retransmission suit à peu de chose près la quantité de données échangées.

Si l'on regarde le comportement d'un réseau normal, quand le nombre de données augmente, le niveau de saturation des points d'accès augmente également, ce qui diminue la fiabilité du média et par conséquent augmente le nombre de retransmissions [29]. Nous le remarquons avec l'AP2. Cependant il s'avère que l'AP3 ressent aussi, dans 2/5 des tests, des retransmissions dépassant celles ressenties par l'AP2 alors qu'il subit un trafic d'une plus faible intensité. La signification d'un tel comportement nous échappe.

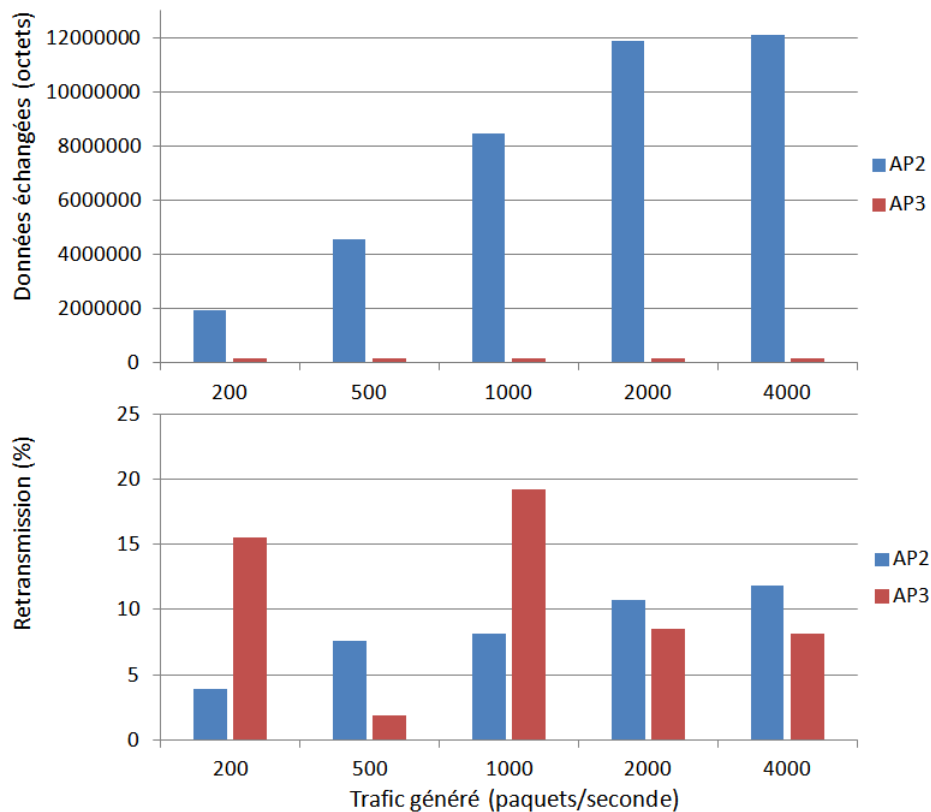


FIGURE 5.6 – Données échangées et pourcentage de retransmission pour les tests allant de 1 à 5 (voir tableau E.1)

Pour cette raison, la décision a été prise de ne pas utiliser ce filtre car nous ne pouvons pas mettre un indice de priorité logique par rapport au pourcentage de retransmission.

5.2.2.6 Consommation de l'airtime : trames Beacon

Ce filtre fait partie des quelques filtres ne pouvant montrer de résultats dans un environnement contrôlé. Ainsi pour montrer des résultats quant à l'exécution de ce filtre, le deuxième environnement a été utilisé (voir section 5.1). Comme beaucoup de points d'accès sont présents dans cet environnement, les résultats apportés par ce filtre seront assez variés.

Pour calculer l'airtime par canal (voir section 4.2.1.4) par seconde, nous avons besoin de connaître les différents BSSID présents sur le canal et leur standard. Ce standard a été récupéré grâce aux trames Beacon reçues. Grâce aux différentes valeurs liées aux standards et récupérées dans la trame, une estimation théorique de la consommation de temps pour envoyer les trames Beacon en ressort. Cette estimation est certes théorique mais elle permet néanmoins de donner une valeur minimale à l'airtime qui est consommé. Une valeur minimale car le calcul de l'airtime

dépend d'une fenêtre de contention que nous ne connaissons pas et que nous mettons à sa valeur la plus petite possible, comme expliqué dans le point 4.2.1.4.

On peut voir les résultats du filtre dans le tableau 5.4 pour l'environnement d'un kot étudiant de Louvain-la-Neuve. Un pourcentage d'*airtime* faible est intéressant, cela veut dire que l'utilisation du canal pour envoyer les trames *Beacon* sera faible en terme de pointss d'accès présents sur ce canal.

Canal	N ^{bre} VAP	<i>airtime</i> [%]
1	10	22.62
3	4	8.98
6	15	34.78
7	2	4.80
9	3	5.80
11	16	38.99
36	1	1.22
44	1	0.70
48	1	1.05

TABLE 5.4 – Tableau reprenant l'*airtime* calculé pour les différents canaux dans un kot étudiant ¹

Louvain-la-neuve étant une ville étudiante, il est assez rare de trouver un endroit où il n'y a pas de réseau Wi-Fi, ce qui facilite ce genre de filtre. On peut ainsi profiter d'un large panel de données. Dans cette figure, on remarque assez facilement que les canaux dit "sans chevauchement" à savoir les canaux 1,6 et 11 (2.3.2.1) se présentent comme ceux disposant du plus grand nombre de VAP. Plus ce nombre est important sur un canal, plus les VAP consommeront d'*airtime* afin d'envoyer les trames *Beacon* diminuant l'*airtime* disponible pour les autres type de trames. C'est d'ailleurs pour cette raison qu'il est recommandé de limiter le nombre de réseaux sans fil sur un canal [31].

Un fait intéressant ressort de cette figure : les réseaux prolifèrent beaucoup plus sur la bande des 2.4 GHz que la bande des 5 GHz malgré un plus grand nombre de canaux et moins de chevauchements entre ces canaux. Cependant comme nous ne nous intéressons uniquement aux réseaux diffusés par les hostpots, seule la bande des 2.4 GHz est prise en compte dans cette analyse.

Les valeurs d'*airtime* présentes dans la figure 5.4 sont toutes assez différentes. Bien qu'elles utilisent le même algorithme, les valeurs utilisées afin de calculer l'*airtime* ne sont pas toutes des données statiques dépendantes du standard utilisé. Le débit

1. Tous les canaux du 5 GHz n'ont pas été repris pour la simple et bonne raison qu'un grand nombre d'entre eux sont utilisés par seulement 1 VAP. Nous estimions inutile de tous les lister.

auquel les trames *Beacon* sont reçues ainsi que leur taille jouent un rôle majeur dans ce calcul. Plus une trame sera grande et son débit de réception faible, plus l'*airtime* consommé par seconde pour envoyer ces trames sera important. Avec *Proximus*, cette taille varie entre 179 octet et 229 octet et peut représenter une consommation entre 1.8% et 2.2% par VAP avec un débit de réception de 1 Mbit/s. Cela combiné avec d'autres VAP peut rapidement augmenter la consommation de l'*airtime*.

Comme nous l'avons défini en section 4.3, les VAP sur le canal qui consomment moins de 10% posséderont la priorité la plus haute. Dans le tableau ci-dessus, cela s'appliquera uniquement aux VAP présents sur les canaux 3, 7 et 9 pour le 2.4 GHz.

Via ce filtre il est donc possible de voir l'impact du nombre de réseaux sans fil sur un canal et ainsi définir sur quel canal on est susceptible d'avoir le moins de données circulant. L'importance de ce filtre est donc prouvée.

Notons, cependant, que l'idéal pour les points d'accès serait d'éliminer les taux de transmissions les plus faibles pour l'envoi des trames *Beacon*. Ceci permet de diminuer l'impact de l'*airtime* des trames *Beacon* sur le reste du trafic.

Afin de démontrer ce fait, nous avons réalisé une autre analyse dans la salle Siemens du bâtiment Réaumur de l'UCL (Place Sainte-Barbe, 2 à Louvain-la-Neuve). Dans le tableau 5.5, on remarque que même si le nombre de VAP est de 6, l'*airtime* consommé se maintient à une valeur très faible à la différence du tableau 5.4. La raison quant à une telle différence s'explique uniquement via le débit de transmissions des trames *Beacon*. Alors que les hotspots de *Proximus* utilisent le 1 Mbit/s (le plus faible possible), les points d'accès de cette salle diffusent les trames *Beacon* à des taux complément différents : on retrouve alors des taux comme 9 Mbit/s ou encore 24 Mbit/s.

Canal	N ^{bre} BSSID	<i>airtime</i> [%]
1	6	1.68
6	3	0.83
9	1	2.21
11	5	3.49
36	4	2.47
40	3	0.71
48	3	0.71

TABLE 5.5 – Tableau reprenant l'*airtime* calculé pour les différents canaux dans une salle de l'UCL

5.2.2.7 Estimation de l'influence des interférences des canaux adjacents

Pour réaliser ce filtre, nous nous sommes entièrement basés sur les observations et les expériences décrites à la section 4.2.1.7. Nous avons décidé de ne pas reproduire les expériences réalisées par les auteurs de l'article sur lequel nous nous sommes basés car celles-ci se rapprochent fortement d'un environnement réel et aucune contrainte n'est imposée. Les seules différences qui pourraient survenir par rapport à un environnement réel est qu'ils ont utilisé des points d'accès en 802.11b et qu'un seul client est connecté par AP. Donc, le seul risque que nous avons est que, pour d'autres standards ou plus de stations connectées, nous sommes susceptibles d'avoir un trafic plus important sur le canal et donc une influence plus importante par rapport aux autres canaux. Cependant, les résultats de leurs expériences peuvent être utilisés comme une borne inférieure en fonction de ce que l'on pourra rencontrer dans une situation réelle.

On peut voir les résultats des poids calculés dans le tableau 5.6 pour l'environnement d'un kot étudiant. Un poids faible serait intéressant, signifiant qu'un canal n'est pas fortement influencé par les BSSID présents sur les autres canaux.

Canal	N ^{bre} BSSID	Poids
1	10	27.40
3	4	51.10
6	15	42.50
7	2	70.1
9	3	66
11	16	36.70
36	1	1
44	1	1
48	1	1

TABLE 5.6 – Tableau reprenant le poids de l'interférence des canaux adjacents calculé pour les différents canaux

Aussi étonnant que cela puisse être, ce n'est pas forcément le canal possédant le moins de BSSID qui sera le moins affecté par les canaux adjacents. À l'inverse, si on reprend la définition de l'indice de priorité pour ce filtre (section 4.3), il est spécifié qu'une fois les valeurs triées par ordre croissant, les résultats avec le poids le plus faible auront la priorité la plus haute. Dans le cas des valeurs présentes dans la figure 5.6, il s'agira alors du canal n° 1 qui possédera la plus grande priorité, suivi des canaux n° 11, 6, 3, etc.

Il s'avère alors que, dans le cas de cet exemple, le meilleur canal soit en fait l'un des trois possédant le plus de VAP. La différence avec le filtre précédent est que, ici, nous ne nous intéressons pas au canal qui consomme le plus d'*airtime* mais bien

à l'influence que peuvent avoir les canaux environnants sur les interférences présentes. Lorsque les bandes de fréquences se chevauchent et lors de transmissions, ce chevauchement peut causer des interférences et donc altérer les données. Dans le cas du 2.4 GHz, cela est un problème majeur à l'inverse du 5 GHz qui de par sa configuration réduit fortement cet impact comme le montre la figure 5.6 ci-dessus.

L'importance du filtre ne fait dès lors aucun doute quant à l'estimation de l'influence des interférences des canaux adjacents. De plus, ce filtre est complémentaire avec le filtre s'occupant de la consommation de l'*airtime* des trames *Beacon* afin de déterminer le meilleur canal à utiliser.

5.2.3 Récapitulatif de l'impact des filtres sur la couche 2

Dans cette section, nous revenons sur l'impact des filtres discutés précédemment afin d'en faire un récapitulatif.

Dans le tableau 5.7 sont listés les différents filtres que nous pouvons retrouver dans les sections ci-dessus. Cette liste offre un récapitulatif de l'importance de ces derniers et de leur efficacité.

Nom du filtre	Utilisable ?
Consommation de l' <i>airtime</i> : trames de données	✓
Perte des trames <i>Beacon</i>	✗
Délai des trames <i>Beacon</i>	✗
Estimation de la bande passante	✗
Retransmission des données	✗
Consommation de l' <i>airtime</i> : trame <i>Beacon</i>	✓
Interférence des canaux adjacents	✓
Nombre de stations connectées	✓

TABLE 5.7 – Tableau reprenant les différents protocoles testés

La validation dans un environnement sain où le trafic généré est contrôlé s'annonce comme un échec. La détermination du délai des trames *Beacon*, leur impact sur l'estimation de la bande passante potentielle, ...donnent des résultats négatifs bien qu'il soit possible de calculer l'*airtime* consommé par les trames de données. Quand il s'agit d'un environnement réel avec une multitude de réseaux Wi-Fi, les filtres en question s'attèlent correctement à la tâche afin de fournir un résultat. Cependant comme les réseaux sont inconnus et donc non maîtrisés, la validation de leur résultat est très difficile.

En annexe E se trouve un tableau reprenant le résultat final de notre algorithme qui donne une priorité pour chacun de ces filtres en fonction de notre environnement de validation contrôlé. Nous simplifions ce tableau au moyen de la figure 5.8.

Cette dernière nous indique que si on ne prend en compte que les filtres considérés comme utilisables, alors on peut s'attendre à avoir un résultat favorable à savoir l'élection de l'AP2 comme étant préférable à l'AP3.

N° test	Point d'accès	Priorité filtres utilisables
1	AP2	4
	AP3	4
2	AP2	5
	AP3	4
3	AP2	6
	AP3	4
4	AP2	7
	AP3	4
5	AP2	7
	AP3	4

TABLE 5.8 – Indice de priorité des filtres utilisables

Dans le cas de ce tableau, c'est majoritairement le filtre s'occupant de la consommation de l'*airtime* des trames de données qui vient faire pencher la balance en faveur de l'AP2 étant donné que cet environnement de validation ne tend uniquement qu'à valider l'algorithme en fonction du trafic généré.

5.2.4 Couche 3

Nous allons maintenant vérifier si l'évaluation de la bande passante grâce à des échanges de paquets UDP est possible pour l'évaluation de la qualité d'un hotspot tel que ceux proposés par *Proximus* (voir section 4.2.2.2). Cette méthode d'évaluation va injecter des trains de paquets et mesurer leurs délais afin de estimer la bande passante potentielle.

Après avoir réalisé quelques tests sur les hotspots de *Proximus*, il s'avère que le routeur n'autorise pas l'envoi de paquets à partir d'une station vers une adresse connue (par exemple : l'adresse de notre serveur) via le port 53. Nous avons aussi essayé de passer par les *DNS resolver* de *Google* (8.8.8.8 et 8.8.4.4) mais les paquets n'arrivent pas à destination. Donc, il n'autorise que la communication avec son "*DNS resolver*".

Comme nous ne pouvons pas faire de connexion directe avec l'adresse de notre serveur, le programme a dû être modifié. Nous créons des paquets DNS échangés en UDP avec comme adresse de destination, l'adresse du *DNS resolver* du routeur. Ensuite, étant donné qu'un échange d'informations est nécessaire entre l'émetteur et le destinataire, tous les paquets DNS échangés ont dû contenir des informations additionnelles (structure de données). Pour cela, les *Additional Record (AR)* ont été

utilisés sous la forme d'un champ de type *TXT*. Afin que notre algorithme échange correctement nos données, le programme qui envoie les trains de paquets envoie donc des requêtes DNS pour connaître l'adresse de notre serveur et ce dernier, sur lequel le programme de réception est lancé, répond à la station grâce à des réponses DNS.

Pour la requête, nous avons donc besoin de mettre le nom de domaine géré par notre serveur. Pour pouvoir être compris par les serveurs, ce nom de domaine doit être dans un certain format pour ensuite être traduit en hexadécimal. L'explication détaillée de ce format n'est pas décrite dans ce travail. Nous avons implémenté la gestion des questions de type A (IPv4) et de type AAAA (IPv6).

Pour la réponse, il contiendra l'adresse correspondant au type demandé.

Un problème qu'on peut rencontrer est que les réponses aux requêtes soient mises en cache par le *DNS resolver* et donc que les requêtes ne soient plus traitées par notre serveur. Pour contrer cela, il faut créer une requête avec une nouvelle demande à chaque fois. Ceci est réalisable en mettant un préfixe différent devant un nom de domaine. Il est alors nécessaire de pouvoir gérer tous les sous-domaines.

Comme précisé en section 3.2.3, deux types de serveurs ont été utilisés afin de valider l'approche : un serveur de l'UCL et un autre d'OVH. L'UCL nous fournissait un nom de domaine ("dnstfe.xxxxxxx.be"). Cependant, dû aux mesures de sécurité mises en place, avoir un serveur qui peut gérer tous les sous-domaines d'un domaine particulier n'est pas réalisable. OVH s'est alors imposé comme une solution viable.

5.2.4.1 Validation avec OVH

Pour pouvoir réaliser cette évaluation de bande passante, nous avons utilisé un autre serveur (voir section 3.2.3.2). Ce serveur est responsable de la zone "tests.dnstfe.be". Grâce à ce serveur, nous pouvons envoyer de nouvelles requêtes pour que la réponse soit toujours une réponse venant du serveur et non de la mémoire cache. Notre programme envoie des paquets vers notre serveur en faisant une requête DNS demandant l'adresse IP d'un sous-domaine de "tests.dnstfe.be" (par exemple : "06062016.tests.dnstfe.be"). Nous avons choisi d'utiliser l'*epoch*² en microseconde comme sous-domaine pour ne pas avoir de paquets identiques lors d'un test. Si nous sommes amenés à déployer cela à grande échelle et avec une multitude d'utilisateurs, il sera intéressant de faire un *hash* de cette valeur pour être certain qu'il n'y ait pas de tests avec le même nom de domaine réalisés par des stations différentes. Nous pourrions utiliser par exemple l'adresse MAC de la station pour créer le *hash* de cette valeur.

2. Temps en microseconde depuis le 1 janvier 1970

Nous avons donc créé une requête DNS valide, voir figure 5.7, contenant un *Additional Record* qui sert à transporter, dans le champ TXT, la structure nécessaire au fonctionnement du programme.

```
Domain Name System (query)
[Response In: 5]
Transaction ID: 0x0000
> Flags: 0x0100 Standard query
Questions: 1
Answer RRs: 0
Authority RRs: 0
Additional RRs: 1
▼ Queries
> 1464849993190024.tests.dnstfe.be: type AAAA, class IN
▼ Additional records
> 1464849993190024.tests.dnstfe.be: type TXT, class IN
```

FIGURE 5.7 – Requête DNS envoyée par le client

En envoyant cette requête au *DNS resolver* du routeur, notre serveur reçoit notre paquet modifié, voir figure 5.8. Nous pouvons voir que notre *Additional Record* de type TXT a été remplacé par un AR de type OPT. Ceci nous pose un problème car nous ne retrouvons donc plus les données de la structure. Nous pouvons aussi remarquer que le champ *Flags* est lui aussi modifié.

```
Domain Name System (query)
[Response In: 55]
Transaction ID: 0xad76
> Flags: 0x0000 Standard query
Questions: 1
Answer RRs: 0
Authority RRs: 0
Additional RRs: 1
▼ Queries
> 1464849993190024.tests.dnstfe.be: type AAAA, class IN
▼ Additional records
> <Root>: type OPT
```

FIGURE 5.8 – Requête DNS reçue par le serveur

Nous pouvons observer un problème similaire lors de la réponse du serveur vers le client. Nous envoyons aussi, à partir du serveur, une réponse DNS valide contenant un AR de type TXT transportant une structure (voir figure 5.9). Le client reçoit aussi ce paquet mais modifié et nous ne retrouvons maintenant aucun AR (voir figure 5.10).

Dans ce cas-ci, nous sommes de nouveau bloqués pour avoir une communication entre le client et le serveur.

Pour essayer de comprendre d'où vient ce problème, nous avons tenté de trouver le ou les serveurs qui modifient nos paquets. Nous avons utilisé trois outils différents

```

Domain Name System (response)
[Request In: 54]
[Time: 0.000551000 seconds]
Transaction ID: 0xad76
> Flags: 0x8180 Standard query response, No error
Questions: 1
Answer RRs: 1
Authority RRs: 0
Additional RRs: 1
▼ Queries
  > 1464849993190024.tests.dnstfe.be: type AAAA, class IN
▼ Answers
  > 1464849993190024.tests.dnstfe.be: type AAAA, class IN, addr 2001:41d0:e:870::1
▼ Additional records
  > 1464849993190024.tests.dnstfe.be: type TXT, class IN

```

FIGURE 5.9 – Réponse DNS envoyée par le serveur

```

Domain Name System (response)
[Request In: 1]
[Time: 0.053800000 seconds]
Transaction ID: 0x0000
> Flags: 0x8180 Standard query response, No error
Questions: 1
Answer RRs: 1
Authority RRs: 0
Additional RRs: 0
▼ Queries
  > 1464849993190024.tests.dnstfe.be: type AAAA, class IN
▼ Answers
  > 1464849993190024.tests.dnstfe.be: type AAAA, class IN, addr 2001:41d0:e:870::1

```

FIGURE 5.10 – Réponse DNS reçue par le client

pour observer nos paquets : *Traceroute* (A.6), *DNSTraceroute* (A.5) et *Tracebox* (A.7). Ces trois outils utilisent les paquets ICMP reçus afin de retracer le chemin emprunté par le paquet UDP. Vu que *Proximus_FON* ne laisse pas passer ce protocole, nous n'avons pas pu faire les tests en étant simplement connectés sur ce type de hotspots. Nous avons donc fait nos tests à partir du réseau privé que nous possédons chez *Proximus*. Nous n'avons donc aucune contrainte au niveau du client.

Traceroute utilise par défaut le port 53 (utilisé par le DNS) pour envoyer ses paquets UDP mais n'envoie pas de paquets DNS valides. Ce n'est pas possible avec cet outil de créer soi-même les paquets qui seront envoyés. Les paquets sont traités par les routeurs comme de simples paquets UDP contenant des données. Nous avons donc pu observer le chemin emprunté par les paquets UDP jusqu'à notre serveur (voir figure 5.11). Nous pouvons voir que les paquets passent par 3 serveurs de *Proximus* et ensuite par 3 autres serveurs de *OVH* pour ensuite arriver à notre serveur.

Le but d'utiliser *DNSTraceroute* était d'obtenir le chemin de la requête DNS et de le comparer au chemin emprunté par le paquet UDP. Nous avons remarqué que cet outil retourne le trajet emprunté par un paquet UDP jusqu'au serveur qui va gérer ensuite la requête DNS. Le trajet de cette requête DNS n'est pas donné. Donc si on

```

$ sudo traceroute -U 5.196.74.112
traceroute to 5.196.74.112 (5.196.74.112), 30 hops max, 60 byte packets
 1 dsldevice.lan (192.168.1.1)  3.857 ms  3.802 ms  3.769 ms
 2 129.198-4-62.adsl-dyn.isp.belgacom.be (62.4.198.129)  30.245 ms  30.262 ms
30.236 ms
 3 ae-92-100.iarstr3.isp.belgacom.be (91.183.241.168)  30.222 ms  31.742 ms  31.730 ms
 4 ae-22-1000.ibrstr6.isp.belgacom.be (91.183.246.110)  33.533 ms  33.505 ms  33.476 ms
 5 * * *
 6 be10-1162.rbx-g2-a9.fr.eu (213.251.130.58)  37.750 ms be10-1161.rbx-g1-a9.fr.eu (213.251.130.56)  32.159 ms be10-1162.rbx-g2-a9.fr.eu (213.251.130.58)  32.134 ms
 7 be100-1188.ldn-5-a9.uk.eu (91.121.128.89)  37.009 ms be100-1187.ldn-1-a9.uk.eu (91.121.128.87)  37.078 ms be100-1188.ldn-5-a9.uk.eu (91.121.128.89)  37.050 ms
 8 be10-1193.gra-g1-a9.fr.eu (91.121.128.90)  40.591 ms be10-1194.gra-g2-a9.fr.eu (91.121.128.92)  40.525 ms be10-1193.gra-g1-a9.fr.eu (91.121.128.90)  40.241 ms
 9 * * *
10 * * *
11 * * *
12 * ns3000438.ip-5-196-74.eu (5.196.74.112)  38.024 ms  38.004 ms

```

FIGURE 5.11 – *Traceroute* : Chemin emprunté par les paquets UDP

utilise notre serveur comme destination, l'échange des AR se fait correctement. Par contre lorsqu'on met un *DNS resolver*, les AR ne sont pas échangés.

Nous avons aussi utilisé *Tracebox*. Grâce à une option de cet outil, nous avons pu envoyer un paquet DNS personnalisé en ajoutant un AR à la requête DNS. Nous avons utilisé l'option "-p IP/udpdst=53,src=xxx/raw({byte_en_decimal})", cette option permet de créer un paquet UDP avec comme port de destination le port 53 et un port libre en source. Les données contenues dans *octet_en_decimal* sont tous les octets de la partie DNS du paquet en décimal (pas en hexadécimal comme on pourrait par exemple le voir dans Wireshark A.3). Avec cet outil, il n'est pas possible d'avoir le trajet de la requête DNS mais seulement le trajet du paquet UDP.

Après toutes ces observations, nous avons donc pu voir que les paquets UDP et les requêtes DNS ne sont pas traités de la même manière. Cependant, nous n'avons pas pu déterminer exactement quel serveur sur le chemin modifie nos requêtes et nos réponses DNS. Elles sont donc sûrement modifiées par le *DNS resolver* utilisé. Nous avons essayé aussi avec des AR contenant d'autres types que le TXT, tels que CN ou NS, mais nous avons rencontré le même problème.

La solution qui nous était laissée était d'utiliser seulement le champ contenant le nom de la requête pour le client et le champ contenant l'adresse IP de réponse pour le serveur. Dans notre configuration et pour le client, il est possible de remplacer le nom de sous-domaine par les données que nous voulons échanger, de mettre la structure de données utilisée comme un hash comme préfixe du domaine "test.dnstfe.be". La taille maximale sans que le paquet ne se fasse refuser est de 81 bytes pour le nom entier de la requête, il reste 63 bytes utilisables pour une structure ($81 - \text{longueur}(\text{tests.dnstfe.be}) - 2$, le 2 provient du caractère de fin, 00, et du

caractère contenant la longueur du préfixe).

Par contre, les données que le serveur veut envoyer doivent être contenues dans le champ de l'adresse IP de réponse et ne peuvent être plus grandes que 128 bits en utilisant des requêtes en IPv6. Nous avons essayé de changer la taille du champ de l'adresse mais le paquet n'est pas transmis dans ce cas. La plus grande structure que nous devons échanger contient 512 bits, nous avons tout de même essayé de réduire un maximum cette structure mais nous ne sommes pas arrivés à descendre en dessous des 128 bits.

Nous pouvons conclure que cette méthode d'évaluation de la bande passante grâce au protocole DNS ne sera pas réalisable à cause des vérifications et des modifications réalisées par les serveurs présents sur le chemin emprunté par les paquets DNS, qui probablement suppriment les informations additionnelles par mesure de sécurité.

5.3 Conclusion

Cette section s'occupe de faire le point sur la validation opérée sur la couche 2 et la couche 3.

Comme résumé en section 5.2.3, certains filtres se montrent prometteurs tandis que d'autres n'ont aucun impact pour déterminer le meilleur AP sur lequel se connecter. Lors de notre validation, quand un certain trafic est généré, l'algorithme permet globalement de déterminer le meilleur point d'accès. De plus si ce dernier est combiné avec les filtres s'occupant de l'environnement, cela peut venir renforcer cette détermination. Cependant, dans le cas où peu de trafics se trouvent présents sur un réseau, il faut uniquement se baser sur l'estimation des contraintes environnementales (adjacence de canaux, *airtime* des trames de *Beacon*, etc.).

Concernant la couche 3, tous les problèmes de validation établis en section précédente ne font que renforcer l'idée qu'une estimation de la bande passante sur les hotspots ouverts de *Proximus* n'est tout simplement pas faisable actuellement due aux contraintes fixées sur ce type d'infrastructure.

Il est donc difficile de dire si cette solution peut s'avérer intéressante pour un développement sur des appareils mobiles. Ci-dessous sont listées les limitations que nous avons rencontrées lors des différentes phases d'observation et de tests :

- Les antennes ne permettent pas toutes de se mettre en mode *monitor*.
- Lors du scan des différents canaux, la connexion internet n'est pas utilisable.
- La bande passante potentielle ne peut être déterminée.

- Les trames *Beacon* semblent avoir une priorité plus élevée que celle des données.
- Les paquets DNS sont probablement modifiés par le *DNS resolver*.
- etc.

Nous pouvons par contre voir le problème sous un autre angle et penser qu'on aura plus facilement des informations supplémentaires directement sur le routeur qu'en écoutant le canal avec la station. Nous allons donc détailler une autre solution, dans le chapitre 6, qui pourra reprendre une partie des filtres que nous avons implémentés.

Chapitre 6

Solution alternative

Suite aux différentes observations et différents tests réalisés dans les chapitres précédents, nous avons dû penser à une solution alternative pour pouvoir déterminer la qualité d'un réseau Wi-Fi à partir d'un appareil mobile. Nous avons pensé qu'il serait plus intéressant que le routeur envoie lui-même des informations importantes pour pouvoir déterminer sa qualité et que ces données soient réceptionnées par l'appareil mobile. La démarche que nous avons prise pour arriver à cette solution est résumée dans la section 5.3. Ce chapitre va détailler la réalisation de cette solution. Dans la section 6.1, les informations récoltées grâce au routeur seront détaillées. Ensuite, en section 6.2, il sera expliqué comment ces données sont récupérées et envoyées. Enfin, dans la troisième section (6.3), nous ferons état des améliorations possibles pour notre solution.

6.1 Informations récupérées

Dans cette section, nous allons expliquer les différentes informations qui vont être récupérées sur le routeur. Ces informations seront diffusées ensuite par le routeur permettant d'aider un terminal à faire un choix entre différents points accès. Une partie des informations récupérées ont déjà fait l'objet d'une implémentation dans le chapitre 4.

Nombre de stations connectées

Si l'on connaît le nombre de stations connectées, on pourra déterminer quel point d'accès sera le moins susceptible de subir du trafic. En effet, plus il y a de stations connectées, plus les chances que la bande passante soit utilisée sont grandes. Par contre, le nombre de stations ne nous dit pas la charge que chaque station consomme et si la connexion à internet sera rapide. On peut donc très bien avoir un point d'accès avec 5 stations connectées qui donnera de meilleures performances qu'un point d'accès avec une 1 seule station connectée.

Bande passante

Pour pouvoir apporter plus de précisions, nous avons trouvé intéressant de connaître la bande passante du routeur coté fournisseur. Le premier avantage est que, si un routeur propose un point d'accès sans avoir de connexion à internet, on verra tout de suite que ce n'est pas intéressant de s'y connecter. Le deuxième avantage est qu'on pourra déterminer une différence entre deux routeurs qui ont le même nombre de stations connectées par exemple. Cela pourra permettre de se connecter au point d'accès qui aura une bande passante élevée.

Airtime

Nous avons réutilisé le calcul de l'*airtime* consommé par les trames *Beacon* sur le canal défini dans le chapitre 4. Grâce à ces valeurs, on peut aussi différencier les points d'accès qui auront plus d'*airtime* pour envoyer leurs données. Vu que c'est un calcul théorique selon le standard utilisé par les points d'accès présents sur le canal, on pourra choisir celui sur le canal qui aura moins d'*airtime* utilisé. Pour un rappel sur ce qu'est l'*airtime* et son implémentation, voir section 4.2.1.4.

Influence des interférences des canaux adjacents

Nous avons réutilisé le calcul de l'influence des canaux adjacents (voir section 4.2.1.7). Grâce à ces données, il est possible de déterminer si un point d'accès est meilleur qu'un autre en fonction du nombre d'AP qui se trouvent sur le même canal ou sur les canaux proches. Cela permettra de différencier les point d'accès selon les risques d'interférences et de collisions qu'ils pourraient subir.

6.2 Implémentation

Nous avons réalisé l'implémentation et les tests sur routeur différent de celui décrit en section 3.2.4. Ce dernier étant fortement limité en espace disque (voir section 6.2.2). Dans cette section, il sera décrit comment les informations sont récoltées et comment elles sont traitées.

6.2.1 Récolte des données

Pour récupérer les données nécessaires, il a fallu faire quelques manipulations afin de les obtenir. Nous allons expliquer les démarches dans cette section.

Nombre de stations connectées

Il y a différentes manières de connaître les stations qui sont connectées sur le routeur.

Premièrement, le routeur garde en mémoire les adresses IP qu'il donne aux stations qui se connectent et qui en demandent une grâce au protocole DHCP. Ces adresses sont généralement stockées dans le fichier `/tmp/dhcp.lease` sous OpenWrt. On peut

donc facilement voir le nombre de stations qui se sont connectées dernièrement. Le désavantage d'utiliser ce fichier est que, si le temps de validité est long (par exemple 4h), le nombre d'adresses stockées sera probablement plus élevé que le nombre de stations réellement connectées.

La deuxième solution est d'utiliser la table de ARP, cette table contient les couples adresse MAC-adresse IP. On peut donc voir le nombre d'entrées que cette table contient pour en déduire le nombre de stations. Encore une fois, si le temps de réactualisation de cette table est long, nous n'aurons pas un nombre précis de stations connectées.

Enfin, la troisième solution est d'utiliser les options de la commande *iwinfo* pour voir combien de stations sont connectées sur les interfaces Wi-Fi. Cette solution est intéressante car elle permet de connaître le nombre précis de stations connectées. Par contre, avec cette solution, nous ne connaissons pas les stations qui seront connectées en câblé et qui pourront faire diminuer la bande passante.

Bande passante

Pour pouvoir calculer la bande passante, nous avons utilisé un outil *Speedtest-cli* [35]. C'est un outil écrit en python qui permet de faire un test de bande passante grâce au site *speedtest.net* [46]. Cet outil permet voir la rapidité de la réponse à une requête *ping* vers un serveur proche et ensuite voir la vitesse d'*upload* et de *download* en Mb/s. En faisant quelques tests l'un à la suite de l'autre, on peut remarquer que les vitesses ne sont pas constantes mais qu'elles restent dans le même ordre de grandeur. Ce test est aussi gourmand en temps et en ressources, il faut entre 10 et 30 secondes pour le réaliser et une utilisation d'un peu plus de 50% du cpu en moyenne sur le routeur. Sur 5 tests réalisés l'un à la suite de l'autre, nous avons mesuré en moyenne 179.2 paquets par seconde avec une taille moyenne de 972 bytes. Cela fait donc en moyenne 2.6MB de données échangées en 15 secondes. Vu que ce test est assez gourmand, il ne faudra pas le faire à intervalle trop régulier au risque de saturer le routeur et de diminuer ses performances. Cet outil a été choisi car il permet de faire un test de bande passante en ligne de commande et il n'y a pas besoin de mettre en place un serveur.

Airtime et Influence des interférences des canaux adjacents

La récupération de ces données se fait grâce à l'analyse des trames que l'on peut observer au niveau de la couche 2 et plus particulièrement dans les trames *Beacon*. Ces trames sont récupérées grâce à un *sniffing* en mode *monitor* sur les différents canaux du 2.4 Ghz. En mettant l'antenne en mode *monitor*, celle-ci ne pourra plus gérer les connexions avec les différentes stations, émettre un/des AP, etc. Ceci est donc une forte limitation car le but premier d'un routeur est de fournir une connectivité. Pour limiter l'impact de cet inconvénient, nous trouvons judicieux de faire une seule fois par jour maximum le *sniffing* en mode *monitor*. Ces données vont

- En 1, le nombre de stations connectées selon DHCP et ARP.
- En 2, le nombre de SSID diffusé par le routeur (*nb_ssid*), le nombre de stations connectées sur les interfaces Wi-Fi (*total_co*) et ensuite les différentes interfaces Wi-Fi avec leur SSID, leur nombre de connectés ainsi que leur canal respectifs.
- En 3, le résultat du *speedtest*.
- En 4, par canal, le nombre de BSSID, le résultat du filtre de l'interférence des canaux adjacents et le pourcentage de l'*airtime* consommé par les trames *Beacon*.

----- NB_STATION -----		----- INTERFERENCE -----			
1	dhcp 5	chan	bssid	prior	airtime
	arp 5	1	0	0.000000	0.000000
	nb_ssid 4	2	0	0.000000	0.000000
	total_co 1	3	0	0.000000	0.000000
2	wlan0 "OpenWrt" 0 36	4	0	0.000000	0.000000
	wlan1 "test_openwrt" 1 11	5	0	0.000000	0.000000
	wlan1-1 "Helloworld" 0 11	6	6	12.680000	1.732920
	wlan1-2 "bonjour" 0 11	7	0	0.000000	0.000000
	----- EVAL_BW -----	8	0	0.000000	0.000000
3	Ping 4.628000	9	1	16.930000	2.199219
	Down 75.570000	10	0	0.000000	0.000000
	Up 49.520000	11	3	7.710000	0.866460

FIGURE 6.2 – Paquet réceptionné par l'application

La démarche complète pour compiler et mettre en place nos paquets est expliquée à l'annexe G.

6.3 Améliorations possibles

Nous avons fait le choix d'envoyer les données dans un paquet UDP en *Broadcast*. Pour pouvoir comparer les différents points d'accès disponibles, il va falloir se connecter à tous les points d'accès que l'on voudra tester. Cela s'éloigne de l'objectif premier qu'est de pouvoir déterminer le meilleur point d'accès de manière la moins intrusive possible. Pour palier à ce problème, une amélioration intéressante serait de modifier le driver qui s'occupe d'envoyer les trames *Beacon* et d'y ajouter un nouvel élément qui contiendrait les différentes données déjà récoltées.

Si l'ajout de ces données est possible dans les trames *Beacon*, il sera donc intéressant de pouvoir adapter le *network-manager* pour qu'il puisse tenir compte de ces données et donc choisir automatiquement le meilleur AP. Si ce n'est pas possible, une solution serait de créer un nouveau type de trames avec son propre intervalle de génération.

Nous utilisons un programme en python pour pouvoir déterminer la bande passante. Installer python demande beaucoup d'espace disque pour un routeur, il serait donc intéressant de développer un programme, en C par exemple, qui ferait

le même type de travail tout en étant moins gourmand. Vu que nous avons maintenant accès au routeur, il sera intéressant aussi de rechercher une autre méthode d'évaluation de la bande passante, peut-être différente de celle décrite dans la section 4.2.2.2 et du *speedtest*, qui sera plus rapide et moins gourmande en CPU et en données échangées. Ce qui permettra d'augmenter la fréquence des tests et donc d'avoir une évaluation de la bande passante la plus proche de la réalité.

Il faut noter également que la solution proposé dans ce chapitre se rapproche de l'élément *QBSS load* que l'on peut retrouver dans certaines trames *Beacon* des appareils permettant de supporter le QBSS [57]. Le QBSS, *QOS enhanced Basic Service Set*, est un élément d'information que l'on peut retrouver dans le standard 802.11e et qui est principalement utilisé par *Cisco*. On peut y retrouver le nombre de stations connectées, le pourcentage de temps d'utilisation du canal, etc. Pour calculer l'utilisation du canal, ils prennent le pourcentage de temps durant lequel le mécanisme *carrier sense* est occupé. Il serait donc intéressant d'ajouter les champs que le QBSS propose dans notre solution.

Chapitre 7

Conclusion

Notre travail de fin d'étude s'est réalisé dans le contexte des réseaux Wi-Fi hotspots, particulièrement ceux de *Proximus*. En effet, dans les grandes villes, nous pouvons trouver presque constamment plusieurs hotspots accessibles. Le but de ce travail était de pouvoir déterminer celui qui donne les meilleures performances de manière non-intrusive.

Pour réaliser ce travail de fin d'étude, nous avons dû comprendre les différents mécanismes qui sont utilisés pour le fonctionnement d'un réseau Wi-Fi. Pour cela, nous nous sommes intéressés à tout ce qui se passait au niveau de la couche 2. Nous avons remarqué que beaucoup de trames et beaucoup d'informations sont échangées à ce niveau. Nous nous sommes donc concentrés principalement sur les informations que transportent les trames *Beacon*, trames qui sont constamment présentes car leur rôle est de prévenir la présence d'un point d'accès.

Nous avons ensuite cherché à récupérer un maximum d'informations grâce à ces trames afin de juger la qualité du réseau Wi-Fi. Différents articles parlent déjà de l'utilisation de ces trames pour estimer la bande passante ou le délai des trames *Beacon* par exemple. Nous avons reproduit ce qui était décrit dans la littérature afin de coupler toutes les données exploitables et de pouvoir déterminer la qualité d'un réseau Wi-Fi. Cependant des difficultés ont été rencontrées pour les valider. En effet, nous avons remarqué que, lors d'une utilisation d'une qualité de services, les trames *Beacon* ont une priorité plus grande que les trames de données et, par conséquent, ne subissent pas l'impact de la charge du trafic.

En plus de ces faits, il a été aussi remarqué que toutes les antennes ne peuvent pas se mettre en mode *monitor* afin d'écouter les trames sur la couche 2. C'est plus particulièrement le cas pour les smartphones. Suite à ces différentes observations, il nous était difficile de dire que notre solution pour estimer la qualité d'un réseau

Wi-Fi est la meilleure.

Nous nous sommes également intéressés au fonctionnement d'un hotspot de chez *Proximus* afin de pouvoir estimer la qualité d'un réseau Wi-Fi sans être authentifié mais tout en étant associé au point d'accès. Il a été remarqué que ceux-ci autorisent les échanges de paquets DNS. Nous avons alors tenté d'exploiter ce protocole afin d'estimer la bande passante. Nous nous sommes rendu compte que l'échange de requêtes fonctionne mais les ajouts de données additionnelles à ce protocole sont supprimées par des serveurs sur le chemin. Cela ne nous permet donc pas d'estimer précisément la bande passante.

Finalement, afin de pouvoir estimer la qualité d'un réseau, nous proposons une solution alternative. Au lieu de demander aux stations de faire tout le travail, nous proposons que ce soit le routeur qui fournisse les informations sur la qualité de son réseau afin d'être réceptionnées par les stations. Pour cela, nous avons développé différents paquets sur *OpenWrt* reprenant une partie du travail réalisé pour la couche 2 qui a été validé et en ajoutant d'autres informations. Le routeur va donc annoncer ces informations en diffusant un paquet UDP en *Broadcast*. Cette solution est probablement plus intéressante pour les stations mobiles.

7.1 Réalisations futures

Dans ce travail, nous nous sommes principalement intéressés aux informations que l'on peut récupérer grâce aux trames *Beacon*. Nous avons vu que ces trames ont une priorité plus grande que les trames de données. Il serait alors peut-être intéressant de regarder plus en profondeur les autres trames présentes sur la couche 2 et de trouver un autre moyen d'estimer la qualité du réseau Wi-Fi.

Suite à nos observations, il a été remarqué que la majorité des appareils mobiles tels que les smartphones ne sont pas capables de réaliser un scan sur la couche 2. En effet, leur antenne ne permet pas de se mettre en mode *monitor*. Ces appareils ne pourront donc pas faire une estimation de la qualité du réseau sans avoir au préalable une connexion avec le point d'accès. Cela va donc allonger le temps de l'estimation et ne pas respecter le caractère non-intrusif de la méthode de la couche 2. Une amélioration à apporter serait de permettre aux antennes présentes sur les appareils mobiles de se mettre en mode *monitor*.

Afin d'ajouter une plus-value aux réseaux *Proximus*, il pourrait être avantageux de mettre en place l'idée de la solution alternative (chapitre 6) pour leur hotspot et de coupler cette dernière avec une application mobile qui réceptionnerait ces données. Vu qu'il s'agit d'appareils mobiles, il serait intéressant que cette application se connecte au point d'accès le plus performant de manière dynamique sans que

l'utilisateur s'en aperçoive.

Une dernière amélioration serait de ne plus envoyer les informations grâce aux paquets UDP mais directement dans les trames *Beacon*. Vu que ces trames sont envoyées de manière régulière, cette amélioration n'aura pas d'impact sur le réseau. Pour cela, il faudra réaliser des modifications au niveau du driver qui s'occupe d'envoyer les trames *Beacon* et de pouvoir modifier le *network-manager* qui s'occupe de sélectionner le réseau Wi-Fi afin qu'il prenne en compte les données reçues dans les trames *Beacon*.

Annexe A

Outils utilisés pour les mesures

Pour réaliser notre Travail de Fin d'Étude, nous avons utilisé différents outils. Cette annexe a pour but de les répertorier et d'expliquer rapidement leur but et leur fonctionnement.

A.1 iw

Iw [33] est un outil en ligne de commande créé par Johannes Berg et d'autres développeurs. utilisé sur les réseaux sans fil. Il est développé par la communauté des développeurs Unix et, comme le kernel de Linux, il est constamment en développement. Actuellement, cet outil a atteint sa version 4.3. Il est utilisé par d'autres utilitaires tels que aircrack-ng pour mettre la carte en mode monitor.

A.2 TCPdump

TCPdump [12] est un outil en ligne de commande créé par Van Jacobson, Steven McCanne et Craig Leres. C'est un analyseur de paquets qui permet de voir le trafic à partir d'une interface réseau. *Libpcap* est la librairie liée à cet outil mais qui est utilisable par d'autres analyseurs de paquets. Elle est aussi distribuée séparément pour pouvoir compiler ou développer de tels outils. Nous avons utilisé la version 4.7.4.

A.3 Wireshark

Wireshark [66] est un outil développé par Gerald Combs. C'est une application qui permet le dépannage et l'analyse de réseaux informatiques, le développement de protocole, etc. Il permet de voir tout ce qu'il se passe sur le réseau à partir d'une interface. Nous avons utilisé la version 2.0.3.

A.4 DITG

D-ITG [6] est un outil en ligne de commande permettant de générer du trafic avec un grand choix dans la façon dont les paquets seront envoyés et avec précision. Cet outil supporte aussi bien l'IPv4 que l'IPv6 et il est capable de générer du trafic au niveau de la couche "réseau", "transport" et "application".

A.5 DNSdiag

DNSdiag [17] est un outil écrit en python qui permet de voir si le comportement des paquets DNS est normal et voir s'ils ne sont pas redirigés. Il y a 3 programmes disponibles : "*dnsping*", "*dnstraceroute*" et "*dnseval*". Le premier peut mesurer le temps de réponse de n'importe quel serveur DNS, comme un traditionnel "*ping*" mais en utilisant des requêtes DNS. Le second peut tracer le chemin utilisé par une requête DNS jusque sa destination pour être sûr qu'il ne soit pas redirigé ou hacké. Cela peut être réalisé en envoyant plusieurs requêtes différentes au même serveur et comparer le chemin entrepris. Le troisième va évaluer différents *DNS resolver* pour choisir le meilleur serveur. Il permet donc de comparer différents serveurs suivant leur performance (latence) et leur fiabilité (perte).

A.6 Traceroute

Traceroute [14] est un programme utilitaire qui permet de voir le chemin emprunté par des paquets de données pour aller d'une machine locale vers une autre machine connectée au réseau IP. Le principe de fonctionnement est qu'il va envoyer des paquets UDP avec un *Time-To-Live (TTL)* égal à 1 pour le premier paquet envoyé et en incrémentant ce TTL tant que la destination ne sera pas atteinte.

A.7 Tracebox

Tracebox [13] est un outil qui permet de détecter des *middleboxes* sur n'importe quel chemin, c'est-à-dire entre une source et une destination. Il peut être vu comme un outil similaire à *Traceroute* (A.6) car il utilise les réponses *ICMP* pour détecter les changements dans les paquets. *Tracebox* est capable de détecter les *middleboxes* grâce à l'observation que les messages *ICMP* ne sont pas tel que définis dans le RFC792.

A.8 iPerf

iPerf [30] est un outil qui réalise des mesures actives afin de déterminer la bande passante maximale sur des réseaux IP. Il prend en charge le réglage des différents paramètres liés à la synchronisation, les tampons et les protocoles (TCP ; UDP, SCTP

avec IPv4 et IPv6). Pour chaque test, il signale la bande passante, les pertes et d'autres paramètres.

A.9 DITG

DITG [6] est une plate-forme capable de produire du trafic de paquets reproduisant avec précision les processus stochastiques appropriés (exponentiel, uniforme, Cauchy normal, pareto, etc.) pour les deux variables aléatoires IDT (*Inter Departure Time*) et PS (*Packet size*).

Annexe B

Antennes compatibles mode *monitor*

Cette annexe reprend un tableau des différentes antennes, pilotes et fabricants de chipset Wi-Fi qui peuvent être utilisés pour réaliser un scan en mode *monitor*. Les éléments de ces tableaux n'ont pas été testés. Il se peut dès lors qu'ils présentent des limitations ou encore des dysfonctionnements. Cependant, selon leurs spécifications, elles devraient être fonctionnelles avec tout matériel.

De nombreuses listes existent indiquant la compatibilité de certains pilotes ou certaines antennes. Pour plus d'informations, nous vous invitons à consulter [49, 9].

Le tableau ci-dessous ne reprend que les cartes dont l'interface de connexion est en USB. Il ne sera alors pas fait état des cartes PCI, mini-PCI, etc. Les éléments de cette liste proviennent entièrement de [22].

Fabricant	Standards supportés	Identité produit	Chipset	Pilote
Alfa	802.11a/b/g/n	AWUS051NH	Ralink	RT2870sta ou RT3070
Belkin	802.11a/b/g	F5D9050 V.4	Ralink	RT2870sta
Edimax	802.11a/b/g/n/ac	EW7811UTC	Realtek	RTL8812au*
TP-Link	802.11a/b/g/n	TL-WDN3200	Ralink	RT5572sta*
TRENDware	802.11a/b/g/n/ac	TEW-805UB	Realtek	RTL8812au*

* : pilote du vendeur seulement

TABLE B.1 – Antennes compatibles mode *monitor*

En plus de ces éléments, on peut aussi noter que d'autres antennes existent mais avec des restrictions quant aux fréquences et aux standards supportés. Le tableau ci-dessous montre les antennes compatibles uniquement sur le 2.4 Ghz testé selon [53].

Fabricant	Standards supportés	Identité produit	Chipset	Pilote
Alfa	802.11b/g/n	AWUS036NHA	Atheros	AR9271
TP-LINK	802.11b/g/n	TP-WN722N	Atheros	AR9271
TP-LINK	802.11b/g/n	TP-WN722NC	Atheros	AR9271
Alfa	802.11b/g/n	AWUS036NH	Ralink	RT3070
Alfa	802.11b/g/n	AWUS036NEH	Ralink	RT3070
Alfa	802.11b/g/n	AWUS036H	Realtek	RTL8187L
Alfa	802.11b/g	AWUS036EW	Realtek	RTL8187L

TABLE B.2 – Antennes compatibles mode *monitor* sur 2.4Ghz

Annexe C

Valeurs des données utilisées

Dans cette annexe sont reprises les valeurs utilisées pour les différents calculs de l'*airtime*. Comme annoncé dans le chapitre 4, ces données sont également disponibles dans [25] mais afin d'éviter de longues recherches, elles sont listées ci-dessous.

	802.11b	802.11g (legacy mode)	802.11g	802.11a
SIFS [μ s]	10	10	10	16
<i>aSlotTime</i> [μ s]	20	20	9	9
<i>aCWmin</i>	15	15	31	31

TABLE C.1 – Valeurs pour SIFS, *aSlotTime* et *aCWmin*

	HR DSSS		OFDM
	1 Mb/s	2, 5.5 ou 11Mb/s	
<i>en-tête preamble</i> [μ s]	48	24	4
<i>preamble</i> [μ s]	144	72	16

TABLE C.2 – Valeurs pour le *preamble* et *sont en-tête*

	Rates [Mb/s]					
	1	2	5.5	11	6	9
bits/symbole	1	2	4	8	24	36
symboles/sec	1000000	1000000	1375000	1375000	250000	250000
	Rates [Mb/s]					
	12	18	24	36	48	56
bits/symbole	48	72	96	144	192	216
symboles/sec	250000	250000	250000	250000	250000	250000

TABLE C.3 – Valeurs pour le nombre de bits par symbole et le nombre de symboles par seconde

Annexe D

Pseudo code pour le calcul des pertes

Cette annexe a la tache de lister l'intégralité du pseudo-code relatif au calcul des pertes des trames *Beacon* développé dans la section 4.2.1.3.

Algorithme D.1 Calcul de la perte des trames *Beacon*

```

1: for each node in list_BSSID do
2:   for each Beacon in node do
3:     if Beacon.next! = Beacon.seq_nbr + inc then
4:       prev_seq_number  $\leftarrow$  Beacon.seq_nbr
5:       expected_seq  $\leftarrow$  seq_nbr + inc
6:       timestamp_beacon  $\leftarrow$  Timestamp of the Beacon
7:       node_frames  $\leftarrow$  node where the BSSID of Beacon is found (in
         all_frame)
8:       for each frame in node_frames do
9:         if frame.seq_nbr  $\geq$  expected_seq and frame.timestamp >
           timestamp_beacon then
10:          if the detected frame is a Beacon then
11:            backward_max  $\leftarrow$  Index of previous frame with same BSSID
12:            while backward_max > 0 and frame.seq_nbr <
              frame[backward_max].seq_nbr do
13:              nbr_frame_backward ++
14:              backward_max  $\leftarrow$  Index of previous frame with same BSSID
15:            end while
16:          else {The detected if is of another type}
17:            if frame.seq_number - inc! = prev_seq_nbr and
              frame.timestamp - timestamp_beacon  $\leq$  beacon_int then
18:              nbr_not_beacon_loss  $\leftarrow$  frame.seq_nbr - prev_seq_nbr - 1
19:            end if
20:            count_frame  $\leftarrow$  count_frame + 1
21:            prev_seq_nbr  $\leftarrow$  frame.seq_nbr
22:          end if

```

```

23:         end if
24:     end for
25:     if Beacon has been detected then
26:          $count\_frame \leftarrow count\_frame - nbr\_frame\_backward$  {could
           possibly be minus 0}
27:          $diff\_seq \leftarrow frame.seq\_nbr - expected\_seq - count\_frame -$ 
            $nbr\_not\_loss\_beacon$ 
28:         for  $t$  to  $MAX\_LOST$  do
29:             if  $diff\_seq < inc * t$  then
30:                  $frame\_lost \leftarrow t - 1$ 
31:             end if
32:         end for
33:     end if
34:      $total\_frame\_loss += frame\_lost$ 
35: end if
36: end for
37:  $percentage\_loss \leftarrow [total\_frame\_loss / (node.nbr\_frame +$ 
    $total\_frame\_loss)] * 100$ 
38: print "Total frame loss percentage : "  $percentage\_loss$ 
39: end for

```

Annexe E

Résultats utilisés pour la validation

Dans cette annexe seront présents les résultats bruts des différents tests réalisés durant ce travail. On peut retrouver en E.2, les résultats obtenus directement depuis DITG. Ce tableau ne reprend que la variation de l'AP2 car il s'agit du seul, parmi nos deux points d'accès qui profite d'une variation dans le trafic généré.

Les tableaux E.3 nous donnent les résultats bruts provenant des filtres lors de la phase de validation développée dans le chapitre 5. Dans ce tableau, on retrouve pour chaque filtre son résultat en fonction du test qui a été réalisé et du point d'accès concerné. Ces valeurs sont, plus précisément, les médianes calculées en fonction des six répétitions de tests.

Le dernier tableau E.4, représente l'indice de priorité donné à chaque filtre en fonction du résultat délivré dans le tableau E.3. La mise en place de ces indices a été définie précédemment, dans la section 4.3.

Pour rappel, les tests effectués sont les suivants :

N° test	Paquets/sec	Octets
1	200	1000
2	500	1000
3	1000	1000
4	2000	1000
5	4000	1000

TABLE E.1 – Tableau reprenant les valeurs de la charge produite sur l'AP2

Pour l'AP3, il ne s'agit que d'un envoi de 10 paquets de 50 octets par seconde.

TABLE E.2 – Résultats fournis via DITG pour l'AP2

N° du test	N° répétition	Débit envoi (Kbit/s)	Nbr paquets envoyés	Pourcentage de pertes	Débit réception (Kbit/s)
1	1	1600	15999	0,05%	1599
	2	1600	15999	0,05%	1599
	3	1600	16000	0,26%	1595
	4	1600	15999	0,04%	1599
	5	1600	15998	0,09%	1598
	6	1600	16000	0,26%	1597
2	1	4000	37497	0,28%	3988
	2	4000	37496	0,28%	3988
	3	4000	37494	0,66%	3973
	4	4000	37496	0,32%	3986
	5	4000	37500	0,40%	3983
	6	4000	37497	0,37%	3986
3	1	8000	79995	0,00%	7999
	2	7999	79991	0,00%	7998
	3	8000	79998	0,10%	7990
	4	8000	79988	0,01%	7998
	5	8000	79995	0,10%	7990
	6	7999	79991	2,93%	7764
4	1	12235	122344	23,41%	9365
	2	11900	118980	20,46%	9452
	3	13350	133449	31,75%	9097
	4	12473	124700	23,08%	9578
	5	12983	129830	27,57%	9391
	6	13343	133408	32,81%	8953
5	1	11939	298449	24,91%	9772
	2	11893	295985	23,57%	9892
	3	11810	295268	29,04%	9339
	4	11972	299281	22,15%	10043
	5	12105	302570	23,49%	9903
	6	11991	299786	22,28%	9956

TABLE E.3 – Résultats des filtres

N° du test	AP	<i>Airtime (Beacon)</i> [%]	Nbr. BSSID	Nbr. stations	Pertes <i>Beacon</i> [%]	Médiane délais <i>Beacon</i> [μ s]
1	AP2	1,67	1	2	6,12	383
	AP3	1,67	1	2	5,04	383
2	AP2	1,67	1	2	6,12	383
	AP3	1,67	1	2	6	383
3	AP2	1,67	1	2	11,355	383,5
	AP3	1,67	1	2	9,18	383
4	AP2	1,67	1	2	8,08	496,5
	AP3	1,67	1	2	5,145	383
5	AP2	1,67	1	2	7,205	488
	AP3	1,67	1	2	6,12	383

N° du test	AP	<i>Airtime</i> (Donnée) [%]	Estimation bande passante [Mbps]	Retransmissions [%]	Moyenne paquets/sec
1	AP2	12,54	12,28	3,875	375
	AP3	0,675	2,515	15,485	20
2	AP2	29,38	11,36	7,57	793
	AP3	0,54	4,4	1,89	24
3	AP2	53,535	12,13	8,17	1550
	AP3	0,54	2,57	19,185	34
4	AP2	77,125	13,27	10,72	2154
	AP3	0,695	2,54	8,535	17
5	AP2	78,4	12,055	11,85	2241
	AP3	0,845	4,69	8,135	21

TABLE E.4 – Indice de priorité donné aux filtres

N° du test	AP	Priorité totale	Priorité filtres utilisables	Priorité <i>airtime</i> (<i>Beacon</i>)	Priorité canaux adjacents	Priorité nbr. stations
1	AP2	8	4	1	1	1
	AP3	10	4	1	1	1
2	AP2	9	5	1	1	1
	AP3	9	4	1	1	1
3	AP2	12	6	1	1	1
	AP3	10	4	1	1	1
4	AP2	12	7	1	1	1
	AP3	10	4	1	1	1
5	AP2	12	7	1	1	1
	AP3	9	4	1	1	1

N° du test	AP	Priorité pertes <i>Beacon</i>	Priorité délais <i>Beacon</i>	Priorité <i>airtime</i> (Donnée)	Priorité bande passante
1	AP2	1	1	1	2
	AP3	1	1	1	4
2	AP2	1	1	2	2
	AP3	1	1	1	3
3	AP2	3	1	3	2
	AP3	1	1	1	4
4	AP2	1	2	4	2
	AP3	1	1	1	4
5	AP2	1	2	4	2
	AP3	1	1	1	3

Annexe F

Problèmes rencontrés avec les smartphones

Nous détaillons dans cette annexe les différents problèmes que nous avons rencontrés avec les smartphones. Nous nous sommes intéressés seulement aux smartphones qui utilisent *Android* comme système d'exploitation, celui-ci est plus ouvert que *iOS* ou *Windows*.

Premièrement, pour pouvoir se mettre en mode *monitor*, le smartphone doit être en *root*. C'est-à-dire qu'il faut avoir les droits d'administrateurs. Cela va être un frein pour avoir un déploiement à une grande échelle avec beaucoup d'utilisateurs. En effet, on va devoir les obliger à faire des manipulations, parfois risquées, pour pouvoir utiliser notre programme.

Deuxièmement, toutes les antennes ne peuvent pas supporter le mode *monitor*. Les fabricants d'antennes ne permettent pas ce mode car il est assez gourmand en énergie. Le mode *monitor* est généralement offert avec les drivers SoftMAC et, la plupart du temps, ce sont des drivers FullMAC qui sont utilisés afin de réduire la consommation d'énergie. Les FullMAC sont implémentés au niveau hardware. Ce mode n'est donc pas mis en place mais il existe un programme, NexMon, qui va étendre le firmware du Nexus 5 pour pouvoir l'activer. Le problème de cette solution est qu'il désactive le *wpa_supplicant* et d'autres services qui permettent d'utiliser le Wi-Fi de manière normale. Il n'est donc plus possible de se connecter à un point d'accès afin de profiter de sa connectivité [56].

Finalement, en dehors du mode *monitor*, les informations que le smartphone est capable de réceptionner sont limitées. Il ne peut récupérer que :

- SSID
- BSSID

- Informations sur le chiffrement
- Niveau du signal/bruit
- Fréquence
- Timestamp

On ne retrouve pas certaines informations importantes comme le TU par exemple. En effet, cette valeur est importante pour connaître exactement le nombre de trames et le délai des trames *Beacon*.

Suite à ces différentes observations, nous avons décidé de ne pas utiliser les smartphones mais d'utiliser un Raspberry Pi qui a plus de libertés tout en restant facilement transportable.

Annexe G

Compiler et installation pour OpenWrt

G.1 Compiler pour OpenWrt

Dans cette partie, nous allons détailler la démarche pour pouvoir compiler nos programmes dans une architecture d'un routeur sous *OpenWrt*.

Installation des dépendances

```
$ sudo apt-get install git-core build-essential libssl-dev  
$ sudo apt-get install libncurses5-dev unzip subversion mercurial
```

Clone du répertoire Git, correspondant à la version d'*OpenWrt*

```
$ git clone git://git.openwrt.org/14.07/openwrt.git
```

Installer des feed additionnels

```
$ cd openwrt  
$ ./scripts/feeds update -a  
$ ./scripts/feeds install -a
```

Copier le dossier du paquet (**evaluate** par exemple) dans `package/utils` et vérifier que tout est bien installé

```
$ make defconfig  
$ make prereq  
$ make menuconfig
```

Dans le menu, vérifier que l'architecture du routeur est bien choisie et que le paquet est choisi (avec un "M" devant), sinon le paquet ne sera pas compilé. Ensuite, réaliser ces deux commandes suivante (cela peut prendre du temps la première fois)

```
$ make tools/install
$ make toolchain/install
```

Compiler le package

```
$ make package/evaluate/compile
```

Pour compiler un nouveau package ou recompiler après avoir fait une fois cette manipulation

```
$ cp -r /home/scanning package/utils/
$ make menuconfig
$ make tools/install
$ make toolchain/install
$ make package/scanning/compile
```

Le binaire se trouve dans `bin/mpc85xx/packages/base/`. Il faut maintenant le mettre sur le routeur, l'installer et l'exécuter

```
$ opkg install evaluate_version.ipk
$ evaluate
```

G.2 Installation des packages

G.2.1 speedtest

Pour pouvoir exécuter le speedtest, il faut installer différents packages sur le routeur qui prendront 5MB.

```
$ opkg update
$ opkg install python-mini
$ opkg install python
$ opkg install python-openssl
$ opkg install python
$ opkg install python-expat
```

Ensuite suivre les instructions sur le Github de l'auteur (partie *Just download*) [35].

G.2.2 scanning et evaluate

Il faut suivre la démarche détaillée ci-dessus (G.1) en copiant les dossier `evaluate` et `scanning`.

Table des figures

2.1	Composant de l'architecture	6
2.2	canaux sur le 2.4 GHz (20 MHz largeur de bande)	12
2.3	Canaux sur le 5 GHz (20 MHz largeur de bande)	13
2.4	Couche liaison de données et physique 802.11	13
2.5	Contrôle d'accès de la sous-couche MAC	15
2.6	Format de la trame MAC [25]	16
2.7	Algorithme de la méthode d'accès DCF [62]	18
2.8	Hidden node problem	19
2.9	Trame RTS [25]	19
2.10	Trame CTS [25]	19
3.1	Structure d'une trame capturée	26
3.2	SSID et BSSID chez des VAP de <i>Proximus</i>	28
3.3	Numéro de séquence pour les trames de management	29
3.4	Numéro de séquence pour les trames de données	30
3.5	Timestamp envoyé dans les trames <i>Beacon</i>	30
3.6	Graphe représentant l'intervalle entre la génération des trames <i>Beacon</i>	32
3.7	Signal capturé par A	34
3.8	Signal capturé par B	34
3.9	Information sur les priorités depuis les trames <i>Beacon</i>	35
3.10	Redirection vers hotspot ouvert	36
3.11	Réponse à la demande d'authentification de <i>Proximus</i>	37
3.12	Capture de packets DNS sur le hotspot de <i>Voo</i>	37
4.1	Vue générale de l'algorithme	40
4.2	Structure de données afin de mémoriser les trames	41
4.3	Incrément de 3 pour le numéro de séquence d'un BSSID	46
4.4	Structure secondaire	47
4.5	Transmission de <i>Beacon</i> d'un point d'accès [64]	54
4.6	Variation du débit en fonction des interférences des canaux adjacents[65]	58
4.7	<i>The gagged stations problem</i>	61

4.8	Résultats expérimentaux obtenus en présence d'un trafic à un taux constant avec une variation des taux de 0 à 64 Mbps [21]	63
4.9	Distribution des paquets dans un flot REACH [20]	64
5.1	Organisation de l'environnement de test sain	70
5.2	Nombre de paquets envoyés par seconde	72
5.3	Variation de la consommation de l' <i>airtime</i> des trames de données pour différent trafic	75
5.4	Médiane des délais observés pour les tests allant de 1 à 5 (voir tableau E.1)	76
5.5	Pourcentage de perte des trames <i>Beacon</i> les tests allant de 1 à 5 (voir tableau E.1	78
5.6	Données échangées et pourcentage de retransmission pour les tests allant de 1 à 5 (voir tableau E.1	80
5.7	Requête DNS envoyée par le client	87
5.8	Requête DNS reçue par le serveur	87
5.9	Réponse DNS envoyée par le serveur	88
5.10	Réponse DNS reçue par le client	88
5.11	<i>Traceroute</i> : Chemin emprunté par les paquets UDP	89
6.1	Paquet envoyé par le package evaluate	96
6.2	Paquet réceptionné par l'application	97

Liste des tableaux

3.1	Tableau reprenant les champs importants rencontrés dans les trames <i>Beacon</i>	28
3.2	Tableau reprenant les différents protocoles testés	38
4.1	Tableau reprenant les filtres développés et envisagés	43
4.2	Comparaison	44
5.1	Add caption	71
5.2	Tableau reprenant les valeurs de la charge produite sur l'AP2	73
5.3	Tableau reprenant les filtres développés et envisagés	74
5.4	airtime beacon	81
5.5	Tableau reprenant l' <i>airtime</i> calculé pour les différents canaux dans une salle de l'UCL	82
5.6	Tableau reprenant le poids de l'interférence des canaux adjacents cal- culé pour les différents canaux	83
5.7	Tableau reprenant les différents protocoles testés	84
5.8	Indice de priorité des filtres utilisables	85
B.1	Antennes compatibles mode <i>monitor</i>	107
B.2	Antennes compatibles mode <i>monitor</i> sur 2.4Ghz	108
C.1	Valeurs pour SIFS, <i>aSlotTime</i> et <i>aCWmin</i>	109
C.2	Valeurs pour le <i>preamble</i> et sont en-tête	109
C.3	Valeurs pour le nombre de bits par symbole et le nombre de sym- boles par seconde	109
E.1	Tableau reprenant les valeurs de la charge produite sur l'AP2	113
E.2	Résultats fournis via DITG pour l'AP2	114
E.3	Résultats des filtres	115
E.4	Indice de priorité donné aux filtres	116

Liste des Algorithmes

4.1	Calcul de l'incrément du numéro de séquence des trames <i>Beacon</i> . . .	47
4.2	Calcul de l' <i>airtime</i> pour les trames Beacon	53
4.3	Calcul de l' <i>airtime</i> pour les trames de données	54
D.1	Calcul de la perte des trames <i>Beacon</i>	111

Bibliographie

- [1] B. ABOBA et D. SIMON. *The EAP-TLS Authentication Protocol*. RFC 5216. 2008. URL : <http://www.rfc-editor.org/info/rfc5216>.
- [2] Wi-Fi ALLIANCE. *Wi-Fi CERTIFIED WMM Programs*. URL : <http://www.wi-fi.org/discover-wi-fi/wi-fi-certified-wmm-programs> (visité le 16/07/2016).
- [3] Johannes BERG. *Radiotap*. URL : <http://www.radiotap.org/> (visité le 02/10/2016).
- [4] Giuseppe BIANCHI. « Performance analysis of the IEEE 802.11 distributed coordination function ». In : *Selected Areas in Communications, IEEE Journal on* 18.3 (2000), p. 535–547.
- [5] Sanjit BISWAS et al. « Large-scale Measurements of Wireless Network Behavior ». In : *SIGCOMM Comput. Commun. Rev.* 45.4 (août 2015), p. 153–165. URL : <http://doi.acm.org/10.1145/2829988.2787489>.
- [6] Alessio BOTTA, Alberto DAINOTTI et Antonio PESCAPÈ. *D-ITG, Distributed Internet Traffic Generator*. URL : <http://traffic.comics.unina.it/software/ITG/> (visité le 22/03/2016).
- [7] B. Aboba L. Blunk J. Vollbrecht J. CARLSON et H. Levkowetz ED. *Extensible Authentication Protocol (EAP)*. RFC 3748. 2004. URL : <http://www.rfc-editor.org/info/rfc3748>.
- [8] CISCO. *Vni mobile forecast highlights*. URL : http://www.cisco.com/c/dam/assets/sol/sp/vni/forecast_highlights_mobile/index.html (visité le 15/07/2016).
- [9] Aircrack ng COMMUNITY. *Compatibility drivers*. URL : http://www.aircrack-ng.org/doku.php?id=compatibility_drivers (visité le 09/04/2016).
- [10] CentOS COMMUNITY. *CentOs*. URL : <https://www.centos.org/> (visité le 25/05/2016).
- [11] Linux Wireless COMMUNITY. *Modes*. 2015. URL : <https://wireless.wiki.kernel.org/en/users/Documentation/modes> (visité le 08/05/2016).

- [12] Wireshark COMMUNITY. *TCPdump & Libcap*. URL : <http://www.tcpdump.org/> (visité le 25/05/2016).
- [13] Gregory DETAL et al. « Revealing Middlebox Interference with Tracebox ». In : *Proceedings of the 2013 ACM SIGCOMM conference on Internet measurement conference*. ACM, 2013.
- [14] Linux DEVLOPPERS. *traceroute(8) - Linux man page*. URL : <http://linux.die.net/man/8/traceroute> (visité le 25/11/2015).
- [15] W. Kumari O. Gudmundsson P. EBERSMAN et S. SHENG. *Captive-Portal Identification Using DHCP or Router Advertisements (RAs)*. RFC 7710. 2015. URL : <http://www.rfc-editor.org/info/rfc7710>.
- [16] H. Haverinen ED. et J. Salowey ED. *Extensible Authentication Protocol Method for Global System for Mobile Communications (GSM) Subscriber Identity Modules (EAP-SIM)*. RFC 4186. 2006. URL : <http://www.rfc-editor.org/info/rfc4186>.
- [17] Babak FARROKHI. *DNS Diagnostics and Performance Measurement Tools*. URL : <https://github.com/farrokhi/dnsdiag> (visité le 01/06/2016).
- [18] Matthew GAST. *802.11 wireless networks : the definitive guide*. " O'Reilly Media, Inc.", 2005.
- [19] Matthew GAST. *TTLS and PEAP Comparison*. URL : <http://www.opus1.com/www/whitepapers/ttlsandpeap.pdf> (visité le 25/11/2015).
- [20] Emanuele GOLDONI, Giuseppe ROSSI et Alberto TORELLI. « Assolo : an Efficient Tool for Active End-to-End Available Bandwidth Estimation ». In : *International Journal on Advances in Systems and Measurements* 2.4 (2009), p. 283–292.
- [21] Emanuele GOLDONI et Marco SCHIVI. « End-to-end available bandwidth estimation tools, an experimental comparison ». In : *Traffic Monitoring and Analysis*. Springer, 2010, p. 171–182.
- [22] HJ HEINS. *Assistance Linux matériel réseau sans fil*. URL : <http://linux-wless.passys.nl/> (visité le 09/04/2016).
- [23] Xueheng HU et al. « Is There WiFi Yet ? How Aggressive WiFi Probe Requests Deteriorate Energy and Throughput ». In : *arXiv preprint arXiv :1502.01222* (2015).
- [24] IEEE. *What 802.11 is doing*. 2015. URL : <http://www.ieee802.org/11/presentation.html> (visité le 27/04/2016).

- [25] « IEEE Standard for Information technology–Telecommunications and information exchange between systems Local and metropolitan area networks–Specific requirements Part 11 : Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications ». In : *IEEE Std 802.11-2012 (Revision of IEEE Std 802.11-2007)* (2012), p. 1–2793. DOI : 10.1109/IEEESTD.2012.6178212.
- [26] ALFA NETWORK INC. *AWUS051NH*. URL : https://www.alfa.com.tw/products_show.php?pc=67&ps=241 (visité le 19/05/2016).
- [27] National INSTRUMENTS. *WLAN - 802.11 a,b,g and n*. URL : <http://www.ni.com/tutorial/7131/en/> (visité le 02/05/2016).
- [28] Manish JAIN et Constantinos DOVROLIS. « Pathload : A measurement tool for end-to-end available bandwidth ». In : *In Proceedings of Passive and Active Measurements (PAM) Workshop*. Citeseer. 2002.
- [29] Amit P JARDOSH et al. « Understanding link-layer behavior in highly congested IEEE 802.11 b wireless networks ». In : *Proceedings of the 2005 ACM SIGCOMM workshop on Experimental approaches to wireless network design and analysis*. ACM. 2005, p. 11–16.
- [30] Dugan JON et al. *iPerf - The network bandwidth measurement tool*. URL : <https://iperf.fr/> (visité le 14/03/2016).
- [31] Thenu KITTAPPA. *Virtual Access Points - Performance Impacts in an 802.11 environment and Alternative*. URL : <http://community.arubanetworks.com/aruba/attachments/aruba/115/1358/1/AppNote.MultipleBSSIDs.pdf> (visité le 12/04/2016).
- [32] Heeyoung LEE et al. « Available bandwidth-based association in IEEE 802.11 Wireless LANs ». In : *Proceedings of the 11th international symposium on Modeling, analysis and simulation of wireless and mobile systems*. ACM. 2008, p. 132–139.
- [33] Jared LEMKE. *Linux wireless*. URL : <https://wireless.wiki.kernel.org/en/users/Documentation/iw> (visité le 28/09/2015).
- [34] LINKSYS. *Differentiating the 5 GHz and 2.4 GHz frequency bands*. URL : <http://www.linksys.com/us/support-article?articleNum=134478> (visité le 27/04/2016).
- [35] Matt MARTZ. *speedtest-cli*. URL : <https://github.com/sivel/speedtest-cli> (visité le 22/07/2016).
- [36] MICROSOFT. *Présentation du protocole EAP (Extensible Authentication Protocol)*. URL : <https://technet.microsoft.com/fr-fr/library/bb457039.aspx> (visité le 12/11/2015).
- [37] Thompson MIKE et Green PETER. *Raspbian*. URL : <https://www.raspbian.org/> (visité le 06/07/2016).

- [38] David Camps MUR. *Linux Wi-Fi open source drivers - mac80211, ath9k/ath5k* - . URL : http://www.campsmur.cat/files/mac80211_intro.pdf (visité le 04/07/2016).
- [39] Andrew von NAGY. *SSID Overhead Calculator*. URL : <http://www.revolutionwifi.net/revolutionwifi/p/ssid-overhead-calculator.html> (visité le 15/03/2016).
- [40] Andrew von NAGY. *Wireless QoS Part 5 - Contention Window and Final Thoughts*. URL : <http://www.revolutionwifi.net/revolutionwifi/2010/08/wireless-qos-part-5-contention-window.html> (visité le 02/05/2016).
- [41] Rasika NAYANAJITH. *802.11 Mgmt : Action Frames*. URL : <https://mrncciew.com/2014/10/09/802-11-mgmt-action-frames/> (visité le 22/03/2016).
- [42] Rasika NAYANAJITH. *CWSP - 4 Way Handshake*. URL : <https://mrncciew.com/2014/08/19/cwsp-4-way-handshake/> (visité le 25/03/2016).
- [43] Anthony J NICHOLSON et al. « Improved access point selection ». In : *Proceedings of the 4th international conference on Mobile systems, applications and services*. ACM. 2006, p. 233-245.
- [44] NXADM. *PROXIMUS AUTO FON and TelenetWifree (Belgium) from GNU Linux (or Windows 7)*. URL : https://nxadm.wordpress.com/2015/04/14/post-it-proximus_auto_fon-and-telenetwifree-belgium-from-gnulinux-or-windows-7/ (visité le 25/11/2015).
- [45] *Official IEEE 802.11 working group project timeline*. 2016. URL : http://grouper.ieee.org/groups/802/11/Reports/802.11_Timelines.htm (visité le 27/04/2016).
- [46] OOKLA. *Speedtest.net by Ookla - The Global Broadband Speed Test*. URL : <http://www.speedtest.net/> (visité le 22/07/2016).
- [47] OVH. *OVH.com*. URL : <https://www.ovh.com/fr/> (visité le 23/05/2016).
- [48] Ashish PATRO, Srinivas GOVINDAN et Suman BANERJEE. « Observing home wireless experience through wifi aps ». In : *Proceedings of the 19th annual international conference on Mobile computing & networking*. ACM. 2013, p. 339-350.
- [49] Xose Vazquez PEREZ. *Existing Linux Wireless driver*. URL : <https://wireless.wiki.kernel.org/en/users/drivers> (visité le 09/04/2016).
- [50] PROXIMUS. *Activez Smart Wi-Fi maintenant*. URL : http://www.proximus.be/fr/id_cr_wifispots/particuliers/produits/internet/internet-mobile/smart-wi-fi-gratuit.html (visité le 15/07/2016).
- [51] RASPBERRY. *Raspberry Pi 2, Model B*. URL : <https://cdn-shop.adafruit.com/pdfs/raspberrypi2modelb.pdf> (visité le 06/07/2016).

- [52] RASPBERRY. *Raspberry Pi Blog*. URL : <https://www.raspberrypi.org/blog/> (visité le 18/02/2016).
- [53] RAYMOND. *Best Compatible USB Wireless Adapter for BackTrack 5, Kali Linux and Aircrack-ng*. 2013. URL : <https://www.raymond.cc/blog/best-compatible-usb-wireless-adapter-for-backtrack-5-and-aircrack-ng/> (visité le 24/03/2016).
- [54] Alan RITACCO, Craig WILLS et Mark CLAYPOOL. « How's My Network? A Java Approach to Home Network Measurement ». In : *Computer Communications and Networks, 2009. ICCCN 2009. Proceedings of 18th International Conference on*. IEEE. 2009, p. 1–7.
- [55] Maya RODRIG et al. « Measurement-based characterization of 802.11 in a hotspot setting ». In : *Proceedings of the 2005 ACM SIGCOMM workshop on Experimental approaches to wireless network design and analysis*. ACM. 2005, p. 5–10.
- [56] Matthias SCHULZ, Daniel WEGEMER et Matthias HOLLICK. « NexMon : A Cookbook for Firmware Modifications on Smartphones to Enable Monitor Mode ». In : *arXiv preprint arXiv :1601.07077* (2015).
- [57] Arunkumar SEKAR. *What is QBSS? What information does it provides?* URL : <http://community.arubanetworks.com/t5/Controller-Based-WLANs/What-is-QBSS-What-information-does-it-provides/ta-p/184840> (visité le 11/08/2016).
- [58] Joao Luis SOBRINHO, Roland DE HAAN et José Manuel BRÁZIO. « Why RTS-CTS is not your ideal wireless LAN multiple access protocol ». In : *Wireless Communications and Networking Conference, 2005 IEEE*. T. 1. IEEE. 2005, p. 81–87.
- [59] Joel SOMMERS, Paul BARFORD et Walter WILLINGER. « Laboratory-based calibration of available bandwidth estimation tools ». In : *Microprocessors and Microsystems* 31.4 (2007), p. 222–235.
- [60] Kristof Van Der STADT. *Proximus va simplifier et accélérer l'accès aux bornes wifi*. 2014. URL : <http://datanews.levif.be/ict/actualite/proximus-va-simplifier-et-accelerer-l-acces-aux-bornes-wifi/article-normal-321987.html> (visité le 21/11/2015).
- [61] YC TAY et Kee Chaing CHUA. « A capacity analysis for the IEEE 802.11 MAC protocol ». In : *Wireless networks* 7.2 (2001), p. 159–171.
- [62] Michel TERRÉ. *Wi-Fi Le Standard 802.11*. URL : <http://easytp.cnam.fr/terre/images/WiFi.pdf> (visité le 16/05/2016).
- [63] UBUNTU. *Programmer des tâches avec CRON*. URL : <https://doc.ubuntu-fr.org/cron> (visité le 27/07/2016).

- [64] Sudarshan VASUDEVAN et al. « Facilitating access point selection in IEEE 802.11 wireless networks ». In : *Proceedings of the 5th ACM SIGCOMM conference on Internet Measurement*. Usenix Association. 2005, p. 26–26.
- [65] Eduard Garcia VILLEGAS et al. « Effect of adjacent-channel interference in IEEE 802.11 WLANs ». In : *Cognitive Radio Oriented Wireless Networks and Communications, 2007. CrownCom 2007. 2nd International Conference on*. IEEE. 2007, p. 118–125.
- [66] *Wireshark*. URL : <https://www.wireshark.org/> (visité le 25/05/2016).
- [67] Haitao WU et al. « IEEE 802.11 distributed coordination function (DCF) : analysis and enhancement ». In : *Communications, 2002. ICC 2002. IEEE International Conference on*. T. 1. IEEE. 2002, p. 605–609.

