Rapport de Stage fin d'études

France Telecom Recherche et Développement

Amélioration de la qualité de service pour les jeux vidéos sur réseaux ad-hoc

Du 16 Mars au 15 septembre 2009

Auteur
Ali BOUHLEL

Tuteur entreprise
Sidi Mohammed SENOUCI

Tuteur enseignant
Emmanuel CHAPUT

Remerciement

Je tiens tout d'abord à adresser mes plus vifs remerciements à monsieur Sidimohammed SENOUCI pour avoir suivi ce travail de près, pour son esprit de travail dynamique et rigoureux et pour l'autonomie d'action qu'ils m'ont accordée tout au long de la durée de ce stage,

Je remercie également Monsieur Emmanuel CHAPUT mon responsable pédagogique pour l'intérêt porté à ce stage.

Ma gratitude est également adressée à l'ensemble de personnel de l'équipe M2I à France Telecom R&d Lannion pour leur accueil et convivialité et à Nadjib ACHIR et Khaled BOUSSETTA pour leurs conseils et aides techniques.

Tous mes remerciements a mes amis, ma famille et tous ceux qui m'ont soutenue et qui de près ou de loin ont contribué à l'aboutissement de ce travail.

Merci a tous

Table des matières

Chapitre 3
Solution de la couche transport: TCP-WELCOME

3.1 Problème de TCP dans un environnement mobile ad hoc
3.2 TCP WELCOME : une alternative adaptée aux MANETS
3.2.1 Principe de fonctionnement
3.3 Algorithmes de base : LDA et LRAPlateforme de tests SEDLANE: Simple Emulation of Delays and Losses for Ad hoc Network Environment.
3.4 Résultats de tests : une étude comparative des variantes TCP

Chapitre 4
Solution de routage : OLSR niveau 2

4.1 Routage proactif : le protocole OLSR
4.2 Une implémentation OLSR au niveau MAC
4.2.1 Le AWDS
4.2.2 Les deux types de tables : MAC et Forwarding
4.2.3 Les messages AWDS
4.3 Apport de la solution : résultats de tests.
4.3.1 Temps de changement de lien

Chapitre 5
Solution de niveau MAC Communications Coopératives

5.1 Solution retenue : Modification du pilote noyau Madwifi
5.1.1 Le Driver noyau linux Madwifi
5.1.2 Solution étudiée et horizons

Conclusion

Annexes

Figures

Introduction

Ce stage se place dans le cadre d'un projet RIAM MADGAMES (Middleware for AD-hoc networked vidéo GAMES). Les partenaires du projet sont France Telecom R&D, l'université de Paris 13 (chef de file), l'université de Paris 6 et un éditeur de jeux Fandango. L'objectif de MADGAMES est de développer un réseau middleware permettant le support de jeux vidéo multijoueurs sur une infrastructure de réseaux sans fil, en mode ad hoc.

Chapitre 1
Le contexte et les objectifs du stage

Avant de passer sur les aspects techniques du stage, nous allons tout d'abord présenter le groupe France Telecom, tout comme sa filière R&D où j'ai pu évoluer pendant ce stage. Nous évoquerons également le contexte économique et technique du stage, expliquant ainsi pourquoi celui-ci a été proposé.

1.1 Présentation de l'entreprise

Voici les caractéristiques principales du groupe France Telecom, à travers quelques chiffres et une vue globale de ces activités.

1.1.1 Le Groupe France Telecom

France Telecom est l'opérateur de télécommunications historique en France. C'est une Société Anonyme (SA) qui a été créée en 1996, après la privatisation de l'entreprise. Avec un chiffre d'affaires 2005 de plus de 49 milliards d'euros, France Telecom est le plus grand groupe de Télécommunications européen et un des plus importants du monde.

Le Groupe emploie 203 000 personnes dans 220 pays à travers le monde, dont :

_ 60,1% en France
_ 16,5% en Pologne
_ 11,9% au Royaume-Uni
_ 2% en Espagne

40% de l'effectif travaille donc à l'international, pour servir plus de 147 millions de clients à travers le monde, dont 62 en France. Les différents métiers de l'entreprise sont :
_ La téléphonie (mobile et fixe)
_ L'accès Internet
_ Les systèmes d'information et les réseaux
_ Les services liés aux nouvelles technologies

La téléphonie France Telecom, en tant qu'opérateur historique, dispose de l'ensemble du réseau téléphonique câblé du territoire français. En France, le groupe dispose ainsi de plus de 26 millions de lignes fixes grand public, touchant un nombre de clients de près de 34 millions.

En ce qui concerne la téléphonie mobile, France Telecom est, à travers sa marque Orange, le plus grand opérateur français, avec près de 22 millions de clients. La téléphonie mobile est à l'heure actuelle le principal secteur d'activité du groupe, puisque parmi ces 147 millions de clients à travers le monde, 86 millions sont des clients des services mobiles.

L'Internet Le groupe est le premier opérateur européen en nombre de lignes ADSL, avec 8,1 millions de clients haut débit ADSL Grand Public en Europe. Il est 2ème opérateur mondial de l'ADSL derrière China Telecom, et devant les plus grands groupes américains et japonais. Le domaine de l'Internet, via les services Wanadoo, représentent ainsi près de 12 millions de clients à travers le monde.

La stratégie NExT (Nouvelle Expérience des Télécoms) A l'heure actuelle, le monde des Télécoms est en plein bouleversement, avec l'avènement d'un grand nombre de nouvelles technologies, mais aussi celui de la convergence, notamment la convergence fixe / mobile. La stratégie Next s'inscrit donc dans la continuité de la logique du groupe qui est de s'assumer comme un opérateur intégré. Les objectifs sont multiples :

-Avoir un réseau et un système d'information intégré et mondial
-Une relation client harmonisée
-Un portail unique pour les clients, offrant un accès simplifié aux services
-Une culture commune, des valeurs partagées

Dans cette logique d'harmonisation et de convergence des services, le groupe France Telecom a donc tout intérêt à se regrouper à travers une marque unique. Pour cela, elle compte s'appuyer sur sa marque commerciale la plus forte et la plus représentée à l'étranger : la marque Orange. C'est ainsi qu'au premier juin 2006, le groupe France Telecom est devenu le groupe Orange, permettant de construire une identité forte très orientée vers l'international.
Pour affirmer encore plus sa dimension internationale et dans cette logique d'harmonisation, le groupe France Telecom est devenu le groupe Orange au 1er juin 2006. A travers cette nouvelle dynamique, le groupe compte donc renforcer sa suprématie européenne en terme d'innovation technologique.

1.1.2 France Telecom et sa branche R&D

En investissant pas moins de 1,5% de son chiffre d'affaires annuel dans la recherche, le groupe possède le plus grand centre de recherche européen dans son domaine des télécommunications.

France Telecom compte ainsi plus de 3900 ingénieurs, scientifiques et chercheurs au sein de sa division R&D, cela sur trois continents : l'Europe, l'Asie et l'Amérique. Le groupe possède ainsi
16 sites de recherche à travers le monde :

En France, Orange dispose donc de 8 centres de recherche :
_ Issy-les-Moulineaux (siège)
_ Lannion
_ Rennes
_ Caen
_ Grenoble
_ Sophia-Antipolis
_ Belfort
_ La Turbie

Issy-les-Moulineaux et Lannion sont les deux plus gros centres. Chaque centre a sa spécialité, Grenoble est par exemple spécialisée dans ce qui concerne la télémédecine, et Caen dans la cybernétique. Issy-les-Moulineaux et Lannion quant à eux touchent à tous les aspects de la recherche du groupe, et sont réellement polyvalents.

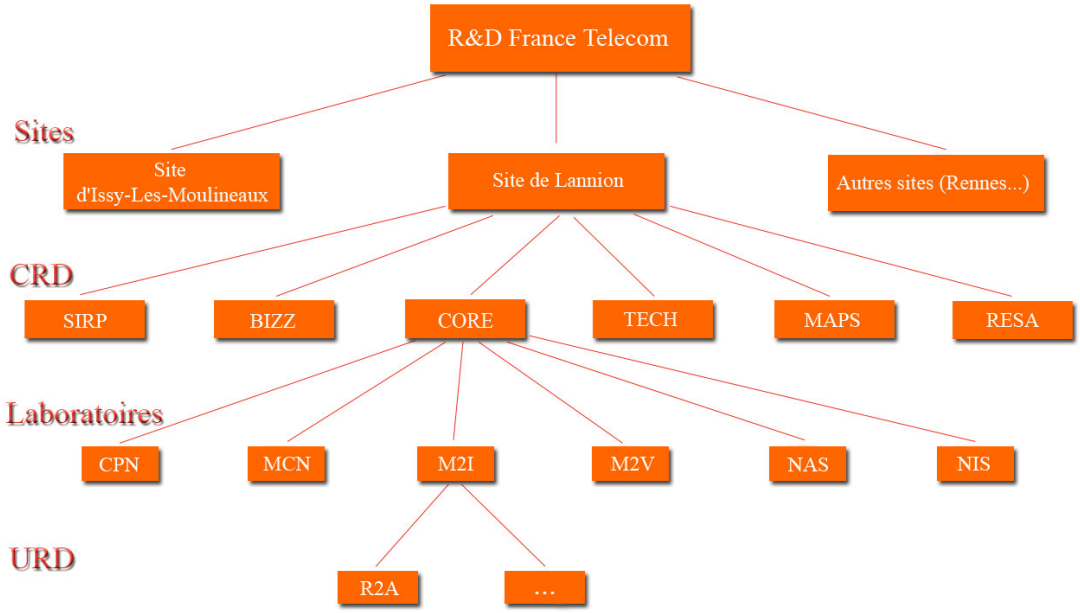
1.1.3 Le site de Lannion

France Telecom R&D est implantée à Lannion depuis 1963, au coeur de la technopole Anticipa, sous l'impulsion du ministre des PTT de l'époque : monsieur Pierre Marzin. En une quarantaine d'années, Lannion est devenue l'un des premiers lieux du développement des télécommunications dans le monde. Dans ce centre ont par exemple été inventés le minitel, la carte à puce, le GSM ou encore l'ATM. Située dans les Côtes d'Armor, proche de la côte de granit rose, Lannion est une ville de plus de dix-sept milles habitants. Avec l'implantation de nombreuses entreprises dans le Trégor, la région compte aujourd'hui plus de trois mille chercheurs et ingénieurs.

Le site de France Telecom Lannion est fort de près de 1600 employés, dont plus de 1200 scientifiques.

1.1.4 L'organisation de France Telecom R&D

France Telecom R&D a une organisation pyramidale assez simplement descriptible. Voici un organigramme de cette structure :



Tout d'abord, la R&D est divisée en CRD (Centres de R&D). Un CRD correspond à une grande spécialité, telle que le Cœur de Réseau ou encore les Réseaux d'Accès. A Lannion et à Issy-les-Moulineaux, tous les CRD sont représentés.
Ensuite, chaque CRD est divisé en différents laboratoires.
Le stage que j'ai effectué s'est par exemple déroulé dans le laboratoire M2I : Multimédia networks for non-conversational Fixed /mobile services : Image, Internet.

Au sein d'un même CRD, les laboratoires peuvent être répartis sur différents sites (Lannion, Issy...).
Enfin, chaque laboratoire, comportant généralement entre 50 et 100 personnes, est divisé en URD (Unité de R&D). Chaque URD a en général une spécialisation très précise.
Mon stage s'est déroulé dans l'URD R2A, spécialisée dans les réseaux ambiants et spontanés, dont les réseaux ad-hoc font typiquement partie.
Enfin, tout comme les laboratoires, les URD d'un même laboratoire peuvent être répartis sur plusieurs sites. Par exemple, dans mon laboratoire, 3 des 6 URD sont à Lannion, et les 3 autres sont à Issy-les-Moulineaux.

Ainsi, mon stage s'est déroulé :
- Dans le CRD CORE, les réseaux ad-hoc représentant un cœur de réseau en eux mêmes.

- Dans le laboratoire M2I, concernant tout ce qui touche aux réseaux et au multimédia (hors services de voix et de visio)

- Dans l'URD R2A, concernant les réseaux spontanés dont font partie les réseaux ad-hoc.

1.1.5 L'unité de recherche R2A

Comme nous l'avons vu, l'unité de recherche R2A traite de tout ce qui touche aux réseaux ambiants et spontanés. L'équipe est composée d'une vingtaine de personnes, dont 8 thésards, plusieurs maîtres de conférences, plusieurs ingénieurs, et de 5 stagiaires pendant ce semestre ; le responsable est Yvon Gourhant, maître de conférences. Cette équipe dispose également de plusieurs chefs de projet, dont Sidi-Mohammed Senouci, mon responsable.
Sidi-Mohammed Senouci est ainsi responsable du PRP (Projet de Recherche Pluridisciplinaire) Spontex, projet développé par France Telecom traitant des réseaux spontanés.
Le projet spontex fait intervenir différentes entités et personnes, dont des universitaires. C'est donc dans le cadre de ce projet que mon stage s'est déroulé.

1.2 Le contexte économique et technique

1.2.1 Le contexte économique

Le marché du jeu vidéo est en pleine expansion depuis une quinzaine d'années, et ne s'est jamais aussi bien porté qu'à l'heure actuelle. En 2005, Le chiffre d'affaires a progressé de 5,3% par rapport à 2004, atteignant ainsi le chiffre de 19 milliards de dollars. Cette progression n'est pas prête de s'inverser, car les prévisionnistes voient le marché augmenter d'au moins 50% d'ici à 2009-2010.

Cette progression fulgurante s'accompagne d'un accès à l'Internet haut débit qui se généralise dans tous les foyers occidentaux, particulièrement en France, et fait que le jeu vidéo voit autant de nouvelles opportunités s'ouvrir à lui, via le jeu dit "On Line". A l'heure actuelle, les jeux vidéos multi-joueurs pour PC permettent quasiment tous de jouer sur Internet.

En parallèle à cet essor, une autre technologie est apparue et s'est rapidement imposée comme un standard incontournable des réseaux locaux : la norme IEEE 802.11, mieux connue sous le nom de Wifi. Avec cette apparition, c'est une nouvelle manière de jouer aux jeux vidéo multi-joueurs qui s'offre au grand public. Premier pas vers cette révolution annoncée : la sortie en 2005 des consoles Nintendo DS et Playstation PSP, des consoles portables proposant un mode multi-joueurs sur réseaux Wifi en mode ad-hoc.

1.2.2 Le contexte technique

1.2.2.1 Projet RIAM MAD GAMES

Avec l'expansion rapide de la technologie sans fil et son intégration sur les consoles de jeux, on peut raisonnablement s'attendre à ce que la portabilité des jeux vidéo sur les réseaux mobiles constitue une évolution naturelle de ce domaine. L'affranchissement des liaisons câblées et la mobilité offerte aux joueurs sont autant d'arguments qui vont dans le sens d'une interaction plus aisée et plus riche entre joueurs. Ainsi, on peut facilement imaginer le cas de jeux improvisés dans une cour de récréation et impliquant plusieurs joueurs munis de consoles de jeux portables. Ce scénario peut être réaliste surtout si l'on suppose que le réseau de support est de type ad-hoc. L'absence d'infrastructure centralisée étant l'argument principal qui pourrait favoriser le succès des jeux multi-joueurs sur les réseaux ad-hoc. A ceci il faut ajouter le faible délai entre joueurs et l'absence de coût, en opposition aux délais relativement importants auxquels doivent faire face les joueurs qui souhaitent utiliser les liaisons supportées par les réseaux cellulaires.

L'objectif du projet **MAD GAMES** est de développer un middleware réseau permettant le support de jeux vidéo multi-joueurs sur une infrastructure de réseaux sans fil, en mode ad-hoc. Ce middleware offrira une interface entre le moteur de jeu et les couches réseau de bas niveau. En comparaison avec la technologie actuellement disponible, l'apport de ce middleware, réside dans les éléments suivants :

1. Il s'appuiera sur une infrastructure de jeu distribuée, en mode pair-à-pair, mieux adaptée à un réseau sans fil en mode ad-hoc.
2. La qualité du jeu perçue par l'utilisateur, étant un paramètre critique pour l'application de jeux vidéo, l'un des objectifs de ce projet est de contribuer à la réflexion et à la conception de mécanismes de gestion de la qualité de service

au niveau réseau, qui permettront d'améliorer la Qualité du Jeu (*QdJ*), dans l'environnement des consoles de jeu et des réseaux ad-hoc.

3. Ce middleware offrira la possibilité de concevoir de nouveaux moteurs de jeux et de nouvelles façons de jouer, exploitant pleinement les avantages offerts par un réseau sans fil en mode ad-hoc.

1.2.2.2 Jeux vidéo sur réseaux ad hoc

Actuellement dans les parties multi-joueurs, les consoles portables utilisent un mode ad-hoc mono-saut (n'impliquant donc pas de routage). Les consoles portables s'utilisent de façon spontanée entre amis, entre écoliers dans la cours de récréation ou dans les transports en commun. Ainsi les réseaux ad-hoc répondent ainsi parfaitement à cette demande qui devrait exploser dans les années à venir.

En effet, sans le moindre câble ni installation particulière, il est possible pour les joueurs de se connecter directement l'un à l'autre en sans fil rapidement et Spontanément. En parallèle de ce constat, nous avons pu remarquer que très peu d'études sont parues sur ce sujet, qui pourtant semble être extrêmement prometteur.

➢ Architecture réseau

L'architecture Client Serveur (C/S) est l'architecture dominante dans le monde des jeux.
L'architecture Pair à Pair (P2P) représente une architecture alternative. Nous allons étudier dans cette partie les principes de fonctionnement de chaque architecture, les comparer et distinguer leurs modèles de communications.

Cette étude va être faite du point de vue architecture du jeu vidéo multi joueurs sur réseau ad-hoc. A la fin de cette partie, nous présenterons également une architecture hybride proposée comme solution aux problèmes rencontrés dans les architectures C/S et P2P.

Architecture Client/serveur

Dans cette architecture tous les clients se connectent sur une machine centrale, le serveur, qui est installé sur une simple machine d'un des joueurs. Le serveur est responsable de maintenir l'état global du jeu alors que les clients sont responsables de le mettre à jour, en lui communiquant l'ensemble de leurs actions.

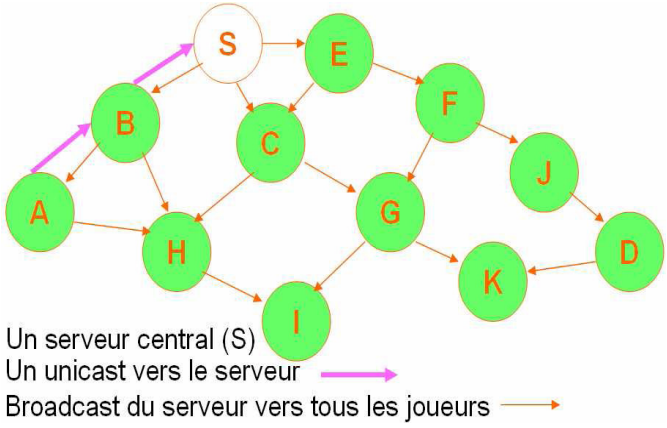


Figure 1 Architecture Client / Serveur

La figure 1 représente une architecture client serveur. Le nœud S est le serveur qui communique avec ses clients en utilisant un mécanisme de diffusion, alors que ses clients lui acheminent leurs paquets en utilisant une communication unicast.

L'une des avantages les plus importants de cette architecture est la cohérence du jeu qui vient de la centralisation du calcul de l'état global du jeu au niveau de serveur. Cette centralisation de calcul permet aux nœuds de ressources limités d'économiser le maximum possible de leur énergie. Malheureusement, la centralisation du serveur a aussi des inconvénients. Ceci représente un point de défaillance, car si le serveur tombe en panne (par exemple pour manque de batterie venant de la forte consommation de maintient et de diffusion d'état de jeu) ou se déplace vers des zones non couvert par le réseau, le jeu va s'arrêter obligatoirement. Il faut noter encore que plus le nombre de joueurs est grand plus le serveur doit assurer une bande passante et une énergie élevée pour satisfaire les besoins de ses clients.

Architecture Pair à pair (P2P)

Cette architecture représente une alternative à l'architecture C/S. Elle est souvent utilisée dans les jeux de stratégie. L'établissement du jeu passe par une phase C/S, un joueur crée le jeu alors que les autres se connectent à lui pour participer au jeu. Une fois connectés, les joueurs communiquent en mode P2P.

L'architecture P2P représente une solution complètement distribuée (voir Figure 2) où chaque nœud est responsable de maintenir l'état global du jeu et de mettre à jour l'ensemble des nœuds du réseau.

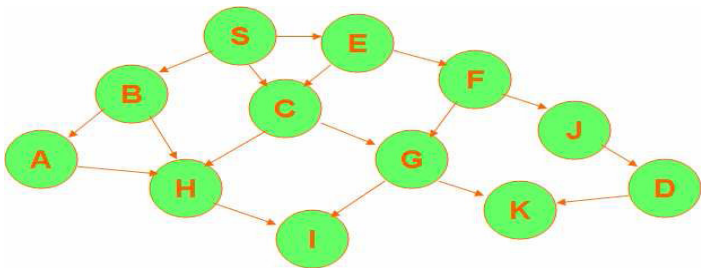


Figure 2 Architecture Pair à Pair

Pour créer un jeu, un trafic unicast est requis. Ce trafic n'est pas contraignant en termes de QoS.
Pourtant, les communications entre les nœuds durant la session passent par des diffusions.

L'avantage principal de cette architecture vient de l'inexistence de point de gestion/calcul central.
Comme il n'est pas nécessaire de maintenir une communication persistant vers un nœud central, cette architecture est plus adaptée au MANET.
Pourtant, la gestion de cohérence du jeu dans un contexte complètement distribué n'est pas une tâche facile. Au niveau de l'énergie, dans cette architecture, chaque nœud fait plus de calcul et consomme donc plus d'énergie.

Architecture hybride

Une architecture hybride est un compromis entre C/S et P2P. Elle est proposée pour résoudre le problème de centralisation du serveur de C/S et de cohérence de P2P.

Nous allons nous concentrer sur les travaux de Riera et al qui proposent une architecture hybride pour les jeux vidéo sur réseaux ad hoc.

Le principe est de diviser le réseau en zones, voir figure 3, et dans chaque zone de sélectionner un nœud comme serveur de zone. Le jeu est centralisé dans les zones et distribué entre
les zones.

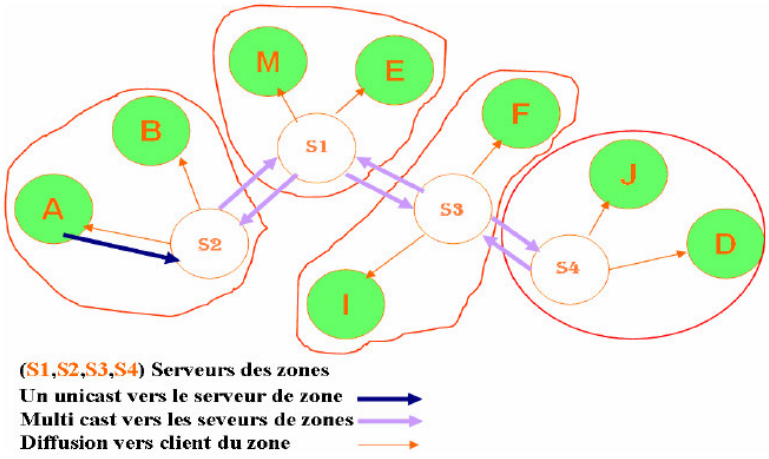


Figure 3 Architecture Hybride

➢ Types de jeux

Jeux d'action

- Beat Them All : le joueur se bat contre tous. Exemple : le jeu Double Dragon
- Combat : combat un contre un. Exemple : Street Fighter
- Inflitration. Exemple : Splinter Cell
- Jeux de plateformes. Exemples : les célèbres Super Mario et Sonic

Jeux d'aventures

- Classique : le héros vit une aventure et résoud des énigmes. Exemple : le jeu Les Chevaliers de Baphomet
- Jeux de rôle (RPG : Role Playing Game). Exemples : Final Fantazy, Diablo, Baldur's gate
- Survival Horror. Le joueur est plongé dans un univers très sombre et e_rayant. Exemples : Resident Evil, Silent Hill.

Jeux de tir

- Les FPS (First Person Shooter). Le but est simplement de tuer les autres. Les jeux phares sont Counter Strike, **Quake**, Doom, Half Life, Unreal Tournament
- Shoot them up : le joueur est en général dans un vaisseau, et doit tirer et anéantir les autres vaisseaux. Exemple : Space Invaders.

De tous ces types de jeu, on comprend qu'un jeu d'échec ou de puzzle n'a pas les mêmes besoins en temps réel qu'un jeu de tir. De plus, il y a des types de jeux qui n'ont pas de mode multijoueurs, c'est le cas par exemple des jeux de plateforme comme Super Mario, ou encore les jeux d'aventures où il faut résoudre des énigmes. Parmis les jeux multijoueurs, différentes études ont été menées pour connaitre les interactions entre ces jeux et les réseaux qu'ils utilisent.
Les jeux multijoueurs les plus populaires, et ayant ainsi fait l'objet de bon nombre d'études, sont les suivants :

- Les FPS (First Person Shooter)
- Les RTS (Real Time Strategy

Dans la suite, les jeux FPS et notamment le Quake 3 ARENA sera la cible de notre étude. On procèdera donc à identifier les métriques de la qualité de service qui influence la jouabilité puis à présenter les différentes solutions améliorant cette QOS.

Chapitre 2
Paramètres de la QOS pour les jeux FPS : Exemple de Quake 3

2.1 Le jeux Quake 3 Arena

➢ Présentation du jeu

Maintenant que nous en avons connu le contexte technique des travaux, étudions le cas d'un jeu en particulier : Quake III Arena. Ce jeu est un FPS (First Person Shooter), qui est sorti le 10 décembre 1999. Le choix d'étudier ce jeu plutôt qu'un autre s'appuie sur divers raisons. Tout d'abord, nous avons vu précédemment que les FPS sont les jeux les plus joués sur Internet, et ce sont également ceux qui sont les plus exigeants en termes de qualité de réseau. Pour cela, il était intéressant de se pencher sur ce type de jeu. Ensuite, Quake III Arena est une sorte de référence dans le monde des jeux vidéo, de par sa qualité pour un jeu de cette époque, mais aussi en ce qui concerne la recherche.

En effet, il a très souvent servi de "benchmark", tout comme peut l'être par exemple l'image "Lena" en imagerie. Cela est peut-être du au fait que ce jeu est codé de façon très académique et typique, et qu'il est très facile d'avoir accès à tous les paramètres et à toutes les variables du jeu, via sa ligne de commande.

En ce qui concerne l'installation de Quake 3 Arena sur une plateforme linux et le paramétrage nécessaire au lancement du jeu en mode automatique (slugbot), veuillez vous referez à l'annexe 1.

➢ Trafic réseaux généré par le jeu

Une petite expérimentation permet d'identifier et de qualifier les flux réseau généré par le jeu; elle consiste à lancer deux joueur sur un réseau Ethernet et de capturer le trafic avec l'outil WireShark.

WireShark a permis d'enregistrer tout le trafic, mais l'intégralité des données ne nous intéresse pas (différentes requêtes de maintenance du réseau wifi par exemple). Le logiciel permet de filtrer les paquets désirés. Ainsi, nous avons pu obtenir la courbes suivantes, qui correspond au nombre de paquets transférés seconde par seconde.

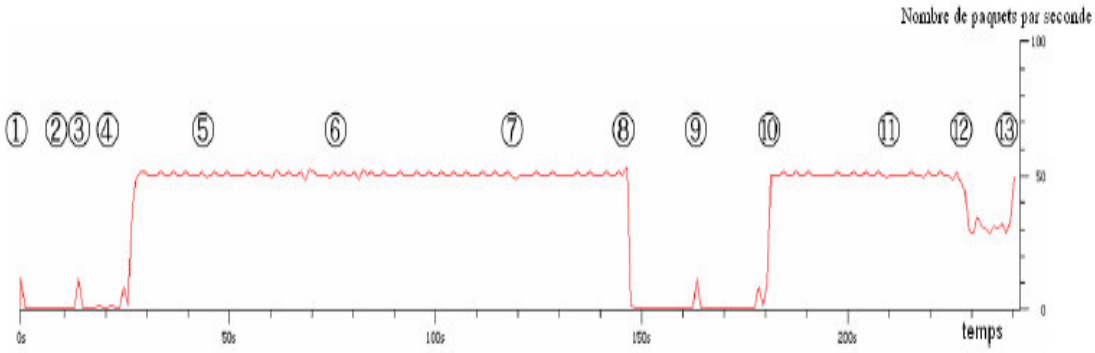


Figure 4 Le nombre de paquets transmis par seconde durant l'expérience

L'étape 1) Cette première étape est le point de départ de l'expérimentation. C'est le moment où un des joueurs clique sur "multi-joueurs". Nous avons constaté lors de précédentes expériences que le jeu n'émet strictement aucun paquet sur le réseau tant que l'on n'a pas cliqué sur ce mode. Lorsqu'un des joueurs a donc choisi ce mode multi-joueurs, le jeu va rechercher s'il y a des serveurs disponibles. Ceci explique pourquoi il y a un pic d'envoi de messages au temps 0, la station émet des requêtes en broadcast pour savoir si un serveur existe déjà.
Nous avons remarqué qu'à chaque fois, ces broadcasts de recherche de serveur étaient constitués de 8 paquets.
L'étape 2) Elle a lieu au bout de 10 secondes : le serveur est lancé. Comme on peut le constater sur cette figure, cette phase n'a pas d'impact sur le réseau, contrairement à ce que l'on aurait pu penser. Il n'y aura plus d'émission de paquet tant qu'un client ne se sera pas manifesté pour joindre la partie.
L'étape 3) Il s'agit de la même étape que la 1, sauf que cette fois ci c'est le second joueur qui clique sur "multi-joueurs". Le fait de cliquer sur ce mode génère automatiquement une recherche de serveur, et donc... L'envoi en broadcast de 8 paquets. Nous obtenons donc là le second pic d'envoi de messages.
L'étape 4) Un serveur a répondu au client : il en existe bien un de disponible. Le client clique donc sur celui-ci pour joindre la partie. Les stations s'échangent des informations générales ayant pour but d'intégrer le client dans la partie ; ces informations impliquent un pic d'envoi juste avant le début du jeu.

L'étape 5) A partir de l'étape 4, le jeu est lancé, et on voit qu'en mode "jeu", le nombre d'envoi de paquets par seconde est incroyablement stable : il y a constamment un envoi de 50 paquets par seconde, à un ou deux près. La taille de paquet est également stable, comme on. On constate que les paquets ont alors approximativement une taille de 100 octets.

A l'étape 5, il y a eu un meurtre : le joueur client a tué le joueur serveur. On constate ici que cela n'a pas de conséquence sur le réseau, pas plus que les différents mouvements qui ont été réalisés.
L'étape 6) Ici, la partie est redémarrée. Une fois de plus, on constate que le nombre d'envoi de paquet est stable, même si ces paquets sont légèrement plus volumineux.
L'étape 7) A cette étape, il y a un double meurtre. Encore une fois, peu d'impact sur le réseau, étape assez similaire à l'étape 6 (comme à celle-ci les joueurs redémarrent la partie, sauf que les scores ne retombent pas à 0).
L'étape 8) Le client se déconnecte. On constate que le trafic chute très brutalement pour atteindre 0 octets transmis par seconde. Cela signifie que le serveur n'envoie des messages que lorsqu'au moins un client est connecté ; sinon, il reste silencieux.
L'étape 9) Etape similaire à la 1) et la 3), le client cherche à se reconnecter, donc il envoie un broadcast pour connaitre les serveurs qui sont disponibles. Evidemment, l'ancien serveur est toujours là, donc il répond et s'affiche à l'écran.
L'étape 10) Le client se reconnecte, étape identique à la 4 ; une fois de plus, la synchronisation génère beaucoup de trafic d'un coup, avant que le jeu ne reparte à un débit moyen de 50 paquets par seconde.
L'étape 11) Le joueur du serveur tue le joueur du client. Une fois de plus, pas d'impact notable sur le réseau.
L'étape 12) Le serveur se déconnecte. Pendant ce temps, le client est toujours connecté à la partie. Sur l'écran est alors indiqué qu'il est déconnecté, et le jeu cherche à retrouver la connexion. On constate nettement que l'envoie de paquet est exactement deux fois moins important à cet instant, et en regardant de près les messages qui sont envoyés, on constate qu'il n'y a plus que le client qui émet, à la même cadence qu'avant. En revanche, le serveur n'émet bien évidemment plus.
Cette étape est assez intéressante, car on aurait pu penser que les choses se passeraient comme lors de l'étape 8. On constate donc que **le trafic entre client et serveur est asymétrique.**
L'étape 13) Le joueur du serveur quitte totalement le jeu, et la simulation s'arrête. On remarque que le nombre de paquets envoyés augmente. En fait, il s'agit de réponses ICMP au client indiquant que le serveur n'est plus disponible sur le port du jeu. Il y a autant de ces requêtes que d'envoi de paquets de la part du client, qui n'arrête toujours pas ses émissions. Ceci explique donc pourquoi le nombre de paquets envoyés augmente.
Au bout d'environ une minute, si le client n'a toujours pas réussi à récupérer la connexion, le jeu quitte automatiquement la partie et renvoie le joueur à l'écran d'accueil.
Un dernier aspect à noter : lorsque l'on fait quitter l'arène au serveur puis quitter le jeu, le client continue ses envois de paquets, espérant retrouver la connexion. En revanche, si on quitte directement le jeu sur le serveur, la partie du client est directement coupée et on se retrouve sur l'écran d'accueil.

2.2 La plateforme d'émulation de l'environnement mobile ad hoc

2.2.1 Simulation / Emulation

Dans cette partie nous présentons les différentes approches pour le test et l'évaluation de réseaux et en particulier des réseaux sans fil. Nous suivrons donc une démarche comparative qui identifie les apports et les inconvénients de chaque proposition afin de pouvoir choisir la méthode la plus adaptée à nos besoins compte tenu de la spécificité de l'application (jeu vidéo de type FPS) et de l'environnement de déploiement (réseau mobile ad hoc).

Comment cette évaluation peut elle être mise en œuvre ?

Une compagne de tests peut être effectuée soit en environnement réel et dans ce cas on est face à une utilisation réel du système à mettre en œuvre (matériel & logiciel) et à des résultats réalistes mais aussi à un coût important de l'infrastructure, mise en œuvre (un nombre non négligeable de nœuds, de personnes, conditions limites, etc…) et parfois l'impossibilité de la reproduction des conditions d'expériences qui permettent de garantir une certaine équité lors de la comparaison de plusieurs solutions.

La deuxième alternative consiste à utiliser la simulation qui repose sur des modèles validés ou reconnus (NS-2, OPNET, QUALNET, GLOMOSIM) et qui garantit la possibilité de **reproductibilité** pour effectuer une comparaison avec d'autres protocoles ainsi que la facilité de l'implémentation par rapport à une implémentation réelle.
Sauf que cette méthode engendre plusieurs problèmes ; tests **non réels**, **Modélisation totale du système**, Difficultés de la modélisation dans certains cas, Simplification des modélisations, temps d'exécution et passage à l'échelle.

L'émulation constitue la troisième approche et garantit un compromis entre les deux solutions précédentes et en conséquences elle assure la possibilité de tester l'implémentation réelle, d'effectuer des tests mêmes dans des conditions limites,
La possibilité de reproduire les mêmes scenarios et par la suite une comparaison équitable avec d'autres protocoles.

2.2.2 Plateforme de test

➢ Emulation

La solution retenue pour les travaux d'évaluation étant l'émulation sauf que plusieurs types d'émulation peuvent être candidats: émulateur niveau physique, 1, 2, voire 3.

L'émulateur niveau 3 qui sera utilisé et exige l'utilisation d'un serveur pour le conditionneur de trafic, la construction de la topologie, la caractérisation des liaisons, etc.
La figure ci-dessous donne une vue globale sur ces différents émulateurs

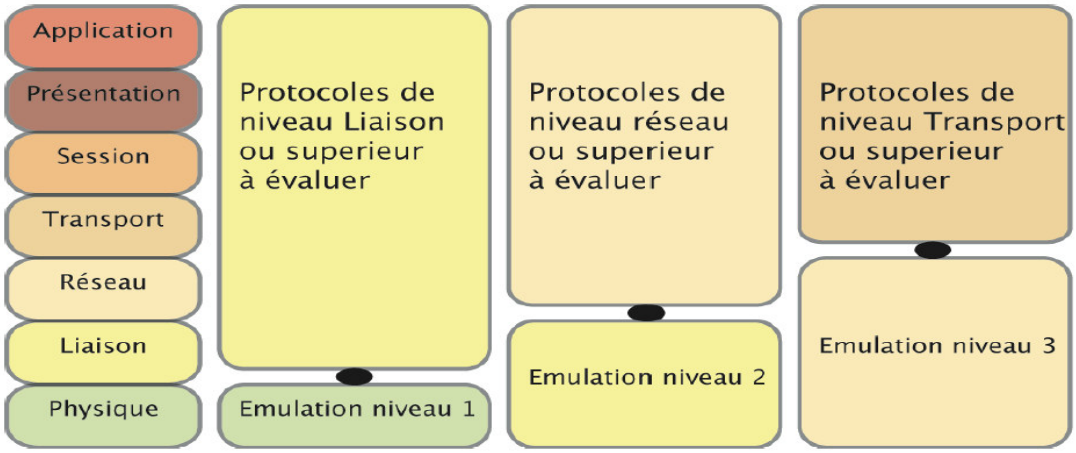


Figure 5 Emulateurs de différents niveaux

➢ Description de la plateforme

Côté client

Au niveau du matériel la plateforme est constitué de : 3 PC identiques : Dell optiplex 755, Processor Intel Core 2 Duo 2 GHz, 2 Go RAM, Integrated ethernet/graphic card et 3 portables identiques : Dell Latitude D510, Processor Intel Pentium M730 1.7 GHz, 512 Mo RAM Integrated ethernet/graphic card.

Au niveau software, on utilise des OS, kernel : Linux, 2.6.22.14 de la distribution : Ubuntu 7.10 Gutsy Gibbon, Jeux: IOQuake 3 Arena (JeuFPS), Joueurs : slugbot, programme bot Quake 3un module de routage Ad Hoc et Module d'émulation.

Côté serveur de jeu

Le matériel utilisé est un PC Dell optiplex 755de Processor Intel Core 2 Duo 2 GHz 2 Go RAM Integrated Ethernet/graphic card.

Le logiciel est compose de OS, kernel : Linux, 2.6.22.14 de la distribution : Ubuntu 7.10 Gutsy Gibbon avec le Jeux: IOQuake 3 Arena (Jeu FPS), Un serveur ioquake3 dédié, Un module de gestion des parties, un module de routage Ad Hoc et un Module d'émulation

Côté serveur d'émulation

On utilise le même matériel et logiciel que le serveur de jeu sauf qu'on n'aura pas besoin du module de routage ad hoc ni du module de gestion de parties.

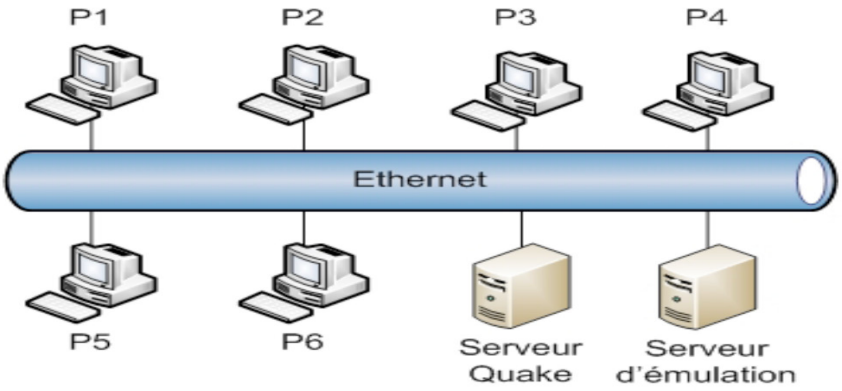


Figure 6 Plateforme d'évaluation des métriques de QOS

Tous les terminaux sont reliés entre eux via une liaison Ethernet, ils sont donc branchés sur un même Switch.

Les différents modules logiciels sont présenté dans la section suivante.

Pour assurer le routage dans l'environnement ad hoc on a choisi d'installer le Daemon de routage olsrd relatif au protocole OLSR. Une description de se protocole sera fournie dans le chapitre 4 qui traite la parie routage.

Dans le module d'émulation, quatre fonctionnalités ont été traités pour répondre aux exigences d'un contexte sans fil en mode ad hoc le module doit ainsi émuler :

- **La connectivité entre terminaux**

Le serveur d'émulation donne à chaque terminal la liste des terminaux non visibles par lui. Le client de jeux procèdera alors à filtrer toutes les trames Ethernet provenant des machines non visibles par le terminal

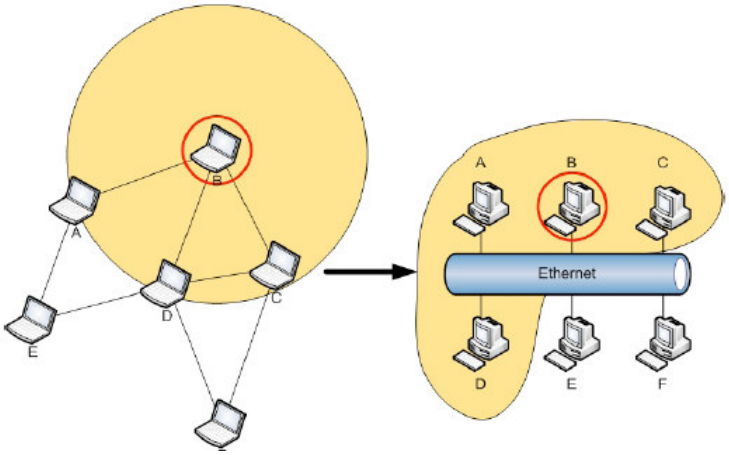


Figure 7 Emulation de la connectivité

Dans la figure précédente, seules les machines A, C et D sont dans la portée de la machine B, pour émuler ce fait, le serveur de jeu signale au terminal B qu'il ne peut communiquer qu'avec A, C et D et par la suite toute trame provenant de E ou de F sera filtrée.

- **La mobilité des terminaux**

Le serveur d'émulation calcule les nouvelles positions des terminaux et transmet périodiquement les nouvelles connectivités à chaque terminal.
Le modèle de mobilité le plus souvent utilisé est Random Way Point.

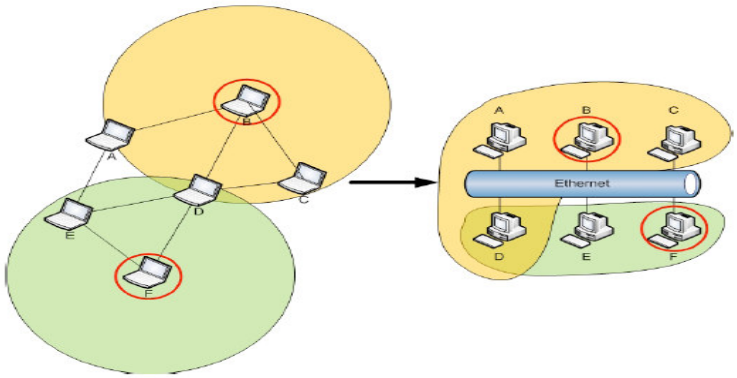


Figure 8 Emulation de la mobilité

- **Le délai et les pertes**

Le serveur d'émulation informe chaque terminal du délai et des taux de pertes avec leurs voisins.

2.3 Les différents types d'évaluations

- **Le score**

Lorsque la jouabilité baisse, cela se ressent sur les performances du joueur, et donc sur son score. Une technique utilisée est donc de faire jouer différents joueurs sur plusieurs parties, et de noter leurs scores. Petit à petit, on dégrade le réseau, et on note les scores obtenus.
- Avantage : On prend en compte le critère le plus proche de la notion de jouabilité : le score. Ce critère est important à prendre en compte, puisque le but des joueurs est bien avant tout d'avoir le score le plus élevé.
- Inconvénient : Ce critère est très difficilement mesurable objectivement et rigoureusement. D'une partie à l'autre, les joueurs peuvent avoir des scores très différents, sans pour que cela soit du à de quelconques problèmes de réseau.

- **Le questionnaire sur Internet**

Dans certaines études, on a demandé à des joueurs sur des forums d'estimer quels étaient leurs besoins en terme de qualité de réseau, notamment en ce qui concerne la bande passante et la latence, pour avoir une bonne jouabilité.
- Avantage : Possibilité d'avoir un grand nombre de réponses, et prise en compte de l'avis des premiers concernés : les joueurs.
- Inconvénient : Une fois de plus, un critère très subjectif, ne permettant pas de se rendre compte rigoureusement de la qualité du jeu.

- **La qualité perçue par le joueur lors de la partie**

Une autre méthode consiste à faire jouer des joueurs en dégradant le réseau, et à leur demander de dire à chaque fois ce qu'ils pensent de la jouabilité de la partie qu'ils effectuent via une note.
- Avantage : La jouabilité c'est avant tout la jouabilité perçue par le joueur lui-même, donc ici on capte cette jouabilité par l'intermédiaire de notes.
- Inconvénient : Même si cette méthode semble plus fiable et adoptant une démarche plus digne d'un scientifique, elle introduit toujours beaucoup de subjectivité et pas de critère terre à terre permettant de définir clairement la qualité du jeu.

- **Le calcul du FPS**

La jouabilité c'est aussi le nombre d'images par seconde affichées à l'écran ; plus il y en a, théoriquement plus la qualité s'accroît.
- Avantage : Le nombre de FPS est un critère mesurable et totalement objectif, ayant une proche relation avec la jouabilité.
- Inconvénient : Un nombre faible d'images par seconde traduit nécessairement une mauvaise qualité visuelle du jeu. Mais inversement, un nombre élevé d'images par seconde n'implique pas forcément un jeu de qualité. Prenons un exemple. Dans une partie, un joueur subit de graves problèmes réseau ; son personnage se déplace de façon très saccadée. Pendant ce temps, en haut à gauche de l'écran, une écriture dit en clignotant : "attention, vous êtes presque déconnectés". Le personnage s'affichera peut-être à la vitesse de 3 images par seconde, mais l'écriture, elle, s'affichera à la vitesse de 90 images par seconde. De ce fait, le nombre de FPS mesuré sera... De 90 images par seconde, alors que le jeu sera haché.

De plus, le FPS traduit plus la qualité de la vidéo plutôt que la jouabilité, de façon analogue à la résolution. Ce n'est donc pas vraiment le meilleur critère à prendre en compte, même s'il peut être intéressant.

2.4 Impact du délai, pertes de paquets et de la gigue

➢ Scénarios et Outputs

Dans cette partie nous décrivons le type d'évaluation retenu et le scénario de tests ainsi que le type d'outputs de chaque évaluation.

Afin de réaliser des tests complètement objectifs et pour pouvoir reprendre à chaque fois les mêmes expériences, nous avons utilisé des bots autonomes pour reproduire le comportement de chaque joueur.

Dans toutes les expériences un seul critère de victoire a été utilisé: le "deathmatch"; ce qui veut dire que le premier joueur qui atteint un certain score en tuant les autres joueurs sera considéré comme vainqueur. Dans chaque expérience cent parties ont été joué et la durée d'une partie varie généralement entre huit et neuf minutes ce qui veut dire approximativement quinze heures de jeu pour chaque expérience.

En détails, chaque expérience s'est déroulée en cinq étapes :

1) Le serveur envoie un message START à chaque client pour qu'il se connecte au jeu.
2) Une partie est jouée jusqu'à ce qu'un client atteigne le score de 40.
3) Le serveur envoie un message KILL à chaque client pour qu'ils déconnectent du serveur et tuent le client quake.
4) Le serveur envoie éventuellement un message CHANGE FILTER si un client doit changer ces paramètres réseau.
5) Enfin, le serveur envoie un autre message START et par la suite tous les clients quake se reconnectent au serveur de jeux.

Les expériences on été réalisé en pénalisant un seul joueur à la fois pour émuler toutes les combinaisons possibles des paramètres réseaux: le délai de bout en bout, la gigue et la perte de paquets. Les délais on été modifiés entre les valeurs de 0, 25, 50 et 100 ms et la gigue prend les valeurs de 0, 10 et 20.

➢ Résultats

Le score à la fin de chaque partie est considéré comme la meilleur métrique pour comprendre si un joueur est bien placé sur l'aréna ou plus généralement si il/elle joue d'une façon satisfaisante.

La figure suivante montre La fonction de répartition cumulée (CCDF) lorsqu'aucun joueur n'est sujet de délai. Comme nous pouvons le noté la partie est équitable car il ya une très importante probabilité pour chaque joueur d'atteindre un score d'au moins 25. Mais il ya un petit peu moins d'équité dans la gamme de score entre 30 et 40 cela est du au fait qu'il ya un seul joueur gagnant dans chaque partie; c'est celui qui atteint le score de 40 le premier.

Les autre joueur ont dans ce cas généralement un score de 38 ou 39 ils perdent alors la partie même si la qualité de leurs parties ont été identiques à celle du joueur gagnant.

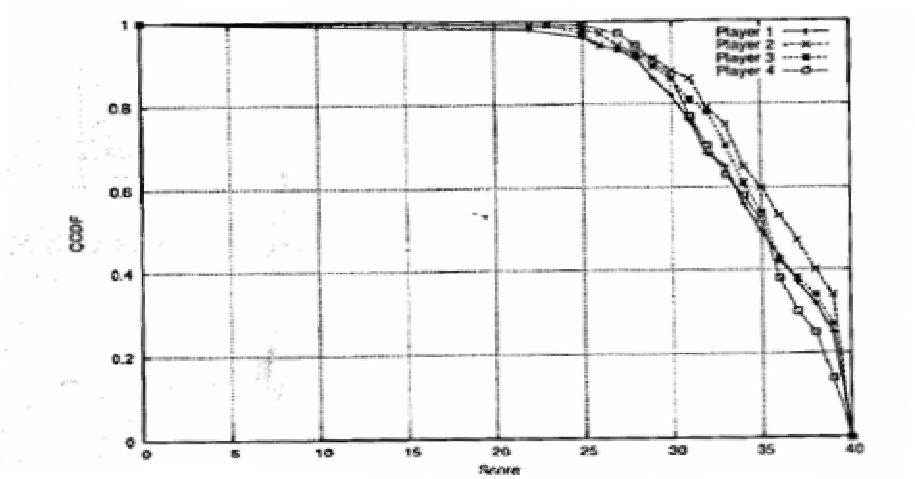


Figure 9 CCDF sans délais

Dans la série de figures ci-dessous on réalise la même expérience que précédemment mais en pénalisant le joueur 1 par l'addition de délais de 50, 100 et 150 ms.

Nous pouvons très facilement constater une dégradation des performances du joueur pénalisé déjà pour seulement 50 ms de délai, cette dégradation croît avec ce délai.

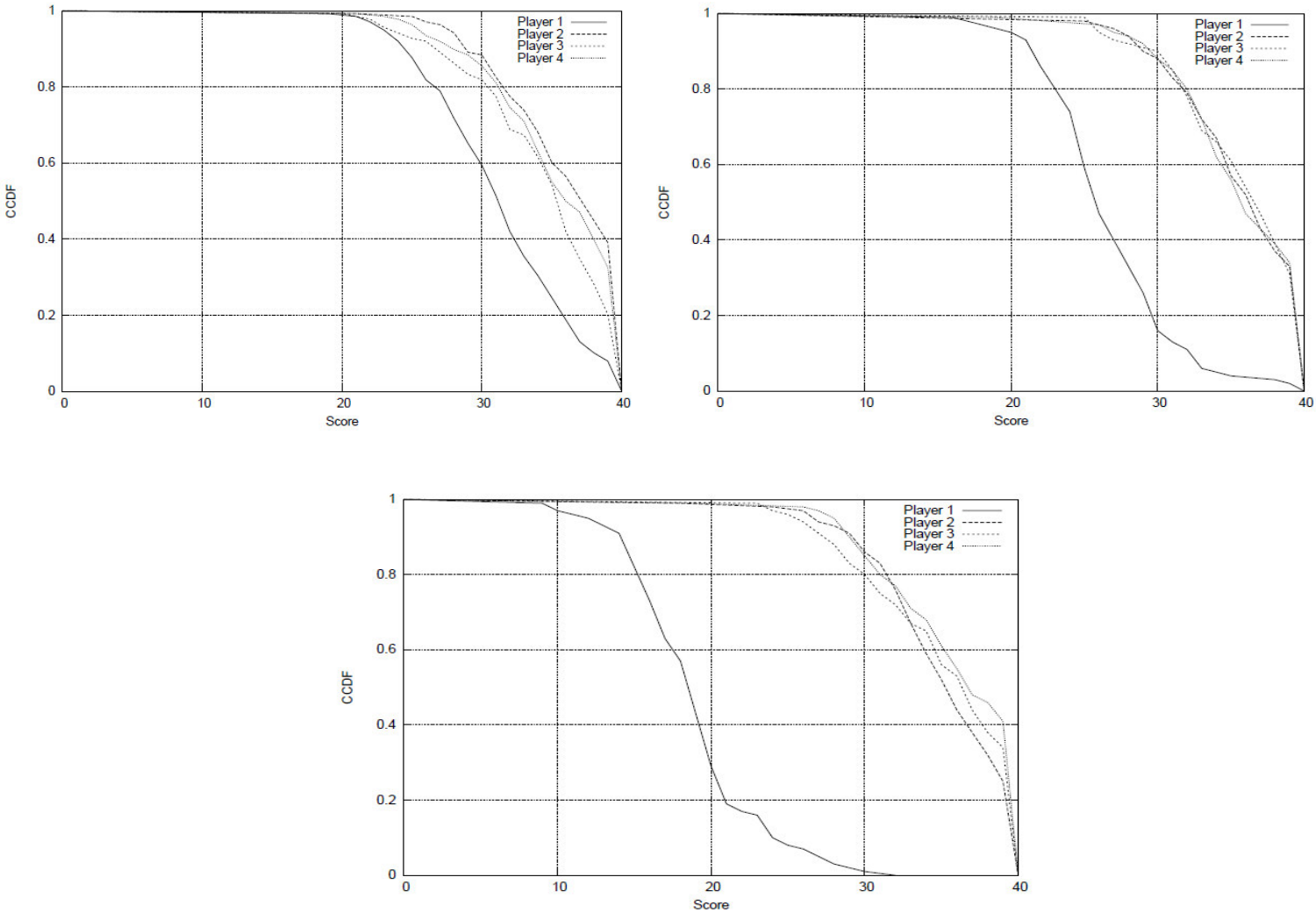


Figure 10 CCDF avec délais de 50, 100 et 150 ms pour joueur 1

Comme décrit ci-dessus, nous croyons fortement que le "débit" d'actions ou de frags c'est-à-dire le fait de tuer ou d'être tué par un autre joueur dans le cas spécifique de quake peut être utilisé pour estimer la satisfaction de l'utilisateur lors d'un jeu en ligne.

Dans les deux figures qui suivent nous présentons respectivement l'évolution du temps moyen mis pour tuer un joueur (Time inter-frags, figure 11) et pour être tué (Time inter-fragged, figure 12) et ce dans le cas d'un délai de 50 ms subit par le joueur 1. Les autres joueur ne subissent aucun délai.

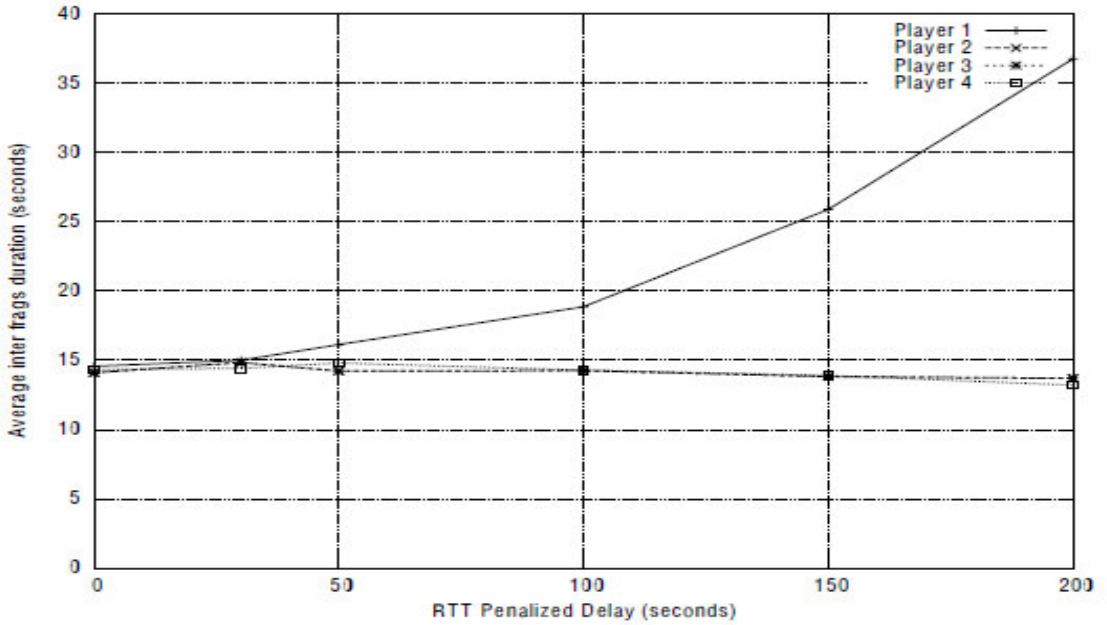


Figure 11 Evolution du temps inter-frags en fonction du délai

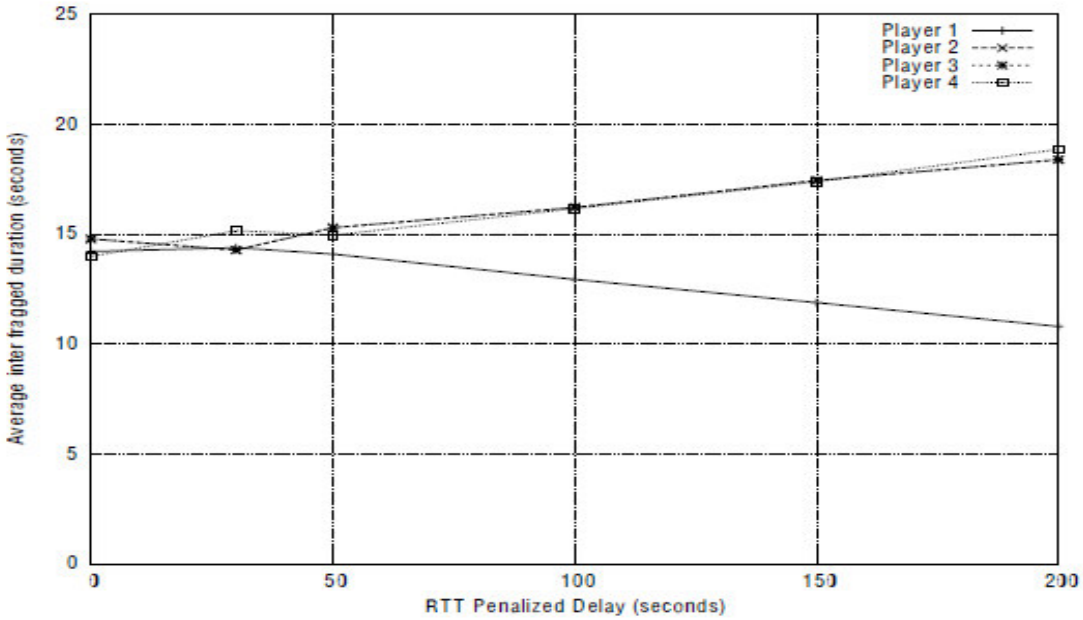


Figure 12 Evolution du temps inter-fragged en fonction du délai

La courbe suivante montre la dégradation de la jouabilité évaluée toujours par la méthode de scores et due au pertes de paquets appliquée uniquement au joueur 1, le taux de perte est de 20 % c'est-à-dire un paquet sur cinq sera détruit si l'émetteur /récepteur est le joueur 1.

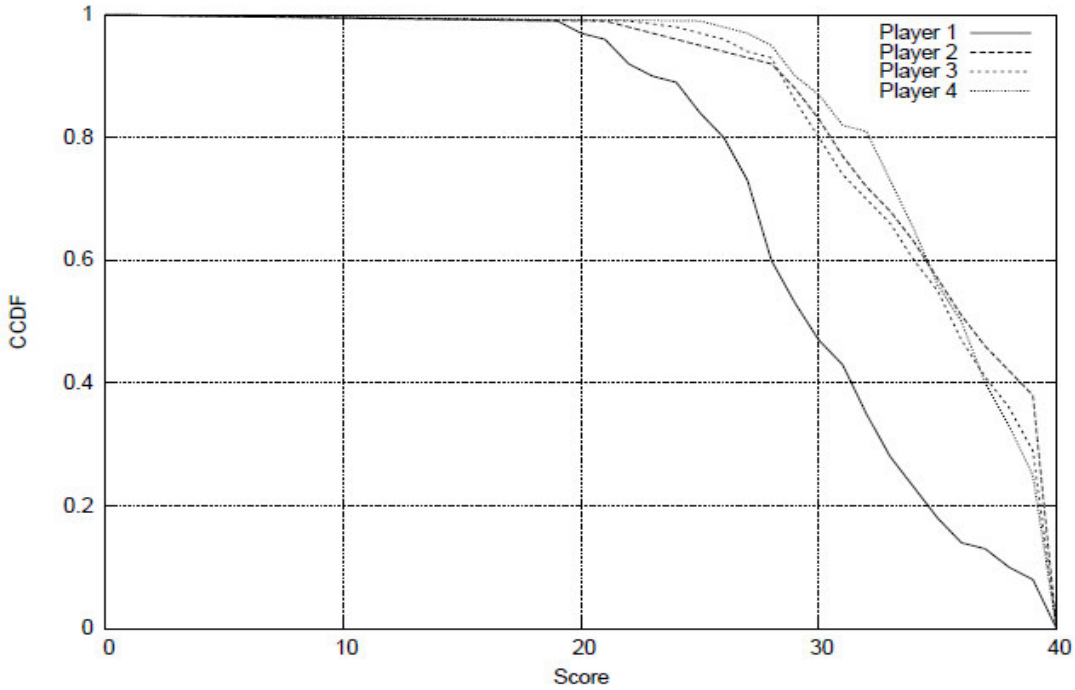


Figure 13 CCDF avec pertes de 20 % (joueur 1)

En ce qui concerne la gigue, nous avons montré que celle-ci observée sur un serveur quake est inférieure à 20 % du délai.
Ce qui veut dire que même avec un délai de 200 ms nous n'avons pas observé un effet sur l'équité.

Nous avons pu voir dans le chapitre 2, malgré que les expérimentations sont relativement longues (15 heures par scénario), les principaux paramètres réseaux qui affectent la jouabilité ou l'équité et les quantifier et nous pouvons à la suite conclure que le délai constitue le paramètre à réduire absolument si nous voulons garantir une certaine satisfaction des joueurs.

Le but étant identifié, la suite du rapport traitera les principales solutions qui peuvent contribuer à réduire le temps de latence dans un environnement mobile ad hoc.

Chapitre 3
Solution de la couche transport: TCP-WELCOME: TCP VARIANT FOR WIRELESS ENVIRONMENT, LINK LOSSES AND CONGESTION PACKET LOSS MODELS

3.1 Problème de TCP dans un environnement mobile ad hoc

Le protocole TCP, élaboré par Vinton Cerf en 1973, fonctionne en commutation de paquets et détient les propriétés suivantes:

- il est *orienté connexion*, c'est à dire qu'une communication fiable est établie entre les deux entités communicantes. Ceci est assuré par une phase de connexion et de déconnexion (three-way handshakes).
- il est *fiable* c'est à dire qu'il assure la délivrance de la totalité des données envoyées, de façon ordonnée et exempte d'erreur.
- c'est un protocole *bout à bout* (end-to-end) c'est à dire que tout le contrôle réside uniquement chez l'émetteur et le destinataire.
- il est *indépendant* vis à vis *des données*
- il régule son débit à travers un mécanisme de *contrôle de flux*, qui prend en compte le volume de données maximum que le récepteur peut recevoir, et un *contrôle de congestion* qui s'active lors de saturation dans le réseau.

  Ceci répond au problème de limitation des ressources dans le réseau en terme d'occupation des buffers.

TCP a été initialement conçu pour porter remède au pertes de paquets liées à la congestion dans les réseaux informatique filaires. Nous discutons brièvement les problèmes de performances de TCP dans un environnement mobile sans fil.

➢ Particularités d'un environnement ad hoc

Les réseaux ad hoc sans fil sont des réseaux sans infrastructure et qui ne nécessitent pas un configuration préexistante ou une administration centralisée. Les entités ad hoc sont totalement connectées via des canaux radio qui sont considérés comme non fiables. Dans de telles réseaux, les nœuds sont indépendantes et auto-organisés.

Les MANETS héritent les problèmes liés aux canaux radio-mobiles en plus des problèmes liés à ses caractéristiques spécifiques comme les partitions du réseaux, les échecs de routes (route failure) engendrés par la mobilité des nœuds, la limitation en termes de batterie et les communications multi-sauts …
Pour améliorer les performances de TCP dans de tels environnements, il doit être capable de distinguer les différentes causes de perte de paquets et pouvoir réagir conformément à ces causes.

Les nouveaux défis que TCP doit confronter dans ce type de réseaux sont canaux radio non fiables, routage multi-trajets, partitions de réseaux, sa topologie, pertes de liens et contraintes de durée de vie de la batterie.

- **Canaux radio mobiles (Wireless Lossy Channel)**

Le médium radio est connu pour avoir un fort taux d'erreur par bit, BER, c'est à dire de paquets au contenu altéré, dû aux interférences, ce qui constitue encore des destructions de paquets.
La couche MAC effectue des retransmissions des paquets, dans la limite de quatre, lorsqu'elle n'a pas reçu d'acquittement. Ce mécanisme qui a pour but d'améliorer la fiabilité rentre en conflit avec celui de retransmission de TCP.

Tout d'abord il influe considérablement sur le calcul du RTT qui est crucial quant au problème de la congestion. Enfin, il n'est pas rare que TCP et la couche MAC effectuent une retransmission d'un même paquet. Ceci va avoir pour conséquence la génération d'acquittements dupliqués. en dessus d'un certain taux d'erreur la plupart des paquets sont retransmis par les deux couches causant alors une perte de l'ordre de 30% du débit par rapport au scénario où TCP est seul garant de la fiabilité.

- **Multi-routes (Multi-path Routing)**

La construction des tables de routage implique l'envoi de nombreux paquets de signalisation. Dans le cas des protocoles réactifs, durant une période assez longue le médium va leur être quasiment dédié.
Le fort coût d'accès au médium en temps et en espace fait que cette période peut-être suffisamment longue pour produire des timeouts sur certaines connexions.
Cependant le problème le plus fréquent, se situe au niveau de l'un des nœuds, dans la perte d'une route pendant la phase de recherche distribuée, ce qui peut provoquer des destructions de paquets conséquentes selon la façon dont le protocole réagit.
Certains protocoles de routage maintiennent plusieurs routes pour une même destination dans le but de diminuer la fréquence de recherche de routes ou encore d'améliorer la connectivité dans un environnement très mobile.
Les différentes routes que vont emprunter les paquets ne sont pas équivalentes en terme de délais, ni de compétions d'accès subit, ce qui va résulter au niveau du récepteur de la connexion en une arrivée plus désordonnée des paquets. La conséquence de ces paquets dits *out-of-sequence* est bien sûr la génération de nombreux acquittements dupliqués qui sont des indices de congestion, mais également un point de difficulté pour le calcul du RTO.

- **Partitions de réseaux (Network partitions)**

Les réseaux mobiles ad hoc peuvent être périodiquement partitionnés pour plusieurs secondes. Si les entités TCP émettrice et réceptrice se retrouvent dans des partitions différentes, tous les paquets de données seront détruits. Si cette phases dure un temps relativement important par rapport au RTO, la source continuera à retransmettre à une destination déconnectée. Plusieurs retransmissions vont engendrer une phase d'inactivité pendant une ou deux minutes même dans le cas ou la source et la destination sont à nouveau dans la même partition.

- **Topologie et environnement**

Le positionnement géographique des nœuds est un facteur très déterminant; si les nœuds sont proches les unes des autres, il va y avoir une grande chance que les données vont emprunter des chemins plus courts.
Il est évident que la consommation d'énergie augmente avec la distance séparant la source et la destination et que les réseaux ayant une dense concentration de nœuds sont marqués par de fortes contentions et par conséquent de plus en plus de collisions et d'interférences ce qui veut dire une grosse perte au niveau des données TCP.

- **Ruptures de liens (Link Failures)**

Le caractère mobile du réseau implique que deux nœuds puissent à tout moment être déconnectés, créant ainsi de nombreuses pertes de paquets, notamment d'acquittements, qui pour TCP vont engendrer de nombreux timeouts consécutifs.

- **Contraintes énergétiques (Power Constrains)**

Dans un environnement ad hoc sans fil, les entités sont indépendantes et ont des batteries en tant que source d'énergie.
Pour assurer la connectivité, il est important que la durée de vie des nœuds soit la plus longue possible. Cela peut être mis en œuvre par la minimisation de la consommation de l'énergie car perdre u nœuds à cause de l'épuisement de sa batterie conduit à des ruptures de sessions (link failure) même si le nœud n'est ni la source ni la destination du paquet en cours.

➢ Vers des conceptions inter-couches

L'architecture actuelle des protocoles réseaux, que ce soit le modèle OSI ou le modèle TCP/IP, repose sur un ensemble de modules devant chacun fournir un service précis et ce de façon autonome. Ces modules on été organisés hiérarchiquement dans une pile, chaque module se trouvant au dessus du service dont il a besoin pour fournir le sien.

Une telle conception a l'avantage de fournir un niveau d'abstraction permettant de découper le problème en sous-problèmes, plus facile à aborder, et par là même d'améliorer la compréhension que l'on a du problème entier.
Le succès de TCP/IP réside certainement plus dans la conception de son architecture que dans les algorithmes qu'il met en œuvre.

Cependant, dès l'apparition d'influences entre deux couches, l'envie est forte que ces couches puissent communiquer entre elles un ensemble de variables, statistiques, qui aideront à la résolution d'un problème. C'est ce que proposent les architectures inter-couches (cross-layers) notamment à travers des retours d'informations (en anglais, feedbacks).
De nombreux *feedbacks* ont été proposés pour les réseaux ad hoc, comme par exemple la couche physique informant la couche MAC de la qualité de la liaison

permettant à ce dernier de décider d'utiliser ou non des mécanismes de contrôle d'erreur. Au niveau de la consommation d'énergie, très importante dans le cas des réseaux de capteurs, on souhaite par exemple que la couche MAC adapte sa portée de transmission ou encore que la couche réseau préfère des nœuds à d'autres lors du routage.
On notera tout de suite que ceci accroît la difficulté pour TCP de fournir son propre service. En effet, la considération précédente sur l'état des batteries mène par exemple directement à une sorte de handoffs.

Au final, comme le prophétise les réseaux sans fil devraient s'orienter vers une architecture comptant un module *cross-layer* auquel les évènements de chaque couche seraient relayés, et qui aurait pour rôle de base de données des variables du réseau.

- **Contrôle des échecs de routage**

L'idée consiste à imposer à un nœud qui détecte la perte d'une route l'envoi d'un paquet de *feedback*. A la réception de celui-ci l'émetteur TCP, que ce soit dans le protocole TCP-F ou ELFN, va bloquer son fonctionnement: gel des horloges, des émissions et des fenêtres de régulation de débit. Ainsi à la fin du blocage (TCP-F attend un paquet de notification de reconstruction de la route alors que ELFN teste par l'envoi de paquets spéciaux si la route est de nouveau disponible) TCP va pouvoir reprendre immédiatement ses envois à un débit concordant avec la réalité du réseau et donc ne pas souffrir d'un départ lent de sa fenêtre de congestion.

- **Contrôle de la congestion**

Venu du monde filaire, le mécanisme ECN permet à un nœud routeur, selon l'occupation moyenne de son buffer, de positionner un bit dans l'entête IP du paquet qu'il est en train de traiter, pour signifier qu'une congestion est probable. Le paquet, en arrivant au récepteur de la connexion TCP va positionner le bit ECN-echo dans quelques-uns des acquittements suivants. l'ECN-echo va avoir pour effet d'inhiber l'accroissement de CWND à l'arrivée de l'acquittement au niveau de l'émetteur TCP. ECN atteint de bonnes performances en prévenant la congestion dans les réseaux ad hoc.
ATCP est l'un des protocoles *cross-layers* le mieux conçu actuellement. Il inclut une nouvelle couche entre la couche réseau et la couche TCP qui a pour but de filtrer les signes de congestion émanant du réseau.

Celle-ci garde trace des acquittements reçus et des horloges de retransmissions. Ainsi lorsque trois acquittements dupliqués arrivent, ou lorsqu'un timeout est proche de s'effectuer, ATCP bloque TCP et effectue la retransmission lui-même des paquets nécessaires, jusqu'au moment où un acquittement non dupliqué arrive. TCP est alors débloqué et l'acquittement lui est délivré.

Enfin ATCP inclut également le mécanisme ECN et un contrôle d'échecs de routage similaire à ceux vus précédemment. L'avantage de ATCP est sa transparence tant au niveau du nœud où il opère que pour l'ensemble du réseau. Malheureusement, si des simulations ont été entreprises, elles ne sont pas de nature à rendre compte de

l'amélioration réelle apportée, et nous ne pouvons donc reporter ici de résultats.
Une dernière approche qui nous a paru prometteuse est celle nommée Split-TCP. Le protocole élit un ensemble de *proxys* tout au long d'une connexion TCP qui vont acquitter, par un paquet nommé Local ACK, l'arrivée du paquet au proxy précédent. Un proxy fait suivre les paquets au débit auquel il reçoit les acquittements Local ACK du proxy suivant. Afin de préserver la sémantique bout-à-bout de TCP les acquittements standards sont conservés.

Au final, la fenêtre de congestion de l'émetteur TCP est calculée à partir de l'ajout de la fenêtre bout-à-bout (ancienne CWND) et de la fenêtre locale variant selon le débit du proxy suivant, cette dernière étant implémentée également au niveau de chaque proxy.

Une amélioration de 5 à 30 % du débit par rapport à TCP. Le plus grand inconvénient réside dans le trafic généré par les acquittements Local ACK et par l'élection des proxys, surtout dans un environnement mobile. Cependant on insistera sur le fait que c'est le seul protocole qui ne propose pas de bloquer TCP mais de réguler son débit selon ce qu'expérimente la connexion sur des tronçons du réseau, à savoir entre deux proxys voisins.

Pour conclure, la figure suivante montre la dégradation de débit dans le cas de l'utilisation d'une variante TCP classique en environnement ad hoc et comparé à celle en environnement filaire et ce en fonction du nombre de nœuds.

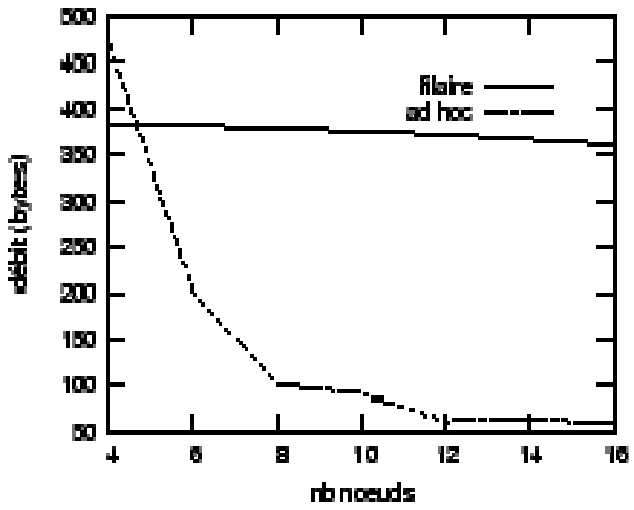


Figure 14 Variations du débit TCP dans un environnement MANETS

3.2 TCP WELCOME : une alternative adaptée aux MANETS

3.2.1 Principe de fonctionnement

Afin d'améliorer les performances de TCP face aux différentes pertes de paquets dans un environnement ad hoc sans fil, TCP doit être capable de réagir différemment et selon la cause de perte. Le comportement de TCP doit être comme suit:

Dans le cas d'un taux d'erreur binaire (TEB ou BER) important, il n'est pas nécessaire d'arrêter la transmission ou de diminuer le débit d'émission TCP suite à l'apparition de la perte.

Dans le cas d'une perte de lien (Link Failure) dans le réseau (ie une rupture de route entre les nœuds extrêmes d'une connexion), il sera suffisant d'arrêter la transmission jusqu'à ce que une nouvelle route sera calculée. Dans ce cas de figure le débit de transmission doit être ajusté en fonction de la bande passante disponible sur la nouvelle route. Il est évident que la longueur du chemin de communication a son impact sur le temps aller-retour (Round Trip Time = RTT) entre les nœuds extrêmes donc il est nécessaire de recalculer les valeurs du débit de transmission TCP (CWND) et le temps avant retransmission (Retransmission Time Out = RTO) selon les caractéristiques de la nouvelle route.

Dans une situation de congestion dans le réseau, TCP garde son comportement ordinaire; il réagit selon la manière dont on a détecté cette congestion c'est-à-dire soit trois acquittements dupliqués soit un RTO. Dans tous les cas de congestion, TCP arrête sa transmission pendant une certaine durée puis il reprend mais avec une réduction du débit.

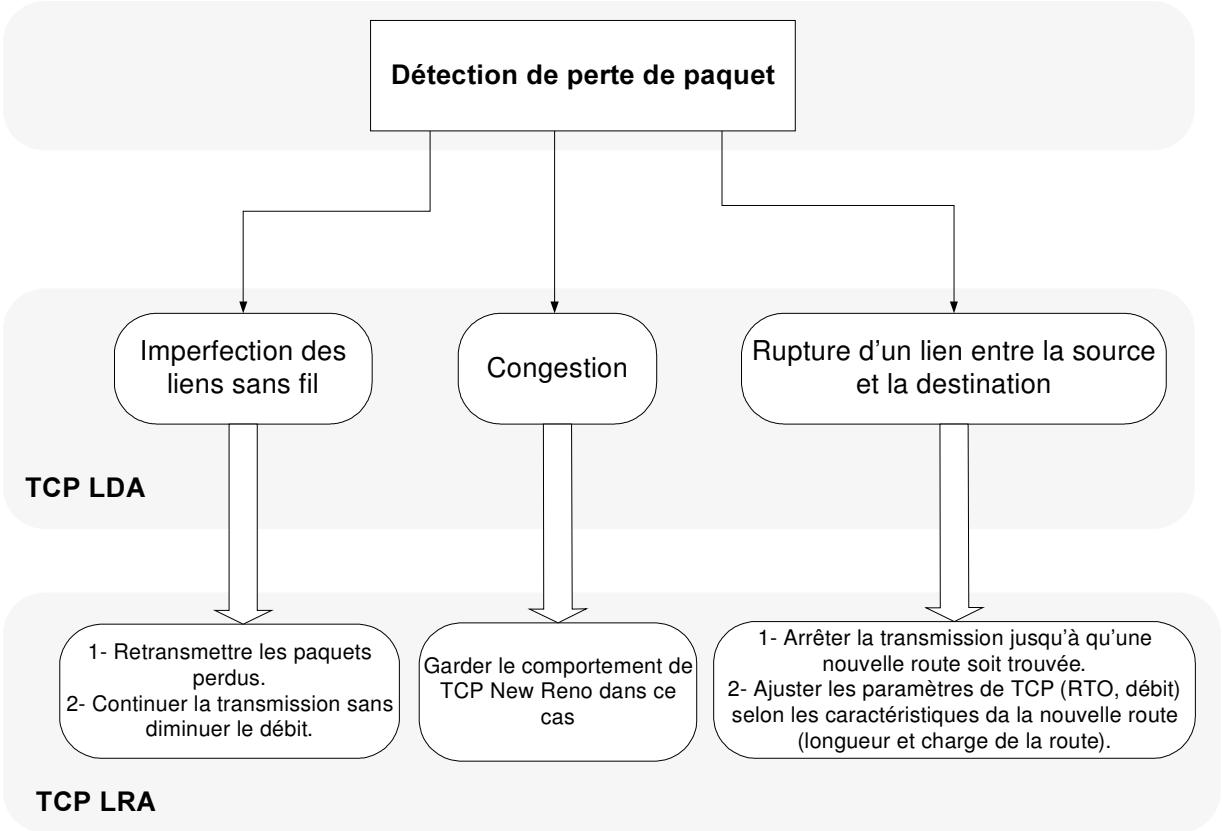


Figure 15 Algorithmes Proposés pour TCP WELCOME

La figure ci-dessus résume le principe de fonctionnement de TCP WELCOME face aux différentes pertes. Ce comportement est géré par deux algorithmes. Dans la suite on propose de décrire les règles sur lesquelles TCP WELCOME se base pour distinguer entre les différents cas de pertes définies précédemment (***Loss Differentiation Algorithm = LDA***) et le comportement face à chaque type de perte (***Loss Recovery Algorithm = LRA).***

3.2.2 Algorithmes de base : LDA et LRA

➢ LDA: Loss Differentiation Algorithm

- **RTT base de classification**

Pour incarner les solutions proposées dans la partie précédente on besoin d'un algorithme LDA adapté et qui permet de classer correctement les causes de pertes des paquets dans un réseau ad hoc utilisant le sans fil.

Nos deux algorithmes LDA et LRA sont conçus pour fonctionner de bout en bout et ce dans le but de garder la notion de connexion TCP intacte. Ils sont basés sur l'évolution de l'historique des échantillons RTT du côté de la source pour pouvoir décider. L'évolution de cette historique constitue une indication très efficace sur la cause des pertes au sein d'une connexion et on verra dans la suite comment les échantillons de RTT peuvent être utilisés pour classifier les causes de pertes.

On note dans la suite qdi (t), Pdi (t) et pdi (t) respectivement le temps de séjour dans la file d'attente (Queuing), de traitement (processing) et de propagation calculés sur un nœud i à une date donnée t.

La figure ci-dessous illustre les délais de transmission d'un segment TCP jusqu'à la réception de l'acquittement par le nœud source.

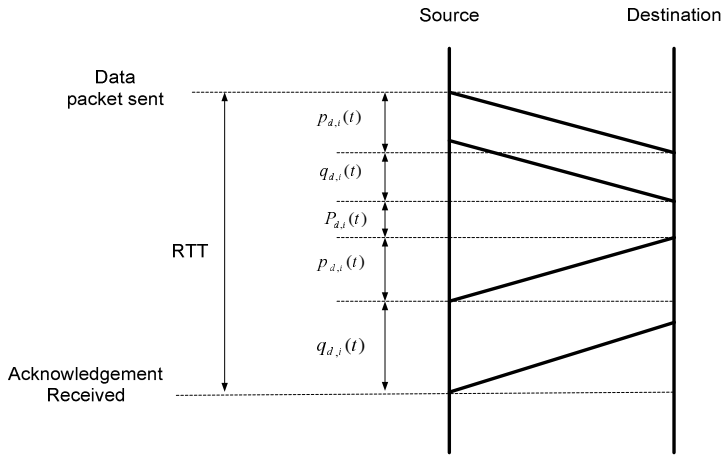


Figure 16 RTT d'un segment TCP

Les valeurs de RTT d'une connexion TCP sur une route contenant m sauts (hops) à une date t peut être calculé à l'aide de la formule suivante :

$$RTT(t) = 2\sum_{i=1}^{m}(q_{d,i}(t) + p_{d,i}(t) + P_{d,i}(t))$$

Dans cette équation, seulement le temps de propagation et de Queuing sont affectés par les conditions du réseau, le temps de traitement quant à lui dépend uniquement des capacités en termes de processeur du nœud en question.

Par exemple dans le cas d'une rupture de lien c'est le temps de propagation qui va changer en fonction de la longueur de la nouvelle route. Par contre dans le cas d'une congestion, c'est le temps de séjour dans la file d'attente qui augmentera.

L'historique de l'évolution des RTT étant établi algorithme LDA repose sur l'observation de cette historique et selon l'évolution des RTT, TCP doit être capable d'identifier la cause de la perte en l'occurrence.
Dans la figure suivante, on décrit les règles de cette classification.

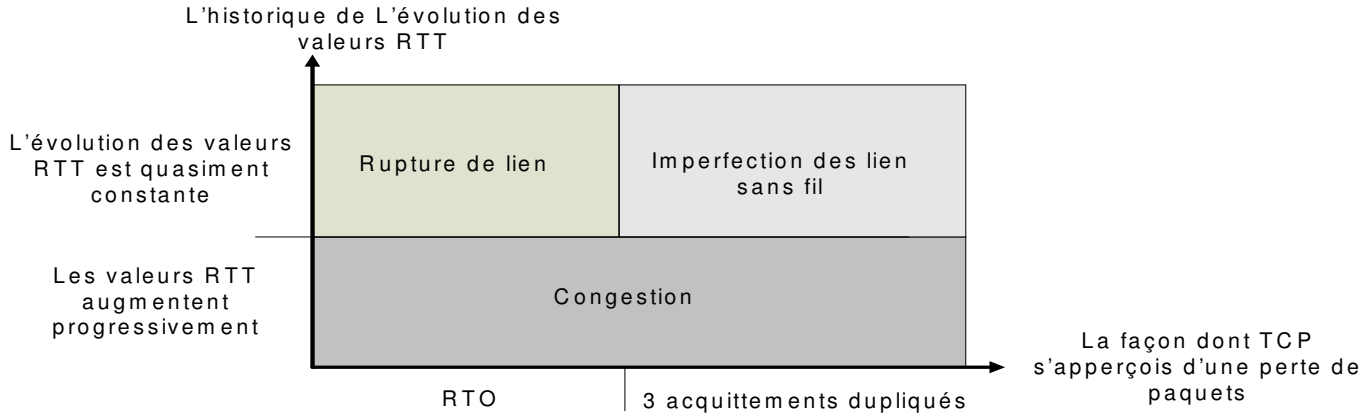


Figure 17 Règle de classification du LDA

- **Les pertes liés au canal radio (wireless channel related losses)**

Si l'évolution des valeurs de RTT sur une connexion ne présente pas beaucoup de fluctuations et reste autour d'une valeur moyenne et que la perte a été identifié via trois acquittements dupliqués, on pourra conclure alors que le problème est du aux imperfections du canal radio.

Pour ce type de canaux, et dans le cas particulier d'un taux d'erreur binaire élevé, les temps de propagation et de Queuing restent quasiment constant ainsi les RTT ne doit pas trop fluctuer dans le temps.

D'autre part, quand on a une route disponible entre la source et la destination et malgré la présence d'erreurs de transmission sur un ou plusieurs liens, la source continue à recevoir les acquittements de la destination; la corruption d'un acquittement est peu probable grâce à sa taille relativement petite.

- **Rupture de lien (Link Failure Loss EVENT)**

Dans le cas ou l'évolution des échantillons de RTT sur une connexion TCP est relativement constante et que la reconnaissance de perte se fait grâce à une expiration du retransmission time out (RTO) alors on peut conclure que ces pertes sont due à une rupture de lien le long d'une route menant à la destination.

Dans cette situation, après rétablissement d'une nouvelle route, on peut remarquer que les temps de propagation et de Queuing changent brusquement. Ceci est du au fait que (i) le nouveau chemin menant à la destination n'a pas la même longueur que son prédécesseur et que (ii) il peut être plus chargé que lui.

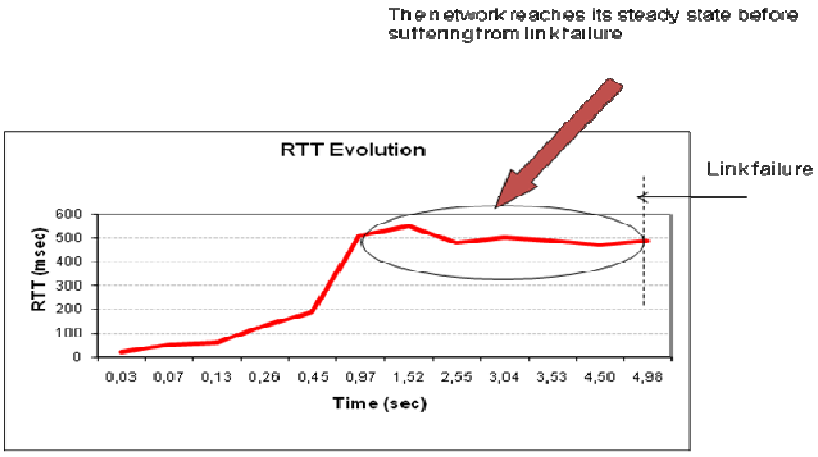


Figure 18 évolutions des RTT suite à une rupture de lien

Soit k le nombre de sauts (hops) le long du nouveau chemin

$$RTT(t) = 2\sum_{i=1}^{k}(q_{d,i}(t) + p_{d,i}(t)) \approx Const.$$

Selon le temps Tr mis par le protocole de routage pour trouver un autre chemin ; (i) si Tr est inférieur à RTO, les pertes sont découverts grâce au DUP ACK et dans ce cas TCP considère que c'est un problème lié au canal radio. (ii) sinon il considère que c'est une rupture de lien.

- **Congestion sur le réseau**

Si les valeurs des échantillons RTT évoluent progressivement alors les pertes sont dues à une congestion dans le réseau. Le problème étant détecté soit par expiration du RTO soit par 3 DUP ACK.
Ici le temps de propagation reste presque constant et le temps de Queuing évolue progressivement.

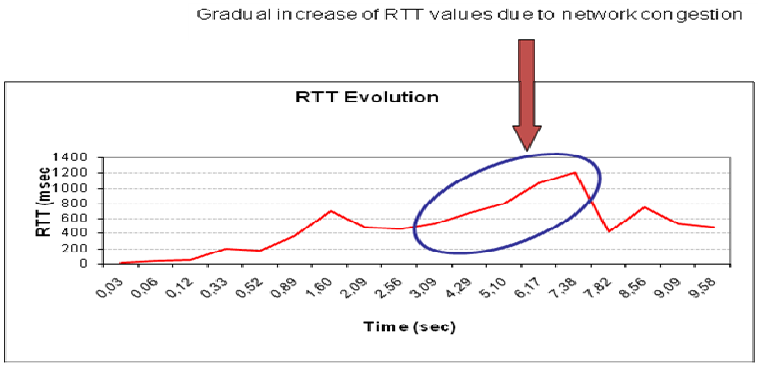


Figure 19 évolutions des RTT suite à une congestion

➢ LRA: Loss Recovery Algorithm

Dans les sections suivantes nous décrivons comment les paramètres d'une connexion TCP (CWND et RTO) sont ajustés suite à une perte de paquets de données et identification de la cause utilisant l'algorithme LDA; c'est l'algorithme LDA.

- **Adaptation du RTO (RTO Adaptation Algorithm)**

L'estimation du RTO est différente de celle du RTT pour plusieurs raisons; un RTO doit respecter un compromis entre une valeur importante pour ne pas passer le temps à retransmettre des paquets sans une nécessité réelle et une valeur petite permettent de réagir le plus rapidement possible face à une perte.
TCP WELCOME ajuste la valeur de RTO selon la cause de la perte ; dans le cas d'une imperfection du canal, aucune estimation du RTO ne doit être effectuée. Mais dans le cas d'une rupture de lien, elle doit être ajustée selon la longueur et la charge de la nouvelle route.

On note RTTold l'ancienne valeur de RTT sur la route perdue et RTT new la valeur du RTT sur la nouvelle route. La nouvelle valeur de RTO peut être calculée à l'aide de la formule suivante :

$$RTO_{new} = \left(\frac{RTT_{new}}{RTT_{old}} \right) RTO_{old}$$

Dans le cas d'une congestion, on gardera la même valeur du RTO.

- **Adaptation du débit de transmission**

L'approche de TCP WELCOME est d'effectuer une estimation de la bande passante disponible du côté de l'entité émettrice pour éliminer les effets des interactions inter-nœuds le long d'une connexion TCP.
Dans le cas d'une imperfection du canal radio TCP WELCOME ne modifie pas son débit de transmission. Par contre dans le cas d'une rupture de lien le débit doit être ajusté selon les caractéristiques de la nouvelle route et dans ce cas plusieurs types d'ajustement peuvent être effectués ; (i) soit on garde le même débit (ii) soit être plus conservatif et réduire à moitié la fenêtre CWND par rapport à celle avant les pertes et le nouveau ssThreshold sera calculé par $ssThreshold = Bw_{estimated} * RTT_{\min}$.

3.3 Plateforme de tests SEDLANE: Simple Emulation of Delays and Losses for Ad hoc Network Environment.

Dans cette section on propose de présenter SEDLANE notre outil d'émulation conçue pour être utilisé dans les travaux de tests et validation de la variante TCP WELCOME du protocole de transport dans l'environnement mobile ad hoc.

➢ Motivations

Le Throughput TCP peut être calculé en utilisant le temps RTT et le taux de perte de paquet sur une connexion.
On propose dans ce qui suit de calculer le throughput en utilisant la formule suivante:

$$r_{TCP} = \frac{1.22 * M}{\tau * \sqrt{l}}$$

r_{TCP} : the TCP connection throughput
M : the maximum packet length,
τ : the round trip time (RTT) of the connection
l : the average loss measured during the lifetime of the connection.

Comme cette formule accorde une grande importance à l' RTT et au taux de perte pour évaluer le throughput, pour tester des protocoles et des applications, on propose alors d'utiliser un émulateur de réseaux ad hoc qui manipule ces paramètres.

➢ Outils

Dans cette section nous proposons les outils logiciels qui permettent de réaliser la plateforme SEDLANE.

- **Dummynet**

C'est un trafic shaper qui a été conçu pour tester les protocoles réseaux, avec Dummynet nous pouvons fixer le délai, les pertes de paquets, les talles des files d'attente et les limitations en termes de bande passante.
Cet outil est entièrement contrôlé par la commande système ipfw (IP Firewall). Ipfw permet de définir les règles à appliquer à un trafic qui traverse une interface en input ou en output.
Dummynet implémente le concept de pipes qui sont des canaux de communication entre la source et la destination. Chaque paquet doit être géré selon les règles associées à chaque pipe avec un overhead très réduit.

- **Network Simulator – 2**

C'est un simulateur open source utilisé pour simuler une très grande variété de réseaux.

➢ SEDLANE

- **Principes**

L'idée du SEDLANE est de configurer les pipes de Dummynet (règles) en utilisant les traces issues de NS-2. SEDLANE utilise les traces TCP de NS-2 pour identifier les différentes classes de données.

Ceci est effectué en regroupant les paquets qui ont des RTT proches. Puis SEDLANE configure un pipe, ou un canal de communication, pour chaque groupe de paquets. Les taux de pertes de paquets peuvent aussi être calculé à partir des traces TCP de NS-2 et ensuite appliqués aux pipes de Dummynet.

La figure ci-dessous donne une vue panoramique sur le fonctionnement de SEDLANE.

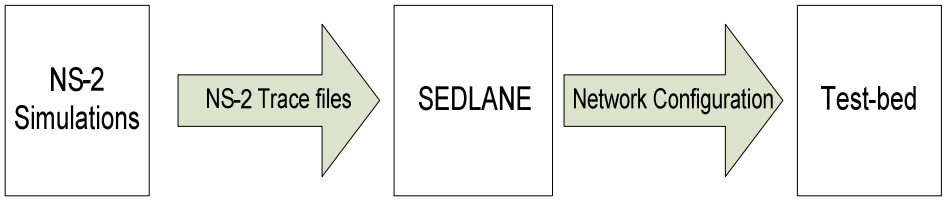


Figure 20 Principe de fonctionnement de SEDLANE

- **Modes opérationnels**

SEDLANE peut suivre l'un des deux modes opérationnels ; le mode simultané et le mode séquentiel.

Dans le mode simultané on suit exactement le fonctionnement ordinaire de Dummynet car on configure tous les pipe avant le début de la communication et tous à la fois. Une probabilité est attribuée à chaque canal (pipe). ce qui veut dire qu'aucune contrainte temporelle n'est utilisé lors de la simulation..
Le schéma suivant illustre le mode simultané.

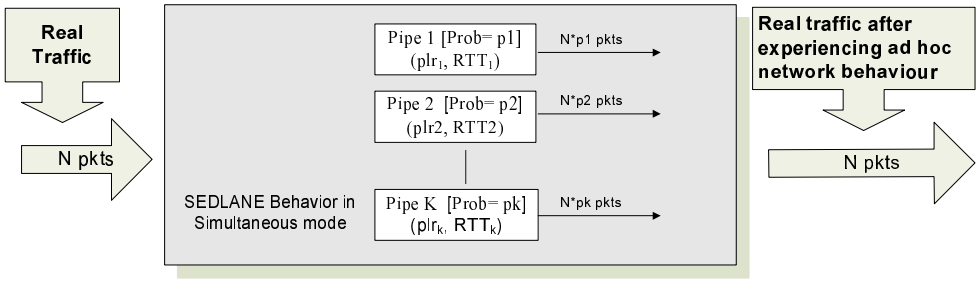


Figure 21 Mode opérationnel simultané de SEDLANE

Dans le mode séquentiel, SEDLANE configure une seule règle ipfw à la fois et chaque règle sera détruit après un certain temps; temps au cours duquel la connexion simulée garde un RTT particulier.

- **Architecture**

SEDLANE extrait les données suivantes de fichiers trace de NS-2 : Le nombre et les valeurs de différents échantillons RTT, taille totales des données transmises et durant quels RTT, taille totales des données retransmises et durant quels RTT, temps de la connexion, temps de vie de chaque RTT, taille total des données détruites et au cours de quels RTT.

- **Algorithmes**

Le nombre de pipes ne doit pas être trop grand pour qu'il n'affecte pas la plateforme de test en monopolisant des ressources importantes.

Si le nombre de valeurs (K) de RTT trouvés dans le fichier trace est inferieur à celui (N) fournies par l'utilisateur alors SEDLANE consacre un pipe Dummynet à chaque RTT.

Sinon SEDLANE recalcule les valeurs de RTT selon l'équation suivante :

$$RTT_{new_1} = \frac{RTT_i + RTT_{i+1}}{2}$$

$$RTT_{new_2} = \frac{RTT_{i+2} + RTT_{i+3}}{2}$$

…

Ce calcul est répété jusqu'à ce que Knew <= N.

La probabilité de chaque pipe représente le poids de chaque règle pour être appliquée sur un paquet en entrée.
Elle est calculée comme suit :

$$prob_{RTT_i} = \frac{txbytes_{RTT_i}}{txbytes_{total}}$$

3.4 Résultats de tests : une étude comparative des variantes TCP

➢ Matériel et logiciel

Pour valider l'implémentation linux de la variante WELCOME de TCP à l'aide d'une plateforme de test réelle. Nous étudions le throughput et la consommation d'énergie ; les algorithmes de contrôle de congestion de TCP exigent l'utilisation de d'opérations mathématiques complexes pour calculer les valeurs des différents timers…

La figure suivante représente notre plateforme de test sur la quelle on utilise:

- Source : Un PC Dell Latitude D610. OS: Linux noyau 2.6
- Destination : Un PC Dell Latitude D420. OS: Linux noyau 2.6
- Emulateur SEDLANE : Un PC HP compaq nc6000 avec OS : FreeBsd 7.2 Release et un noyau recompilé pour tenir compte de Firewall IP
  - une connexion wifi est établie entre interface ath1 de la Source et ath0 de l'Emulateur en mode ad hoc.
  - une connexion filaire est établie entre interface bge0 de l'Emulateur et eth0 de la Destination.
    - ==> Tout flux entre PC1 et PC3 passe nécessairement par PC2 (SEDLANE).
- Un multimètre sur ma machine source qui mesure l'énergie dissipée.

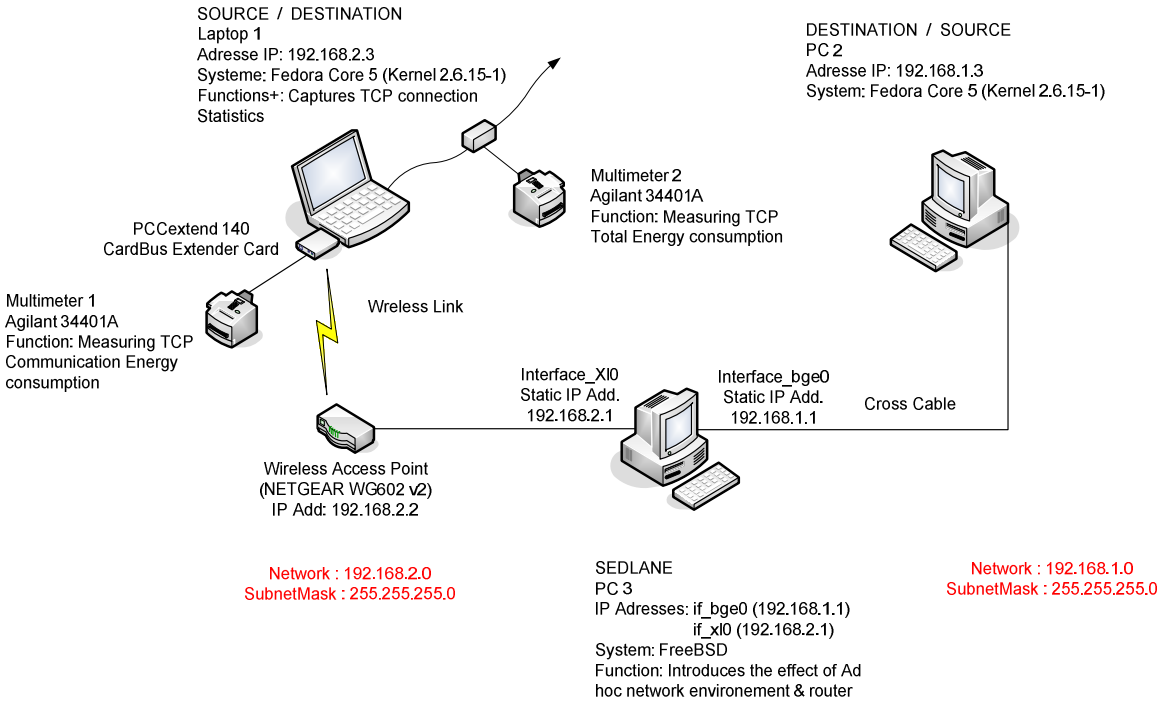


Figure 22 Plateforme de test de TCP WELCOME

➢ Scénarios testant l'application jeu

Le premier test consiste à :

-Lancer le jeu ioquake 3 sur la source (rôle de serveur de jeu) et la destination (rôle client connecté au serveur) en mode automatique (Slugbot) afin de comparer les débits moyen des flux échangés entre la source et la destination et ce en utilisant les différentes variantes de TCP. La durée de chaque expérience est fixée à une heure (3600s).

La figure ci-dessous illustre les résultats du test. Nous pouvons clairement noté que le débit moyen des flux TCP échangés entre le serveur du jeux et le clients dans le cas de TCP WELCOME est nettement supérieur au débits moyens des autres variantes.

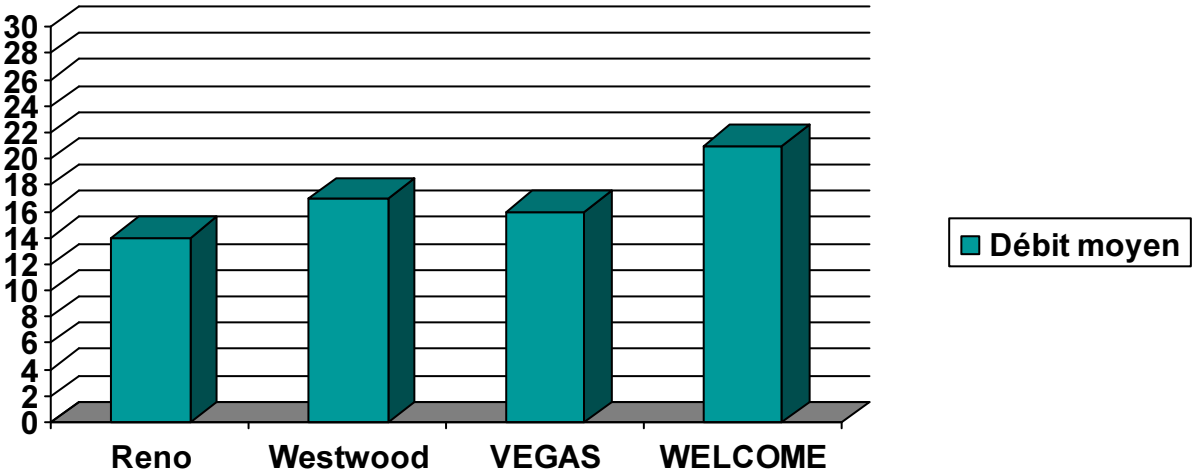


Figure 23 débits moyens des flux de jeux générés en une heure

Le second test consiste à

-Lancer Iperf sur le client. Ceci simule le client qui télécharge une scène/personnage/vidéo (taille utilisée 20Mo)

-Ce test est effectué avec différentes variantes de TCP (Reno, Westwood, Vegas et Welcome).

Et les métriques de performances sont :

- Temps de téléchargement
- Débit moyen des flux générés
- Consommation d'énergie

Dans le cas de TCP RENO les résultats est donné dans la capture suivante

```
pfxh7245@l-at9216:~$ iperf -c 192.168.10.3 -n 20m
------------------------------------------------------------
Client connecting to 192.168.10.3, TCP port 5001
TCP window size: 16.0 KByte (default)
------------------------------------------------------------
[  3] local 192.168.11.2 port 34019 connected with 192.168.10.3 port
5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-63.0 sec  20.0 MBytes  2.67 Mbits/sec
```

Le transfert se fait donc en 63 secondes avec un débit moyen de 2.67 Mbits/s

La même expérience a été réalisée mais en utilisant le TCP WELCOME et les résultats sont dans la capture suivante:

```
pfxh7245@l-at9216:~$ iperf -c 192.168.10.3 -n 20m
------------------------------------------------------------
Client connecting to 192.168.10.3, TCP port 5001
TCP window size: 16.0 KByte (default)
------------------------------------------------------------
[  3] local 192.168.11.2 port 59603 connected with 192.168.10.3
port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-26.4 sec  20.0 MBytes  6.34 Mbits/sec
```

Le transfert se fait donc en 26.4 secondes avec un débit moyen de 6.34 Mbits/s

Dans la figure ci-dessous nous représentons l'évolution des nombres de séquences TCP (RENO et WELCOME) en fonction du temps lors du test précédent.

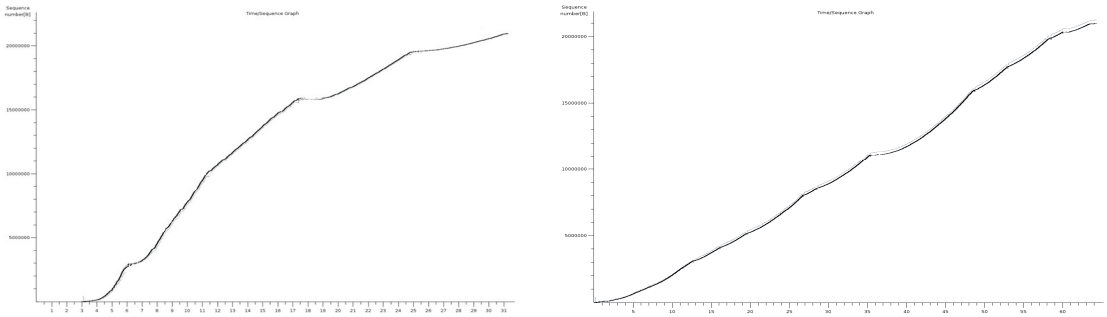


Figure 24 Evolution du numéro de séquences TCP en fonction du temps

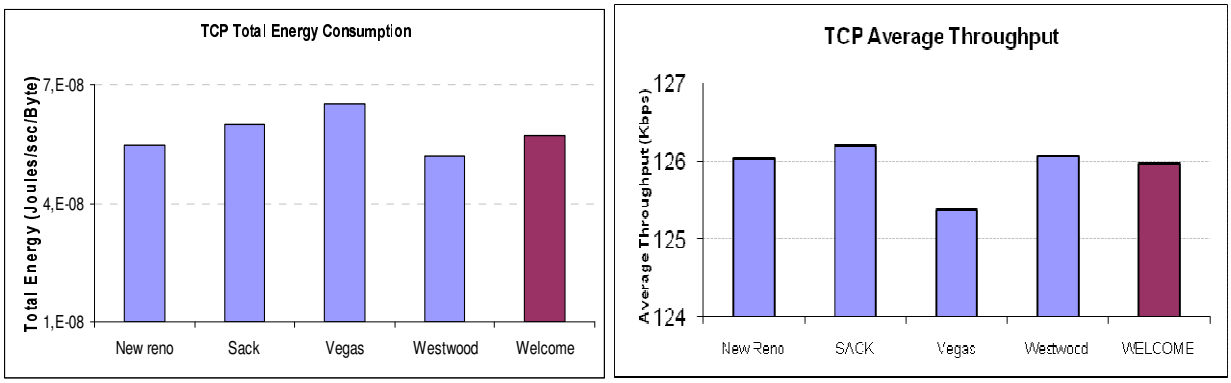
Nous pouvons noter que :
- Une évolution quasi linéaire du numéro de séquence TCP en fonction du temps dans le cas RENO.

- L'évolution devient plus importante (exponentielle) dans le cas de TCP WELCOME.

➢ Tests transparents à l'application: Throughput (débit) et consommation d'énergie

Dans les la suite on donnera les résultats des expériences effectuées sur plusieurs variantes de TCP afin de comparer leurs performances en termes de throughput et de consommation de l'énergie.

- **Cas de congestion**



Consommation d'énergie totale Débit de connexion

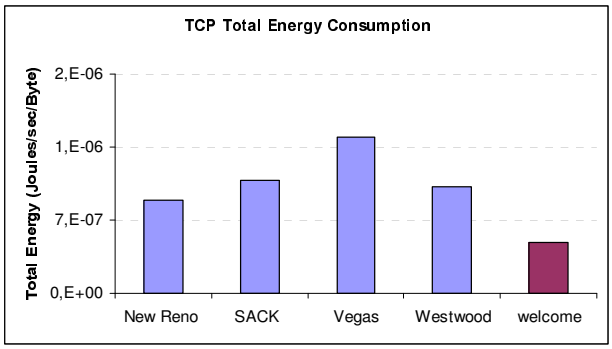
Figure 25 Performances de TCP WELCOME : cas de congestion

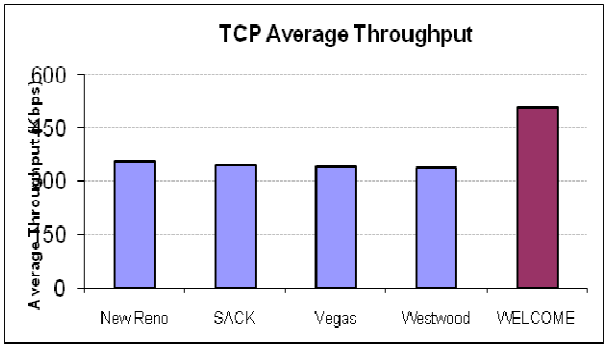
TCP-WELCOME a une performance comparable à TCP New Reno et TCP Westwood :

- Les meilleurs en cas de congestion

- TCP-WELCOME est capable de bien classifier la cause de la perte

- **Cas d'interférences**





Consommation d'énergie totale

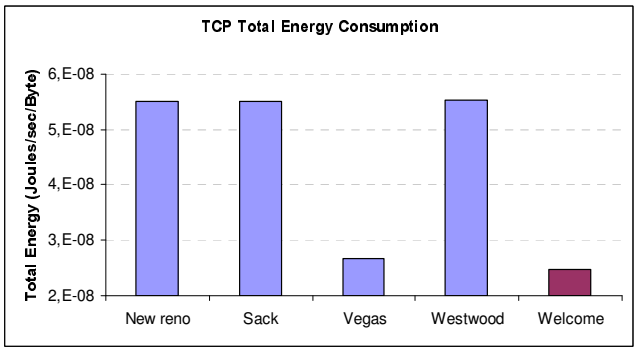
Débit de connexion

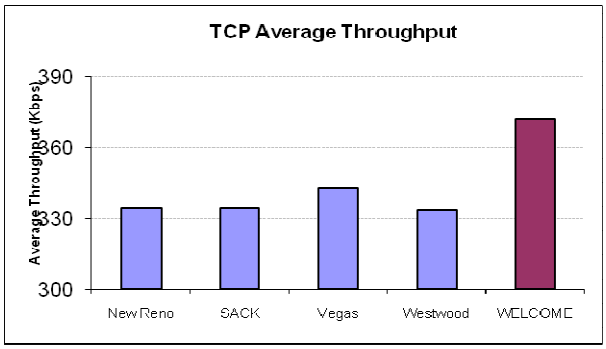
Figure 25' Performances de TCP WELCOME : cas d'interférences

TCP-WELCOME a une meilleure performance que TCP SACK

- TCP-WELCOME ne diminue pas son débit de transmission après une perte due à l'environnement sans fil

- **Cas de perte de lien**





Consommation d'énergie totale

Débit de connexion

Figure 25'' Performances de TCP WELCOME : cas de perte de lien

TCP-WELCOME a une meilleure performance que TCP Vegas

- TCP-WELCOME adapte ses paramètres de performance selon les caractéristiques de la nouvelle route

Nous avons vu dans ce chapitre que TCP souffre d'une dégradation de performance dans les réseaux ad hoc multi-sauts (ressources énergétiques et débit) Car TCP n'est pas capable d'identifier la cause de pertes de paquets dans le réseau Il fallait donc procéder à bien classifier et bien réagir selon la cause de la perte .

TCP-WELCOME réalise cela grâce à :

- Un nouvel algorithme de classification de pertes de paquets (LDA)
- Un nouvel algorithme de récupération des paquets après perte (LRA)

L'évaluation des performances montre que TCP-WELCOME optimise le débit obtenu et la consommation d'énergie grâce aux nouveaux algorithmes introduits.

Dans le chapitre suivant nous essayons de descendre un à deux pas dans L'architecture du modèle OSI pour examiner la couche de routage et proposer les améliorations convenables.

Chapitre 4
Solution de routage : OLSR niveau 2

4.1 Routage proactif : le protocole OLSR

Les algorithmes proactifs construisent les tables de routage avant que les demandes de trames n'apparaissent sur le réseau. A tout moment, un nœud connaît la topologie du réseau.
Ici, sera présenté le protocole OLSR

OLSR : Optimized Link State Routing Protocol

Ce protocole est une adaptation du protocole d'état de lien. Il réduit la taille des messages de contrôle et minimise l'inondation du trafic de contrôle. Il se base sur le nombre de sauts pour fonctionner, ie le nombre de nœuds à traverser pour atteindre un autre nœud.

Ce protocole échange régulièrement les tables de routage sur le réseau. Pour ce faire, il utilise les relais multipoints (MPR). Chaque nœud du réseau élit ses MPR parmi ses voisins à un saut avec un lien symétrique. Ceci permet à un nœud d'accéder à tous ses voisins à deux sauts.

Les nœuds MPR annoncent régulièrement leur rôle de MPR à leur voisinage. Les MPR permettent la mise en place des routes sur tout le réseau, car ce sont les seuls à pouvoir retransmettre les paquets sur le réseau. L'avantage de choisir les MPR de cette façon permet d'éviter la retransmission des paquets sur les liens unidirectionnels. Le fonctionnement L'idée de ce protocole est de minimiser la diffusion dans le réseau.

Pour cela, chaque nœud doit élire le minimum de multipoints réseaux, qui sont les seuls à pouvoir retransmettre dans le réseau. La suite présente donc cette élection.

Tout nœud émet des paquets Hello présenté ci après, indiquant la liste de ces voisins. Ainsi, si un nœud récupère tous les paquets Hello de ses voisins, il peut connaître tous ses voisins à deux sauts.

A partir de cette topologie, le but est qu'un nœud puisse accéder à tous ses voisins à deux sauts. Pour illustrer ceci, la figure 26 montre une situation. Nous allons nous intéresser au cas du nœud du milieu. Celui-ci est à deux sauts de tous les nœuds, il doit donc pouvoir tous les atteindre par l'intermédiaire des MPR.

Les MPR possibles sont les points d'accès A, B, C et D, puisqu'ils sont reliés directement au nœud central. Si on élit C et D, on se rend compte que les nœuds liés seulement à A et B ne seront pas accessibles. On comprend donc qu'il faut élire A et B. Si seuls A et B peuvent retransmettre les paquets, il n'y a pas de points d'accès inaccessibles directement ou par le biais
de la retransmission de A ou de B. Le but est donc atteint, tous les nœuds sont accessibles par ses deux MPR.

Il est assez facile de résoudre visuellement ce problème dans de nombreux cas, mais ce problème est NP-complet. Pour le résoudre, les nœuds utilisent une heuristique.

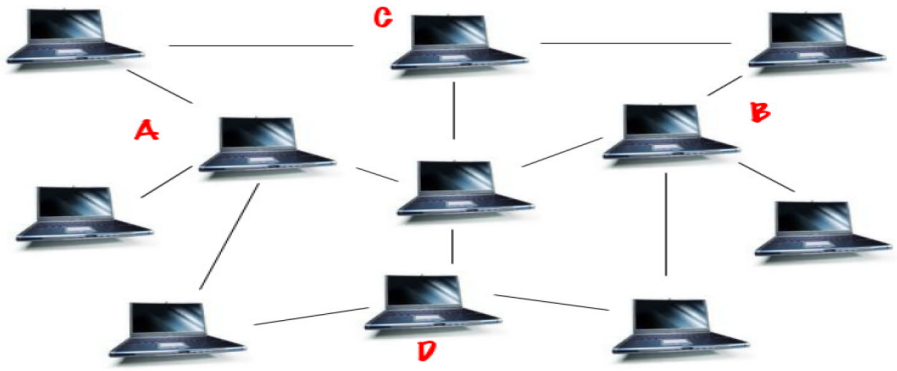


Figure 26 L'élection des multipoints relais

Une fois l'élection faite, chaque nœud informe son choix à son ou ses MPR. Les MPR savent donc qu'ils sont les seuls à retransmettre les messages. Chaque nœud ne connaît qu'une topologie locale du réseau. Il connait la liste de tous les nœuds accessibles sur le réseau et stocke son MPR associé vers lequel il doit envoyer le paquet pour que celui-ci arrive à destination.

Les paquets Hello et TC (Topologie Control) Les paquets Hello sont toujours envoyés en diffusion. Ils servent à découvrir les réseaux en transmettant l'état et les types de lien entre l'expéditeur et ses voisins à un saut. Il sert aussi à choisir son MPR.

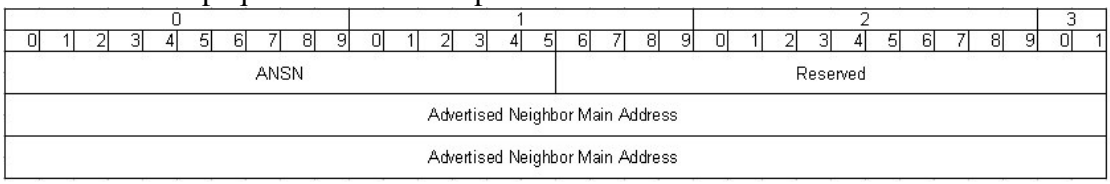
Le format du paquet Hello est donné par :

<table>
<tr><td colspan="10">0</td><td colspan="10">1</td><td colspan="10">2</td><td colspan="2">3</td></tr>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>0</td><td>1</td></tr>
<tr><td colspan="16">Reserved</td><td colspan="8">Htime</td><td colspan="8">Willigness</td></tr>
<tr><td colspan="8">Link Code</td><td colspan="8">Reserved</td><td colspan="16">Link Message Size</td></tr>
<tr><td colspan="32">Neighbor Interface Address</td></tr>
<tr><td colspan="32">Neighbor Interface Address</td></tr>
<tr><td colspan="32">..</td></tr>
<tr><td colspan="8">Link Code</td><td colspan="8">Reserved</td><td colspan="16">Link Message Size</td></tr>
<tr><td colspan="32">Neighbor Interface Address</td></tr>
<tr><td colspan="32">Neighbor Interface Address</td></tr>
</table>

Voici la description des champs importants :
- Reserved : ce champ vaut toujours 0.
- Htime : durée entre 2 émissions de paquets Hello.
- Willigness : pour imposer à un nœud de devenir MPR
- Link Code : Code identifiant le type de lien (pas d'information, symétrique, asymétrique,etc.) entre l'expéditeur et les interfaces listées ("Neighbor Interface Address ")

On ne route pas les messages Hello, sauf pour l'élection des MPR.
Concernant les paquets TC pour Topologie Control, ils ne sont seulement envoyés que par les MPR. Ils permettent d'établir les tables de routage.

Le format du paquet TC est donné par :



Voici la description des champs importants :
- Reserved : ce champ vaut toujours 0.
- ANSN (Advertised Neighbor Sequence Number) : Entier permettant de savoir la "fraîcheur" de l'information. Quand un nœud reçoit un paquet TC, il vérifie que son numéro est égal ou inférieur à celui du paquet. S'il est supérieur le paquet n'est pas traité.
- Advertised Neighbor Main Address : Adresse IP des nœuds à un saut qui sont annoncés par les paquets Hello.

4.3 Une implémentation OLSR au niveau MAC

4.3.1 Le AWDS

AWDS (Ad Hoc Wireless Distribution Service) est une implémentation d'OLSR pour le routage dans les réseaux sans fil multi-sauts qui fonctionne au niveau 2 (niveau MAC). Il fournit un accès transparent similaire à celui de l'Ethernet à tous les nœuds qui constituent le réseau et permettant ainsi une utilisation des différents protocoles et applications de niveaux supérieurs comme le IP, IPV6, jeux …

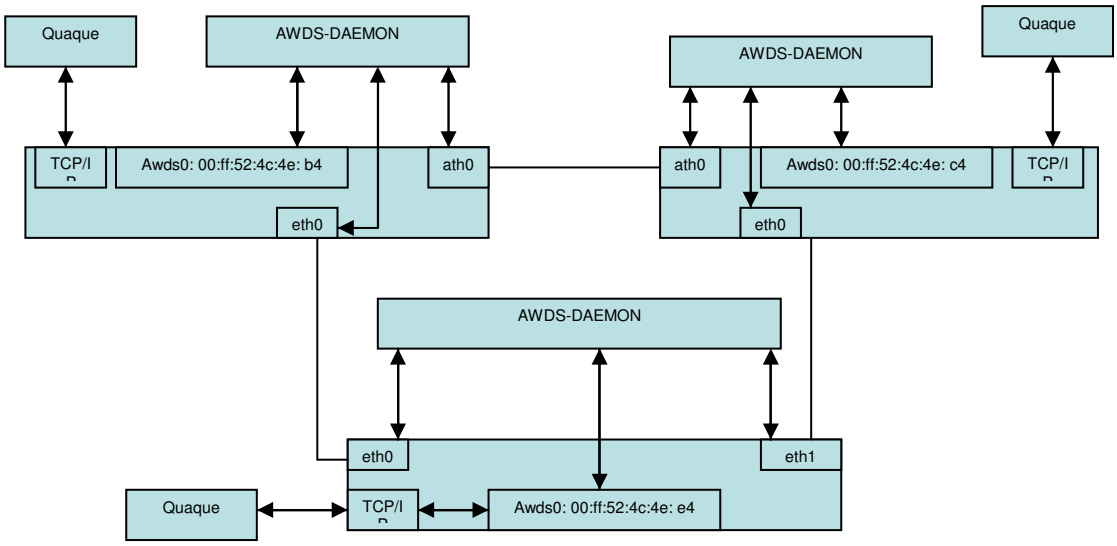
AWDS est un protocole proactif qui utilise des tables. Il a été implémenté en tant que Daemon et fonctionne complètement dans l'espace utilisateur. Nous avons utilisé la version 7.6 du protocole qui est la dernière implémentation stable (Août 2008). Les Daemon de routage sur toutes les entités du réseau communiquent entre eux afin de construire les tables de relayage. Pour atteindre ce objectif, le Daemon utilise deux types de messages : beacon et paquets de topologie. Sachant que le protocole fonctionne au niveau 2, les Daemon communiquent entre eux en utilisant des sockets Ethernet.

Tous les paquets générés parle Daemon sont identifiés en tant que paquets AWDS via le EtherType = 0x8334. le EtherType étant le champ de la trame Ethernet qui indique quel protocole des couches hautes est transporté par la trame Ethernet.

Une documentation sur ce projet est disponible sur http://awds.berlios.de/

4.2.2 Les deux types de tables : MAC et Forwarding

AWDS défini deux types de tables. La première est celles des adresses MAC (figure 27). Dans le protocole implémenté, cette table contient l'adresse MAC d'une interface virtuelle AWDS0 et le NodeID qui identifie le nœud dans le protocole de routage dans lequel l'interface AWDS0 a été crée.
Dans la suite on donne un exemple de topologie.



Les tables MAC et de Forwarding on la forme suivante :

MAC Address	NodeID
00:ff:52:4c:4e:c4	00:19:5b:79:3a:d1
00:ff:52:4c:4e:e4	00:30:18:4b:22:83

NodeID Destination	NodeID NextHop
00:19:5b:79:3a:d1	00:30:18:4b:22:83
00:30:18:4b:22:83	00:30:18:4b:22:83

Figure 28 Table MAC et Table de Forwarding du AWDS

Sur réception d'un paquet sur l'interface awds0, le AWDS Daemon regarde dans la table MAC et il cherche l'adresse destination. Si un NodeID est trouvé, il cherche le nNodeID du prochain saut (next hop) et il encapsule le paquet en question dans un paquet AWDS de type unicast, le rôle des nœuds intermédiaire se limite à analyser l'entête de ce paquet unicast et de le relayer vers le NodeID destinationen utilisant leurs table de Forwarding.

4.2.4 Les messages AWDS

AWDS défini différents types de message(le diagramme d'héritage est donné par la figure 29). Les paquets Beacon et TC (Topology Packet) sont suffisant pour construire la topologie, la table de Forwarding et le calcul des métriques de la qualité de service). Les paquets Beacon sont transmis en broadcast toutes les 1.4 secons; ils sont utilisé pour la découverte de voisinage et l'évaluation de la qualité de liens. Cet information est ensuit diffusée tousles 1 seconde vers tous les nœuds via les paquets TC.
Les message Beacon et Toplogy Packets sont construits et envoyés directement vers la couche liaison de données à travers les sockets Ethernet par le Daemon.

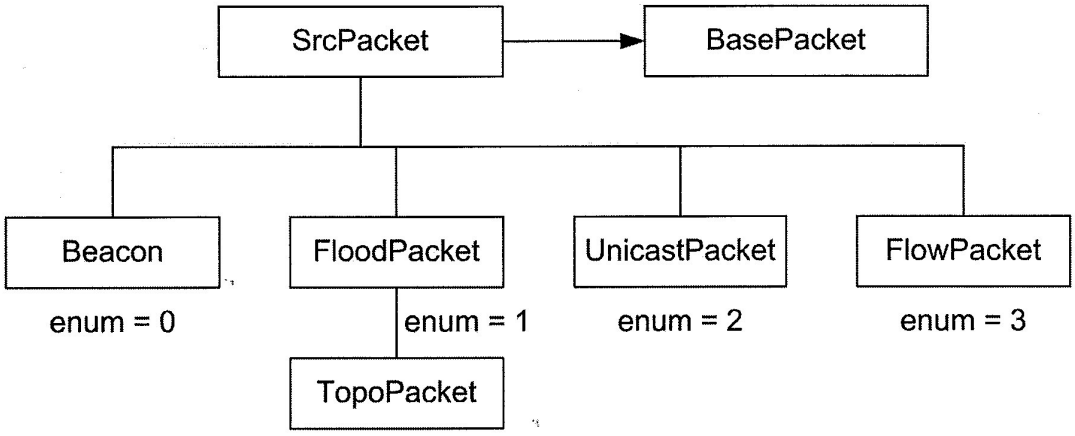


Figure 29 Diagramme d'héritage des messages AWDS

Les paquets générés par la couche application sont écrites sur l'interface awds0, qui à son tour, relaye ces paquets au Daemon AWDS.
Le Daemon encapsule les paquets Ethernet reçues en UnicastPackets ou FloodPackets.

4.3 Apport de la solution : résultats de tests.

4.3.1 Temps de changement de lien

Le but de ce test est de mesurer le temps de basculement lors du relayage d'"un chemin vers à un autre suite à une dégradation de performance d'un ou de plusieurs liens du premier chemin.

Ceci suppose qu'une métrique de performance mesurant la qualité de transmission sur un lien soit implémenté; en effet on a choisi d'utiliser une version d'OLSR niveau 2 qui implémente la métrique ETX *.

Le temps de basculement vers le nouveau chemin, lui seul, ne suffit pas pour mettre en évidence l'apport de OLSR de niveau 2, c'est pourquoi on décide de le comparer avec son homologue lorsque OLSR est implémenté à son niveau ordinaire c'est-à-dire au niveau routage 3.

Dans la figure 30 on propose une topologie simple constituée de trois nœuds et qui permet de réaliser cette expérience.

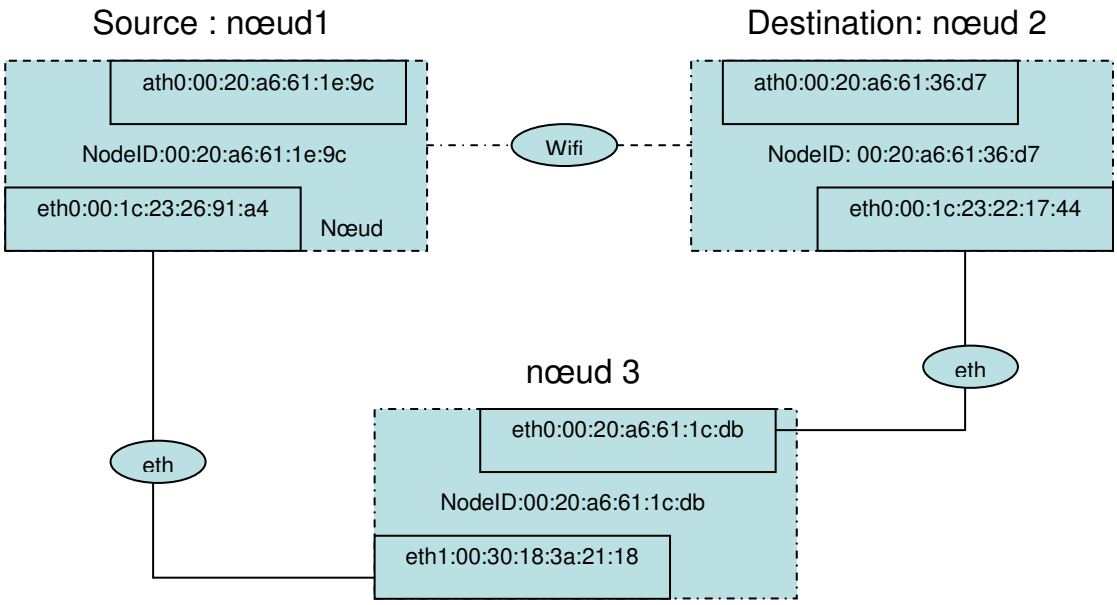


Figure 30 Scenario pour calcul du temps de changement de lien

L'expérience consiste à lancer une transmission vidéo du nœud 1 (source) vers le nœud 3 (destination). Ce nœud peut être atteint soit directement par la liaison wifi soit indirectement via le nœud 3 et dans ce cas les données emprunteront des liaisons filaires.

La figure 31 illustre les résultats obtenus avec un algorithme de routage OLSR implémenté au niveau 3

. Nous pouvons noter l'instant à laquelle le nœud 1 source "commute" ou bascule du lien sans fil vers le lien filaire. Initialement, le trafic vers la destination (NodeID 00:20:a6:61:36:d7) est directement délivré par la liaison sans fil, c'est pourquoi le prochain saut (the next hop) vers la destination est la destination elle-même.
Sur augmentation excessive du trafic le long de la liaison wifi, l'algorithme de routage commute vers la liaison filaire à deux sauts. Le prochain saut vers la destination est maintenant le nœud 3 (NodeID 00:20:a6:61:1c:db). Le temps mis pour ce changement (premier paquet envoyé sur l'interface eth0 – dernier paquet envoyé sur l'interface ath0) est de 0.014 secondes.

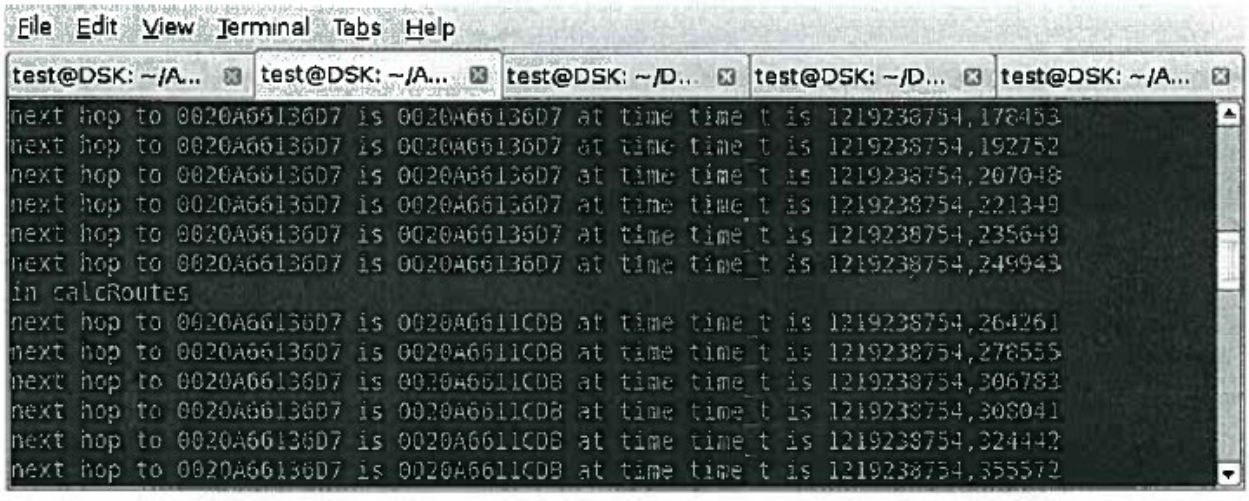


Figure 31 Temps de changement de lien (cas d'OLSR niv2)

Le même test est répété en utilisant notre implémentation de OLSR au niveau 2. Dans ce cas, comme on peut le voir sur la figure 32, la source bascule de la liaison filaire à deux sauts vers la liaison sans fil. Le temps de ce basculement est de 0.001 seconde ce qui est nettement plus petit que son homologue de la première expérience; c'est un temps négligeable et n'affecte pas la qualité de la vidéo transmise.

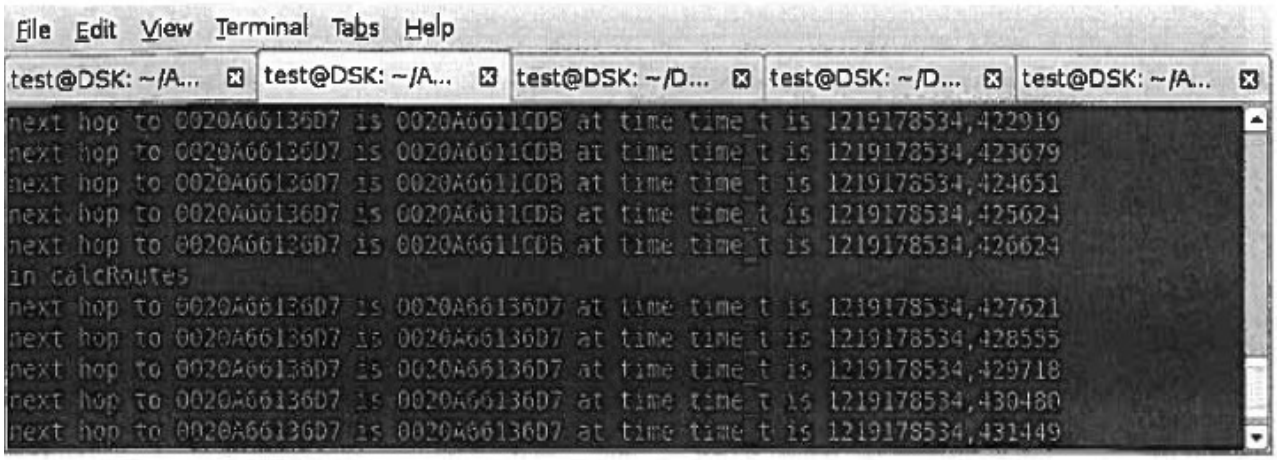


Figure 31 Temps de changement de lien (cas d'OLSR niv 3)

* ETX : Expected Transmission Count est défini comme le nombre de transmissions prévues pour acheminer un paquet sur un lien donné incluant les transmissions erronées . si on considère df comme étant le taux de livraison d'un lien et dr le taux inverse alors

$$ETX = 1/(df*dr).$$

Par exemple, si 3 parmi 10 paquets hello sont perdues sur un lien donné alors la probabilité d'une transmission réussie est de 70 % et si on suppose que dans le sens inverse la probabilité est de 0.6 alors

$$ETX = 1/(0.7*0.6) = 2.38.$$

** Iperf c'est un logiciel réseau qui permet de génerer des flux (TCP, UDP …) tout en choisissant le débit, le temps de transmission …

Chapitre 5

Solution de niveau MAC Communications Coopératives

5.1 Solution : Modification du pilote noyau Madwifi

5.1.1 Le Driver noyau linux Madwifi

Vu le contraintes de temps et de coût, il vient de soit qu'on adopte la première approche qui se contente de modifier le pilote de noyau Madwifi afin d'implémenter des fonctionnalités de coopération.

Par manque de documentation sur le Kernel Driver Madwifi, nous procédons dans ce qui suit à analyser son implémentation en langage C est d'en déduire l'architecture et la structure de code afin d'avancer dans l' implémentation dans la solution.

Sam LEFFLER est l'auteur original de MadWifi, il continue de maintenir et améliorer le pilote pour FreeBSD seulement. Néanmoins, il fournit les binaires pour la couche d'abstraction matérielle (HAL pour Hardware Abstraction Layer). En 2005 Sam LEFFLER a arrêté le développement de MadWifi, et une de poignée de développer a repris le projet pour crée le projet madwifi.org et continuer d'améliorer le plus avancé des pilotes 802.11 pour Linux. Le nombre de développeurs a augmenté et de plus en plus de personnes se sont attachées à fournir un support aux utilisateurs.

Logo du projet Madwifi

Le projet madwifi.org est d'abord une équipe de développeurs volontaire qui travaille sur les pilotes Linux pour les équipements de réseaux locaux sans fil utilisant un chipset Atheros. Il est composé de deux pilotes : MadWifi et ath5k.

MadWifi est un des pilotes les plus développés pour Linux. Il est stable et utilisé par une grande communauté d'utilisateur. Le pilote est libre, mais il dépend d'une HAL propriétaire qui ne lui permet pas d'avoir un contrôle complet sur le matériel.

Le projet ath5k a pour objectif de libérer le pilote MadWifi de ces contraintes propriétaire grâce aux pilotes openHAL et MadWifi. ath5k sera un pilote entièrement libre et gratuit. Ce pilote doit remplacer à terme la HAL propriétaire actuellement utilisée dans MadWifi.

MadWifi signifie Multiband Atheros Drivers for Wireless Fidelity. C'est-à-dire, un pilote Linux basé sur des équipements Atheros pour les réseaux sans fil. Le pilote apparaît dans le système comme une interface traditionnelle. La figure 34 schématise son architecture.

La HAL compose la couche la plus proche du matériel. C'est elle qui agit directement sur le matériel grâce à des fonctions de lecture et d'écriture. Aussi basiques que soient ces fonctions, la tolérance aux erreurs est nulle, car cela impliquerait une erreur dans le cœur du système et par conséquent un gel du noyau. Le module ath permet de faire le lien entre l'accès au matériel et les interruptions déclenchées par les couches supérieures.

Le module net802.11 est l'interface qui gère le transfert de l'information au périphérique réseau.

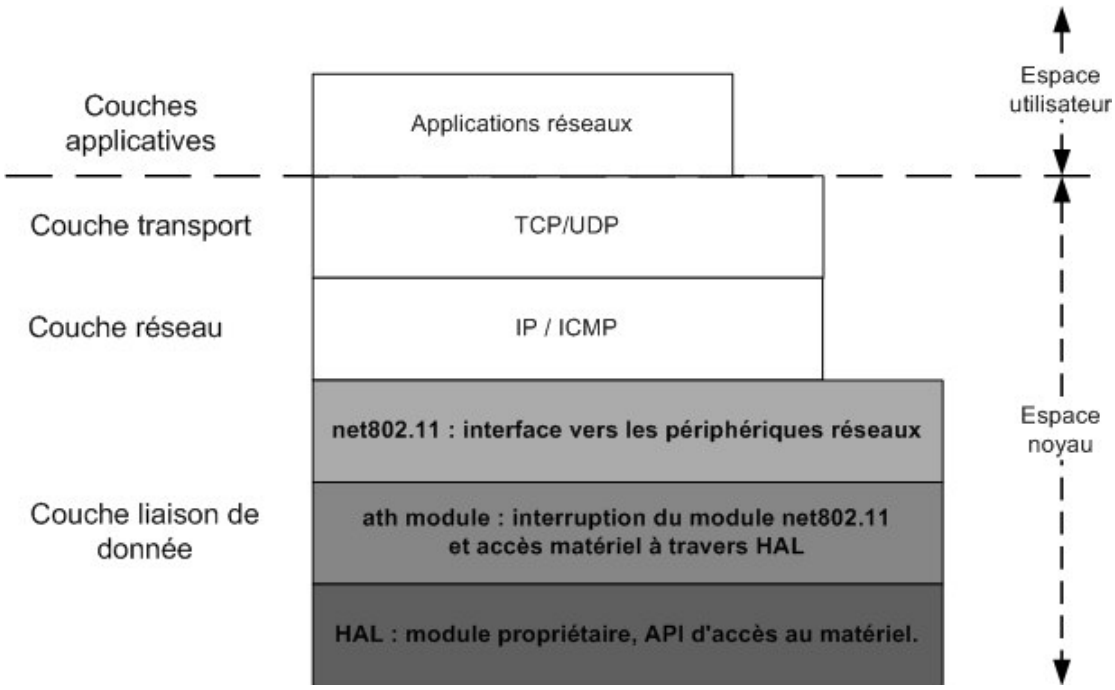


Figure 34 Architecture de Madwifi

Nous avons réalisé une analyse du code c implémentant le Madwifi afin d'en extraire son architecture, son organisation, les principaux modules et les structures de données. Une version en anglais de cette analyse est attachée en annexe à ce rapport.

5.1.2 Solution étudiée et horizons

L’objectif de cette section est de décrire le mode de fonctionnement de la solution de coopération proposée par l'article "CoopMAC: A Cooperative MAC for Wireless LANs" (Pei Liu*, Zhifeng Tao†, Sathya Narayanan‡, Thanasis Korakis* and Shivendra S. Panwar*)*.

L'idée fondamentale de la solution à relais est pouvoir utiliser le caractère diffusion de la transmission sans fil.
En effet, plusieurs nœuds sur un réseau de type wifi reçoivent un certain nombre de trames sans qu'ils lui sont destinés, dans le cas particulier ou un nœud reçoive une trame et que celle-ci soit destiné à un de ses voisin, il pourrait éventuellement intervenir dans sa retransmission si le lien direct entre source et destination est moins performante.

1. Lorsque une source à une nouvelle trame MAC (MPDU) à envoyer, elle peut soit la transmettre directement vers la destination soit utiliser un nœud support si le temps total de livraison de la trame est plus petit que celui mis au cours de la transmission directe.
2. à coté de son fonctionnement habituel, un message RTS est aussi utilisé par CoopMac pour signaler à une station qu'elle a été élue en tant que relai. CoopMac introduit un nouveau message appelé HTS (Helper-ready To send) qui sera utilisé par un relai pour indiquer sa disponibilité après la réception du message RTS de la source. Lorsque la destination reçoive un HTS elle émet un message CTS afin de réserver le canal radio pour une transmission à deux sauts. Si non elle réservera le canal pour une transmission directe.

3. Si les messages HTS et CTS sont reçues par la source, la trame doit être envoyée au relai et par la suite délivrée à la destination finale. Si la source ne reçoit pas le message HTS, elle transmet sa trame directement à la destination.
4. un acquittement ordinaire sera utilisé pour accuser réception la trame.

Une analyse de l'implémentation de MadWifi étant faite, il ne rest qu'à implémenter les nouvelles fonctinnalité (message HTS …) afin de réaliser la coopération.

Conclusion

Dans ce stage nous avons essayé d'améliorer la qualité de jeu dans le cadre de jeux vidéo multi-joueurs sur réseau mobile ad hoc en terme de QOS.

L'étude bibliographique avec une compagne de test ont permis de mette en évidence les métriques de la qualité de service les plus contraignantes; c'est notamment le temps de latence qui en constitue le principal.

Une approche cross-layer a été suivi afin de répondre à cette contrainte temporelle; indépendamment de l'application (jeu vidéo), l'objectif est d'optimiser les différentes couches réseaux (TCP, RoutageET COUCHE mac). Plusieurs solution ont alors été testé.

Sur le plan TCP, la variante TCP-WELCOME permet d'identifier les différents types de pertes sur un réseau sans fil et de répondre correctement à dans chaque cas.

Au niveau du routage, une approche originale a été testée : c'est l'implementation du protocole de routage proactif OLSR au niveau MAC ce qui permet de garantir une certaine flexibilité au niveau du forwarding.

Quant à la couche MAC, l'amélioration consiste à ajouter des fonctionnalités de coopération à l'implémentation du IEEE 802.11, un travail qui n'a pas été entamé par manque de temps mais qui est extrêmement facilité par la préparation d'une documentation sur le pilote noyau MadWifi qui implémente le 802.11 en open source.

Ce travail peut être continué par l'implémentation de la solution MAC et l'étude de la possibilité de marier deux voire plus des solutions précédents.

Annexes

1) TCP-WELCOME et Plateforme SEDLANE

Configurer une station en mode ad hoc

Dans le cas de l'utilisation de la carte wifi intégrée :

- passer en route
- iwconfig eth1 essid (nom de la cellule) mode ad hoc freq 2.4….G rate 11M
- ifconfig eth1 (adresse ip)

Dans le cas de l'utilisation des cartes Atheros et pilote Madwifi :

- wlanconfig ath0 destroy
- wlanconfig ath0 create wlandev wifi0 wlanmode ad-hoc
- iwconfig ath0 essid (nom de la cellule) freq 2.4…G rate 11M.
-

Installer le jeu Quake 3 sur une plateforme Linux

2) TCP-WELCOME et Plateforme SEDLANE

Enabling IP Fire-Wall of FreeBSD

Dummynet uses the ip fire-wall of the system to execute its commands (as explained earlier). Then, ip fire-wall of FreeBSD must be enabled before starting using dummynet. The following steps show how to enable the ip fire-wall of the system (FreeBSD, in our case).

- Add these lines at the end of the configuration file:
  - **options IPFIREWALL**
  - **options IPFIREWALL_VERBOSE**
  - **options IPFIREWALL_VERBOSE_LIMIT=5**
  - **options IPFIREWALL_DEFAULT_TO_ACCEPT**

  The last option defines default policy for ip fire-wall. This policy accepts all the packets that enters or leaves that host. The default policy is applied when there are no other rules defined into the fire-wall.

Then, the file /etc/rc.conf must be modified in order to activate the fire-wall and to name the file that contains the script of ip fire-wall rules to be applied.

- **firewall_enable="YES"**
- **firewall_script="/etc/ipfw.rules"**
- **firewall_logging="YES"**

- Now the ip fire-wall of the system is enabled and activated.
- To modify the Bandwidth, packet delays and packet losses, the system kernel must be recompiled with the following options:

  - **options DUMMYNET**
  - **option HZ=1000**

- Recompile the FreeBSD Kernel (see Annexe B).

Recompiling FreeBSD Kernel

To start the compilation process, you must log in by root account.

- Copy the file named GENERIC to have an example in the case that something is going wrong.
  - **cp GENERIC TEST**
- Create a configuration file that named TEST
  - **config GENERIC**

  This command creates new sub-directory ../compile/TEST and displays the following:

  ```
  Kernel build directory is  ../compile/GENERIC
  Don't forget to do a "make cleandepend; make depend"
  ```

  The sub-directory GENERIC now contains many files that ends with .h (header) and makefile and config.c file.
- According to the message on the screen, run make depend in the GENERIC folder.
  - **make cleandepend**

  Then
  - **make depend**

  This command will create depend file, files that ends with .c which corresponds to .h files and also creates a modules sub-directory.
- Make
  - **make**

  This step will take a lot of time.
- Make install
  - **make install**

  This command renames /boot/kernel into /boot/kernel.old, then it creates a new /boot/kernel with the new compilation options.
- To be sure that the new kernel is the one that running on your machine;
  - **uname –a**

We can verify, from the output of this command, the date of compilation that must correspond to the current day.

Verification of IP Fire-Wall

- To verify that the ip fire-wall is turned on, enter the following commands:
    - **`ipfw list`**

  This command displays the list of active rules in the fire-wall. The output must be like this:

  `65535 allow ip from any to any`

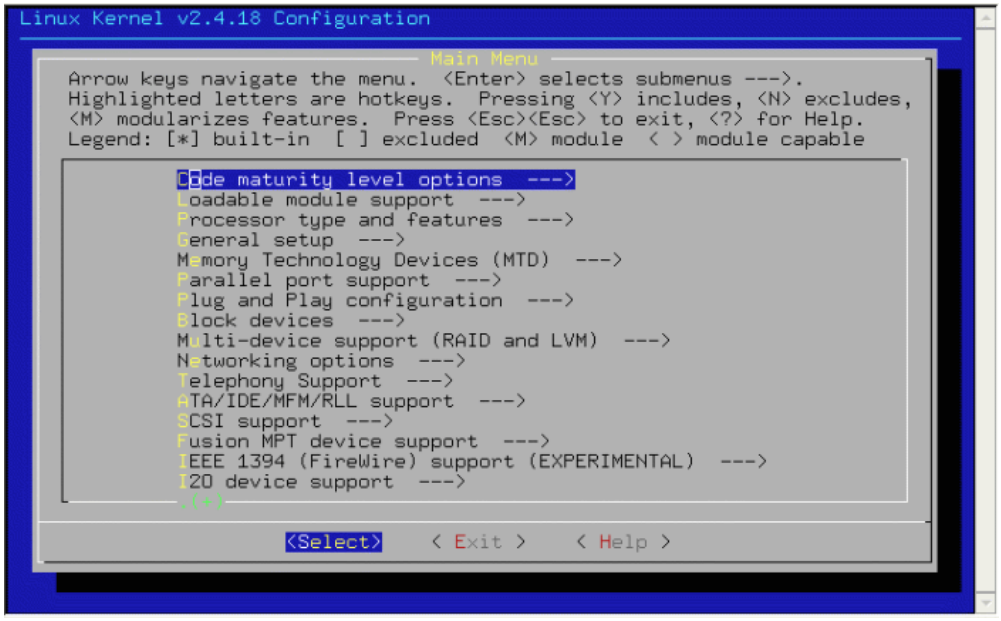
  This output corresponds to the default policy configured earlier. Try to add a new rule in the list:
    - **`ipfw add deny ip from any to any`**

  Note that this rule will block your machine from accepting any ip traffic that enters or leaves your system.
- To delete all the rules defined within ip fire-wall, use:
    - **`ipfw  flush`**

  This command will delete all the defined rules except the default policy.
- Now, we can use dummynet and configure the communication channels (refer to section 1. Dummynet).

Activer les différentes variantes de TCP sous Linux kernel 2.6 et après

Tu peux recompiler le kernel Linux avec les variantes de TCP en utilisant la commande :

- **make menuconfig**
- **puis choisir l'option Networking options**
- 

Aussi, tu peux vérifier les variantes disponibles dans le kernel en utilisant la commande :

- **sysctl net.ipv4.tcp_available_congestion_control**

Installer SEDLANE

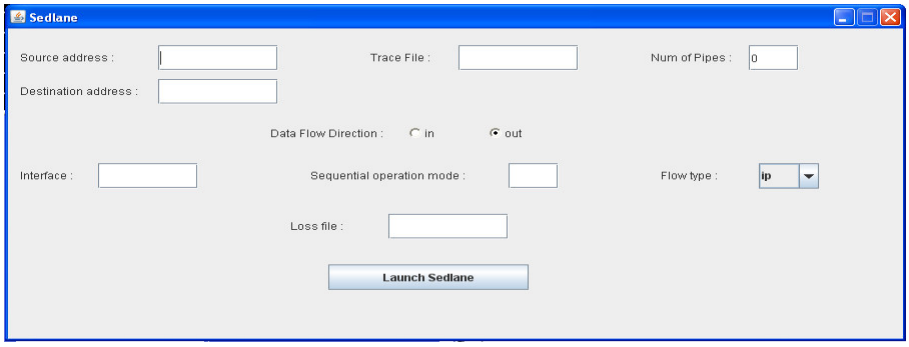
Comme dans le fichier README.

changer l'algorithme de contrôle de congestion sous Linux

Pour changer l'algorithme de contrôle de congestion/la variante de TCP utilise la commande suivante :

- **sysctl -w net.ipv4.tcp_congestion_control=reno**
- **sysctl -w net.ipv4.tcp_congestion_control=lda3 (c'est le nom de TCP-WELCOME dans le code)**

Lancer SEDLANE



-If faut mettre le nom de fichier de trace que t'ai envoyé dans le champ « «Trace File ».

-Source / Destination adresse : les adresse comme dans la configuration du réseau.

-Num of pipes : tu peux mettre 10 par exemple.

-Interface : tu précise l'adresse de l'interface sur la machine SEDLANE.

-Pour le « Data Flow Direction » met « out »

-Met rien dans le champ « Sequential operation mode ».

- Dans le champ « Flow type » : met « TCP ».

-Ne met rien dand le champ « Loss file ».

Et tu appuie sur « Launch SEDLANE » après avoir entrer tous les champs précisé avant.

Cette étape doit être terminée avant de commencer le transfert du fichier.

Lancer TTCP

Après avoir configuré et lancer SEDLANE, tu peux commencer le transfert du fichier sur la connexion.

Ce lien peut être utile : http://linux.die.net/man/1/ttcp

Tu peux utiliser ces deux commandes :

Exemple sur la destination: ttcp –r –s –n 5120 –l 1024

Exemple sur la source : ttcp –t –s –n 5120 –l 1024 source_adresse

3) Analyse de l'architecture et des structures du code C du MadWifi

Introduction to Madwifi

a) **Transmission procedure**
   It starts by calling ***ath_hardstart***. There are descriptions for this function and for any other difficult and main function, which is called by ***ath_hardstart*** (immediately or not).
b) **Reception Procedure**
   It starts by calling ***ath_rx_tasklet***. There are descriptions for this function and for any other difficult and main function, which is called by ***ath_rx_tasklet*** immediately or not.
c) **The query of the packets**
d) **Rate adaptation algorithms**

<u>The common structures which you should know, before start learning...</u>

- *Structure net_device*
  This structure describes the underlying device. In this case the device is the wireless card. (Whenever there is a pointer that is called ***dev***, it should point to an instance of this structure...)

- *Structure ath_softc*
  This structure describes the software adjustments of the device. Every ***dev*** has its own instance of this structure. (Whenever there is a pointer that is called ***sc***, it should point to an instance of this structure...)

- *Structure ieee80211vap*
  Virtual AP (Access Point), a technique to allow one physical access point to behave like it is multiple access points. In some contexts, it also refers to having multiple modes of operation (e.g. station, monitor, wds) running at the same time. (Whenever there is a pointer that is called ***vap***, it should point to an instance of this structure...)

- *Structure ieee80211com*
  Data that is common to one or more virtual AP's. State shared by the underlying device and the net80211 layer is exposed here. Every ***vap*** points to the corresponding common data via the pointer ***vap->iv_ic***. (Whenever there is a pointer that is called ***ic***, it should point to an instance of this structure...)

- *Structure ieee80211_node*
  This structure keeps node specific information. Note that drivers are expected to derive from this structure to add device-specific per-node state. Each node represents another station in the network. (Whenever there is a pointer that is called ***ni***, it should point to an instance of this structure...)

- *Structure sk_buff*

This structure keeps all the necessary information about a package. It has a variety of information and a pointer ***data*** that keeps the whole packet (header, body, tail). (Whenever there is a pointer that is called ***skb***, it should point to an instance of this structure...)

- *Structure ath_buf*
  A structure similar to the previous one, but it is more specific. The previous structure keeps packets for every device. This structure keeps these packets that are related to the 80211 Atheros wireless device. Every instance of this structure is associated with an instance of the *sk_buff* structure, via the pointer ***bf_skb***. (Whenever there is a pointer that is called ***bf***, it should point to an instance of this structure...)

- *Structure ieee80211_cb*
  This structure keeps a control buffer for every ***skb***. Every ***skb*** points to its control buffer via the pointer ***skb->cb***. (Whenever there is a pointer that is called ***cb***, it should point to an instance of this structure...)

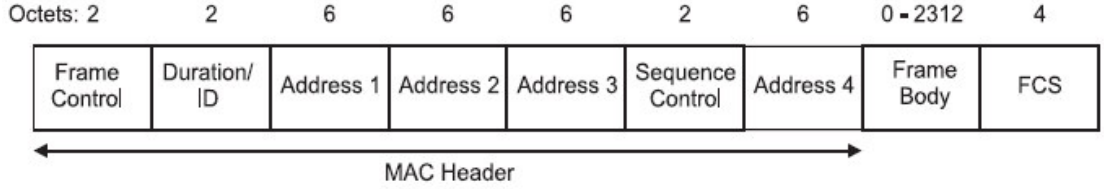
- *Structure ether_header*
  A structure, that keeps information about the Ethernet header. (Whenever there is a pointer that is called ***eh***, it should point to an instance of this structure...)

- *Structure ieee80211_frame*
  A structure, that keeps information about the 80211 frame header. (Whenever there is a pointer that is called ***wh***, it should point to an instance of this structure...)

- *Structure ath_desc*
  This structure describes an instance of the structure *ath_buf*. So every ***bf*** is associated with an instance of this structure, which is its description, via the pointer ***bf->bf_desc***. (Whenever there is a pointer that is called ***dc***, it should point to an instance of this structure...)

Other useful information...

- The MAC frame format... *(Table 1)*



- Address field contents (for Data packets)... *(Table 2)*

To DS	From DS	Address 1	Address 2	Address 3	Address 4
0	0	DA	SA	BSSID	N/A
0	1	DA	BSSID	SA	N/A
1	0	BSSID	SA	DA	N/A
1	1	RA	TA	DA	SA

DS means Distribution System and actually is an AP (Access Point).

sk_buff a basic kernel defined structure we are using for packet

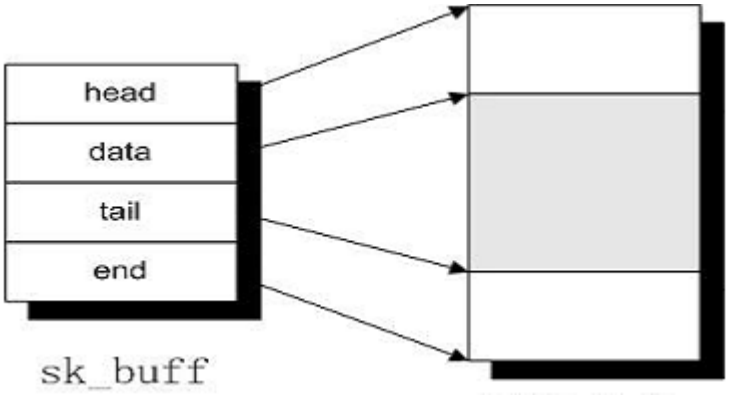
Structure sk_buff keeps all the necessary information about a package, from this structure we have access to the information of a packet, including pointers about the packet and variables that describe the packet. (Whenever there is a pointer that is called ***skb***, it should point to an instance of this structure). This document is meant to briefly discuss the members in sk_buff and some functions operating on skb.

(1) Let's view what's in structure sk_buff ?

```
struct sk_buff {
/* These two members must be first. */
struct sk_buff *next;              // Next buffer in list
struct sk_buff *prev;              // Previous buffer in list
struct sk_buff_head *list;         // List we are on

// this 3 pointers put sk_buff into a bi-direction cycle link.

struct sock *sk;           // Socket the pkt is owned by
struct timeval stamp;      // Time the pkt arrived
struct net_device *dev;    // the network device which received the pkt
struct net_device *input_dev;
struct net_device *real_dev;
union {
struct tcphdr *th;
struct udphdr *uh;
struct icmphdr *icmph;
struct igmphdr *igmph;
struct iphdr *ipiph;
struct ipv6hdr *ipv6h;
unsigned char *raw;
} h;
union {
struct iphdr *iph;
struct ipv6hdr *ipv6h;
struct arphdr *arph;
unsigned char *raw;
} nh;
union {
struct ethhdr *ethernet;
unsigned char *raw;
} mac;

// the above 3 unions are the header structures of transport layer, network layer, and
link layer

struct dst_entry *dst;
struct sec_path *sp;


char cb[40];
```

```
//This is the control buffer. It is free to use for every
// layer. Please put your private variables there. If you
// want to keep them across layers you have to do a skb_clone() first
unsigned int len,  //Length of actual data ( including header and data)

data_len,
mac_len,
csum;   // Checksum
unsigned char local_df,
cloned,   // may have cloned sk_buff but with only one original
pkt_type,  //Packet class
ip_summed;  // Driver fed us an IP checksum
__u32 priority; //Packet queueing priority
unsigned short protocol, // link layer protocol
security;
void (*destructor)(struct sk_buff *skb);
#ifdef CONFIG_NETFILTER
unsigned long nfmark;
__u32 nfcache;
struct nf_ct_info *nfct;
#ifdef CONFIG_NETFILTER_DEBUG
unsigned int nf_debug;
#endif
#ifdef CONFIG_BRIDGE_NETFILTER
struct nf_bridge_info *nf_bridge;
#endif
#endif /* CONFIG_NETFILTER */
#if defined(CONFIG_HIPPI)
union {
__u32 ifield;
} private;
#endif
#ifdef CONFIG_NET_SCHED
__u32 tc_index; /* traffic control index */
#ifdef CONFIG_NET_CLS_ACT
__u32 tc_verd; /* traffic control verdict */
__u32 tc_classid; /*traffic control classid */
#endif
#endif
unsigned int truesize; // storage region length, large than the length of pkt
atomic_t users;
unsigned char *head,*data,*tail,*end; // important pointers to storage region of pkt
};
```

The relationship between sk_buff and pkt and be illustrated as follows:



(The shaded field is packet.)
Note that it's not necessary for a pkt to fill up all the space in storage region. So **head** point to the beginning of storage region, **end** points to the end of storage region, **data** points to where the pkt begins, and **tail** points to where pkt ends. The order of storage is as: header of link layer, header of network layer, header of transport layer, actual data. In a particular protocol layer, the **data** points to the header of this layer. Also note that the memory allocated for sk_buff and the packet are not continuous.

(2) functions about sk_buff that we might need to know

struct sk_buff*alloc_skb(unsigned int size, int gfp_mask)
allocate the memory of **size** size for pkt and its sk_buff. **Size** should be 16 bytes alignment. gfp_mask is the priority for memory allocation. After successful allocation, data and tail in sk_buff both point to the beginning of storage region. And len is 0, isclone and cloned variable are 0.

struct sk_buff*skb_clone(struct sk_buff*skb, int gfp_mask)
clone a new sk_buff from the original sk_buff, both of them point to the pkt. After sucessful clone, is_clone and cloned are set to 1. And the reference counter number will be increased.

struct sk_buff*skb_copy(struct sk_buff*skb, int gfp_mask)
Copy skb and the content which it points to. The new skb is independent of the older skb and the storge region. So is_clone, clone is set to 0, and new storage region's reference number is 1.

void kfree_skb(struct sk_buff*skb)
release skb and the memory which it points to.

unsigned char *skb_put(struct sk_buff *skb, unsigned int len)
Move **tail** down by size len, and increase **len** in skb. This operation enlarges memory for pkt, but **tail** should be larger than **end.**

unsigned char *skb_push(struct sk_buff *skb, unsigned int len)
Move **data** up by size len, and increase **len** in skb. This operation enlarges memory for pkt, but **data** should be less than **head.**

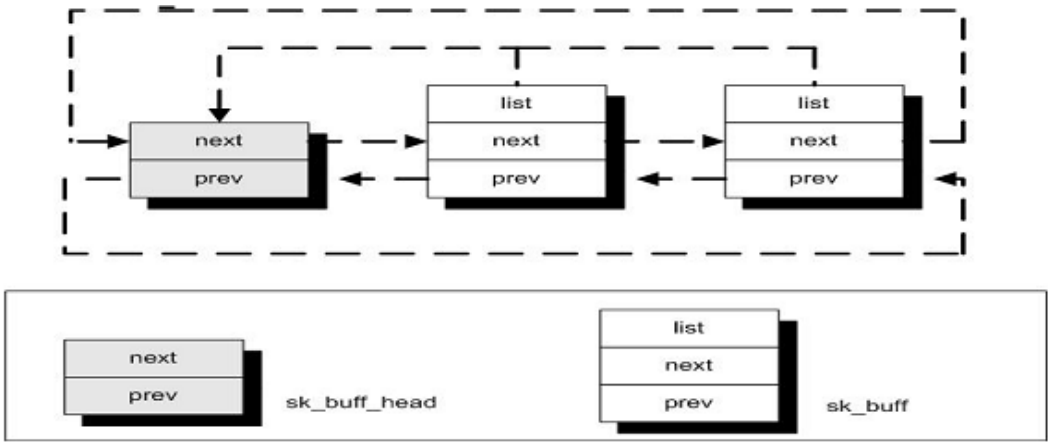
unsigned char *skb_pull(struct sk_buff *skb, unsigned int len)
Move **data** down by size len, and decrease **len** in skb. By this operation, **data** can point to the header of next layer.

void skb_reserve(struct sk_buff *skb, unsigned int len)
data and **tail** move down simultaneously by size len.

void skb_trim(struct sk_buff *skb, unsigned int len)
Truncate the length of pkt to **len**, discard the portion of data which is outside of **len.**

int skb_cloned(struct sk_buff *skb)
To decided whether it is a cloned skb or not.

(3) sk_buff_head concept
Sk_buff_head is the head of a bi-diresction link , for linking sk_buff. Please refer to the figure below.



Transmission Procedure
ath_hardstart
File: ath/if_ath.c

static int **ath_hardstart**(struct sk_buf ***skb**, struct net_device ***dev**)
skb is a pointer to a structure that includes a buffer, where the data of a packet is located. Every packet is associated with an instance of the *struct sk_buff.*
dev points to all necessary information about a device. An example of device is a wireless card.

Returns properly zero. In case there is any problem, then returns a negative number.

I don't consider about the case of fast frames. So I assume that the constant ***ATH_SUPERG_FF*** = 0.

1. The first check is about the device. The function checks if it is not operating (or running) or if it is detached, so it doesn't process interrupts. In this case the function prints an appropriate message and then returns *–ENETDOWN*, which means that the interface has since been taken down.

2. ***STAILQ_INIT(&bf_head)****;*
   The function initializes a list of instances of the structure *ath_buf* which is called ***bf_head***.

3. ***sc = dev->priv*** and ***cb = skb->cb***
   So ***sc*** describes ***dev*** and ***cb*** controls ***skb***. By using the ***cb***, the function checks if the packet, that is included in the ***skb*,** is raw. If it is raw, then it calls the macro...
   ATH_HARDSTART_GET_TX_BUF_WITH_LOCK. This macro fills the ***bf*** with the first instance of the list ***sc->sc_txbuf***. The list ***sc->sc_txbuf*** includes instances of the *ath_*buf, which are going to be used for transmitted packets. Also, the ***bf*** is inserted in the list ***bf_head.***

   Finally it calls the function ***ath_tx_startraw*** which associates the ***bf*** with the ***skb***, and then it sends it via the device ***dev.*** The association is succeeded in this way: ***bf->bf_skb = skb***. The function returns whatever ***ath_tx_startraw*** returns.

4. ***eh*** *= (struct ether_header *)* ***skb->data****;*
   eh points to the start of the data that is included in the ***skb***. At the start of the packet's data there is the Ethernet header, which is pushed to the ***skb*** by kernel. The kernel doesn't recognize any other header except for Ethernet's. It is up to the driver to replace this header, with the appropriate one...

5. ***ni*** *=* ***cb -> ni****;*
   In this way, ***ni*** describes the receiver of the packet ***skb***.
   If this pointer is null, then the receiver is unknown and the function fails (*goto hardstart_fail)*

6. ***ATH_HARDSTART_GET_TX_BUF_WITH_LOCK***
   It fills the ***bf*** as I described before...

7. ***skb** = **ieee80211_encap(ni, skb, &framecnt)**;*
   This function does the necessary encapsulation of the packet. It takes the packet's ***skb*** and the receiver node ***ni,*** and returns the new encapsulated ***skb***. If the ***skb***, which the ***ieee80211_encap*** received, had too much data, then the ***ieee80211_encap*** has to fragment the ***skb*** and produce new instances of the *sk_buff*, in a way that each of them will have a part of the whole data. Each of these new instances is called also frame. So the function returns also the number of the frames that it has produced, via the variable ***framecnt***.
   In case of fragmentation, the ***skb*** that the function returns is the first of the list, which contains all the instances that have the whole data. Each instance shows the follow via the pointer ***next (skb_follow = skb->next).***

8. Then checks if the returned ***skb*** is null. In this case there is a problem and the function fails.

9. Then we have 2 cases... The packet has been fragmented (if ***framecnt*** == 1) or not (if ***framecnt*** > 1).

10. If there was fragmentation, then we get new instances from the ***sc->sc_txbuf***, and we insert them in the list ***bf_head***. ***bf*** is already there, because of the macro *ATH_HARDSTART_GET_TX_BUF_WITH_LOCK.*
    The function gets (***framecnt***-1) instances of the *ath_buf*, because it has to associate the same number of new frames (or instances of the *sk_buff)*.

11. Also it calls the ***ieee80211_ref_node(ni)***. It should be called every time that a new reference to the node ***ni*** arises. So in this case, because of the existence of (***framecnt***-1) new frames, which must be sent to the node ***ni,*** the function increases the same the reference-counter of ***ni***.

12. Then the function checks if there were enough instances of the *ath_buf* in the list ***sc->sc_txbuf***. If not, then the function fails.

13. Then the function puts the first instance of the list ***bf_head*** to the variable ***bf*** and then removes it from the list. Also, it keeps in the variable ***nextfraglen*** the length of the frame which follows the ***skb***. After, the function calls the...
    ath_tx_start(dev, ni, bf, skb, nextfraglen).
    The ***ath_tx_start*** associates the ***bf*** with the ***skb*** and the receiver node ***ni*** and it sends the frame to the physical layer, ready for transmission.
    No success of the ***ath_tx_start*** means that ***ath_hardstart*** will fail.

14. The next step is to replace the ***skb*** with the ***skb->next***. So the function is ready to repeat the previous step, until no frame remains in the list.

15. But if we didn't have any fragmentation, then the function calls only one time the function ***ath_tx_start(dev, ni, bf, skb, 0)***. It is natural because there is only one ***skb*** and ***bf.***

16. The function returns properly 0.

17. In case of failure, the function empties the list ***bf_head*** and puts again its instances in the previous list ***sc->sc_txbuf.*** Also, it decreases the references on the node ***ni*** and releases the frames which have been produced due to the fragmentation.