N° d'ordre : 2385

THESE

DÉFINITION ET MISE EN ŒUVRE D'UNE SOLUTION D'ÉMULATION DE RÉSEAUX SANS FIL

Présentée pour obtenir

Le Titre De Docteur De l'Institut National Polytechnique De Toulouse

École doctorale : Informatique et Télécommunications
Spécialité : Réseaux, Télécommunications, Système et Architecture

Par :

Emmanuel CONCHON

Soutenue le 27 Octobre 2006 devant le jury composé de :

M.	*Christian FRABOUL*	Président
MM.	*Michel DIAZ*	Directeur de thèse
	Tanguy PERENNOU	Co-Encadrant de thèse
	Hossam AFIFI	Rapporteur
	Eric FLEURY	Rapporteur
	Christian BONNET	Membre
	Eric FAURE	Membre
	Patrick SENAC	Membre

Remerciements

Je tiens à remercier :

M. Michel Diaz, mon directeur de thèse, pour son soutien et sa vision pragmatique des problèmes. En de nombreuses occasions, son recul et sa vision d'ensemble du monde des réseaux se sont avérés essentiels pour trouver de nouvelles solutions et dégager des perspectives de recherche.

M. Tanguy Pérennou pour son aide quotidienne, ses conseils et sa très grande disponibilité dont j'ai largement abusée au cours de ces quelques années. De manière générale, je tiens à le remercier pour la grande gentillesse dont il a fait preuve et pour la rigueur qu'il m'a obligé à avoir pour la rédaction de ce manuscrit et dans mon travail.

Merci également à l'équipe de l'Université de Karlstad et M. Johan Garcia en particulier pour toutes les discussions et réunions que nous avons menées dans le cadre du réseau d'excellence NEWCOM en vue d'intégrer nos deux outils. Je tiens également à saluer tous les membres du Département 6 de NEWCOM qui, à travers les discussions que nous avons pu avoir, m'ont permi d'ouvrir de nouvelles pistes de recherche.

Merci aux professeurs Eric Fleury et Hossam Afifi pour avoir accepté d'être rapporteurs de ma thèse, ainsi qu'au professeur Christian Fraboul pour avoir présidé le jury. Tous trois ont accepté malgré un emploi du temps extrêmement chargé et le délai très court, ce dont je leur suis infiniment reconnaissant. Je suis de plus très honoré d'avoir pu les compter parmi les membres du jury au même titre que les professeurs Christian Bonnet et Patrick Senac et M. Eric Faure.

Un grand merci à l'ensemble des membres du Département Mathématiques et Informatique de l'E.N.S.I.C.A. et du groupe O.L.C. du L.A.A.S. pour leur accueil chaleureux, et en particulier à Laurent Dairaine pour m'avoir permis, en m'encadrant lors de mon stage de D.E.A., de me familiariser avec le monde de la recherche, Yves Caumel, René Espourteau, Fabrice Frances, Bernard Jarlan, Jérome Lacan, Pierre de Saqui Sannes. Je salue également tous les doctorants actuels et anciens que j'ai pu côtoyer au cours de ces quelques années : Laurent, Ernesto, Romain, Tarek, Ahlem, Ali, Amine, Juan, Benjamin, les derniers arrivés Lei et Nouredine et bien sur Mathieu et Florestan qui ont partagé mon bureau et contribué grandement à l'excellent souvenir que je garderai du DMI. Je n'oublie pas non plus Hervé et Jérome mes deux camarades de promotion avec qui j'ai partagé non seulement le bureau mais également de fantastiques vacances à l'autre bout du monde avant de démarrer la rédaction de ce manuscrit.

Merci également à tous ceux qui m'ont accompagné et soutenu dans cette grande expérience de la vie que constitue une thèse. Je salue donc tous mes amis de Toulouse et d'ailleurs : Anne, Jérome, Laury, Patrick, Stéphanie et Sophie la néo-rennaise bien qu'elle ait réussi à soutenir avant moi :-P. Je salue également tous les gens de Limoges (Audrey, Baptiste, Laurent, Pierre-Yves, Philippe, Remi...) avec qui j'ai passé de super week-ends loin de la pression toulousaine ;-)

Et pour finir merci à mes parents et mes grand-parents pour leur soutien sans faille, leurs encouragements et sans qui rien n'aurait été possible.

Table des matières

Table des figures

Table des figures

Liste des tableaux

1 Introduction

Les réseaux de données sans fil ont connu une véritable explosion depuis la fin des années 90 aussi bien dans la vie de tous les jours pour se connecter à l'Internet que dans le monde de la recherche. L'environnement sans fil présente de nombreuses différences avec le monde des réseaux filaires en particulier au niveau des couches basses de communications que sont les couches physiques et liaisons de données. Les communications s'effectuant à l'aide d'un signal radio, les risques de voir ce signal brouillé ou atténué sont très importants. Un autre point important est que la topologie de ce type de réseau peut évoluer de manière très dynamique car les terminaux composant un réseau sans fil peuvent être mobiles. La qualité de service dans les réseaux sans fil est aujourd'hui faible et nécessite d'être améliorée. Les réseaux filaires ont également été confrontés à ce problème avec l'explosion de l'Internet et de nombreuses solutions ont été proposées au niveau transport pour améliorer les communications. Malheureusement, du fait des différences des niveaux physiques la plupart des solutions retenues dans le monde filaire ne sont pas directement transposables aux réseaux sans fil. De nouveaux protocoles de transport utilisant les particularités sans fil sont régulièrement proposés pour répondre à ce besoin de qualité de service.

Grâce à l'explosion d'Internet, de nombreuses applications réparties ont vu le jour au cours des dernières années. Ces applications proposent notamment de la vidéo à la demande, de la voix sur IP permettant de téléphoner à moindre coût d'un bout à l'autre du globe ou encore des applications pair à pair pour l'échange de fichiers. La plupart de ces applications ont profité de l'augmentation constante des débits dans les réseaux filaires pour se développer. Dans les réseaux sans fil, le débit disponible reste inférieur et ne peut pas augmenter indéfiniment mais par contre la demande en applications réparties ne devrait, elle, pas diminuer. De nouveaux mécanismes permettant de s'adapter à ces réseaux, en utilisant par exemple des techniques de codage pour faciliter les transmissions, sont donc proposés et des applications exploitant ces mécanismes devraient bientôt apparaître.

Que ce soit pour un protocole ou une application, le développeur est confronté aux mêmes problèmes de test et d'évaluation que dans les réseaux filaires. En effet, lors du développement d'un protocole ou d'une application répartie, il est nécessaire d'évaluer le comportement des différents mécanismes proposés et de pouvoir le comparer avec celui des mécanismes existants. C'est également lors de cette phase de test que le paramétrage des mécanismes est mis au point (par exemple déterminer le délai acceptable entre deux images pour de la vidéo à la demande). Cette phase est donc critique et conditionne le succès ou l'échec d'un prototype.

Pour pouvoir tester et évaluer un nouveau protocole ou une nouvelle application, il existe à ce jour trois grandes solutions :

- La première solution consiste à utiliser un réseau sans fil réel pour mener son évaluation. Bien que permettant d'avoir des conditions parfaitement réalistes, cette solution n'est pas complètement satisfaisante car elle ne permet pas de contrôler l'ensemble des paramètres de l'environnement (par exemple la météo pour les réseaux sans fil extérieurs). De plus, cette solution ne permet pas de reproduire plusieurs fois de suite les mêmes conditions avec précision. En effet, les conditions de propagation du signal à l'intérieur du réseau

ne sont pas contrôlables, si bien que d'une expérience à l'autre les conditions observées peuvent évoluer et parfois modifier considérablement le résultat des tests.

- La seconde solution, souvent privilégiée par les chercheurs, est la simulation. Celle-ci permet d'évaluer un modèle d'application ou de protocole dans un environnement entièrement contrôlable. Pour cela, la simulation s'appuie sur des modèles décrivant l'environnement, des modèles décrivant les couches de communication utilisées par les terminaux sans fil et les autres équipements du réseau et un modèle du trafic circulant sur le réseau. Cependant, la simulation ne travaille pas en temps réel ce qui interdit par exemple l'évaluation d'applications interactives. Utiliser un modèle au lieu de l'implémentation réelle est également pénalisant : la validité du modèle ne garantit pas que son implémentation réelle et le déploiement de cette implémentation s'effectueront sans problème. Des erreurs de programmation peuvent toujours survenir au moment de l'implémentation si bien que cette implémentation doit ensuite être évaluée en environnement réel pour vérifier qu'elle se comporte bien comme son modèle.
- L'émulation se veut comme un compromis entre les deux solutions précédentes en permettant d'évaluer un protocole ou une application dans un environnement contrôlable et reproductible qui simule en temps réel les conditions telles que les débits, les délais et les pertes que l'on observe dans le réseau cible. Pour cela, l'émulation va simuler les effets des couches basses de telle manière qu'un protocole ou une application s'exécute dans les mêmes conditions que celles existant dans un environnement réel. Cette solution de test peut par ailleurs être vue comme une étape supplémentaire dans le cycle de développement entre la phase de simulation, qui permet de concevoir les mécanismes spécifiques d'un protocole ou d'une application, et le déploiement dans un environnement réel.

Nous avons choisi de nous intéresser à cette troisième solution mais celle-ci impose cependant certaines contraintes. Comme l'émulation travaille en temps réel, les modèles utilisés pour simuler les couches basses ne peuvent pas être trop complexes sous peine de ne pas pouvoir tenir les contraintes de temps. Le fait de ne pas pouvoir utiliser de modèles trop coûteux en temps de calcul a un impact négatif sur le réalisme de l'émulation rendue. La plupart des solutions d'émulation existantes ont proposé des solutions permettant d'émuler des réseaux de grande taille au détriment parfois du réalisme de l'émulation rendue, pour être en mesure de tenir ces contraintes de temps. En effet les quelques solutions d'émulation qui tentent de simuler en temps réel des modèles complexes ne peuvent tenir la charge, en particulier lorsque le trafic est important, et ceci même lorsque des machines très puissantes sont utilisées. En utilisant des grappes de machines (*cluster*) et en distribuant le processus de calcul, il semble possible d'améliorer les performances. Cependant, cette distribution du calcul va introduire de nouveaux délais car il faut maintenir la cohérence globale de l'émulation si bien que du trafic d'administration va devoir circuler entre les machines du *cluster*. Cette solution est donc relativement coûteuse à mettre en œuvre et ne résout pas à elle seule les problèmes de temps réel. A l'échelle d'un laboratoire, il est donc nécessaire de trouver un moyen pour limiter les calculs à effectuer en temps réel pour permettre ainsi d'améliorer le réalisme de l'émulation.

Nos travaux se sont attachés à proposer une nouvelle solution d'émulation qui permette de réduire la part de calculs effectuée en temps réel, sans sacrifier la précision de l'émulation rendue et en préservant l'aspect dynamique et temps réel de l'émulation. La plate-forme d'émulation W-NINE résultant de ces travaux s'appuie sur une phase de simulation préalable à l'émulation proprement dite. Cette phase de simulation, dé-corrélée des contraintes temps réel permet d'utiliser des modèles précis, potentiellement complexes et coûteux en temps de calcul, pour déterminer les conditions à émuler à la suite de la simulation. La phase d'ému-

lation qui s'effectue toujours en temps réel est chargée de reproduire les conditions calculées par simulation lors de la phase précédente et de prendre en compte les effets résultant du trafic qui circule sur le réseau. En effet, l'approche d'émulation ne nécessite pas l'utilisation de modèle de trafic. Ceci simplifie les tests de l'utilisateur car il est parfois complexe de fournir un tel modèle comme par exemple dans le cas d'une application utilisant des tailles de paquets variables. Pour préserver cet avantage, la phase de simulation utilisée préalablement à l'émulation fonctionne sans modèle de trafic si bien que des effets liés au trafic réel instantané devront être pris en compte en temps réel au cours de la phase d'émulation. Un exemple des effets liés au trafic peut être observé lorsque deux terminaux émettent simultanément vers un même troisième terminal ce qui provoque des perturbations du signal (cas des terminaux cachés détaillé en section 2.6.2.3).

La plate-forme d'émulation proposée ici se présente comme une boîte noire composée de machines reliées par un réseau filaire. Ces machines sont chargées de reproduire les caractéristiques d'un réseau sans fil en fonction des résultats obtenus par simulation. Pour cela, le trafic circulant sur la plate-forme subit des limitations de débit, des introductions de délai ou des pertes de paquets semblables aux conditions que l'on peut observer dans un réseau sans fil.

Ce principe de boîte noire permet de connecter à la plate-forme des machines hétérogènes sur lesquelles les protocoles et les applications à tester sont déployés. Ceci laisse donc une grande liberté à l'utilisateur de la plate-forme quant au développement de son prototype. La seule contrainte pour ces machines est, bien sûr, qu'elles disposent d'une interface réseau filaire leur permettant de se connecter à la plate-forme.

Le manuscrit est découpé en cinq chapitres. Le chapitre 1 présente les différentes catégories de réseaux sans fil existant puis s'intéresse plus particulièrement aux réseaux locaux sans fil. A travers la présentation de la norme IEEE 802.11, les principales différences qui existent entre ce type de réseau et les réseaux filaires sont mises en évidence, notamment en ce qui concerne l'accès au canal de communication et son partage, en se limitant cependant aux fonctionnalités nécessaires à une bonne compréhension de la suite du document.

Le chapitre 2 présente les différentes solutions d'émulation existantes à ce jour pour les réseaux sans fil et discute notamment des avantages et inconvénients de chacune. Dans ce chapitre nous discutons également de la notion de "niveaux d'émulation" et nous justifions notre choix d'utiliser une solution d'émulation centralisée pour émuler des réseaux sans fil au niveau IP. C'est également dans ce chapitre que sont mises en évidence les limitations du calcul en temps réel de l'ensemble des caractéristiques de qualité de service (débits, pertes, délais, etc.) d'un réseau sans fil.

Le chapitre 3 montre l'intérêt d'avoir une simulation préalable à l'émulation et présente les différents modèles nécessaires à une bonne description de l'environnement sans fil. Ces modèles sont regroupés en trois grandes catégories :

- les modèles de mobilité, qui sont chargés de déterminer les positions occupées dans le temps par les terminaux qui composent le réseau sans fil ;
- les modèles de propagation, qui permettent d'estimer les effets de la distance et de l'environnement sur une communication radio en fonction des puissances d'émission et des positions de chaque terminal ;
- les modèles de communication, qui permettent par exemple de déterminer le débit utile maximum atteignable au niveau IP, mais également de prendre en compte des événements liés au trafic qui circule sur le réseau comme les cas de terminaux cachés ou encore de construire les cellules pour un réseau en mode Infrastructure.

Une fois ces modèles présentés, nous présentons les principes de notre plate-forme d'émulation puis nous proposons une solution de communication permettant à une phase de simulation d'être utilisée préalablement à une phase d'émulation. Pour cela, deux types de fichier au format XML (*eXtensible Markup Language*) respectant des schémas que nous définissons, sont utilisés :

- les fichiers de description de haut niveau, qui permettent à l'utilisateur de définir les modèles de mobilité, de propagation et de communication qu'il souhaite utiliser dans sa simulation ainsi que la description des terminaux (puissance d'émission radio, adresse IP, etc.) et des obstacles qui composent le réseau sans fil dans lequel ces expériences vont avoir lieu ;
- les scénarios d'émulation, qui organisent sous forme temporelle les résultats obtenus par le simulateur et qui devront être reproduits en temps réel lors de la phase d'émulation.

Le chapitre 4 présente plus en détail les deux phases utilisées par notre solution d'émulation. Pour la phase de simulation, un nouveau simulateur de réseau pour l'émulation est spécifié. Pour la phase d'émulation, de nouvelles techniques permettant de prendre en compte le trafic qui circule sur le réseau sont présentées au même titre que les solutions retenues pour assurer la reproduction d'une expérience. Ce chapitre est également l'occasion de présenter les travaux menés dans le cadre du réseau d'excellence européen NEWCOM (*Network of Excellence in Wireless Communications*) et qui ont conduit à l'intégration de KAUnet, un outil développé par l'Université de Karlstad (Suède) pour améliorer le réalisme de l'émulation rendue.

Le chapitre 5 propose une évaluation expérimentale des outils développés à travers différents cas d'utilisation. Le premier cas d'utilisation permet notamment de valider notre approche en montrant la concordance des résultats obtenus lors de la phase de simulation et de la phase d'émulation. Le second cas d'utilisation présente les résultats obtenus par la solution proposée lors de la prise en compte du trafic réel qui circule sur le réseau au cours d'une expérience. Le dernier cas d'utilisation permet de présenter les améliorations obtenues grâce à l'intégration des nouvelles fonctionnalités améliorant la gestion des pertes qui sont proposées par l'outil de l'Université de Karlstad.

Enfin, nous concluons ce manuscrit par une discussion des différents résultats obtenus et des conclusions que nous en avons tirées puis en présentant les différentes pistes de travail pouvant être explorées dans le futur.

2 Des environnements sans fil

2.1 Introduction

Jusqu'au début des années 1990, aucun type de réseau n'était en mesure de proposer des solutions sans fil pour acheminer de la voix ou des données. Les premières réussites sont à mettre au crédit de la téléphonie qui la première proposa des téléphones dotés de combinés sans fil. Bien que ne disposant que d'une faible autonomie et d'une faible portée de communication, ces téléphones préfiguraient ce besoin de communication et cette envie de pouvoir communiquer de n'importe où. Ils ont également eu le mérite d'habituer les gens à se passer de prises murales.

La véritable explosion des réseaux sans fil pour les réseaux de données a eu lieu à la fin des années 90 et au début des années 2000 grâce à des solutions comme 802.11a ou 802.11b/g qui offrent des débits et une qualité de services se rapprochant de plus en plus des réseaux filaires traditionnels. 802.11a et 802.11g proposent par exemple des débits de l'ordre de 54 Mbit/s et 802.11b des débits de l'ordre de 11 Mbit/s. La prochaine évolution 802.11n annonce même des débits pouvant atteindre plus de 500 Mbit/s. Grâce à ces débits élevés et profitant de l'explosion de l'Internet, ces réseaux sans fil se sont imposés comme une solution viable pour les réseaux d'accès dans les entreprises et les universités et même plus récemment chez les particuliers.

Le succès rencontré par les réseaux sans fil est tel qu'ils sont en passe de remplacer les technologies utilisées par le monde de la téléphonie pour les réseaux dits de quatrième génération qui devront assurer un transport de la voix et de données avec des débits élevés en s'appuyant sur la norme IP. Les dernières technologies comme WiMax vont d'ailleurs dans ce sens en proposant des débits de 74 Mbit/s sur des portées pouvant atteindre une cinquantaine de kilomètres. Toutes ces technologies proposent des portées, et une qualité de service très différentes si bien que plusieurs catégories de réseaux sans fil ont été mises en place, chacune étant dédiée à un type particulier d'application. Dans la suite de ce chapitre nous présenterons ces différentes catégories puis nous nous interesserons à l'une d'elles en particulier : les réseaux locaux sans fil.

2.2 Les différentes catégories de réseaux sans fil

Suivant la portée et le taux de transmission radio disponibles, on parlera de différents types de réseaux sans fil. Les applications vont souvent varier d'un type de réseaux à l'autre c'est pourquoi chaque type de réseau repose sur des normes de communication différentes.

2.2.1 WPAN

Les réseaux sans fil personnels ou *Wireless Personal Area Network* (WPAN), sont des réseaux sans fil à très faible portée (de l'ordre d'une dizaine de mètres). Ils sont le plus souvent utilisés dans le cadre du *Wearable Computing* [55] ou informatique vestimentaire qui consiste

à faire communiquer entre eux des matériels présents sur une personne (par exemple une oreillette et un téléphone portable). Ils sont également utilisés pour relier des équipements informatiques entre eux : par exemple pour relier une imprimante ou un *PDA* (Personal Digital Assistant ou assistant personnel) à un ordinateur de bureau.

Pour mettre en œuvre de tels réseaux, la principale technologie est *IEEE 802.15.1* [2] ou *Bluetooth*. Elle fut proposée par Ericsson en 1994 et fournit un taux de transmission radio théorique de 1 Mbit/s pour une portée maximale d'une trentaine de mètres. Elle est très peu gourmande en énergie ce qui la rend particulièrement intéressante pour être intégrée dans de petits équipements autonomes sans fil comme les *PDA*.

La technologie infrarouge ou *IrDA* est également utilisée dans ce type de réseaux. Cette technologie est cependant beaucoup plus sensible que *Bluetooth* aux perturbations lumineuses et nécessite une vision directe entre les éléments souhaitant communiquer ce qui la limite bien souvent à un usage de type télécommande.

2.2.2 WLAN

Les réseaux locaux sans fil ou *Wireless Local Area Network* (WLAN) sont généralement utilisés à l'intérieur d'entreprises, d'universités mais également chez les particuliers depuis le développement des offres à haut débit. Ces réseaux sont principalement basés sur la technologie *IEEE 802.11 [1]* soutenue par le *WECA (Wireless Ethernet Compatibility Alliance*) ou sur la technologie *HiperLan 1* et son remplaçant *Hiperlan 2* soutenue par l'*ETSI (European Telecommunications Standards Institute)*. Ils offrent des taux de transmission radio théoriques pouvant atteindre 54 Mbit/s pour *IEEE 802.11g* ou *HiperLan 2* c'est-à-dire bien plus élevés que la technologie *Bluetooth*. La portée de ces technologies est également beaucoup plus importante : de l'ordre de 300 mètres en extérieur et 100 mètres à l'intérieur de bâtiments pour *IEEE 802.11b* et d'une centaine de mètres pour *Hiperlan 2*.

Dans la suite du document nous nous intéresserons principalement à ce type de réseaux sans fil locaux. Une description plus détaillée des différentes normes sera présentée dans la section 2.6

2.2.3 WMAN

Les réseaux métropolitains sans fil ou *Wireless Metropolitan Area Network* (WMAN) également appelés boucle locale radio (BLR) étaient à l'origine prévus pour interconnecter des zones géographiques difficiles d'accès à l'aide d'un réseau sans fil. Actuellement ces réseaux sont utilisés dans certaines villes américaines (San Francisco) pour fournir un accès internet aux habitants. Les réseaux basés sur la technologie *IEEE 802.16* ont une portée de l'ordre de quelques dizaines de kilomètres (50 km de portée théorique annoncée) et un taux de transmission radio théorique pouvant atteindre 74 Mbit/s pour *IEEE 802.16-2004 [3]* plus connue sous le nom commercial de *WiMAX*.

C'est également dans cette catégorie que peuvent être classés les réseaux téléphoniques de troisième génération utilisant la norme *UMTS* (Universal Mobile Telecommunication System) pour transmettre de la voix et des données. Cette norme UMTS propose des taux de transmission radio théoriques pouvant aller jusqu'à 2 Mbit/s sur des distances de plusieurs kilomètres.

2.2.4 WWAN

Les réseaux sans fil étendus ou *Wireless Wide Area Network* (WWAN) regroupent notamment les différents réseaux téléphoniques de première et deuxième génération mais également les réseaux satellitaires. Les réseaux cellulaires téléphoniques reposent sur des technologies comme *GSM* (Global System for Mobile Communication), *GPRS* (General Packet Radio Service). Les réseaux satellites s'appuient quant à eux sur les normes comme *DVB-S* (*Digital Video Broadcasting-Satellite*) pour transmettre l'information et proposent des débits élevés (de l'ordre de 40 Mbit/s pour la norme *DVB-S*).

Ces réseaux se posent en concurrents directs des réseaux filaires traditionnels en proposant des services et une qualité de service de plus en plus proche de celle que l'on retrouve dans les réseaux filaires. Ils permettent en particulier de toucher des zones totalement enclavées dans lesquelles les réseaux de types WMAN ne peuvent être utilisés comme les déserts ou les zones montagneuses dépourvues d'infrastructures terrestres.

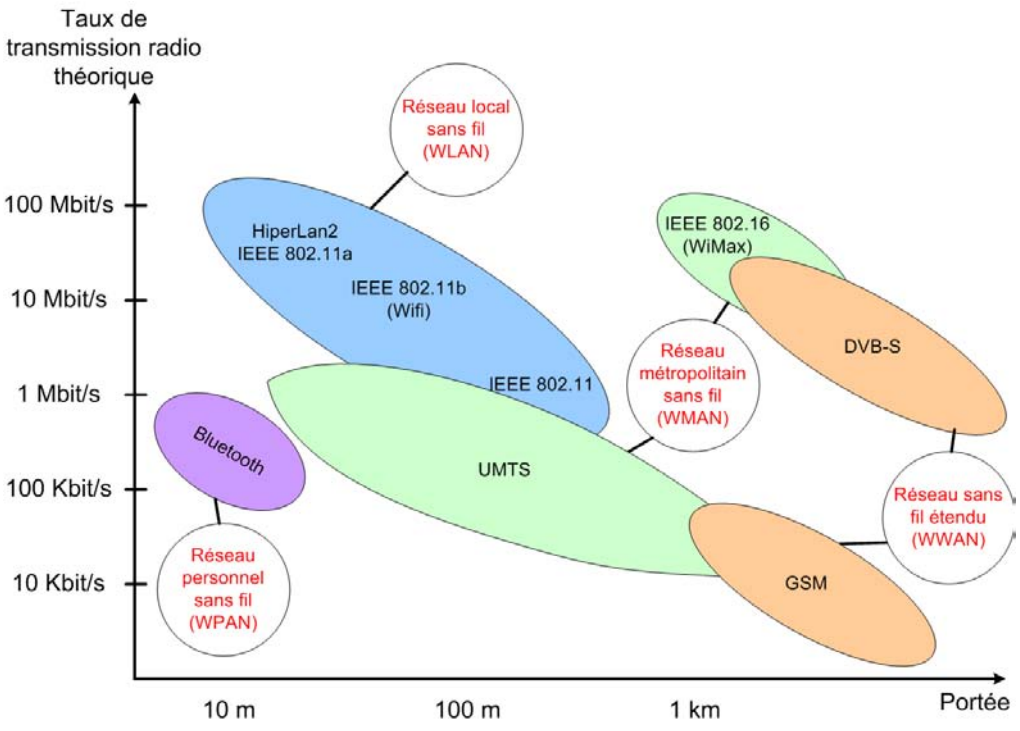


FIG. 2.1: Les catégories de réseaux sans fil.

La figure 2.1 présente ces différentes catégories. Dans la suite de ce document nous nous intéresserons uniquement aux réseaux locaux sans fil (WLAN) dont nous allons maintenant présenter quelques caractéristiques en nous intéressant surtout aux différences entre ces réseaux et les réseaux filaires.

2.3 Contraintes et problèmes spécifiques des réseaux sans fil

Par nature, les communications sans fil entraînent un certain nombre de problèmes n'ayant pas d'équivalent dans le monde filaire.

La propagation du signal radio est un élément clé de la qualité d'une transmission mais le nombre de fréquences et de canaux disponibles pour communiquer est limité. Ainsi, lorsque deux nœuds émettent simultanément sur des bandes de fréquences ou des canaux proches, des interférences seront ressenties sur les deux communications. De plus, la météo, la distance et l'environnement en général vont affecter la qualité d'une transmission radio en perturbant le signal. Le signal s'atténue en fonction de la distance entre l'émetteur et le récepteur mais également en fonction de problèmes de réflexion et de chemins multiples. Ces problèmes de réflexion vont dépendre des matériaux composant les différents obstacles : L'onde sera partiellement absorbée avec du bois ou complètement bloquée avec du plomb. Les différentes réflexions de l'onde sur les obstacles vont faire apparaître des chemins multiples sur une communication générant des interférences (du bruit) au niveau du nœud récepteur. Des sources électriques et/ou électromagnétiques extérieures peuvent également agir sur une communication en brouillant le signal radio émis. Toutes ces perturbations vont déformer le signal et ceci jusqu'à le rendre inutilisable par le destinataire.

Le fait que le signal s'atténue avec la distance pose un autre problème : le signal émis par un nœud est beaucoup plus fort que n'importe quel signal reçu si bien que lorsqu'un nœud est en train d'émettre, il ne lui est pas possible d'entendre une autre communication. Le signal qu'il envoie est tellement fort qu'il va lui masquer toutes les autres transmissions qu'il pourrait recevoir. Les nœuds ne peuvent donc pas émettre et écouter en même temps si bien que nous parlerons de réseaux half-duplex contrairement aux réseaux filaires qui sont par nature full-duplex. Contrairement aux réseaux filaires où un nœud détecte une collision en comparant le message circulant sur le câble avec celui qu'il cherche à émettre, le nœud émetteur dans un réseau sans fil ne peut pas détecter les collisions qui peuvent survenir au niveau du nœud récepteur à cause de l'atténuation du signal en fonction de la distance. Enfin, la dernière conséquence de cette atténuation du signal est que les nœuds ne peuvent communiquer qu'avec des nœuds géographiquement proches (on parlera alors de nœuds voisins). Cette portée de communication va définir des cellules naturelles dans lesquelles les communications pourront avoir lieu à un bond. Une cellule sera centrée sur un nœud et englobera tous les nœuds voisins. Ainsi, deux nœuds appartiendront à la même cellule s'ils sont à portée de communication l'un de l'autre

Les taux d'erreurs rencontrés dans un réseau sans fil sont beaucoup plus importants que ceux que l'on peut rencontrer dans les réseaux filaires. La nature même de ces pertes est différente. En effet, dans un réseau filaire, les pertes sont souvent dues à des congestions contrairement aux réseaux sans fil où les pertes sont majoritairement dues à des problèmes de transmission du signal. Les protocoles de transport filaires reconnus tels que TCP ne sont donc pas les mieux adaptés pour des réseaux sans fil. Ils s'intéressent principalement à la résolution des problèmes de congestion en introduisant des délais entre deux transmissions alors que dans le cas de pertes sans fil, il serait préférable de ré-émettre le plus rapidement possible.

Le débit des réseaux WLAN est beaucoup plus faible que celui que l'on trouve dans les réseaux filaires même si les premiers réseaux haut-débit sans fil commencent à voir le jour (54 Mbit/s pour 802.11g et 500 Mbit/s annoncés pour le futur 802.11n qui devrait le remplacer). La partie réservée à l'établissement et la maintenance d'une communication doit donc être la plus faible possible. Tout comme dans les réseaux filaires, il est impossible de parler

à un nœud en particulier. Tous les nœuds peuvent recevoir les paquets émis par leurs voisins même si ceux-ci ne leurs sont pas destinés et ceci de manière beaucoup plus simple que dans un réseau filaire car les communications se déroulent dans l'air, ce qui pose un problème évident de sécurité.

Les terminaux composant les réseau sans fil ont également la particularité de pouvoir être mobiles ce qui n'est pas le cas dans un réseau filaire. Cette mobilité va augmenter les problèmes de propagation du signal dans la mesure où un nœud mobile va voir les conditions de propagation du milieu qui l'entoure évoluer en fonction de sa position. Ceci va également pouvoir avoir un impact sur la topologie du réseau notamment dans le cas des réseaux Ad-Hoc comme nous le verrons dans la section 2.5. Ces terminaux mobiles peuvent également être limités en énergie ce qui peut poser des problèmes. En effet, un terminal va consommer de l'énergie pour envoyer et recevoir des paquets mais également pour écouter périodiquement le canal si bien qu'il peut disparaître à tout moment par simple manque d'énergie.

2.4 Les réseaux sans fil avec infrastructure

2.4.1 Définition

Les réseaux sans fil avec infrastructure sont également appelés réseaux cellulaires. Ces réseaux se composent de deux types de terminaux : les stations de base ou points d'accès et les terminaux mobiles ou nœuds mobiles. Les points d'accès sont des terminaux fixes reliés en général à une source d'énergie illimitée et chargés de relayer les informations qui circulent dans leur cellule. Ils jouent le rôle de serveur pour chaque terminal mobile présent dans la cellule. Les terminaux mobiles se déplacent librement mais ne communiquent jamais directement les uns avec les autres. Toutes les communications se font systématiquement vers le point d'accès le plus proche qui se charge ensuite de les relayer à la destination. Ces réseaux sont donc une réponse au problème d'affaiblissement du signal. En effet, en déployant plusieurs points d'accès on peut étendre la couverture globale du réseau. Tous les points d'accès sont reliés entre eux par un réseau filaire qui se charge de la partie routage ainsi que de la plupart des fonctions d'administration (authentification centralisée par exemple).

Ce type de réseau est de plus en plus utilisé dans les entreprises et les universités comme dernier bond d'un réseau d'accès à l'Internet. Ces réseaux sont souvent retenus pour leur simplicité d'administration. En effet comme les points d'accès sont fixes et n'ont pas de problème d'énergie, la topologie du réseau demeure dans l'ensemble assez stable. De plus, les problèmes de routage sont ici réduits à leur plus simple expression dans la mesure où toutes les communications (sur la partie sans fil) se font en un bond.

Il existe cependant des problèmes ouverts auxquels s'interessent de nombreux travaux de recherche. Parmi eux nous pouvons citer celui du changement de réseau aussi appelé *handover* ou *handoff*.

2.4.2 Le problème du changement de réseau (Handover ou Handoff)

Le changement de réseau peut être de deux types : horizontal lorsque l'on passe d'un point d'accès à un autre sur la même technologie, ou vertical lorsque l'on change de technologie pour améliorer la qualité de service. Seul le *handover* horizontal sera présenté ici.

Le *handover* survient lorsqu'un nœud se déplace et quitte la zone de couverture d'une cellule pour entrer dans la zone de couverture de la cellule voisine (cf. figure 2.2). Le temps de ré-

association entre le nœud et le nouveau point d'accès est appelé délai de *handover*. Lorsque le nœud atteint la zone de recouvrement des cellules, il rompt sa connexion avec le premier point d'accès et établit une nouvelle connexion avec le point d'accès de la cellule qu'il est en train d'intégrer. Il peut également conserver sa première connexion tout en établissant la communication avec le deuxième point d'accès, on parlera dans ce cas de *handover* doux (*soft handover*). Lorsque les cellules ne sont pas dans les mêmes bandes de fréquences (pour éviter les interférences et les collisions) on parlera de *handover* dur (*hard handover*) car il doit non seulement changer de point d'accès mais également changer sa bande de fréquence d'émission voire sa modulation ; dans ce cas il ne peut pas toujours maintenir la connexion.

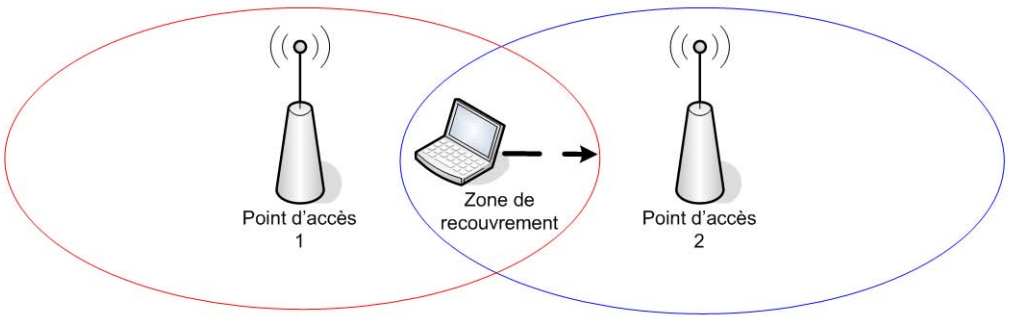


FIG. 2.2: Exemple de *handover* entre deux cellules

Lorsqu'un nœud change d'opérateur en changeant de cellule on dit alors qu'il effectue un forme particulière de *handover* appelée *roaming*. De plus, contrairement au *handover* classique qui suggère que la connexion est maintenue lors du changement de cellule, le *roaming* ajoute une dimension d'interruption de la communication. Un exemple simple de *roaming* se retrouve dans le monde de la téléphonie mobile où un client abonné chez un opérateur dans son pays d'origine va pouvoir utiliser le réseau d'un autre opérateur lors de ses déplacements à l'étranger et ceci de manière transparente.

2.5 Les réseaux sans fil sans infrastructure (Ad-Hoc)

2.5.1 Définition

Contrairement aux réseaux cellulaires qui sont organisés autour de points d'accès jouant le rôle de routeurs, les réseaux Ad-Hoc sont des réseaux distribués et spontanés se composant uniquement de terminaux mobiles. Dans ce type de réseau, présenté sur la figure 2.3, chaque nœud du réseau joue à la fois le rôle d'élément terminal et le rôle de routeur pour relayer les messages de ses voisins vers un nœud qui n'est pas situé dans le voisinage immédiat. On parle alors de réseau coopératif. Sur l'exemple présenté sur la figure 2.3, le nœud M_1 ne peut pas communiquer directement avec le nœud M_5, mais il peut faire relayer son message par M_2, M_3 et M_4. Le groupe de travail *Mobile Ad-Hoc NETworks* (MANET) de l'*Internet Engineering Task Force* (IETF) a formalisé l'ensemble de ces caractéristiques dans le *Request For Comment* (RFC) numéro 2501 [19].

Du fait de la mobilité des nœuds, des problèmes de propagation et d'énergie, la topologie d'un réseau Ad-Hoc évolue beaucoup au cours du temps si bien que les routes permettant aux nœuds de communiquer ne sont pas statiques. Contrairement aux réseaux filaires, une

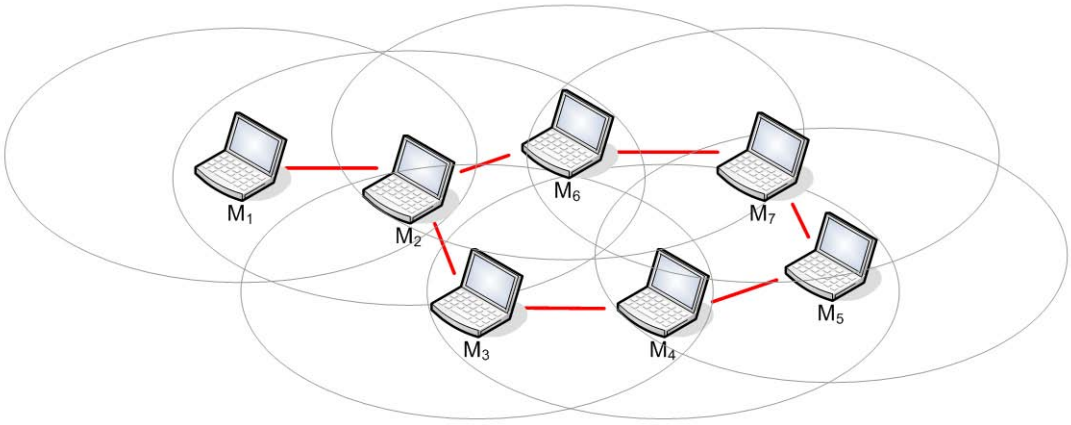


FIG. 2.3: Exemple de réseau Ad-Hoc

rupture de route ne peut pas être considérée comme un événement rare dans un réseau Ad-Hoc. La fréquence de ces évolutions va dépendre principalement de la vitesse de déplacement des nœuds mais également de l'énergie dont dispose chaque nœud. En effet, comme chaque nœud doit servir de routeur à ses voisins, la disparition d'un nœud par manque d'énergie va entrainer des re-calculs de routes. Dans certains cas, cette mobilité et ces aspects énergétiques peuvent également entrainer des partitionnements du réseau. Par exemple, si les nœuds M_3 et M_6 viennent à manquer d'énergie, les nœuds M_1 et M_2 ne pourront plus communiquer avec M_4, M_5 et M_7.

Les réseaux Ad-Hoc sont un des éléments moteurs de la recherche dans les réseaux sans fil car leurs domaines d'application sont très nombreux. Ils vont du scénario militaire où des soldats veulent échanger des informations en zone de combat, jusqu'aux interventions sur les sites de catastrophes naturelles en passant par la création spontanée d'un réseau pour des événements sociaux comme par exemple une conférence scientifique. Pour généraliser, ces réseaux vont être utiles partout où il n'y a pas d'infrastructure.

2.5.2 Le routage dans les réseaux Ad-Hoc

La principale difficulté rencontrée dans ces réseaux très évolutifs est le routage. En effet les réseaux Ad-Hoc sont des réseaux multi bonds contrairement aux réseaux cellulaires. Dans ce type de réseau, lorsqu'une source cherche à communiquer avec une destination distante de plus d'un bond, elle doit soit inonder le réseau, soit mettre en œuvre un algorithme de recherche de route avancé. La technique de routage par inondation permet certes de contacter une destination mais est très coûteuse en énergie et consomme également beaucoup de bande passante. L'utilisation d'un algorithme de routage (qui peut cependant utiliser l'inondation ponctuellement pour établir les routes) est le plus souvent préférée de manière à optimiser la gestion des ressources. Ce type de réseau est donc beaucoup plus sensible aux problèmes de mobilité des nœuds et aux problèmes d'énergie

Comme la topologie du réseau peut changer très rapidement, les protocoles de routage utilisés habituellement dans l'Internet ne sont pas facilement transposables. Dans les réseaux filaires, on considère en effet les déconnexions et les changements de route comme des événements rares, ce qui ne va pas être le cas pour un réseau Ad-Hoc. Pour résoudre ce problème,

de nouveaux algorithmes de routage Ad-Hoc ont été proposés dont les principaux se classent en trois grandes familles : les algorithmes à base de table, les algorithmes à la demande et les algorithmes hybrides.

Les algorithmes à base de table (Table Driven) Le principe des algorithmes à base de table, également appelés algorithmes pro-actifs, repose sur une connaissance complète de l'ensemble du réseau par chaque nœud. Pour avoir une carte complète du réseau, l'ensemble des nœuds maintient une ou plusieurs tables de routage qui lui permettent de joindre n'importe quel autre nœud du réseau. Ces tables sont remises à jour régulièrement par un échange périodique et/ou événementiel de messages. Les algorithmes à base de tables diffèrent principalement par la méthode de mise à jour des tables de routages et par le nombre de tables de routage utilisées par chaque nœud.

Par exemple, l'algorithme de routage *Destination Sequenced Distance Vector* (DSDV [64]) repose sur l'adaptation à l'environnement mobile du protocole de routage classique de Bellman Ford utilisé par le protocole *Routing Information Protocol* (RIP [29]). C'est un protocole à vecteur de distance, ce qui veut dire que chaque station mobile maintient une table de routage où chacune des lignes caractérise une des destinations possibles. Cette table contient le nombre de sauts qu'il y a entre la source et la destination souhaitée, le terminal mobile voisin qu'il faut traverser ainsi que le numéro de séquence qui est affecté à cette route. Ce numéro de séquence va permettre de faire une distinction entre les anciennes et les nouvelles routes. Lors d'un changement de topologie dans le réseau, les tables sont mises à jours et les modifications sont propagées aux autres nœuds du réseau. On peut propager soit la ligne modifiée, soit la table de routage toute entière. Le choix de la méthode va dépendre du nombre de mises à jour à faire dans la table à un instant donné : si le nombre de modifications est trop important il vaut mieux transmettre toute la table de façon à ne pas surcharger le réseau.

Le protocole *Optimized Link State Routing* (OLSR [15]) est un protocole à état de lien standardisé par l'IETF qui peut être vu comme une transposition du protocole filaire *Open Shortest Path First* (OSPF [58]) aux réseaux sans fil. Un algorithme à état de lien permet à chaque nœud du réseau de connaître avec précision la position de tous les autres nœuds composant le réseau. Chaque nœud peut donc ainsi choisir la route la plus adéquate pour contacter une destination en mettant en œuvre un algorithme de Dijkstra. Cette technique permet d'éviter tout ce qui est redondance et propose plutôt d'organiser le réseau pour minimiser le nombre de messages à échanger. Pour ce faire, certains nœuds du réseau sont élus pour jouer un rôle de relai multi-points (MPR). Un nœud est élu pour être MPR après que les nœuds ont échangé entre eux des messages HELLO leur permettant de déterminer si les liens qui les relient sont symétriques ou asymétriques. Un MPR est choisi s'il peut contacter tous les nœuds situés à deux sauts par des liens symétriques. Une fois que les MPRs sont choisis, les autres nœuds du réseaux reçoivent la liste de ces MPRs et mettent à jour leurs tables de routage.

Le principal intérêt de ce type de protocole est donc qu'un nœud sait directement comment communiquer avec n'importe quelle destination ce qui évite des temps de latence lors de l'établissement d'une communication. Par contre lorsque le réseau est hautement mobile, le nombre des paquets de contrôle à échanger pour maintenir les tables va augmenter en grande proportion, ce qui va diminuer d'autant la bande passante disponible pour les applications. Ce type d'algorithme doit donc être réservé à des réseaux sans fil de grande taille et avec une faible mobilité.

Les algorithmes à la demande (On Demand) Contrairement aux algorithmes pro-actifs, les algorithmes à la demande, également appelés algorithmes réactifs, n'ont pas une vision complète du réseau. La recherche de route n'est initiée que lorsqu'un nœud a besoin d'établir une communication. Suivant les algorithmes, les routes peuvent ensuite être stockées dans une table de routage locale et être maintenues par l'envoi périodique de messages de signalisation. Chaque entrée de la table n'est valable que pendant une certaine période de temps qui est remise à jour à l'arrivée des messages de signalisation. Si aucun message de signalisation concernant une entrée n'arrive avant cette période, on dit alors que la route arrive à expiration. Lorsque cela se produit, le nœud supprime simplement l'entrée de sa table et ne cherche généralement pas à la rétablir.

Le protocole *Ad-Hoc On-demand Distance Vector* (AODV) proposé dans [63] propose une amélioration de l'algorithme DSDV présenté précédemment. Contrairement à DSDV, AODV ne construit les routes que lorsqu'un nœud a effectivement besoin de communiquer. Lors d'une découverte de route, le nœud inonde le réseau avec un message de requête de route (RREQ). Les nœuds voisins propagent cette requête et en conservent une trace dans leurs tables de routage. Si un nœud retransmet une requête il va stocker dans sa table l'identifiant du nœud à partir duquel la première copie de la requête a été reçue. Ces informations vont ensuite permettre de construire les routes de retour que va emprunter le message de réponse (RREP) de la destination. La phase de découverte de route n'est re-initiée que si la source bouge ou si elle reçoit d'un de ses voisins une indication selon laquelle la route vers la destination n'est plus disponible. Pour vérifier si une route est toujours valide, chaque nœud diffuse périodiquement à tous ses voisins (*broadcast*) des messages HELLO qui lui permettent de savoir si ses voisins sont toujours présents. Si l'un d'eux ne répond pas, il regarde dans sa table les routes utilisant ce voisin et la supprime. Il prévient ensuite les nœuds utilisant ces routes de cette disparition en diffusant un message de réponse de requête de route avec une métrique infinie.

De manière générale, ces algorithmes à la demande sont utilisés dans des réseaux à forte mobilité où l'établissement et surtout la maintenance des routes, nécessaires aux algorithmes à base de table, s'avèrent trop coûteuses en bande passante et en énergie. Cependant, dans un réseau dense à faible mobilité, ces algorithmes introduisent des délais qui ne sont pas satisfaisants. En effet la recherche d'une nouvelle route implique un certain délai (plusieurs secondes) qui sera forcément ressenti au niveau applicatif si le nombre de nœuds à traverser est important. De plus, l'envoi périodique de paquets pour s'assurer que la route n'a pas disparu, va diminuer la bande passante disponible au niveau applicatif ce qui peut être pénalisant dans la mesure où ces échanges sont superflus quand la topologie est relativement stable.

Les algorithmes hybrides Les protocoles hybrides vont utiliser à la fois des techniques pro-actives et des techniques réactives pour acheminer l'information. Le choix de la technique dépend de l'organisation du réseau et en règle générale de la distance qui sépare la source de sa destination.

Un exemple de routage hybride est *Zone Routing Protocol* (ZRP [28]) qui utilise une technique pro-active à l'intérieur d'une zone et une technique réactive pour communiquer avec un autre mobile se trouvant à l'extérieur de la zone. Une zone est par définition centrée sur un nœud et peut englober tous les nœuds situés à un, deux ou plusieurs bonds de ce nœud. Chaque nœud du réseau va définir une zone, et en donner la taille en nombre de sauts. Cependant, plus le nombre de sauts est grand, plus il y a de risques d'affecter les performances.

D'autres algorithmes hybrides comme l'algorithme *Hierarchical State Routing* (HSR [40])

proposent un découpage hiérarchique en plus du découpage en zone. Ainsi, dans le cas de HSR, chaque zone va élire une tête de zone qui sera chargée de gérer les communications à l'intérieur de la zone. Les têtes de zone vont ensuite reformer des zones de niveau supérieur dont elles seront membres et reprendre le processus d'élection de tête de zone. Ce processus peut ainsi être reproduit sur plusieurs niveaux jusqu'à avoir une organisation hiérarchique satisfaisante. Pour les communications, HSR a mis en place un adressage hiérarchique qui lui permet de savoir jusqu'à quel niveau il est nécessaire de remonter dans la hiérarchie pour pouvoir trouver une destination. Chaque nœud dispose d'une adresse hiérarchique et d'une table lui permettant de déterminer s'il doit contacter sa tête de zone ou s'il peut communiquer directement avec la destination (si celle-ci se trouve dans la même zone).

Les autres algorithmes Quelques algorithmes de routage font appel à des principes différents et ne peuvent donc pas être classés dans ces catégories. En effet, il existe également des algorithmes qui utilisent des informations propres aux réseaux Ad-Hoc pour communiquer, comme l'énergie pour le protocole *Power Aware Routing* (PAR [73]) . PAR va choisir une route non pas en fonction de la vitesse de transmission mais plutôt en fonction de l'énergie nécessaire pour acheminer l'information. La sélection d'une route en fonction de l'énergie de chaque nœud permet d'atteindre plusieurs objectifs dont le principal est de garantir une durée de vie plus importante du réseau en retardant au maximum les risques de partition. Pour atteindre cet objectif, une route est sélectionnée en fonction de l'énergie nécessaire pour émettre un paquet mais également en essayant de diminuer la variance entre les niveaux d'énergie des différents nœuds.

D'autres algorithmes comme *Distance Routing Effect Algorithm for Mobility* (DREAM [6]) utilisent des informations géographiques pour déterminer les routes en diminuant ainsi le nombre d'inondations nécessaires. Pour cela, il s'appuie sur les coordonnées des nœuds qui peuvent être obtenues grâce à l'utilisation d'antennes GPS (Global Positionning System) par exemple. Grâce à ces coordonnées, cet algorithme à la demande n'envoie un paquet que dans la direction géographique où se trouve la destination. La route n'est donc pas obtenue en fonction de la rapidité de transmission mais en fonction de la distance séparant l'émetteur de la destination. Cette technique a également l'avantage de supprimer la phase d'inondation lors de la recherche de route si les terminaux sont équipés d'antennes directionnelles. En effet, seuls les nœuds situés dans le voisinage géographique de la destination ont besoin d'être contactés.

Déploiement des protocoles de routage Ad-Hoc Le groupe de travail MANet recommande de déployer ces protocoles de routage au niveau IP, cependant ceci soulève certains problèmes. Par exemple l'utilisation d'une adresse IP de diffusion (*broadcast*) doit permettre de contacter tous les nœuds du réseau, or, dans un réseau Ad-Hoc elle ne permettra à une nœud de ne contacter que les nœuds situés dans son voisinage immédiat. En effet, la couche IP ne permet pas d'effectuer de routage sur ce type d'adresse et ceci ne peut être modifié sous peine de perdre l'inter-opérabilité avec l'Internet et les réseaux filaires. De même, lorsqu'un nœud ne fait pas parti du réseau IP, il ne lui est pas possible d'effectuer de requête *Dynamic Host Configuration Protocol* (DHCP), dans la mesure où ce type de requête n'est pas non plus relayé. Des travaux ont donc proposé de nouvelles architectures compatibles avec l'architecture IP qui prennent en compte les spécificités Ad-Hoc. Le protocole de routage ABR [75] propose l'utilisation d'adresses Ad-Hoc liées aux adresses IP pour permettre aux nœuds de communiquer et offre pour cela une nouvelle couche protocolaire située entre la couche MAC et la couche IP.

HSR, algorithme hybride présenté précédemment, propose également une technique d'adressage Ad-Hoc permettant au nœuds de communiquer sans passer par la couche IP. Enfin, des architectures Ad-Hoc spécifiques comme ANA4 [11, 7] ont été proposées. Comme ABR, ANA4 propose l'utilisation d'une couche Ad-Hoc située entre la couche MAC et la couche IP qui sera chargée de mettre en œuvre les mécanismes de routage. Cette solution fournit également à la couche IP des solutions pour le *broadcast* notamment et ceci de manière transparente grâce à l'utilisation d'une interface virtuelle Ad-Hoc. Cette interface virtuelle Ad-Hoc va permettre de masquer à la couche IP les différentes interfaces physiques dont peut disposer le nœud mobile et va utiliser une adresse Ad-Hoc indépendante de l'adresse IP. Cette solution permet donc, contrairement à ABR, de relayer les requêtes DHCP à travers le réseau Ad-Hoc de manière transparente pour la couche IP.

2.6 Les réseaux locaux sans fils

Les réseaux locaux sans fils avec infrastructure prennent de plus en plus d'importance dans les entreprises, les universités et même chez les particuliers. Pour cela, plusieurs technologies ont été proposées parmi lesquelles nous avons choisi de présenter HiperLan et IEEE 802.11.

2.6.1 La norme européenne HiperLan

HiperLan est une norme européenne standardisée par l'ETSI (European Telecommunications Standard Institute) qui a proposé deux versions d'HiperLan pour les réseaux de type WLAN : HiperLan 1 [22] en 1996 et HiperLan 2 [23] en 1999. Moins répandue que 802.11 (voir section 2.6.2), elle propose néanmoins des mécanismes très performants et intègre des possibilités Ad-Hoc.

2.6.1.1 HiperLan1

Tout comme la norme IEEE 802.11a [36] présentée dans la section 2.6.2.1 qu'il concurrence, HiperLan 1 fonctionne dans la bande de 5.1-5.3 GHz. Il permet de communiquer sur une portée de 50 m en offrant un débit de l'ordre de 23.5 Mbit/s pour des terminaux se déplaçant à la vitesse maximum de 10 m/s.

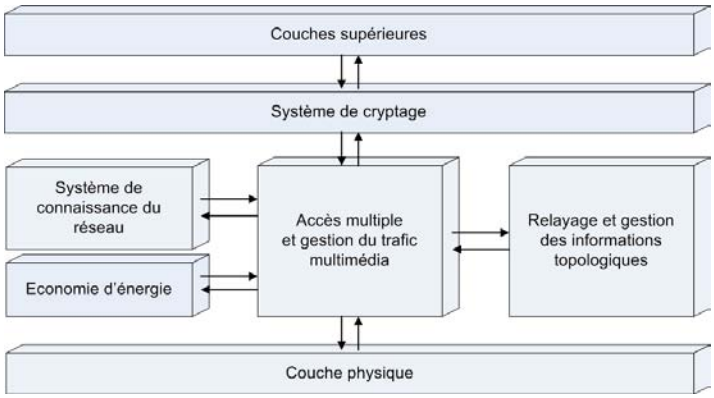


Fig. 2.4: Architecture d'HiperLan de type 1

HiperLan 1 repose sur une architecture complètement décentralisée (présentée sur la figure 2.4) où chaque terminal peut jouer le rôle de relai. Cette technique permet d'étendre la zone de couverture radio en intégrant un système de routage multi-bonds dont s'est d'ailleurs inspiré le protocole OLSR présenté précédemment. En raison de cette architecture décentralisée, la notion de point d'accès n'est pas représentée dans HiperLan 1. Une autre caractéristique d'HiperLan est sa méthode d'accès au canal appelée *Elimination Yield-Non Preemptive Multiple Access* (EY-NPMA). EY-NPMA permet un accès multiple au canal avec des gestions de priorités et se découpe en trois parties :

- La **phase de signalement de priorité** : Lorsqu'un nœud souhaite émettre sur le canal il commence par écouter le canal. Si personne n'est en train d'émettre, il envoie son message de priorité. Cette priorité va de zéro à quatre où zéro représente la priorité maximum. En fait, ce message de priorité correspond à l'envoi d'un *burst* de signalement dans l'intervalle (slot) temporel correspondant à la priorité. Un nœud avec une priorité deux va ainsi attendre durant deux slots temporels puis émettre son *burst* pendant le troisième slot. Par exemple, si un nœud M_1 envoie son burst durant le slot de priorité un, le nœud M_2 qui est en attente pour envoyer son propre *burst* avec une priorité deux, voit le burst émis par M_1 et diffère son émission. On dit alors que le nœud M_2 met fin à son cycle d'accès au canal. Après cette phase, il ne reste plus en concurrence que les nœuds de même priorité. Pour les départager, on entre ensuite dans la phase dite d'élimination.
- La **phase d'élimination** consiste pour chaque nœud à continuer à envoyer le slot de priorité pendant une durée aléatoire et à se remettre en écoute juste après. Si le nœud détecte une autre émission, il considère qu'il a perdu et abandonne le cycle d'accès au canal. C'est donc le nœud ayant tiré le temps maximum d'émission de son slot de priorité qui va gagner l'accès au canal. A la fin de cette phase il ne reste normalement que très peu de nœuds en concurrence.
- Lors de la **phase d'écoute**, le nœud attend une période aléatoire, appelée *backoff*, durant laquelle il écoute le canal pour voir si un autre nœud est en train d'émettre. De la même manière que pour la phase de priorité, si le nœud détecte une activité au cours de sa période d'attente il arrête son cycle d'accès au canal. C'est donc le nœud ayant tiré le *backoff* le plus petit qui va finalement gagner l'accès au canal.

Les différentes valeurs numériques retenues dans la norme pour ces différentes phases permettent de calculer que le taux de collision est inférieur ou égal à $3,5\%$ pour 256 utilisateurs. Ainsi, les priorités seront bien respectées lors des émissions et surtout les transmissions avec des priorités moindres ne viendront pas interférer avec les transmissions de priorités hautes.

Ces priorités sont notamment utilisées pour les trafics ayant de fortes contraintes temporelles telles que les trafics multimédias. La norme HiperLan 1 permet de gérer ce type de trafics en s'appuyant sur les informations fournies par l'application : la durée de vie résiduelle du paquet et la priorité de l'utilisateur. Pour mettre à jour cette priorité de niveau MAC, il faut commencer par mettre à jour la durée de vie du paquet reçu. Pour cela HiperLan 1 tient compte de la priorité utilisateur ainsi que du nombre de sauts nécessaires pour atteindre la destination. En fonction de cette priorité de niveau utilisateur et de la durée que va prendre la transmission du paquet (vie résiduelle), une priorité de niveau MAC est générée assurant ainsi que le paquet sera bien transmis dans les bons délais.

2.6.1.2 HiperLan 2

HiperLan 2 fut proposé en 1999 et définit un réseau d'accès radio pouvant s'interfacer avec différents types de réseaux classiques et notamment TCP/IP, ATM et UMTS. Contrairement à HiperLan 1, HiperLan 2 propose une architecture centralisée autour des contrôleurs centraux (CC). La norme définie dans [23] présente tous les aspects de communication entre un terminal mobile et un CC dont nous ne détaillerons pas ici tous les tenants et aboutissants : une description plus fine des différents mécanismes peut être retrouvée dans la norme ou même dans [57]. Pour résumer, HiperLan 2 propose deux modes de fonctionnement : un mode centralisé (appelé réseau d'accès) où des CCs connectent des terminaux mobiles à une infrastructure réseau (par exemple un réseau de type LAN Ethernet). Le second mode de fonctionnement est un mode Ad-Hoc où les terminaux mettent en œuvre un mécanisme d'élection de CC pour définir quel nœud va être chargé d'administrer le réseau.

Dans les deux cas il existe deux modes de communication : un mode totalement centralisé où toutes les communications inter-nœuds transitent par le CC et un mode de communication directe où les communications entre deux terminaux connectés au même CC peuvent se faire directement sans passer par le CC.

Une autre particularité de HiperLan 2 est de mettre en œuvre un système de contrôle d'erreurs basé sur l'algorithme *Automatic Repeat reQuest* (ARQ) qui permet d'avoir une retransmission sélective des trames. De plus, la bande passante atteint avec cette version 54 Mbit/s.

2.6.2 La norme IEEE 802.11

2.6.2.1 Historique

802.11 est un standard établi par l'IEEE en 1997 [1]. Cette première version propose une technique de communication sans fil avec des débits de 1 ou 2 Mbit/s sur quelques centaines de mètre en extérieur et fonctionne sur la bande des 900 Mhz. Cette norme propose deux modes de communication :

Le mode infrastructure Dans ce mode, des stations de base, également appelées point d'accès, sont reliées par un réseau filaire et assurent les communications radio entre les différents terminaux mobiles situés dans leurs voisinages. Chaque nœud mobile fait partie d'une cellule et un point d'accès va être responsable de gérer l'ensemble des communications à l'intérieur de sa cellule mais également entre les cellules. Toutes les communications passent par le point d'accès, il n'y a pas de communications directes entre les terminaux mobiles même si ceux-ci sont très proches.

Le mode Ad-Hoc Dans ce mode l'ensemble des terminaux mobiles sont égaux et peuvent jouer à la fois le rôle d'élément terminal mais également de routeur pour acheminer les communications distantes. Ce mode routeur n'est cependant pas décrit directement dans la norme, celle ci se contentant de définir une communication point à point. Il n'y a pas de protocoles de routage ad-hoc standardisé dans la norme 802.11.

Pour assurer ces communications sans fil, trois couches physiques peuvent être utilisées : DSSS (Direct Sequence Spread Spectrum), FHSS (Frequency Hopping Spread Spectrum) et IR (Infra Red).

Cette première version de 802.11 a rapidement été améliorée grâce notamment aux groupes de travail mis en place par l'IEEE. Une première amélioration est proposée en 1999 (IEEE 802.11-99 [16]). Elle fait passer les communications dans la bande des 2.4 Ghz (bande ISM = *Industrial, Scientific and Medical*) qui est également utilisée par la technologie Bluetooth. La même année est proposée la norme IEEE 802.11b [37] qui autorise des communications à 5.5 Mbit/s et 11 Mbit/s sur la bande ISM en utilisant une couche physique améliorée HR-DSSS (High Rate DSSS) utilisant une modulation CCK (Complementary Code Keying). Cette nouvelle norme demeure inter-opérable avec la norme 802.11-99 et propose donc également des débits de transmissions de 1 et 2 Mbit/s. Dans la suite de ce document nous nous intéresserons principalement à cette version de 802.11 qui s'est rapidement imposée comme un standard de fait.

IEEE 802.11a [36], offre des débits plus rapides que 802.11b qui peuvent aller jusqu'à 54 Mbit/s mais ceci au détriment de la distance de réception. Pour atteindre des débits aussi importants, la modulation utilisée n'est plus une version modifiée de DSSS mais la modulation *Orthogonal Frequency-Division Multiplexing* (OFDM). Une autre particularité de 802.11a est qu'elle fonctionne dans la bande des 5 Ghz (bande UN-II = *Unified National Information Infrastructure*) comme HiperLan et n'est donc pas inter-opérable avec les deux versions précédemment citées.

Pour répondre à ce besoin d'inter-opérabilité, rendu nécessaire par le succès commercial des équipements 802.11b, le groupe de travail IEEE 802.11g a proposé d'utiliser une couche physique OFDM dans la bande des 2.4 Ghz tout en conservant les modes de fonctionnement utilisant HR-DSSS pour pouvoir communiquer avec des matériels 802.11b.

Il existe de nombreux autres groupes de travail comme IEEE 802.11e qui cherche a améliorer la Qualité de Service (QdS) en fournissant des garanties aux utilisateurs, IEEE 802.11f qui est une recommendation visant à simplifier le changement de point d'accès pour un nœud, ou IEEE 802.11i qui se focalise sur les aspects de sécurité et d'autentification. La plupart de ces travaux sont toujours en cours et n'ont pour le moment pas fait l'objet d'une mise sur le marché. Enfin, d'autres groupes travaillent sur l'internationalisation de IEEE 802.11 comme le groupe IEEE 802.11d ou sur le rapprochement avec d'autre normes comme HiperLan pour IEEE 802.11h et la norme japonaise pour IEEE 802.11j.

Que ce soit pour 802.11b/g ou 802.11a, la bande de fréquence utilisée est séparée en plusieurs bandes de fréquence appelées canaux. Pour la bande ISM, 14 canaux d'une largeur de 22 MHz sont disponibles. Cependant, ces canaux présentés sur la figure 2.5 sont recouvrants si bien qu'en pratique les canaux qui sont le plus communément utilisés sont les canaux 1, 7 et 13 ce qui permet de faire cohabier plusieurs cellules. En effet prendre des canaux éloignés les uns des autres permet d'éviter au maximum les risques d'interférence.

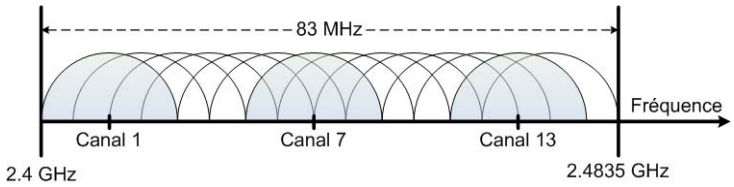


Fig. 2.5: Le recouvrement de canaux dans la bande ISM

Pour permettre d'étendre la couverture du réseau tout en évitant les interférences, il est possible d'utiliser une technique dite de réutilisation spatiale grâce à un plan d'allocation

de fréquences comme celui présenté sur la figure 2.6. Ce plan d'allocation est d'autant plus nécessaire que la bande ISM est surchargée. Sur cette bande de fréquence, 802.11b/g est en concurrence avec Bluetooth (claviers et souris sans fils, PDA, oreillette pour téléphone mobile...) qui connaît actuellement un grand succès dans le domaine des *WPANs* mais également avec les ondes émises par les fours à micro ondes.

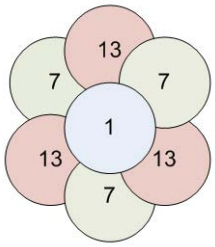


FIG. 2.6: La réutilisation spatiale avec 802.11b/g

Pour la bande UN-II par contre, les canaux sont au nombre de douze et sont disjoints ce qui ne nécessite pas de plan de réutilisation. De plus, cette bande de fréquence est beaucoup moins encombrée que la bande ISM. Le seul autre concurrent à l'utiliser est HiperLan.

La figure 2.7 présente les couches basses de 802.11 en les comparant avec la pile *OSI* (Open System Interconnection [39]) ainsi que la pile Ethernet (IEEE 802.3). Les couches supérieures ne sont pas représentées car elles sont similaires à celles que l'on retrouve dans l'Ethernet ce qui permet aux réseaux filaires et aux réseaux sans fil d'être inter-opérables.

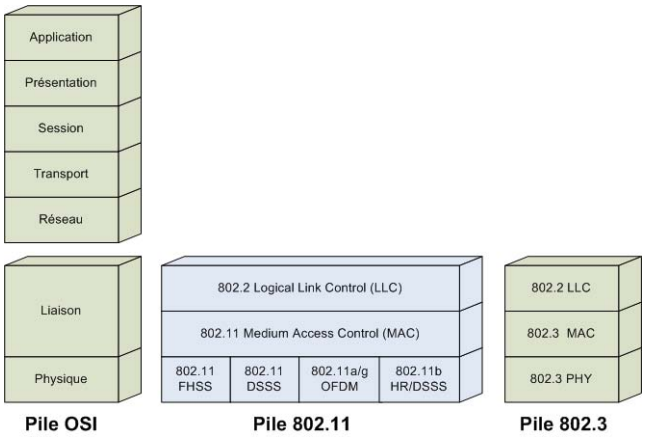


FIG. 2.7: Les couches basses 802.11

2.6.2.2 Le protocole d'accès au médium

802.11 propose deux modes de fonctionnement, le mode ad-hoc utilisant une *Distributed Coordination Function* (*DCF*) et le mode infrastructure utilisant une *Point Coordination Function* (*PCF*) ou bien la *DCF*.

- *DCF* est un mode utilisable par tous les nœuds mobiles qui garantit un accès équitable au canal radio sans recours à une quelconque centralisation. Dans ce mode tous les nœuds sont égaux et choisissent quand ils veulent parler.

- *PCF* est un mode centralisé où les stations de bases sont chargées de distribuer l'accès au canal aux nœuds qui en font la demande. Chaque station de base gère ainsi l'accès au canal radio des nœuds faisant partie de sa cellule.

Le mode *DCF* a donc été retenu pour les réseaux en mode Ad-Hoc mais également pour les réseaux en mode Infrastructure dans la mesure où la plupart des équipements disponibles sur le marché n'implémentent pas le mode *PCF*.

Description du mode DCF Dans le monde filaire, les émetteurs sont full-duplex et peuvent donc lire le canal tout en émettant leurs données. Si la valeur qu'ils lisent est différente de celle qu'ils sont en train d'écrire, ils vont en déduire qu'une collision est survenue et donc qu'un autre nœud est également en train d'émettre. Cette technique de détection de collision est connue sous le nom de *Carrier Sense Multiple Access/Collision Detection* (CSMA/CD) dans les réseaux Ethernet et permet, lorsque c'est nécessaire, de retransmettre la trame après un temps d'attente aléatoire.

Dans un réseau sans fil basé sur un support radio, cette technique n'est pas envisageable. En effet les terminaux ne peuvent pas écrire et lire sur le canal en même temps. Ils ne peuvent donc pas détecter les collisions au moment de l'émission des trames. De plus, même si c'était possible, la puissance du signal émis par le nœud est tellement forte qu'il n'arriverait pas à détecter le signal émis par un de ses voisins : du fait de l'affaiblissement du signal en fonction de la distance, le signal émis par son voisin sera complètement masqué par son propre signal. Il considèrera donc qu'il est seul à émettre. Par contre, pour le nœud destination, la puissance des deux signaux peut être assez proche et se traduire en interférences et donc en pertes de trames.

Pour éviter ce genre de collisions, 802.11 a mis en œuvre plusieurs techniques : la première consiste à acquitter systématiquement toutes les trames qui sont reçues dans le cadre d'une communication entre deux nœuds (unicast). Ainsi un nœud émetteur est informé par un *timeout* qu'une trame n'a pas été reçue par le nœud destinataire et peut la retransmettre. Dans le cadre de communications entre un nœud émetteur et plusieurs nœuds destinataires (multicast) ou lors de diffusions à l'ensemble des nœuds (broadcast) ce mécanisme d'acquittement n'est pas utilisé. Il faut donc limiter au maximum les possibilités de collisions. La solution retenue est basée sur un système CSMA comme pour Ethernet mais cette fois le principe ne sera pas de détecter les collisions lorsqu'elles se produisent mais de les éviter en utilisant des temps d'attente aléatoires entre les transmissions. On parle alors de *CSMA/Collision Avoidance* (CSMA/CA).

Dans ce système présenté sur la figure 2.8, lorsqu'un terminal M_1 souhaite émettre vers un autre terminal M_2, il commence par regarder si le canal est occupé. Si le canal est libre, il tire une duré d'attente aléatoire appelée *backoff*. Ce *backoff* se découpe en plusieurs intervalles temporels de taille fixe appelés slots. Comme dans le cas d'HiperLan, le *backoff* est exprimé en nombre de slots pris dans un intervalle compris entre zéro et un entier CW initialisé à CW_{min}. Lorsque ce temps d'attente est écoulé sans qu'aucune transmission n'ait été détectée sur le canal, M_1 peut envoyer sa trame.

Supposons maintenant qu'un troisième nœud M_3 souhaite également émettre des données en même temps que M_1. Lorsque M_3 sonde l'état du canal, il va détecter la transmission de M_1 et différer la sienne. A la fin de la transmission de M_1, il attend un temps incompressible appelé *Distributed Inter-Frame Spacing* (*DIFS*) puis tire une durée de *backoff*. Une fois ce délai d'attente expiré la station M_3 peut commencer sa transmission si personne n'a accédé au canal.

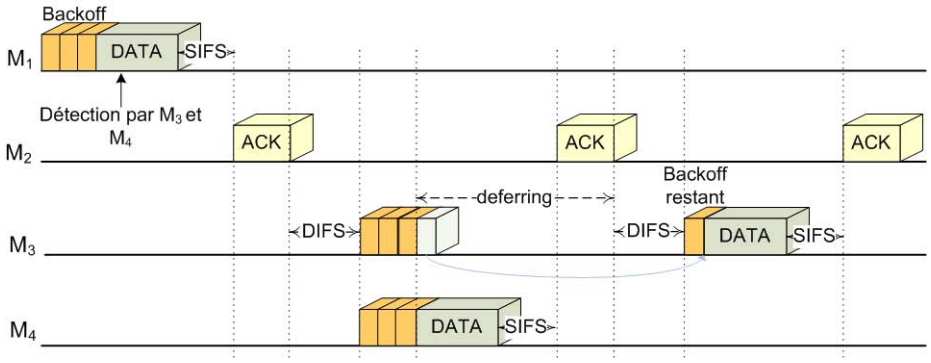


FIG. 2.8: Le *backoff* et le *deferring* utilisés dans 802.11

La transmission est terminée une fois que la station réceptrice a renvoyé un acquittement à la station émettrice. Pour laisser le temps à cette dernière de se mettre en mode d'écoute du canal et pour éviter les problèmes de collisions, le récepteur attend une durée fixe entre la réception de la trame et l'envoi du message d'acquittement. Cette durée est appelée *Short Inter-Frame Spacing* (*SIFS*).

Pendant l'ensemble de la procédure de décompte des slots de *backoff*, le nœud reste à l'écoute de ce qui se passe sur le canal. S'il détecte une autre transmission, il considère qu'il a perdu la procédure d'accès au canal et diffère son émission. De manière à ce que le même nœud ne soit pas toujours pénalisé, un mécanisme appelé *deferring* a été mis en place. Sans ce mécanisme, un nœud pourrait toujours tirer le *backoff* le plus grand et ne jamais avoir le droit de parler. Pour éviter que ceci ne se produise, le nœud qui a tiré le plus grand *backoff* va arrêter de décompter ses slots de *backoff* dès qu'il va détecter une communication. Il va attendre la fin de cette communication puis reprendre le décompte là où il s'était arrêté. Ainsi, la probabilité d'être le nœud avec le moins de slots à décompter va augmenter au fur et à mesure de ses échecs ce qui garantit qu'il finira par avoir accès au canal. Un exemple de *deferring* est présenté sur la figure 2.8.

Au cas où des interférences seraient survenues lors de la transmission de la trame, le récepteur n'acquitte pas le paquet. Le nœud émetteur en déduit donc qu'un autre nœud à émis en même temps et donc que cet autre nœud a tiré le même temps d'aléatoire, le même *backoff*. Pour éviter qu'une telle situation ne se reproduise il va donc doubler sa fenêtre CW pour avoir un espace de tirage plus grand. Ce processus, appelé *backoff exponentiel*, est répété jusqu'à ce qu'il n'y ait plus de pertes ou que CW atteigne une valeur maximum notée CW_{max}. Les valeurs de CW_{min} et CW_{max} sont fixées par la norme. Dès qu'une trame est acquittée, on réinitialise la valeur de CW à CW_{min} pour pouvoir reprendre le processus si de nouvelles pertes surviennent. Si au bout de plusieurs tentatives avec un *backoff* tiré sur CW_{max} le paquet n'est toujours pas acquitté, la connexion est considérée comme rompue. Toujours dans le cas de 802.11b et pour éviter les ruptures de connexions, un mécanisme d'adaptation du débit d'émission a été mis en place pour gérer ces pertes de connexion. Par exemple, si un nœud émet à 11 Mbit/s et ne reçoit pas d'acquittement après trois tentatives avec un *backoff* tiré sur CW_{max}, ce nœud va diviser son débit d'émission par deux pour passer à 5.5 Mbit/s et relancer l'ensemble de la procédure pour émettre sa trame. Ce processus est répété jusqu'à un débit d'émission de 1 Mbit/s en dessous duquel il est impossible de communiquer. Si la communication n'est toujours pas possible à 1 Mbit/s, le nœud émetteur considère la connexion

comme définitivement perdue. Ce mécanisme d'adaptation du débit d'émission est appelé *auto-rate fallback*. L'algorithme utilisé pour mettre en œuvre le mécanisme d'auto-rate fallback ainsi que le nombre de retransmissions à effectuer avant de le déclencher vont différer d'un constructeur à l'autre. Cependant, la plupart se base sur la puissance du signal reçu (ou le rapport signal bruit) pour effectuer cette adaptation du débit.

Description du mode PCF Le mode *DCF* permet un fonctionnement distribué de l'accès au médium, mais pour cela *CSMA/CA* a recours à une durée d'attente aléatoire avant l'émission de chaque paquet. Ce temps d'attente est perdu et diminue d'autant le débit utile au niveau applicatif. Pour résoudre ce problème la *Point Coordination Function (PCF)* a été proposée.

Elle suggère de centraliser la gestion de l'accès au canal dans les stations de base. Ce sont maintenant ces stations de base qui vont devoir distribuer un droit de parole aux différents terminaux mobiles présents dans leur cellule. Grâce à cette centralisation, le *backoff* devient inutile, les risques de collisions devenant nuls. Il n'y a plus de contention pour l'accès au canal : on parlera de *Contention Free Period*. Cependant, il faut que les nœuds n'utilisant pas la *PCF* puissent encore accéder au canal. Pour cela, une période dite de contention (Contention Period) est aménagée. Les deux modes *PCF* et *DCF* sont donc utilisés alternativement par les stations de base pour permettre à tous les nœuds de communiquer. De plus, pour éviter qu'un nœud n'implémentant pas la *DCF* n'accède au canal pendant la *PCF*, il a été nécessaire de définir un nouveau temps inter-frame qui soit plus court qu'un *DIFS*. Ce temps appelé *PCF Inter Frame Spacing (PIFS)* sépare toutes les trames émises pendant la période libre de contention.

Pour que ce mode puisse être utilisé, il faut cependant que les stations de bases et les terminaux mobiles implémentent la *PCF*, ce qui n'est pas le cas dans la majorité des produits vendus dans le commerce.

2.6.2.3 Impact de 802.11 sur les réseaux sans fil

L'impact du partage du canal Les travaux menés par Heusse et al. [34] ont montré que le partage du canal utilisé dans 802.11b avait une influence directe sur le débit maximum théorique de niveau IP (ou débit utile de niveau IP) observé par un utilisateur. Ainsi ces travaux ont permis de mettre en évidence que le débit utile disponible pour un terminal dépend du terminal émettant le plus lentement. Ceci s'explique par la durée de réservation du canal. En effet, si deux nœuds cherchent à émettre simultanément, ils vont devoir périodiquement différer leur émission pour permettre à l'autre nœud de communiquer. Prenons deux nœuds en concurrence M_1 et M_2 avec un débit théorique de 11 Mbit/s pour M_1 et de 1 Mbit/s pour M_2. Lorsque M_1 souhaite accéder au canal il commence par écouter : Si personne n'est en train d'émettre il commence sa transmission à 11 Mbit/s. M_2 va détecter la communication entre M_1 et le point d'accès et va donc différer sa transmission. Lorsque le nœud M_1 a transmis sa trame, le nœud M_2 peut alors émettre sa propre trame à 1 Mbit/s pendant que M_2 se met en attente (ici, nous supposerons que le nœud M_2 a tiré un *backoff* plus petit que M_1). Ce comportement se poursuivra donc ainsi jusqu'à ce que les deux nœuds aient envoyé l'ensemble de leurs données. Le nœud M_1 émet donc bien à 11 Mbit/s mais par contre, ses temps d'attente entre l'envoi de deux trames dépendent de la vitesse d'émission de M_2 soit 1 Mbit/s. Le débit utile observé par l'utilisateur va donc progressivement décroître jusqu'à atteindre un niveau à peine supérieur à 1 Mbit/s. Cette dégradation est bien sur atténuée lorsque le nombre de nœuds émettant à 11 Mbit/s augmente car, dans ce cas là, le nœud à 1 Mbit/s accèdera moins fréquement au médium.

Le problème des terminaux cachées (Hidden Terminal) Le cas dit des stations cachées, illustré sur la figure 2.9, se produit lorsque deux nœuds (M_1 et M_3) n'ayant pas connaissance l'un de l'autre, émettent vers un même troisième nœud (M_2). En effet, comme les nœuds M_1 et M_3 ne sont pas à portée radio l'un de l'autre, ils ne peuvent pas se détecter. Lorsqu'ils souhaitent communiquer avec le nœud M_2, ils commencent par écouter le canal radio. Étant hors de portée l'un de l'autre, ils ne détectent pas d'activité et peuvent donc émettre en même temps à destination de M_2. Cela se traduit généralement par des interférences fortes au niveau de M_2 et donc en pertes ou en corruptions de paquets au niveau IP. Ces interférences ressenties au niveau de M_2 ne peuvent pas être détectées par les deux nœuds émetteurs.

Pour limiter les occurrences de ce problème, 802.11 préconise l'utilisation d'un mécanisme de signalisation appelé *RTS/CTS*. Un exemple de fonctionnement de ce protocole est présenté sur la figure 2.10.

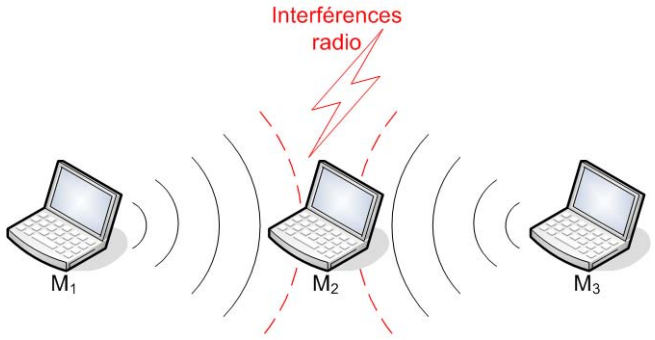


FIG. 2.9: Exemple de terminaux cachés

Préalablement à toute communication, le nœud souhaitant émettre envoie une requête de communication (*RTS : Request To Send*) au nœud destinataire. Le *RTS* contient un champ informant la destination de la durée prévue pour la communication. Ce champ appelé *Network Allocation Vector* (NAV) permet de réserver le canal pour éviter qu'un autre nœud n'essaie d'émettre durant cette période. Le nœud destination décrémente le *NAV* et l'insère dans son message de réponse (*CTS : Clear To Send*). Ce message *CTS* est envoyé au nœud ayant effectué la requête mais également à tous les nœuds voisins du nœud destination. Tout autre nœud souhaitant également communiquer avec la destination va entrer dans une phase d'attente d'une durée égale à la durée spécifiée par le *NAV CTS*.

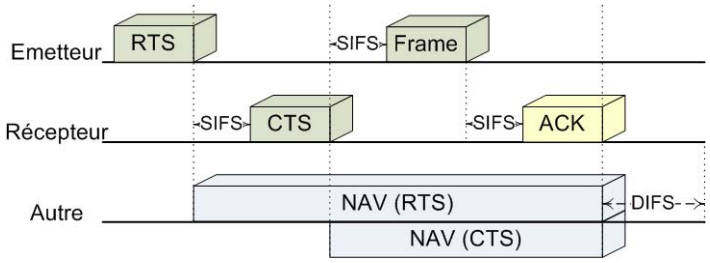


FIG. 2.10: Le mécanisme de RTS/CTS

Ce protocole permet donc de limiter les occurrences de terminaux cachés mais ne les supprime cependant pas complètement. En effet, si nous nous plaçons dans le cas illustré sur la

figure 2.11, il peut arriver que deux paquets *RTS* et *CTS* entrent en collision au niveau du nœud M_3. Supposons que le nœud M_1 veuille communiquer avec le nœud M_2. Il commence par envoyer un message *RTS* pour demander l'accès au médium. Le nœud M_2 ne détecte pas de communications en cours et envoie un message *CTS* à tous ses voisins. Cependant il est possible qu'au même moment le nœud M_4 souhaite également communiquer avec M_3 et envoie un message *RTS*. Le nœud M_3 voit donc arriver simultanément deux signaux : l'un contenant le *CTS* de M_2 et l'autre le *RTS* de M_4. Ces deux signaux vont se brouiller l'un l'autre et être indéchiffrables par M_3 qui ne sera donc pas informé qu'une communication va avoir lieu entre M_1 et M_2. En effet, le nœud M_1 qui a reçu le *CTS* envoyé par M_2, n'est pas informé que des collisions sont survenues au niveau de M_3. Il va donc pouvoir commencer sa transmission. Lorsque le nœud M_4 va de nouveau envoyer son message *RTS* à M_3, celui-ci va penser que le canal est libre dans la mesure où il ne peut pas détecter l'activité entre M_1 et M_2. Par conséquent, M_3 va envoyer un message de *CTS* à tous ses voisins pour les informer qu'une communication va démarrer. Ce message *CTS* a donc de fortes chances d'entrer en collision avec la communication $M_1 \rightarrow M_2$.

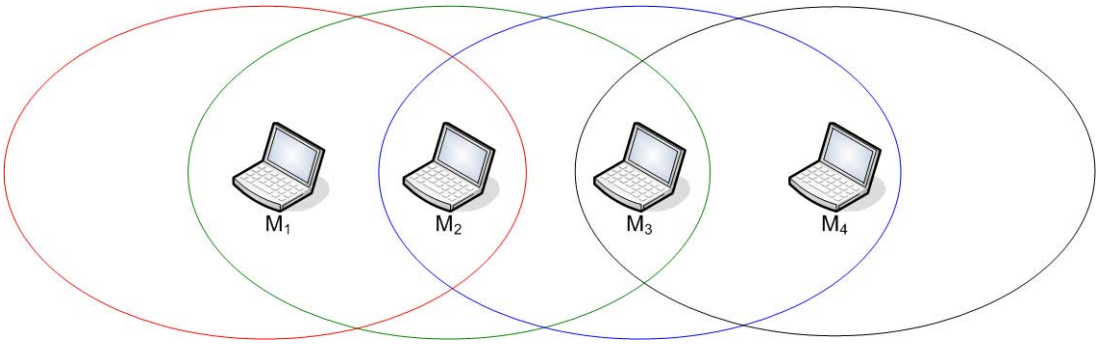


Fig. 2.11: Les terminaux exposés.

Avec la même topologie, un autre problème peut survenir. Supposons que ce soit le nœud M_3 qui souhaite communiquer avec M_4. Si M_3 envoie sa requête *RTS* au moment où M_2 envoie son message *CTS*, M_3 ne peut pas recevoir le *CTS* car il n'est pas en train d'écouter le canal. Le nœud M_4, qui n'est pas au courant de la communication $M_1 \rightarrow M_2$, autorise le nœud M_3 à émettre en lui répondant par un *CTS* si bien que M_3 va commencer à émettre ses données avant que la communication $M_1 \rightarrow M_2$ ne soit finie. Comme les communications ne sont pas directionnelles, le nœud M_2 va donc recevoir le signal émis par M_1 mais également celui émis par M_3 et ceci, même si ce signal ne lui est pas destiné. Les perturbations générées par ces deux signaux au niveau de M_2 vont se traduire par des collisions et donc des pertes de trames. La figure 2.12 illustre cette problématique.

Le problème des stations exposées (Exposed Terminal) La topologie présentée sur la figure 2.11 peut également être source d'un autre problème, celui dit des stations exposées. Ce problème est un résultat de l'utilisation de CSMA/CA couplée à l'utilisation d'antennes omnidirectionnelles et va se traduire par une introduction inutile de délais dans une communication. Par exemple, si le nœud M_2 est en train de communiquer avec le nœud M_1, le nœud M_3 ne peut pas communiquer avec le nœud M_4 car il détecte la communication $M_2 \rightarrow M_1$ (et ceci avec ou sans l'utilisation du protocole RTS/CTS). Cependant, cette communication $M_3 \rightarrow M_4$ pourrait tout à fait avoir lieu. En effet si les nœuds M_2 et M_3 émettent en même

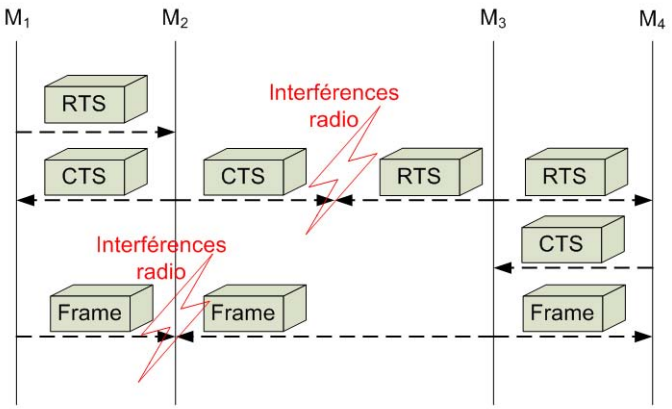


FIG. 2.12: Cas d'échec du mécanisme RTS/CTS.

temps cela n'aura pas d'influence au niveau des nœuds destinataires. M_1 est très éloigné de M_2 et ne recevra donc pas le signal que M_3 émet ou, s'il le reçoit, celui-ci sera couvert par le signal venant de M_2. Les deux communications sont donc possibles car elles ne génèrent pas d'erreurs. Une solution simple pour résoudre ce problème, qui ne nécessite pas de modifications de la méthode d'accès au canal, est d'utiliser des antennes directionnelles. En effet avec ce type d'antenne, le nœud M_3 ne détectera pas la communication $M_2 \rightarrow M_1$ et pourra donc émettre simultanément vers M_4. Cependant, ce type d'antenne est encore onéreux et n'est pas présent dans la plupart des matériels disponibles dans le commerce. De plus, ces antennes résolvent certains problèmes mais en génèrent d'autres comme la difficulté de devoir localiser les nœuds vers lesquels on souhaite émettre puis orienter convenablement l'antenne.

Un autre problème peut survenir dans cette situation. Le nœud M_3 peut détecter la communication $M_2 \rightarrow M_1$ mais ne pas voir l'acquittement envoyé par M_1. Il peut donc émettre au moment où l'acquittement est reçu pas M_2 et provoquer des interférences au niveau de ce nœud. Pour résoudre ce problème, deux mécanismes ont été proposés. Le premier est le fait des industriels et repose sur l'utilisation de deux zones de communications centrées autour du nœud. La première est appelée zone de communication et la seconde, d'une taille supérieure, est appelée zone de détection de porteuse. Ainsi, les nœuds ont une meilleure connaissance du réseau même s'ils ne peuvent pas communiquer avec l'ensemble des nœuds qu'ils détectent. Grâce à ce système, le nœud M_3 va détecter qu'il y a une communication entre les nœuds M_1 et M_2 lors de l'envoi de l'acquittement. Ce message n'est pas compréhensible par M_3, donc la norme introduit un nouveau temps d'attente appelé *Extended Inter Frame Spacing* (*EIFS*) qui est appliqué chaque fois qu'un message n'est pas compris. Ce délai est calculé de telle sorte qu'il soit supérieur à une durée *SIFS* plus la durée de transmission d'un acquittement. Ce délai doit en effet être plus long que *SIFS+ACK* pour éviter les collisions dans le cas d'un échange *RTS/CTS*. Ces explications ainsi que des figures complémentaires sont détaillés dans [75].

2.6.2.4 Le débit utile de 802.11b

Dans la suite de ce document, nous nous interesserons principalement à la technologie IEEE 802.11b. Il est donc nécessaire de clarifier certains points et notamment ceux concernant le débit. En effet le débit annoncé pour une communication (11, 5.5, 2, 1 Mbit/s) est le débit

au niveau physique, c'est-à-dire un débit supérieur au débit utile de niveau applicatif.

Lors de l'émission, des en-têtes de niveau MAC et de niveau physique vont être ajoutés aux données venant du niveau applicatif. Ces en-têtes ont la particularité en 802.11b de ne pas être transmis à la même vitesse que le reste de la trame. L'en-tête physique, est transmis à une vitesse de 1 Mbit/s pour assurer qu'il sera bien reçu par n'importe quel nœud utilisant une carte compatible avec la version d'origine de 802.11. L'en-tête de niveau MAC, jugé moins critique, peut être transmis à la même vitesse que les données : c'est à dire jusqu'à 54 Mbit/s pour 802.11g.

Les paquets de contrôle *ACK, RTS/CTS* sont échangés à des vitesse qui doivent pouvoir être supportées par l'ensemble des nœuds compatibles. Suivant les constructeurs, cette vitesse peut donc varier. La plupart ont retenu une vitesse de 2 Mbit/s pour ces paquets de contrôle en la diminuant à 1 Mbit/s lorsque les taux d'erreurs deviennent trop importants. Cependant, certains ont choisi de les transmettre à 11 Mbit/s pour assurer un meilleur débit au niveau applicatif. De la même manière que les paquets de contrôle, les paquets émis en mode diffusion sont envoyés à une vitesse qui doit être supportée par l'ensemble des nœuds. C'est donc le plus souvent la vitesse de 2 Mbit/s qui est retenue pour la diffusion. Il faut cependant bien noter que les paquets de contrôle ainsi que les paquets diffusés sont soumis à l'encapsulation de niveau physique et donc que leur en-tête physique est transmis à 1 Mbit/s.

Paramètre	Durée (μs)	Taille
SIFS	10	N/A
DIFS	50	N/A
EIFS	364	N/A
backoff	20	N/A
CW_{min}	620	31 slots
CW_{max}	20460	1023 slots
En-tête physique long	192	192 bits
En-tête MAC	dépendant du débit	272 bits
RTS (sans l'en-tête physique)	dépendant du débit	160 bits
CTS (sans l'en-tête physique)	dépendant du débit	112 bits
ACK (sans l'en-tête physique)	dépendant du débit	112 bits

Tab. 2.1: Valeurs retenues dans la norme 802.11b

En se basant sur ces particularités ainsi que sur les valeurs définies dans la norme pour les différents temps d'attente, il est possible de déterminer un débit maximum théorique au niveau applicatif en l'absence de faute. Les différentes valeurs définies par la norme 802.11b sont présentées sur le tableau 2.1. Ainsi, en supposant qu'il n'y a qu'un *backoff* d'une valeur égale à la moyenne entre 0 et CW_{min}, que les paquets de contrôle sont transmis à une vitesse de 2 Mbit/s et que les en-têtes MAC sont transmis à la même vitesse que les données, il est possible de déterminer le délai que va subir un paquet pour atteindre sa destination.

Prenons l'exemple d'un paquet de 1500 octets émis avec un débit théorique de 11 Mbit/s. Les délais qu'il va subir seront :

- Un *DIFS* pour accéder au réseau : 50 μs
- Un backoff moyen : 310 μs
- Durée d'émission du paquet : 272 bits d'en-tête MAC et 1500 octets de données à 11 Mbit/s soit 1116 μs plus l'en-tête physique de 192 bits émis à 1 Mbit/s soit 192 μs ce qui donne :

$1116 + 192 = 1308\,\mu$s
- Un *SIFS* avant de recevoir l'acquittement : $10\,\mu$s
- Une durée d'émission du paquet ACK avec son en-tête physique : $10 + 192 = 202\,\mu$s

Tout ceci nous donne donc un temps total de transmission de $1880\,\mu$s ce qui correspond à un débit réel de niveau IP de 6.38 Mbit/s pour un paquet IP de 1500 octets en-tête compris. De même, on obtient un délai inter-paquets de $572\,\mu$s.

2.7 Conclusion

Les réseaux locaux sans fil présentent des différences majeures avec les réseaux locaux filaires. L'utilisation de transmissions radios génère en effet des contraintes inédites. La première est l'atténuation du signal radio en fonction de la distance qui fait que lorsqu'un nœud émet, il ne lui est pas possible de détecter des interférences situées au niveau du nœud destination. Toujours à cause de cette atténuation, lorsqu'un nœud est en train d'émettre, le signal qu'il envoie va être beaucoup plus fort que tous les signaux qu'il pourrait recevoir ce qui fait qu'un nœud ne peut pas émettre et écouter le canal simultanément (s'il le faisait il n'entendrait que sa propre communication). En plus de l'atténuation en fonction de la distance, le signal radio va être soumis aux contraintes de l'environnement et aux effets de la propagation. De même, à cause de la mobilité, la topologie du réseau va évoluer au cours du temps ce qui va poser des problèmes de routage, voire même dans certains cas pour les réseaux en mode Ad-Hoc, des problèmes de partition du réseau.

Plusieurs technologies ont été proposées pour résoudre ces différents problèmes, la plus célèbre d'entre elle étant la norme IEEE 802.11 qui a connu ces dernières années un grand succès commercial. Pour résoudre les problèmes d'interférences au niveau de la destination et réduire les risques de collisions, ces solutions proposent l'utilisation d'une écoute préalable du canal ainsi que de temps d'attente entre chaque transmission. Ces méthodes d'accès (comme CSMA/CA pour IEEE 802.11) diffèrent de celles que l'on trouve dans un réseau filaire comme Ethernet par exemple. Il n'est donc pas possible de transposer directement les protocoles et applications développés dans le cadre des réseaux filaires dans les réseaux locaux sans fil. Il est nécessaire de développer de nouveaux protocoles permettant de répondre aux besoins de Qualité de Service des réseaux sans fil et de pouvoir les évaluer avant de les déployer. Comme dans les réseaux filaires, le test et l'évaluation des protocoles et applications avant leur déploiement à grande échelle sont des contraintes fondamentales que nous allons maintenant étudier.

3 Les différentes approches de test

3.1 Introduction

Comme nous l'avons vu dans le chapitre précédent, les réseaux sans fils 802.11 sont relativement complexes et présentent de nombreuses différences avec les réseaux filaires. Les protocoles et applications développés dans le cadre des réseaux filaires ne sont donc pas directement transposables dans les réseaux 802.11 sans dégradation de la Qualité de Service (QdS) si bien que de nombreuses propositions sont faites notamment au niveau transport et applicatif pour améliorer la QdS de ces réseaux sans fil. Or, le test et l'évaluation de ces nouvelles propositions sont complexes car ils regroupent plusieurs aspects :

- les mesures de performances
- La comparaison avec les solutions existantes
- L'évaluation des performances dans des conditions réseaux particulières (grande mobilité, cas de terminaux cachés, etc.)

Pour les réaliser, il est possible d'utiliser plusieurs méthodes : le test en environnement réel, la simulation et l'émulation.

3.1.1 Le test en environnement réel

Une première approche pour tester un protocole ou une application consiste à mettre réellement en œuvre le réseau sans fil sur lequel on doit le/la tester. Cette solution est a priori la meilleure dans la mesure où l'on utilise vraiment le système opérationnel ainsi que l'implémentation finale du protocole. Cependant, cette solution peut être assez difficile à mettre en œuvre et peut être très coûteuse dans certains cas. Si nous prenons l'exemple du satellite, il ne paraît pas évident d'avoir un satellite en libre accès pour pouvoir tester un nouveau protocole. De même, dans le cas des réseaux sans fil, il est assez difficile de trouver suffisamment de personnes pour réussir à déployer un réseau ad-hoc mobile de taille importante.

Un autre point négatif de cette solution est l'impossibilité d'évaluer le protocole aux conditions limites de propagation (pendant une tempête par exemple) ou de mobilité (par exemple avec des vitesses de déplacement très importantes). De même, il est très difficile de réussir à mettre en œuvre des cas rares (tous les nœuds qui émettent simultanément par exemple).

Enfin, cette solution n'autorise pas l'exacte reproduction des conditions d'une expérience à l'autre, ce qui peut être très gênant pour réussir à mettre au point son protocole. En effet, un développeur de protocole a souvent besoin de jouer sur un ou deux paramètres de son protocole et il lui est donc nécessaire de pouvoir reproduire fidèlement une expérience pour identifier le paramétrage optimal. Cette reproductibilité est également nécessaire pour pouvoir comparer le protocole en développement avec les protocoles existants. Or, pour que ces comparaisons soient fiables, il faut que les conditions soient exactement les mêmes ce qui ne peut pas être assuré dans un test en environnement réel puisque les conditions de propagation du signal peuvent changer très fortement entre deux tests.

3.1.2 La simulation

La simulation est une solution souvent plébiscitée par la communauté scientifique. Elle possède en effet l'avantage de reposer sur des modèles validés ou au moins reconnus. Il existe plusieurs simulateurs à événements discrets permettant de mettre en œuvre un réseau sans fil tels que NS-2 [8], Opnet [61], QualNet [66] et GloMoSim [81]. Un simulateur à événements discrets se caractérise par le fait que les changements d'états dans le réseau simulé (événements) se produisent à des instants (sans durée) répartis de manière discrète sur l'axe des temps. Classiquement, une simulation consiste à reproduire le comportement du système complet dans un environnement synthétique où le temps est simulé par une horloge à événements discrets. La simulation demande une modélisation complète, allant des applications au réseau physique, ce qui peut se révéler à la fois complexe et coûteux en terme de temps d'exécution et d'utilisation mémoire dans le cas de réseaux de taille importante (100 nœuds et plus). Ainsi, tous les simulateurs que nous avons cités précédemment simulent complètement les piles protocolaires ainsi que le trafic. C'est d'ailleurs en faisant circuler des paquets simulés à l'intérieur de ces piles protocolaires que ces simulateurs fournissent des résultats à l'utilisateur. Plus les modèles utilisés sont précis et plus les résultats obtenus seront pertinents. Malheureusement, le temps nécessaire pour définir ces modèles peut être relativement long.

Il arrive également que ce processus de modélisation soit quasiment impossible sans accepter de faire des simplifications, par exemple lorsque l'on souhaite comparer son application à une application disponible dans le commerce. Le fait de ne pas avoir une connaissance complète et exhaustive des choix faits lors de la conception d'une application commerciale, la rend en général assez difficile à modéliser. Ces simplifications vont tout de même permettre d'avoir une vision du comportement de son protocole/application et de faire les premières modifications pour l'améliorer.

Ce n'est qu'une fois que les résultats de cette phase de simulation atteignent un assez bon niveau que le développement du système réel peut commencer. Sur la base de l'algorithme évalué par simulation, le processus complet de développement logiciel – analyse, conception, mise en œuvre et tests – est alors nécessaire pour la réalisation de la maquette. Ainsi, le processus de simulation oblige l'utilisateur à faire un double développement de son système et ceci sans lui assurer la cohérence du modèle de simulation et de l'implémentation.

3.1.3 L'émulation

L'émulation peut être vue comme un compromis entre les deux solutions précédentes dans la mesure où elle va permettre de tester l'*implémentation réelle* du protocole/application sur un *réseau simulé* en temps réel. Le but de l'émulation sera donc de reproduire le comportement d'un réseau jusqu'au niveau d'une couche donnée en simulant en temps réel les effets des couches sous-jacentes du réseau cible. On dira alors que l'émulateur rend un service d'émulation à ce niveau (e.g. service de niveau réseau pour une simulation en temps réel des couches réseau, liaison de donnée et physique). L'émulation permet d'avoir un contrôle équivalent à celui de la simulation sur les paramètres du réseau mais permet en plus d'utiliser des implémentations réelles à tester. Ainsi, elle permet de tester un protocole aux conditions limites, ce qui est très difficile à faire dans un réseau réel. On parlera alors d'évaluation d'un protocole sur des cas rares. De plus, comme l'environnement simulé est entièrement contrôlable, il est possible de comparer ce protocole avec d'autres protocoles existants en reproduisant plusieurs fois de suite la même expérience. Le fait de pouvoir reproduire précisément une expérience va également

permettre, comme pour la simulation, de bien paramétrer son protocole.

L'émulation pose cependant certains problèmes. Le plus évident vient du fait que le système sur lequel on va émuler un autre réseau doit être sur-dimensionné par rapport au réseau à émuler. Ainsi, il ne sera jamais possible d'émuler un réseau de cœur sur un réseau de type LAN Ethernet. Le second problème vient des contraintes temps réel à respecter lors de la simulation des couches sous-jacentes. En effet, pour pourvoir tenir ces contraintes, il ne faut pas que les modèles décrivant le fonctionnement des couches basses du réseau à émuler soient trop complexes. Si les modèles prennent trop de temps de calcul pour fournir les résultats, les contraintes temps réel ne sont plus respectées et donc l'émulation rendue n'est pas réaliste. A contrario si les modèles sont trop simplistes, les contraintes temps réel seront respectées mais là non plus le réalisme ne sera pas au rendez-vous. La principale difficulté est de trouver un équilibre entre complexité et réalisme des modèles.

Cette solution s'inscrit comme une étape entre la simulation et le test en environnement réel dans le cycle de développement d'un protocole/application.

3.2 Emulation de réseaux

3.2.1 Les différents niveaux d'émulation

Lorsque l'on parle d'émulation, la première étape est de définir ce que l'on souhaite pouvoir évaluer sur la plateforme dont on dispose. En effet, suivant que l'on souhaite évaluer une application ou un protocole de routage, le niveau d'émulation, et donc le niveau d'abstraction qui va être nécessaire, ne sera pas le même. Ainsi, pour évaluer un protocole de routage il faut reproduire en temps réel le comportement des couches liaisons et physiques. L'émulateur doit donc rendre un service de niveau liaison. Pour évaluer une application, il est possible pour l'émulateur de rendre là encore un service de niveau liaison. Cependant, il faut disposer des implémentations réelles des protocoles de niveau transport et de niveau réseau (routage) qui sont utilisés dans le réseau à émuler ce qui n'est pas toujours le cas. Quand les implémentations réelles des protocoles de niveau réseau ne sont pas disponibles, il est préférable d'utiliser un émulateur rendant un service de niveau réseau, c'est-à-dire un émulateur qui va reproduire en temps réel le comportement des couches jusqu'au niveau de la couche réseau. La figure 3.1 présente cette notion de niveau d'émulation et de service rendu par un émulateur.

3.2.1.1 Émulation de niveau 1

L'émulation de niveau 1 aussi connue sous le nom d'émulation de niveau physique (qui va rendre un service de niveau physique), propose d'utiliser les piles protocolaires réelles à partir du niveau liaison de donnée et de ne jouer que sur la couche de niveau physique pour émuler le comportement d'un réseau sans fil. Cette solution permet donc de pouvoir tester des protocoles de niveau liaison ou supérieur. Cependant, comme les niveaux physiques radio et Ethernet sont très différents, il est extrêmement difficile de reproduire fidèlement le comportement d'un canal physique radio sur un réseau de type LAN Ethernet. En effet, pour émuler un réseau sans fil 802.11, la couche MAC utilisée dans ce genre de plate-forme est une implémentation réelle qui va s'appuyer sur des informations fournies par la couche de niveau physique émulée. Celle-ci doit pouvoir fournir à la couche MAC les messages de contrôles qui sont nécessaires à son bon fonctionnement. Il est donc très difficile de rendre une émulation réaliste ce qui explique qu'il

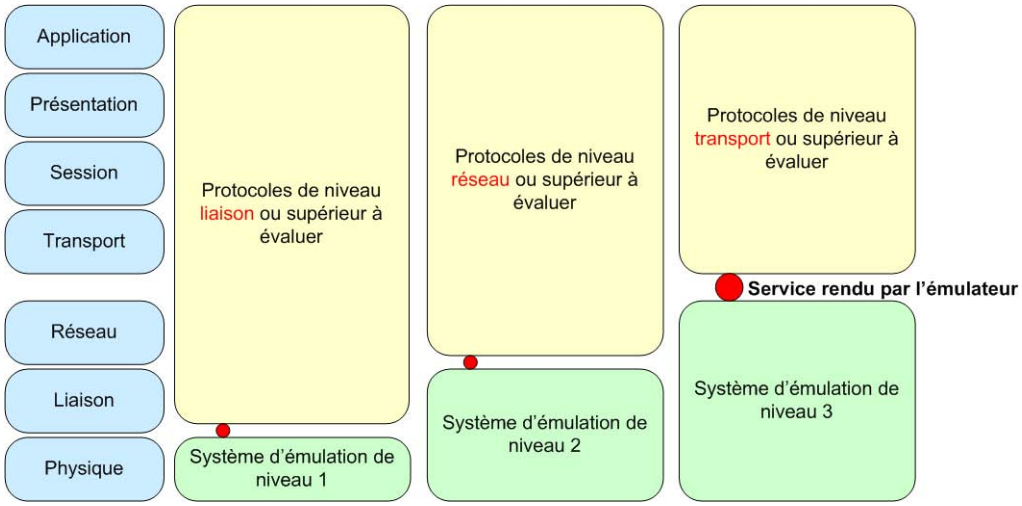


FIG. 3.1: Principe de l'émulation

existe très peu de plate-forme d'émulation de niveau 1 utilisant Ethernet à l'heure actuelle. Un exemple sera donné dans la section 3.3 à travers la plate-forme d'émulation JEMU [25].

3.2.1.2 Émulation de niveau 2

L'émulation de niveau 2, également appelée émulation de niveau liaison, va permettre d'évaluer le comportement de protocoles de routage, de transport ou d'évaluer des applications réparties. Pour cela, il va être nécessaire de simuler en temps réel le service rendu au niveau de la couche liaison de donnée. Ainsi, pour émuler un réseau sans fil 802.11 sur une plate-forme d'émulation basée sur un LAN Ethernet, la principale difficulté va consister à reproduire fidèlement le comportement d'une couche MAC 802.11 réelle. Cette couche MAC 802.11 doit pouvoir fournir aux couches supérieures les informations qu'elles demandent et ceci de manière transparente. La plupart des solutions d'émulation de niveau 2 se présentent sous forme distribuée, chaque élément terminal embarquant une couche MAC virtuelle reproduisant le comportement du réseau cible. Divers exemples de plate-forme sont donnés dans la section 3.3.

3.2.1.3 Émulation de niveau 3 (ou émulation de niveau IP)

Une plate-forme d'émulation de niveau 3 reproduit le comportement d'un réseau cible en simulant en temps réel les effets des couches réseau, liaison et physique et permet l'évaluation de protocoles de niveau transport et applicatif. Pour simplifier, nous dirons qu'elle rend un service de niveau réseau. Avec cette technique d'émulation, il n'y a que peu de paramètres à manipuler pour reproduire des conditions réseaux spécifiques nécessaires à une communication de bout en bout. Ces paramètres sont les délais et les pertes de paquets (qui peuvent être considérées comme des délais infinis) et sont a priori les seuls nécessaires pour caractériser la dynamique du trafic au niveau IP. Cependant, pour simplifier la modélisation du comportement du service rendu par l'émulateur, la bande passante disponible au niveau IP est le plus souvent retenue comme troisième paramètre. Pour désigner ces trois paramètres, nous parlerons de paramètres de QdS de niveau IP.

Il existe des outils appelés conditionneurs de trafic ou *traffic shaper* qui permettent de manipuler ces trois paramètres et de contraindre le trafic sur un réseau d'expérimentation. La plupart des solutions d'émulation de niveau 3 reposent sur l'utilisation de ces conditionneurs de trafic et diffèrent principalement sur la technique de configuration de ces conditionneurs de trafic.

Dans le cadre de nos travaux, nous nous sommes intéressés à cette émulation de niveau 3 pour pouvoir évaluer des protocoles de niveau transport ou applicatif basés sur IP. L'émulation de niveau 3 est également appelée émulation de niveau réseau ou émulation de niveau IP. Nous utiliserons indifféremment ces trois termes dans la suite de ce document.

3.2.2 Les différents mécanismes d'émulation

La figure 3.2 résume un processus d'émulation utilisant un conditionneur de trafic. Lors d'une expérience, le trafic (ou paquets IP) circulant sur le réseau d'expérimentation est intercepté par le conditionneur de trafic puis il est *contraint* en fonction d'un *modèle d'émulation*.

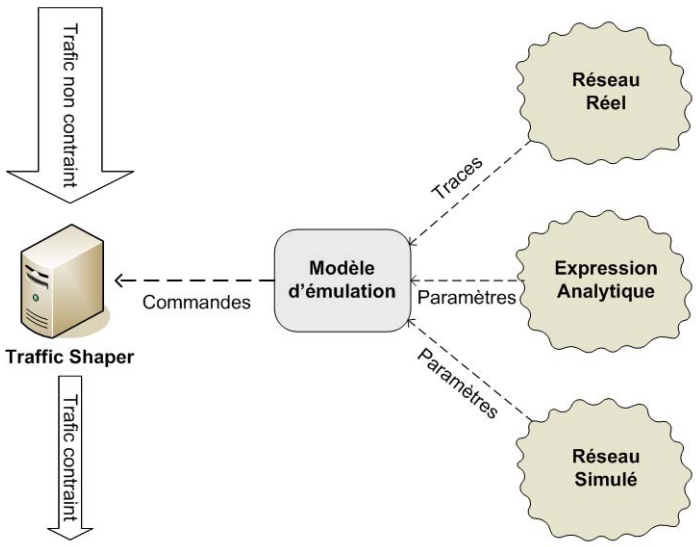


FIG. 3.2: Processus d'émulation

Si nous prenons l'exemple illustré sur la figure 3.3 pour émuler une communication entre deux terminaux sans fil inter-connectés par un point d'accès, il est possible d'utiliser un conditionneur de trafic. Le terminal 1 du réseau physique d'expérimentation va jouer le rôle du nœud 1 et le terminal 2 celui du nœud 2. Si nous supposons que la communication à lieu du nœud 1 vers le nœud 2 dans le réseau émulé avec des taux de transmission radio de 11 Mbps, sur la plate-forme d'expérimentation, le trafic va être envoyé à partir du terminal 1 vers le conditionneur de trafic avec un taux de transmission de 100 Mbps. Une fois dans le conditionneur de trafic, le trafic reçu va être ralenti de manière à être envoyé au terminal 2 avec un débit correspondant à un taux de transmission radio de 11 Mbps. Cette action de ralentir le trafic est une forme de contrainte, les autres formes étant d'introduire des délais ou des pertes dans la communication.

Le modèle d'émulation peut être obtenu à partir de traces capturées sur un réseau réel, à partir de modèles empiriques ou théoriques implémentés dans un simulateur à événements

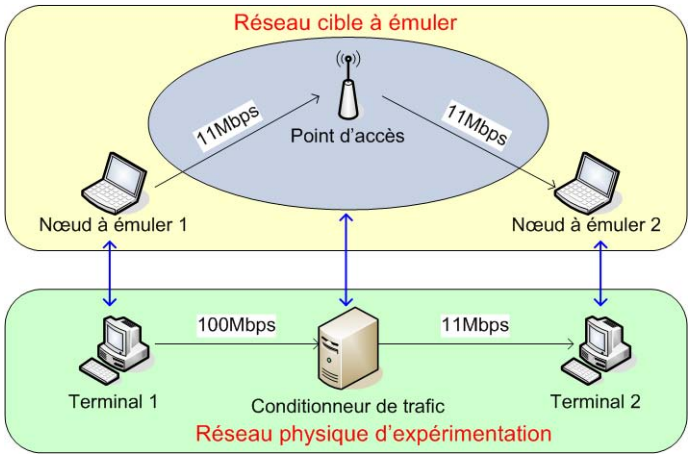


FIG. 3.3: Exemple d'utilisation d'un conditionneur de trafic

discrets, ou encore à partir d'instructions fournies par l'utilisateur. De ce modèle sont issus les paramétrages éventuellement dynamiques du conditionneur de trafic en termes de bandes passantes, de délais et de pertes. Ces différents paramètres sont ensuite traduits en *commandes d'émulation* qui vont permettre de faire changer les conditions reproduites par le conditionneur de trafic.

3.2.2.1 Utilisation d'un conditionneur de trafic (traffic shapper)

Il existe de nombreux conditionneurs de trafic dans la littérature, parmi lesquels nous pouvons signaler ONE [4], Delayline [38], Dummynet [70], NISTNet [10].

ONE (the Ohio Network Emulator) [4] ONE a été développé à l'université de l'Ohio en partenariat avec la NASA. Il fut conçu pour pouvoir émuler fidèlement des liens satellites et se présente comme un routeur entre deux réseaux. ONE modélise les routeurs et le réseau séparant les deux réseaux de test, en reproduisant les pertes de paquets dues aux congestions. Pour cela, il se base sur les paramètres fournis par l'utilisateur. Les paquets qui transitent par l'émulateur sont soumis à un délai calculé en fonction de leurs tailles et des paramètres spécifiés par l'utilisateur. Tout le fonctionnement de ONE repose sur la caractérisation de ce délai et pour cela il va tenir compte de trois aspects : le délai de transmission qui correspond à la bande passante du lien, le délai de file d'attente qui caractérise le délai que le paquet va subir en traversant les routeurs du réseau émulé et le délai de propagation qui correspond au temps que met un paquet pour traverser le canal de communication.

Cet émulateur peut être vu comme un conditionneur de trafic dans la mesure où il ne fait que contraindre les paquets en fonction de paramètres spécifiés par l'utilisateur. Cependant, la principale limitation de cet émulateur vient du fait qu'il ne permet d'inter-connecter que deux réseaux. Pour pouvoir émuler un réseau de taille importante, il faut donc multiplier le nombre de routeurs ONE ce qui ne permet pas de passer à l'échelle.

Delayline [38] Delayline est un émulateur de réseau métropolitain, développé par David Ingham et Graham Parrington de l'université de Newcastle. Il a été développé dans le but de

tester des applications réparties sur un réseau WAN. Delayline se présente comme une plate-forme d'émulation de niveau 3 capable de changer les conditions du réseau sous-jacent sur lequel sont testées les applications réparties. Il permet de reproduire le partitionnement d'un réseau et le comportement des éléments d'interconnexions des sous-réseaux ainsi partitionnés. Ceci permet d'émuler des pannes de routeurs et des pannes de passerelles entre deux sous-réseaux.

Cependant, pour pouvoir intercepter les paquets, Delayline nécessite une recompilation des applications à tester. En effet, Delayline utilise des *sockets* pour intercepter des paquets et doit donc être inclus sous forme de librairie dans le code des applications à tester. Grâce à ces *sockets*, il peut intercepter et manipuler les paquets en introduisant des délais mais également des pertes qui vont reproduire les congestions et les partitionnements du réseau. Cette nécessité de recompiler l'application à évaluer élimine toute possibilité de comparaison entre une application en cours de développement et une application commerciale dont les sources ne sont pas disponibles. Ce besoin de disposer du code source de l'application le rend donc difficilement utilisable. Enfin, il faut noter que les *sockets* utilisées nécessitent de tester l'application sur un système d'exploitation disposant de *sockets Berkeley,* ce qui élimine de fait toutes les applications développées sous Windows.

Dummynet [70] Dummynet est un émulateur développé par Luigi Rizzo et intégré au noyau FreeBSD. Il est contrôlable par l'intermédiaire du pare-feu du système (i.e. *ipfw*). Le pare-feu intercepte les paquets IP selon les règles définies par l'utilisateur, puis Dummynet les traite en appliquant la QdS spécifiée en termes de bande passante, de délai et de perte. Les règles du pare-feu permettent de choisir précisément quel paquet doit être intercepté en tenant compte de paramètres tels que l'adresse de l'émetteur, du récepteur, les protocoles, le numéro de port utilisé, etc. Les traitements appliqués aux paquets filtrés sont ensuite réalisés en passant les paquets d'une file d'attente à une autre. Un exemple est illustré par la figure 3.4.

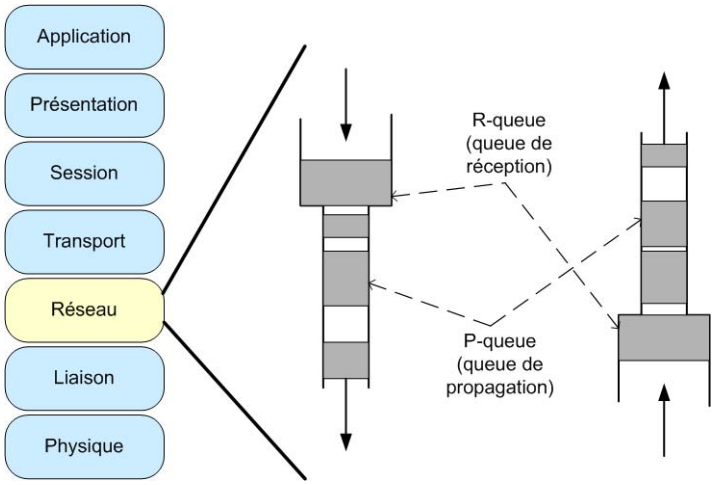


FIG. 3.4: Fonctionnement interne de Dummynet [70]

Dummynet est inséré dans la pile IP et plus particulièrement dans la couche réseau. Il peut donc être utilisé pour évaluer des protocoles de transport et des applications basés sur IP. Détaillons maintenant le fonctionnement de Dummynet et en particulier la méthode utilisée pour contraindre les paquets en termes de bande passante, de délais et de pertes.

Les paquets reçus sont tout d'abord stockés dans la file d'attente R (R-queue) qui reproduit la taille que l'on peut trouver en entrée d'une interface réseau. Les paquets sont ensuite extraits de cette file d'attente et envoyés à la file d'attente P (P-queue) à une vitesse correspondant au débit choisi par l'utilisateur. Les paquets restent ensuite pendant un temps au délai fixé par l'utilisateur dans cette file d'attente P puis sont émis vers le destinataire. Les pertes sont également appliquées au moment de ce passage de la file d'attente R à la file d'attente P en fonction du taux de pertes de paquet (PLR) spécifié par l'utilisateur. Les pertes sont probabilistes, c'est à dire qu'un tirage aléatoire est effectué sur chaque paquet puis comparé avec le taux spécifié par l'utilisateur. Si la valeur tirée est inférieure au taux, le paquet est perdu, sinon il est passé à la file d'attente P pour subir la contrainte de délai.

Plusieurs règles du pare-feu *ipfw* peuvent être enchaînées ; il est ainsi possible pour un paquet vérifiant plusieurs règles de subir plusieurs traitements avant de sortir de l'émulateur. Ceci permet d'émuler une communication de bout en bout traversant plusieurs types de réseaux (par exemple un réseau satellite puis un réseau d'accès ADSL). Enfin, Dummynet peut fonctionner de manière transparente vis-à-vis des stations d'extrémité en agissant comme un routeur ou comme un pont transparent et peut être étendu par l'utilisateur au niveau applicatif en utilisant les *divert sockets* de FreeBSD.

NISTNet [10] NISTNet, qui fonctionne sous le système Linux, reprend et étend certaines des fonctionnalités de Dummynet. Il permet notamment de réordonner, de dupliquer les paquets et de spécifier deux types de pertes, les pertes dues aux congestions, et les pertes aléatoires. En revanche, contrairement à Dummynet, il ne permet pas d'enchaîner plusieurs règles d'émulation les unes à la suite des autres. Plus récemment, en se basant sur le code de NISTNet, le projet NETEM [30] a intégré des fonctions d'émulation (délais, pertes, etc.) à l'outil de contrôle de trafic *tc* proposé dans les dernières versions de Linux qui permet notamment d'appliquer à un trafic des politiques de gestion de files d'attente.

Le point commun de tous ces émulateurs de première génération est qu'ils laissent à l'utilisateur le soin de fournir les caractéristiques à reproduire. Ils se contentent de reproduire les conditions en se basant sur différents paramètres plus ou moins précis, mais aucun ne se charge de déterminer ces conditions. Pour résoudre ce problème plusieurs solutions ont été envisagées : l'utilisation de traces, la simulation temps réel et des solutions permettant de limiter les calculs à faire en temps réel.

3.2.2.2 Émulation par rejeu de traces

Les plates-formes d'émulation basées sur le rejeu de traces comme celle présentée par Noble et al. dans [60] reproduisent le comportement d'un réseau cible sur un réseau d'expérimentation en s'appuyant sur des traces capturées au préalable. Les traces doivent être interprétées puis traduites pour être exploitables par le conditionneur de trafic. La phase de traduction, aussi appelée phase de distillation, extrait de traces réelles les paramètres clés qui vont permettre au conditionneur de trafic de reproduire les conditions du réseau cible sur le réseau d'expérimentation. La difficulté d'utilisation de ce genre d'émulateur vient en particulier de l'obtention des traces et de la phase de traduction. Il ne faut surtout pas perdre d'informations importantes durant cette traduction. D'autre part, cette approche est assez rigide dans la mesure où les traces ne reflètent que des conditions spécifiques issues d'une campagne de mesure donnée. L'expérimentateur ne peut pas choisir les conditions lui-même. Il est limité par les

traces qui sont mises à sa disposition et ne peut faire varier les expériences à sa guise. En particulier, le test des conditions limites d'un réseau avec cette solution s'avère très difficile.

3.2.2.3 Émulation par simulation temps réel

Une autre possibilité consiste à utiliser un simulateur à événements discrets complet pour reproduire en temps réel les conditions d'émulation. Le simulateur doit alors pouvoir se coupler aux protocoles et applications réels à tester et utiliser le trafic réel généré pour alimenter son moteur de simulation. Il sera vu comme une boîte noire pour les éléments terminaux sur lesquels seront déployés les protocoles et applications à tester.

NSE [24] L'un des émulateurs utilisant cette approche n'est autre que NS-2 dans son mode émulation, appelé NSE. Contrairement au mode simulation, il fonctionne ici en temps réel et fait également office de conditionneur de trafic. NSE intercepte le trafic circulant sur la plate-forme d'émulation (généré par l'application à tester ou par un générateur de trafic dans le cas de l'évaluation d'un protocol) et le fait passer dans son moteur d'émulation pour déterminer les contraintes à appliquer à ce trafic. Dans son mode émulation, NS-2 peut fonctionner de deux manières différentes : le mode opaque dans lequel les paquets réels ne sont pas modifiés mais sont soumis aux contraintes du réseau simulé et le mode protocole dans lequel les paquets sont interprétés, transformés en paquets simulés qui serviront à déterminer les conditions dans le réseaux puis re-formés en sortie du simulateur. Cet aspect émulation n'a pas été développé spécifiquement pour les réseaux sans fil, mais une extension présentée dans [45] montre qu'il peut également rendre ce service. Elle tire parti des améliorations apportées à NS-2 dans l'environnement sans fil, pour émuler des réseaux Ad-Hoc.

Le problème de ces solutions basées sur une simulation centralisée vient du fait que le simulateur (que nous appellerons aussi nœud central) doit traiter en temps réel un très grand nombre de paramètres (effets de l'environnement, de la mobilité sur les communications etc...).Quand ce nombre devient trop important, le simulateur n'arrive plus à remplir les conditions de fonctionnement en temps réel ce qui dégrade énormément le réalisme de l'émulation. Le nombre d'événements va en fait dépendre du nombre de nœuds intervenant dans l'expérience et de la complexité des modèles qui sont utilisés par le simulateur. Dans le cas des réseaux de mobiles par exemple, ces modèles ne se limitent pas aux couches de communications mais doivent également permettre de décrire les problèmes de mobilité, de propagation du signal radio dans différents milieux, d'énergie, etc. Certains de ces modèles additionnels peuvent utiliser des techniques relativement lourdes à mettre en œuvre en temps réel comme la génération de nombres aléatoires suivant des distributions non standard (Rayleigh, Rice, etc.) ou le lancer de rayons pour [72].

Pour gérer ces contraintes de temps réel, une solution consiste à distribuer la simulation sur plusieurs serveurs pour pouvoir paralléliser les calculs.

NCTUns [77, 78] NCTUns est un simulateur de réseau développé à l'université d'Harvard qui propose d'utiliser un tel fonctionnement pour faire de l'émulation. Ce mode de fonctionnement distribué n'a cependant pas été évalué pour émuler des réseaux complexes utilisant plusieurs trafics réels ou utilisant des modèles de mobilité complexes dans le cas des réseaux

sans fil. En effet, bien que le fait de distribuer la simulation puisse faire gagner du temps au niveau de la durée des calculs, l'administration d'un tel système s'avère beaucoup plus complexe. Ainsi, le nœud chargé de faire le conditionnement de trafic doit pouvoir traiter et synthétiser en temps réel, les résultats qui lui sont envoyés par les différentes parties du simulateur. Cette phase de traduction peut donc être relativement lourde et le temps gagné en simulation peut de nouveau être perdu lors de cette collecte des résultats. De plus, les différentes parties du simulateur doivent être synchronisées les unes avec les autres pour maintenir la cohérence des résultats obtenus ce qui n'est pas assuré au moment où nous écrivons.

3.2.2.4 Les autres solutions d'émulation

Des plate-formes hybrides d'émulation à grande échelle ont été développées pour évaluer des protocoles et des applications. Ces plate-formes d'émulation peuvent utiliser plusieurs des techniques d'émulation que nous venons de présenter comme dans le cas de NetBed [79] et de TWINE [85] ou bien elles peuvent se rapprocher du test en environnement réel comme dans le cas de ORBIT [69] et EWANT [71].

NetBed (Emulab) [79] Cette plate-forme est basée sur Emulab, une plate-forme d'expérimentation qui utilise du conditionnement de trafic avec Dummynet, de la simulation temps réel avec NSE et même du test réel pour évaluer des protocoles sur des réseaux sans fils. Cette plate-forme repose sur l'utilisation d'une grille de calcul se composant de plusieurs centaines de nœuds inter-connectés par un réseau filaire. Ces nœuds peuvent servir d'élément terminal où l'application à tester est déployée ou d'élément de calcul, suivant qu'ils embarquent un émulateur ou un simulateur temps réel. Pour les test sans fils, Netbed dispose d'une soixantaine de machines équipées de carte réseau sans fil 802.11 a/b/g réparties à travers la plate-forme. Les tests sans-fil sont relativement limités : il n'est pas possible de faire bouger les nœuds et il n'est pas possible de spécifier de modèles de propagation. On ne peut donc pas vraiment parler d'émulation pour la partie sans fil mais plutôt de test réel. Enfin, Netbed ne propose qu'un support des réseaux sans fil en mode Infrastructure, le mode Ad-Hoc n'est pas supporté. Il faut tout de même noter que des efforts sont fait actuellement pour introduire de la mobilité dans Netbed. Les travaux présentés dans [42] proposent d'utiliser des robots mobiles pour reproduire le comportement d'un réseau de senseurs. Ces robots sont dotés de cartes réseaux sans fil fonctionnant sur la bande de 900 MHz, mais là encore les aspects de propagation ne sont pas pris en compte. Les cartes ne sont pas dotées d'atténuateurs si bien qu'il n'est pas possible de jouer sur la puissance d'émission des équipements radio. Cependant cette solution permet d'avoir un contrôle très précis de la mobilité, mais elle se rapproche plus du test réel que de l'émulation.

TWINE [85] TWINE a été développée dans l'optique d'évaluer des protocoles *cross-layers* et des applications dans les réseaux sans fils. Pour cela, elle propose elle aussi d'utiliser de la simulation temps réel, du conditionnement de trafic et du test réel. La principale différence avec Netbed est que TWINE permet d'émuler et de simuler en temps réel des réseaux sans fil et de ne pas uniquement reposer sur du test réel. Un autre point important est que la partie de simulation temps réel ne repose pas sur un simulateur en mode émulation comme NSE mais sur l'utilisation d'un simulateur de réseau classique qui est ralenti par des mécanismes externes pour travailler en temps réel. A cette fin, une couche de simulation a été développée pour intercepter les paquets, les convertir en événements compréhensibles par le simulateur et

les injecter dans le simulateur. L'horloge à évènements discrets du simulateur est ensuite ralentie pour fonctionner en temps réel. Pour éviter qu'une simulation ne prenne trop de temps, le nombre maximum de nœuds pouvant être simulé par une machine est limité à 60 pour un réseau 802.11b. L'autre différence fondamentale avec Netbed est que TWINE ne permet pas de mettre en œuvre de réseau filaire.

Ces plate-formes d'émulation sont relativement complexes à administrer et ne proposent pas encore un degré de contrôle suffisant pour les réseaux sans fil. Netbed ne propose que du test réel et ne permet pas de spécifier les conditions de propagation ni de faire bouger les nœuds. Une solution comme Netbed est de plus extrêmement complexe à administrer car plusieurs expériences peuvent être effectuées en parallèle si bien que les ressources qui sont mises à disposition de l'utilisateur ne lui permettent pas toujours de pouvoir mener son expérience. Dans un expérience sur les réseaux sans fil, il peut arriver par exemple que la plate-forme attribue des nœuds ne se trouvant pas à porté de communication.

D'autres plate-forme d'émulation ont donc été proposées pour pouvoir faire de l'évaluation de protocoles et d'applications dans les réseaux sans fil. Ces solutions peuvent être classées dans les solutions d'émulation même si elles se rapprochent du test réel. Ces plate-formes, dont nous allons présenter deux exemples à travers ORBIT et EWANT, utilisent des terminaux sans fil réel. Ces terminaux sont équipés d'atténuateurs pour pouvoir émuler la topologie ainsi que la mobilité des nœuds. Dans certains cas des générateurs extérieurs de bruits sont ajoutés dans la plate-forme pour pouvoir jouer sur les conditions de propagation dans le réseau sans fil.

ORBIT [69] La plate-forme ORBIT se caractérise par une grille composée d'environ 400 nœuds équipés d'interfaces réseaux sans fil utilisant des technologies 802.11a/b/g. Ces terminaux sont fixes et espacés d'un mètre chacun. Pour reproduire la topologie d'un réseau sans fil quelconque, des atténuateurs de signal sont utilisés au niveau de chaque terminal. Cette solution permet donc de reproduire les caractéristiques de chaque lien. Une version plus récente, présentée en 2005 dans [67], introduit le concept de mobilité dans la plate-forme ORBIT ce qui permet de faire varier la topologie du réseau au cours du temps.

Le fait d'utiliser des terminaux sans fil évite de passer par la phase complexe de déploiement d'une couche MAC 802.11 dans la mesure où celle-ci est déjà présente et permet d'obtenir des informations réalistes sur le comportement physique du réseau sans fil. Un générateur de bruits parasites est également utilisé pour pouvoir reproduire les interférences radios qui peuvent survenir sur un réseau 802.11.

EWANT [71] EWANT, présenté en 2003, est un autre exemple de plate-forme basée sur des ordinateurs reliés par des cartes sans fils équipées d'atténuateur. EWANT a introduit de nouvelles techniques pour reproduire la mobilité d'un réseau sans fil en s'appuyant sur l'utilisation de ces atténuateurs de signal. Ce sont ces techniques qui ont été reprises par ORBIT dans sa version de 2005 pour pouvoir à son tour gérer la mobilité.

Ces plate-formes permettent d'évaluer des protocoles de niveau MAC ou supérieur et sont parfois assimilées à des solutions de niveau 1. Cependant, comme le niveau physique n'est pas simulé, nous avons choisi de ne pas les inclure dans cette catégorie. Cette technique d'émulation, proche du test réel, est très onéreuse à mettre en œuvre. ORBIT par exemple se compose de plusieurs centaines de terminaux. De plus, elle présente quelques incertitudes

au niveau du réalisme résultant du contrôle des communications sans fil entre les nœuds. En effet, comme les communications se passent dans l'air, des sources de perturbations extérieures peuvent survenir à tout moment. Les travaux de Ganu et al. présentés dans [26] ont clairement montré que l'environnement de test sans fil n'est pas complètement contrôlable et va influer sur la qualité de l'émulation. Ils ont également permis de mettre en évidence que même en calibrant correctement les interfaces réseaux sans fil, l'exacte reproduction d'une expérience ne peut pas être assurée.

3.3 Emulation de réseaux sans fil

Les solutions d'émulation vues dans la section précédente ne permettent pas d'émuler correctement les réseaux sans fil soit parce qu'elles ne prennent pas en charge la mobilité des terminaux et le caractère dynamique de la propagation radio(e.g. les conditionneurs de trafic), soit parce qu'elles ne respectent pas les contraintes de temps réel (e.g. la simulation temps réel). Pour répondre à ces besoins, de nombreuses autres solutions ont été proposées. Certaines, toujours basées sur une solution centralisée, ont recherché des moyens de limiter les calculs à faire au niveau du nœud central, tandis que d'autres au contraire ont cherché à distribuer au maximum le processus d'émulation pour que tous les nœuds du réseau participent au calcul des conditions. Ce sont ces différentes solutions que nous allons maintenant présenter.

3.3.1 Émulation centralisée

Les problèmes liés aux contraintes temps réel comme la nécessité de générer des conditions réalistes dans un très petit intervalle de temps et pouvoir gérer plusieurs trafics réels ne sont pas très bien pris en compte par les solutions de simulation temps réel tel que NSE. Pour les résoudre, des solutions utilisant différentes techniques ont donc été proposées.

Seawind [46] Seawind se présente comme un émulateur de réseau sans fil GPRS (*General Packet Radio Service*) proposant une gestion très poussée des délais entre un client et un serveur. Il permet d'insérer des délais pour l'allocation de ressources mais également pour la transmission (bande passante) et la propagation du signal. De plus, il permet de spécifier un trafic de fond circulant sur le lien sans fil ainsi que des priorités sur le trafic émis par le serveur vers le client. La principale limitation de ce conditionneur de trafic vient du fait qu'il ne peut reproduire qu'un seul lien entre un client et un serveur et ne peut donc pas être retenu pour émuler un réseau à grande échelle ou même pour mener des expériences de multicast. En outre, comme il ne permet pas de prendre en compte la mobilité des terminaux ainsi que la propagation du signal radio, nous avons fait le choix de le classer dans la catégorie des conditionneurs de trafic. Il est en effet envisageable de construire une plate-forme d'émulation plus importante chargée de calculer les conditions à émuler qui repose sur Seawind.

WINE2 [74] WINE2 propose une solution qui consiste à utiliser une version allégée de NSE pour émuler des réseaux sans fil UMTS (*Universal Mobile Telecommunications System*) ou GSM-GPRS. Cependant, la technique utilisée pourrait être transposée au réseau 802.11. Ainsi, pour alléger les calculs en temps réel de NSE, les caractéristiques physiques du lien sont précalculées à l'aide d'un simulateur hors ligne puis réinjectés dans NSE. La couche liaison a également été simplifiée en n'intégrant que les modèles nécessaires pour émuler une cellule

unique. De plus, les réseaux de type UMTS et GSM-GPRS ont besoin de beaucoup moins de bande passante que les réseaux mobiles 802.11 : les réseaux UMTS annoncent à l'heure actuelle un débit maximum théorique de 384Kb/s et les réseaux GPRS un débit de l'ordre de 115Kb/s. Ces faibles débits autorisent un temps de calcul plus important de la part du simulateur temps réel. Par contre, il faut noter que cet émulateur n'autorise qu'un seul client réel, tout le reste du réseau étant simulé au sein de NSE ce qui limite les évaluations réalisables avec un tel système. Il est impossible par exemple de tester une application interactive distribuée avec ce type d'émulateur.

MobiNet [51, 52] Contrairement à WINE2 qui a pris le parti d'améliorer une solution existante, des solutions comme MobiNet ont choisi de repartir de zéro. MobiNet est un extension pour les réseau sans fil de l'émulateur de réseau grandes distances ModelNet [76]. La plateforme se compose de deux catégories d'éléments :

- Les nœuds de bordures qui vont héberger des nœuds virtuels sur lesquels seront déployés les applications et protocoles réels à tester. Chaque nœud virtuel va représenter un nœud du réseau à émuler.
- Les nœuds de cœur qui vont être chargés de reproduire les conditions d'un réseau sans fil de type 802.11b.

Pour cela, plusieurs modules ont été développés. Le premier est une couche MAC virtuelle qui va reproduire à l'aide d'une machine à état la technique d'accès au canal 802.11b. Le second est un module de routage embarquant une implémentation du protocole DSR qui permet d'émuler les communications possibles entre les nœuds distants du réseau émulé. Enfin, le troisième module est un module de mobilité qui va être chargé de faire bouger les nœuds. Ce module peut d'ailleurs être alimenté par des traces qui auront été récupérées dans le monde réel ou pré-calculées. Ce découpage en module permet à MobiNet d'être facilement extensible, l'utilisateur pouvant par exemple remplacer le module de routage DSR par un autre module qu'il aurait développé ou par un module utilisant un protocole de routage plus performant comme AODV ou OLSR. Chaque module peut être désactivé en fonction des besoins de l'utilisateur si bien que les résultats annoncés ne tenant pas compte de la mobilité sont très impressionnants. MobiNet peut ainsi supporter 100 trafics concurrents jusqu'à un débit cumulé de 49Mbit/s. Cependant, les modèles utilisés pour décrire l'environnement et la propagation en particulier sont relativement simples dans la mesure où ils sont uniquement fonction de la distance. Il est donc à craindre qu'avec des modèles nécessitant plus de temps de calcul et en ajoutant de la mobilité nécessitant des recalculs fréquents de routes, les performances ne se dégradent rapidement. En effet le protocole de routage implémenté (DSR) est relativement léger en termes de calculs de route et il est à craindre que la simulation en temps réel d'un protocole comme OLSR par exemple ne soit beaucoup plus coûteuse en temps de calcul.

JEMU [25] JEMU est un émulateur de niveau 1 qui s'appuie sur l'utilisation de piles de communication configurables. Chacune de ces piles de communication configurables va jouer le rôle des nœuds composant le réseau à émuler. Tout comme MobiNet, JEMU propose dans ces piles configurables, une implémentation du protocole de routage DSR et de la couche MAC 802.11b pour pouvoir émuler des réseau Ad-Hoc. JEMU se présente sous la forme d'une application client serveur où le rôle du client et joué par une couche radio virtuelle. Cette couche est insérée dans la pile de communication configurable juste en dessous de la couche MAC et va intercepter tous les paquets émis par cette dernière. Ces paquets sont ensuite envoyés à

l'application centrale jouant le rôle de serveur qui va devoir reproduire le comportement de la couche radio. De ce fait, JEMU ne peut pas être considéré comme une solution totalement centralisée, cependant comme le traitement est effectué par une application centrale qui voit passer l'ensemble du trafic, nous avons choisi de le classer dans cette catégorie.

En simulant le mouvement des nœuds composant le réseau et en se basant sur la portée de transmission de chaque nœud, cette application centrale détermine en temps réel si une communication est possible ou si des collisions peuvent survenir (lorsque deux nœuds voisins émettent en même temps par exemple). S'il n'y a pas de collision, les paquets sont acheminés à leur destination. En cas de collision par contre, les paquets peuvent être perdus, corrompus ou acheminés suivant la façon dont l'utilisateur a paramétré JEMU. Notons également que les mouvements des nœuds peuvent être simulés en temps réel ou calculés préalablement à une expérience.

Cette solution d'émulation permet donc de rendre un service de niveau physique en déterminant si des collisions surviennent sur le support physique. Des implémentations des protocoles DSR pour le routage Ad-Hoc et MAC 802.11b sont également fournies dans les piles de communications configurables utilisées par les clients émulant les nœuds du réseau sans fil. Par contre, ce service de niveau physique ne tient compte que d'une zone de couverture, c'est-à-dire la distance, pour déterminer si deux nœuds peuvent communiquer l'un avec l'autre. Pour émuler de manière réaliste un réseau sans fil, il est nécessaire de prendre en compte, par exemple, les rebonds de l'onde sur les obstacles et leurs effets sur la communication radio. Nous reviendrons plus en détail sur ces aspects de propagation dans le chapitre suivant.

NEMAN [65] Enfin, la dernière solution concernant les solutions d'émulation centralisée consiste à utiliser une seule machine pour émuler l'ensemble du réseau. Cette solution d'émulation qui se développe dans le cadre de l'émulation filaire à travers des solutions comme IMUNE [80], fut retenue dans le cadre de NEMAN. NEMAN propose d'émuler l'ensemble des terminaux mobiles à l'intérieur d'une seule machine en utilisant des interfaces réseaux Ethernet virtuelles appelées TAP et de contraindre les communications qui vont circuler entre les TAPs. Ces TAPs sont disponibles dans les noyaux Linux et permettent notamment de faire des tunnels de communications. NEMAN dispose d'une interface graphique permettant de voir évoluer la topologie au cours du temps. Pour faire évoluer cette topologie, il s'appuie sur des scénarios de mobilité générés à l'aide de l'outil *setdest* fourni avec le simulateur de réseau NS-2. Ces scénarios vont décrire l'ensemble des positions que les nœuds vont occuper au cours de l'expérience. Pour obtenir ces positions, *setdest* utilise un modèle de mobilité aléatoire le *Random Waypoint* dont nous présenterons les particularités dans le chapitre suivant.

Pour l'instant NEMAN ne propose qu'un support de la mobilité, la partie propagation ne gérant qu'une portée de transmission des terminaux pour pouvoir déterminer quel terminal peut communiquer avec quel autre à un moment donné. Grâce à sa simplicité en termes de modélisation, NEMAN peut émuler plusieurs centaines de terminaux. Cependant, il est à craindre qu'un support complet de la propagation ne fasse sensiblement chuter ce nombre. En effet les calculs à effectuer en temps réel sont pour l'instant très limités ce qui ne sera plus le cas une fois que la propagation sera supportée, c'est pourquoi les concepteurs de NEMAN s'orientent vers une distribution de la plate-forme sur plusieurs terminaux. En distribuant la charge de calcul, ils espèrent pouvoir garder ce qui fait leur force à savoir le support d'un très grand nombre de terminaux.

Toutes ces solutions centralisées ont essayé de résoudre le problème du passage à l'échelle en diminuant le nombre de calculs à effectuer au niveau du nœud central. Ainsi, WINE 2 a proposé de simplifier les modèles utilisés dans NSE ainsi que le nombre de nœuds à prendre en compte. Cependant, cette solution n'est pas satisfaisante dans la mesure où cela diminue le réalisme de l'émulation. Des solutions comme MobiNet et JEMU ont proposé d'utiliser des couches virtuelles pour émuler un réseau sans fil, cependant aucune de ces deux solutions ne prend en compte de manière convainquante les conditions de propagation du signal radio. De plus, JEMU nécessite de modifier les éléments terminaux qui vont héberger les applications et protocoles à tester. Par contre, le principe de pré-calcul des conditions de mobilité qui est utilisé dans JEMU et dans NEMAN est intéressant. En effet bien que ces deux émulateurs n'utilisent que des modèles de mobilité très simples, le fait de pré-calculer les positions permet a priori d'utiliser des modèles beaucoup plus réalistes et ceci sans affecter les performances de l'émulation.

3.3.2 Émulation décentralisée

Une méthode d'émulation distribuée consiste à déléguer une partie de l'émulation à chaque nœud composant la plate-forme de test. Cette plate-forme se compose la plupart du temps de deux réseaux distincts, l'un servant à l'administration et l'autre aux expériences. Le fait de séparer ces deux réseaux évite les collisions entre le trafic d'administration et le trafic généré pour les besoins de l'expérience. Dans la majorité des cas, un nœud central, situé sur un réseau d'administration, est chargé de synchroniser l'émulation entre les différents terminaux d'expérimentation pour garantir la cohérence de l'expérience. Cette approche fut retenue dans MobiEmu [82], dans NET [32, 31] et dans EMWIN [83].

MobiEmu [82] MobiEmu permet de reproduire la mobilité des nœuds mobiles sur un environnement filaire de type LAN. Il reprend l'architecture maître/esclave avec une application chargée de surveiller l'émulation globale et de contrôler les communications possibles entre les nœuds. C'est cette application qui établit ou rompt les liens en fonction d'un scénario de mobilité prédéfini. Ce scénario peut être obtenu à partir de l'outil *setdest* qui sert également pour les simulations NS-2 ou il peut être fourni par l'utilisateur. A partir de ce fichier de scénario l'application de contrôle va générer une liste de règles permettant de savoir si deux terminaux peuvent communiquer à un instant donné. La liste de règles est ensuite envoyée à tous les nœuds esclaves et la phase d'émulation proprement dite va pouvoir commencer.

Durant l'expérience, le nœud maître va maintenir une horloge et envoyer périodiquement des paquets contenant la date aux esclaves. En fonction de ces informations temporelles, les nœuds esclaves vont savoir quelles règles doivent être appliquées. Cette solution basée sur des scénarios est intéressante, mais n'a cependant pas été poussée assez loin dans la mesure où les couches physiques et MAC ne sont pas émulées. De ce fait, les communications ne sont pas vraiment réalistes et des effets tels que le problème des terminaux cachés ne peuvent pas être reproduits.

NET [31, 33] NET est un émulateur de réseaux filaires et de réseaux sans fil qui utilise des scénarios pré-calculés. Ces scénarios, décrits au format XML(eXtended Markup Language), servent en particulier à modéliser la topologie du réseau ainsi qu'à spécifier les modèles permettant de faire varier dynamiquement les paramètres de QdS. Ces modèles dynamiques peuvent agir soit au niveau temporel, c'est-à-dire qu'à une date prédéfinie on fait varier le paramètre

désiré, soit au niveau paquet ce qui revient à faire varier le paramètre en fonction d'un paquet particulier (le 10ème paquet d'un flux par exemple). Ces modèles de variations dynamiques peuvent être basés sur des tables, sur des distributions gaussiennes (pour les délais essentiellement) ou sur des modèles markoviens (pour la bande passante et les pertes). Dans le cadre d'une émulation de réseau sans fil, ces scénarios vont permettre de décrire les différentes positions des nœuds au cours du temps (en utilisant là encore l'outil *setdest* de NS-2) et de tenir compte de la propagation entre ces nœuds. Comme dans les autres solutions d'émulation, cette partie propagation est très rudimentaire et utilise un modèle ne tenant compte que de la distance. Ainsi, deux nœuds ne peuvent communiquer que si la distance qui les sépare ne dépasse pas un certain seuil. Une fois analysés, les résultats obtenus à partir de ces scénarios sont joués sur NETShaper, un conditionneur de trafic mis au point spécialement pour les besoins de NET qui se présente sous la forme d'une couche d'émulation s'insérant entre la couche liaison et la couche réseau. Chaque nœud composant la plate-forme NET embarque une instance du conditionneur de trafic NETShaper. Le nœud central situé sur le réseau d'administration se base donc sur le scénario pré-calculé pour simuler en temps réel les mouvements des nœuds et déterminer les communications qui sont possibles dans le réseau. Il envoie ensuite périodiquement les résultats de sa simulation aux instances de NETShaper qui vont être chargées de les reproduire.

EMWIN [83] EMWIN est une extension pour les réseaux sans fil de l'émulateur filaire EMPOWER [84] qui utilise un cluster de machines. Dans EMWIN un nœud mobile est représenté grâce à un nœud virtuel qui correspond à un module du noyau Linux attaché à une interface réseau. Ces nœuds virtuels peuvent utiliser des modèles de mobilité différents de l'un à l'autre et il est possible d'en déployer plusieurs sur une seule machine physique du cluster pour émuler un réseau Ad-Hoc. La cohérence des déplacements est ici aussi contrôlée par un serveur de simulation situé sur un réseau d'administration dédié. En plus de ces nœuds virtuels, EMWIN utilise sur chaque nœud une couche MAC virtuelle permettant de reproduire l'accès au canal CSMA/CA sur un réseau Ethernet utilisant du CSMA/CD. Enfin, il est à noter que les routes multi-bonds sont pré-calculées avant l'émulation proprement dite ce qui permet de limiter les calculs à faire en temps réel.

MASSIVE [56] MASSIVE présenté dans [56] est un émulateur de niveau 2 qui est complètement décentralisé contrairement aux solutions précédentes. Dans cette plate-forme d'émulation, il n'y pas de réseau d'administration et donc pas de nœud chargé de maintenir la cohérence entre les nœuds émulés. Il se compose de plusieurs serveurs appelés MASSIVE SERVER qui hébergent chacun un nœud virtuel du réseau Ad-Hoc à émuler. La particularité de MASSIVE est qu'il s'intéresse uniquement à l'évaluation de protocole de routage Ad-Hoc si bien que seuls les aspects de mobilité sont pris en compte. Ici encore, la modélisation des aspects de propagation se limite à la zone de couverture maximale d'un nœud. Par contre, MASSIVE présente la particularité de pouvoir réaliser plusieurs émulations en parallèle, chaque MASSIVE SERVER pouvant héberger un nœud pour chaque expérience. Il permet également grâce à l'utilisation d'un langage de script dédié, de spécifier de nouveaux modèles de mobilité de manière simple ce qui permet d'augmenter le réalisme des émulations rendues.

Ces solutions d'émulation décentralisées sont assez intrusives dans la mesure où chaque nœud d'expérimentation de la plate-forme doit embarquer une partie de l'émulateur. De plus, ces

émulateurs ne fonctionnent en général que sur un seul type de système d'exploitation (Linux la plupart du temps). Ceci peut être pénalisant car bien souvent un développeur d'application préférera effectuer ses tests sur un système d'exploitation de type Windows. Par contre toutes ces solutions proposent de l'émulation de niveau 2 qui permet de tester de nouveaux protocoles de routage. Malheureusement, les aspects de propagation et d'énergie ne sont pas pris en compte avec précision or, comme nous le verrons dans le chapitre suivant, ceux-ci ont un impact très important sur le comportement d'un réseau sans fil.

3.4 Conclusion

Comme nous pouvons le voir sur la figure 3.5, les solutions décentralisées sont surtout utilisées pour rendre un service de niveau 2 et ne s'interessent pour la plupart qu'aux effets de la mobilité (MobiEmu, NEMAN, JEmu). En effet, ces solutions permettent de tester des algorithmes de routage donc, le fait de décentraliser l'émulation permet de ne pas avoir de problèmes lors du déploiement du protocole. Chaque nœud pense se trouver sur un vrai réseau ad-hoc et pourra de manière naturelle effectuer sa recherche et sa maintenance de routes car la topologie du réseau est identique à celle du réseau émulé. Dans une solution centralisée les choses sont plus complexes car il faut masquer au protocole de routage l'existence du nœud central chargé de l'émulation. En effet ce nœud n'existe pas dans le réseau émulé. Cependant, si les protocoles de routage ne sont pas la cible de la plateforme d'émulation cet aspect décentralisé peut devenir pénalisant. En effet, il oblige à travailler de manière intrusive sur les éléments terminaux ce qui complique la mise en œuvre de la plateforme. Une solution centralisée est donc mieux adaptée pour fournir un service d'émulation de niveau 3. Toutes les communications passant par un nœud central, l'administration s'en trouve simplifiée et les éléments terminaux peuvent utiliser des systèmes d'exploitation identiques à ceux que l'utilisateur retrouvera dans l'environnement réel.

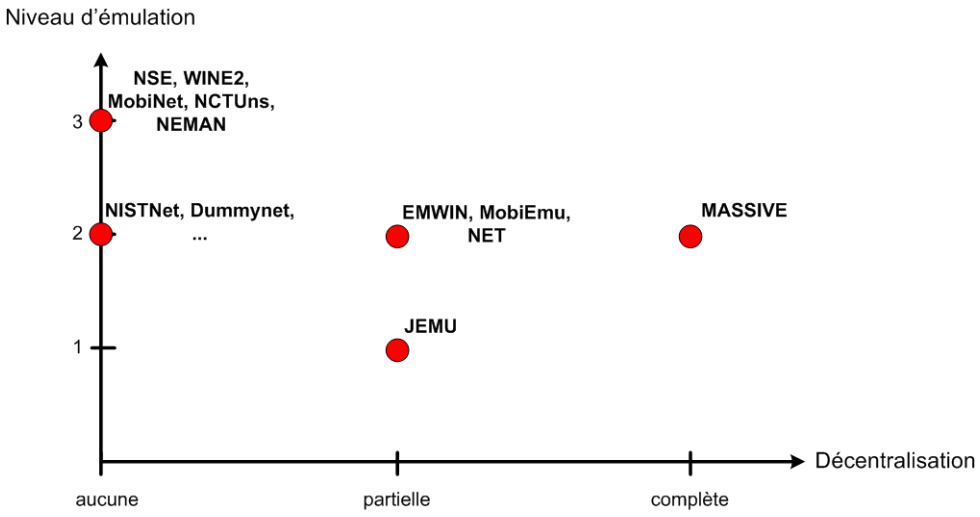


FIG. 3.5: Classement des différentes solutions d'émulation sans fil existantes (Niveau / Décentralisation)

Le nœud central, que nous appellerons émulateur (ou routeur émulateur) pour simplifier, est chargé de reproduire en temps réel les effets d'un réseau sans fil sous-jacent. Comme

nous pouvons le voir sur la figure 3.6, les solutions actuelles de niveau trois fonctionnent principalement en temps réel, c'est à dire que les effets du réseau sous-jacent sont calculés en temps réel. Ainsi, NSE ou WINE2 vont simuler l'ensemble des couches basses et effectuer le conditionnement de trafic en temps réel. Seul la partie mobilité est calculée à l'avance. Or, pour tenir les contraintes de temps réel , les solutions comme NSE sont obligées de faire des concessions au niveau du réalisme des modèles utilisés. Différentes études récentes ([62, 47]) ont montré que le fait d'utiliser des modèles trop simples a un impact non négligeable sur la qualité des résultats obtenus par simulation et donc que des modèles réalistes doivent être utilisés.

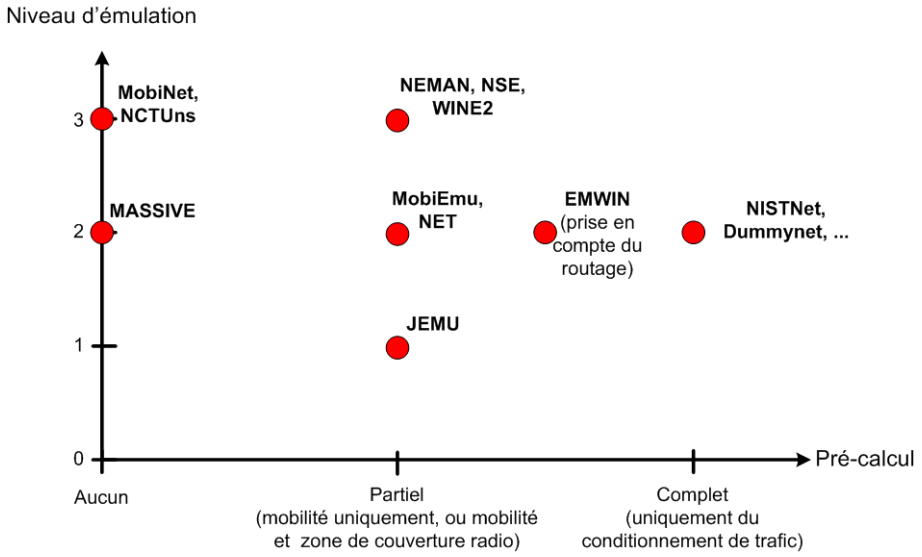


FIG. 3.6: Classement des différentes solutions d'émulation sans fil existantes (Niveau / Pré-calcul)

Des solutions d'émulation de niveau 2 comme NET ont proposé de calculer certains paramètres préalablement à l'émulation proprement dite. Cette technique, qui consiste dans le cas de NET à pré-calculer les positions des mobiles ainsi que la porté radio [32], va donc conduire à utiliser un scénario durant l'émulation. Ce scénario va permettre de faire bouger les différents nœuds du réseau sans fil tout en laissant à l'émulateur le soin de simuler en temps réel les communications pouvant avoir lieu dans le réseau sans fil. EMWIN est une autre solution de niveau 2 qui reprend ce principe en l'étendant au pré-calcul des tables de routages. C'est ce principe de scénario et de pré-calcul des conditions que nous nous proposons d'étendre et de généraliser dans une solution d'émulation de niveau 3, pour diminuer au maximum la charge de travail que va devoir supporter l'émulateur.

4 Spécification d'une solution d'émulation

4.1 Fonctionnalités nécessaires pour l'émulation

Comme nous venons de le voir dans le chapitre précédent, de nombreuses solutions d'émulation ont été proposées. Elles se sont principalement concentrées sur un ou deux aspects caratéristiques des réseaux sans fil comme la mobilité ainsi que sur le moyen de disposer d'un très grand nombre de nœuds dans la plate-forme d'émulation pour pouvoir réaliser des tests à grande échelle. Cependant ceci a bien souvent conduit à faire des concessions au niveau du réalisme de l'émulation rendue. Or pour le test et l'évaluation de protocole, le réalisme des conditions émulées est très important au même titre que la taille du réseau que l'on peut émuler. De plus, comme nous allons le voir par la suite, l'évaluation de protocole nécessite d'avoir un contrôle très précis des expériences qui sont menées et des conditions qui sont émulées. Nous avons donc choisi de nous intéresser à trois fonctionnalités importantes pour l'émulation : la précision et le dynamisme des conditions à émuler qui vont jouer dans le réalisme d'une expérience et le besoin d'avoir des expériences reproductibles.

4.1.1 Précision des conditions

Quand on cherche à émuler un réseau, la première chose à prendre en compte est la précision des conditions à émuler afin de reproduire le plus fidèlement possible les conditions d'un réseau cible. En utilisant des modèles analytiques, de la simulation ou de l'analyse de traces capturées sur un réseau réel, il est possible de passer des conditions que l'on retrouve dans le réseau réel aux conditions émulées.

Ainsi que nous l'avons déjà présenté, l'environnement a un impact direct sur les paramètres de qualité de service (QdS) au niveau IP : la position des terminaux mobiles communicants ainsi que la propagation du signal entre ces terminaux vont faire varier les délais, les pertes et la bande passante au niveau IP. La première difficulté est donc d'émuler la mobilité de manière précise, pour obtenir les positions des terminaux mobiles au cours du temps. Pour cela, il importe de pouvoir utiliser des modèles précis tels que ceux que nous présenterons dans la suite de ce chapitre. En fonction du réalisme que l'on souhaite obtenir, on privilégiera des modèles plus ou moins complexes. Un exemple de modèle de mobilité réaliste a été présenté par Jardosh en 2003 [41]. Ce modèle utilise un découpage du monde en cellules de Voronoi pour décrire le déplacement d'un terminal sur un campus universitaire. D'autres modèles comme celui présenté dans [59] s'appuie sur l'utilisation de bases de données conséquentes, pour prédire les déplacements possibles d'un terminal en les comparant à tous les déplacements précédemment effectués par d'autres terminaux s'étant trouvés dans la même situation.

Dans le même ordre d'idée, la propagation du signal entre deux terminaux doit pouvoir tenir compte de l'environnement qui l'entoure. En se basant sur l'atténuation du signal en fonction de la distance, sur celle résultant des chemins multiples que l'onde emprunte pour atteindre sa destination et sur l'absorption du signal par les obstacles, il est possible de calculer la puissance reçue par un nœud récepteur et ceci pour chaque instant de l'expérience. En fonction de cette

puissance reçue, il est ensuite possible de déterminer le débit utile théorique au niveau IP ainsi que les occurrences des pertes qui peuvent survenir. Ici encore la précision des modèles utilisés pour reproduire les conditions de propagation va jouer sur la qualité et donc le réalisme des résultats.

La précision est donc un élément clef, car si le modèle de propagation est trop simple, le comportement du réseau émulé ne sera pas réaliste et pourra conduire l'utilisateur à tirer des conclusions erronées des résultats qu'il obtiendra. Par exemple, les résultats obtenus dans [48] montrent sur l'analyse de traces réelles que l'utilisation de codes correcteurs d'erreurs basés sur des codes ARQ hybrides de petite taille ont des performances bien meilleures que ce que ne le laissaient supposer les simulations effectuées avec des modèles simples de pertes uniformément réparties et indépendantes entre les récepteurs. Or, ce résultat est important dans la mesure où ces codes peuvent être utilisés dans des terminaux disposant d'une puissance de calcul comme les PDA ou les téléphones mobiles. Plus généralement, plus la précision des modèles sera importante, plus l'émulation rendue sera réaliste.

4.1.2 Dynamisme des conditions

La mobilité et la propagation ont comme principal effet de faire varier les conditions réseaux (débit utile, pertes et délais) et donc de faire varier la connectivité beaucoup plus fréquemment que dans un réseau filaire. L'émulation doit donc être dynamique pour reproduire ces variations de conditions.

L'utilisation de modèles précis va permettre d'obtenir ces variations de conditions réalistes. Une fois déterminées, ces conditions peuvent être traduites en commandes d'émulation compréhensibles par le conditionneur de trafic, qui seront envoyées périodiquement à l'émulateur pour reproduire le dynamisme des conditions. La fréquence à laquelle ces commandes d'émulation doivent être prises en compte par le conditionneur de trafic est donc un point très important. En effet, même si les modèles utilisés pour représenter la mobilité et la propagation sont très précis, le fait de ne remettre à jour les conditions que toutes les secondes par exemple va diminuer d'autant l'efficacité du modèle. L'intervalle de temps entre deux mises à jour résultera donc d'un compromis : il faut qu'il soit suffisamment court pour permettre de garder le réalisme des modèles utilisés mais également qu'il soit assez grand pour que le conditionneur de trafic ait le temps matériel de les appliquer.

La solution d'émulation doit donc être sur-provisionnée pour pouvoir tenir les contraintes temps réel et utiliser un intervalle de temps suffisamment faible entre deux mises à jour des conditions pour garder le caractère dynamique de ces conditions.

4.1.3 Expériences reproductibles

Le troisième point que nous avons souhaité mettre en valeur est la possibilité de reproduire une expérience. En effet pour pouvoir évaluer convenablement un protocole, il est nécessaire de reproduire plusieurs fois de suite la même expérience avec exactement les mêmes conditions. Cette répétition des conditions peut être vue à plusieurs niveaux. Ainsi, il est nécessaire de reproduire les mêmes conditions en termes de bande passante, de délais et de pertes et ceci, aux mêmes instants t. Les différents facteurs pouvant introduire des différences entre plusieurs itérations de la même expérience doivent donc être réduits au minimum.

Cependant, la reproductibilité peut également être vue comme le degré de contrôle que l'on a sur la plate-forme d'émulation. Ainsi, plus on contrôle la plate-forme d'émulation et

meilleurs sont les résultats obtenus. Or, l'une des premières causes d'indéterminisme dans une émulation, provient de l'insertion des pertes de paquet. La plupart des conditionneurs de trafic qui sont utilisés par les plates-formes d'émulation ne proposent qu'un support probabiliste et non reproductible des pertes. Ces pertes sont donc insérées de manière aléatoire et affecteront différents paquets au cours des différentes répétitions de l'expérience. Dummynet par exemple n'assure que la reproductibilité d'une expérience en moyenne, c'est à dire qu'il assure que si on reproduit plusieurs fois la même expérience, on obtiendra toujours le même taux moyen de pertes. Par contre, il n'assure pas que les pertes affecteront toujours les mêmes paquets.

Ces solutions d'émulation sont suffisantes pour évaluer des applications réparties basées sur IP mais ne peuvent pas être utilisées pour évaluer des protocoles. Supposons que l'on veuille comparer deux mécanismes de contrôle d'erreur, l'un utilisant des codes FEC et l'autre utilisant des codes ARQ hybrides. Pour pouvoir les comparer, il est nécessaire que les pertes surviennent sur les mêmes paquets sinon, il sera difficile de conclure qu'un mécanisme est meilleur qu'un autre car les conditions n'auront pas été les mêmes. En effet, si l'on prend l'exemple d'une connexion TCP, les résultats obtenus ne seront pas les mêmes suivant que l'on perd un paquet de données ou que l'on perd le paquet de connexion SYN-ACK. Lorsque ce dernier est perdu, le délai observé est beaucoup plus important. La position des pertes est donc très importante dans l'évaluation d'un protocole : il faut donc être capable de les placer précisément mais également de pouvoir reproduire ce placement de manière parfaite d'une expérience à l'autre.

Enfin, le fait de reproduire précisément une expérience d'émulation va permettre d'augmenter la fiabilité dans les résultats obtenus en diminuant la variance entre les résultats qui pouvait résulter d'un placement aléatoire des pertes.

4.2 Mises en œuvre envisageables

Pour pouvoir répondre au trois besoins énoncés ci-dessus, deux solutions ont été envisagées : la simulation en temps réel ou l'utilisation d'un simulateur hors ligne préalable à celle d'un conditionneur de trafic.

4.2.1 Simulation en temps réel

La simulation en temps réel est la solution retenue par NSE, mais des études [53] ont rapidement montré que cet émulateur n'était pas satisfaisant car il n'arrive pas à tenir les contraintes temps réel lorsque le nombre de flux réels à traiter augmente, et ce, même avec des modèles simples. Les solutions d'émulation basée sur de la simulation en temps réel doivent non seulement être capables de conditionner le trafic au cours d'un intervalle de temps entre deux mises à jour des conditions mais elles doivent également déterminer ces conditions. Lorsque la fréquence de mise à jour est trop petite, ces solutions n'arrivent plus à effectuer la simulation et le conditionnement de trafic en temps réel.

Pour résoudre ce problème Mahrenholz et Ivanov ont proposé dans [53] une amélioration de NSE en optimisant le code du simulateur et notamment en diminuant le nombre d'appels systèmes ainsi que le nombre d'écritures disque. Ils ont également proposé dans [54] une nouvelle technique qui permet de traiter plus d'événements simulés dans le même intervalle de temps. Cependant, bien que cette solution améliore les performances elle souffre encore de limitations en termes de passage à l'échelle. Ainsi, même avec des modèles d'erreurs obtenus à partir de traces capturées sur un réseau réel et en utilisant des terminaux fixes, la nouvelle

version de l'émulateur atteint ses limites avec 6 terminaux émulés. L'ajout de modèles de mobilités et de modèles de propagation ne peut donc qu'augmenter la charge de calcul à effectuer dans un intervalle de temps, si bien que cette solution ne nous a pas paru adéquate pour développer une solution d'émulation de réseau sans fil.

4.2.2 Simulation hors ligne et conditionnement de trafic

Comme il n'est pas possible d'augmenter la puissance de calcul indéfiniment, une solution alternative à la simulation en temps réel consiste à pré-calculer un maximum de paramètres permettant de déduire les conditions à reproduire durant l'émulation. Ainsi comme le proposent déjà des solutions comme MobiEmu ou NET, il est possible de pré-calculer les effets de la mobilité du réseau de manière à ne laisser que la partie propagation à évaluer en temps réel pour déterminer les conditions à reproduire. Cependant cette phase de propagation peut également être partiellement pré-calculée (en faisant abstraction des collisions) dans la mesure où elle dépend principalement de l'environnement et de la position des terminaux mobiles à un instant donné. Il est donc possible de déterminer la puissance reçue au niveau de chaque nœud et ceci pour chaque paire de nœuds du réseau.

Le fait de pouvoir calculer les effets de la propagation et de la mobilité hors ligne va permettre d'utiliser des modèles complexes qui peuvent être coûteux en temps de calcul ce qui est une première solution pour résoudre le problème de précision que nous avons soulevé. En se basant sur une description de l'expérience spécifiant notamment les modèles à utiliser, il va être possible de déterminer les conditions du réseaux à émuler et de générer les commandes d'émulation correspondantes. En effet, les résultats obtenus à partir de ces modèles de propagation peuvent ensuite être traduits en termes de paramètre de QdS de niveau IP (débits utiles, délais et pertes) pour pouvoir être reproduits en temps réel par un conditionneur de trafic. Ces résultats forment des scénarios d'émulation qui seront joués en temps réel lors de la phase d'émulation.

Un diagramme d'activité résumant ce processus en deux phases est présenté sur la figure 4.1. La première phase ou phase de simulation se trouve sur la gauche de la figure. Elle se base sur une description de l'expérience fournie par l'utilisateur pour construire les différents modèles utilisés pour représenter le comportement du réseau à émuler . Grâce à ces modèles précis, la phase de simulation va être capable de déterminer les conditions de QdS présentes dans le réseau à émuler et de les synthétiser dans un scénario d'émulation. Ce scénario est ensuite lu lors de la phase d'émulation décrite sur la partie droite de la figure pour reproduire les caractéristiques du réseau émuler nécessaires à la prise de mesures.

Pour que l'émulation soit le plus réaliste possible, il faut que le scénario d'émulation soit le plus précis possible. Les modèles utilisés lors de la phase de simulation hors ligne sont donc très importants et doivent décrire le plus précisément possible des aspects comme la mobilité, les conditions de propagation et le comportement du réseau sans fil au niveau MAC. Ce sont quelques exemples de ces modèles que nous allons maintenant illustrer.

4.3 Modélisation d'un environnement sans fil

4.3.1 Introduction

Pour pouvoir émuler un réseau sans fil, il est nécessaire de tenir compte de plusieurs aspects :
– La mobilité des nœuds,

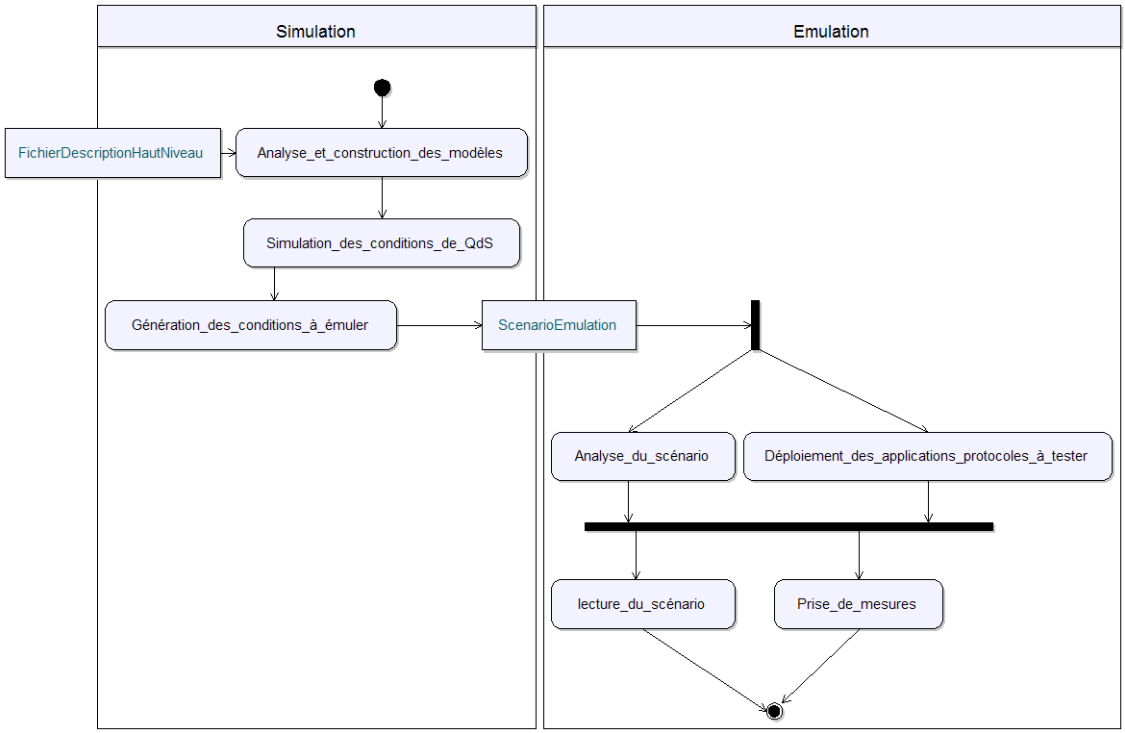


FIG. 4.1: Diagramme d'activité simplifié de W-NINE

- La propagation du signal entre ces nœuds qui va dépendre des différents obstacles ainsi que des sources d'interférence,
- La communication entre les nœuds qui va entraîner des problèmes décrits précédemment comme le cas des terminaux cachés, le cas des terminaux exposés pour les réseaux Ad-Hoc ou le handover pour les réseaux en mode infrastructure. Cette partie communication va également englober la prise en compte des effets du routage au niveau IP et la prise en compte de l'énergie résiduelle de chaque nœud pour pouvoir déterminer la durée de vie du réseau ainsi que les partitions de réseau qui peuvent survenir. Enfin, c'est également dans cette partie communication que devront être pris en compte les protocoles de niveau MAC et de niveau physique propres à une technologie particulière (IEEE 802.11b/g dans notre cas).

Comme nous l'avons vu, la mise en œuvre de la simulation hors-ligne nous libère des contraintes temps réel. De plus, la simulation permet d'obtenir des résultats parfaitement reproductibles. En effet, les tirages aléatoires peuvent être reproduits si le grain (*seed*) utilisé pour calibrer un générateur pseudo-aléatoire est le même d'une expérience à l'autre. Nous pouvons ainsi envisager l'utilisation a priori de nombres de modèles décrits dans la littérature dont nous allons maintenant donner un bref aperçu.

4.3.2 Modèles de mobilité

Un modèle de mobilité a pour vocation de prédire les positions qu'un nœud ou qu'un groupe de nœuds vont occuper au cours d'une expérience. Ces modèles peuvent être relativement

simples et se baser sur des tirages aléatoires ou bien prendre en compte différentes informations telles que les obstacles composant le monde. D'autres modèles encore plus complexes peuvent même s'appuyer sur des bases de connaissances pour déterminer les prochains déplacements d'un nœud.

4.3.2.1 Mobilité Individuelle

Un modèle de mobilité individuelle prédit la position d'un nœud mobile indépendamment des déplacements des autres nœuds mobiles.

Random Walk (marche aléatoire)[43] Ce modèle de mobilité est l'un des premiers présenté dans la littérature et également l'un des plus simples. Ici, un nœud se déplace d'un point à un autre en choisissant aléatoirement sa direction et sa vitesse. Le tirage aléatoire s'effectue dans une distribution uniforme. La direction σ est comprise entre 0 et 2π dans le plan et la vitesse v est comprise entre une valeur minimum (en général 0) et une valeur maximum spécifiée par l'utilisateur. Le mouvement est découpé soit en fonction du temps soit en fonction de la distance, c'est à dire que l'on passe d'un état à un autre soit au bout d'un temps fixé t soit en ayant parcouru une distance prédéfinie. Dans une expérience, cette vitesse peut varier d'un mobile à l'autre, cependant dans la plupart des expériences où ce modèle est employé, la vitesse est souvent la même pour l'ensemble des nœuds. De même, il n'est pas nécessaire de choisir le même critère d'arrêt pour chaque nœud.

C'est un modèle sans mémoire dans la mesure où les paramètres σ et v qui sont tirés à un instant $t+1$ ne dépendent pas des paramètres utilisés à l'instant t. Ainsi, la direction qui va être tirée ne tiendra pas compte de l'endroit d'où vient le nœud. Un nœud utilisant ce modèle peut donc faire des aller-retours incessants tout comme il peut suivre une ligne droite même si ces deux cas ont de très faibles chances de survenir.

Ce modèle qui est généralement associé au mouvement Brownien est régulièrement employé dans la recherche en dépit de son manque de réalisme car il permet de décrire un déplacement erratique et donc de faire des tests avec des conditions extrêmes de mobilité.

Random Waypoint (destination aléatoire) Le modèle dit de *Random Waypoint* est très proche du modèle précédent. La seule différence notable est l'introduction d'une notion de pause dans le modèle de *Random Walk*. Dans ce modèle un nœud commence par rester un certain temps à l'endroit où il se trouve puis il détermine une destination ainsi qu'une vitesse appartenant à l'intervalle $[0, V_{max}]$. Une fois la destination atteinte, le nœud marque à nouveau un temps d'arrêt puis reprend le processus de recherche de destination et de calcul de vitesse. Il faut noter que dans ce modèle la vitesse est constante tout au long du déplacement. Des modifications mineures ont parfois été apportées à ce modèle comme l'utilisation d'une vitesse constante à travers toute la simulation.

Ce modèle, bien que très simple et montrant des limitations certaines, reste cependant l'un des plus fréquemment employé dans le cadre de la recherche sur les réseaux Ad-Hoc. Ce modèle est presque toujours présent dans les simulateurs utilisés par la communauté scientifique (NS-2, OPNET, GloMoSim...). Il constitue un standard de fait pour la comparaison et l'évaluation de protocoles de routage Ad-Hoc. Cependant il est à noter que, comme pour le modèle précédent, les comportements obtenus sont très éloignés de ceux que l'on peut rencontrer dans la réalité.

Ces modèles de mobilité individuels sont relativement peu coûteux en temps de calcul si bien qu'ils peuvent être utilisés dans une solution de simulation temps réel. Cependant, le niveau de réalisme qu'ils procurent n'est pas suffisant pour la plupart des développeurs d'application ou de protocole souhaitant évaluer leur travail sur un comportement réel. D'autres modèles de mobilité simples ont donc été proposés. Ces modèles que l'on classera dans la catégorie des modèles à mémoire vont s'appuyer sur une connaissance de leurs déplacements précédents pour déterminer leur prochain mouvement et, pour les plus avancés, sur des informations relatives à leur environnement comme par exemple leur position géographique.

Version Markovienne du Random Waypoint Model [12] Ce modèle de mobilité, développé par Chiang, est un exemple simple de modèle de mobilité à mémoire. Ici, une chaine de Markov à trois états est définie : l'*état 0* qui est la position courante d'un nœud, l'*état 1* qui représente sa position précédente et l'*état 2* qui représente sa position suivante. Grâce à ces états, on obtient alors la matrice de mobilité suivante :

$$\mathbf{P} = \begin{pmatrix} P(0,0) & P(0,1) & P(0,2) \\ P(1,0) & P(1,1) & P(1,2) \\ P(2,0) & P(2,1) & P(2,2) \end{pmatrix}$$

où chaque entrée $P(a,b)$ représente la probabilité de passer de l'état a à l'état b.

Ce modèle permet d'avoir un comportement plus réaliste que les deux modèles précédents. La complexité est un peu augmentée dans la mesure où avec ce modèle, lorsqu'un nœud souhaite se déplacer il doit non seulement tirer sa nouvelle destination mais également faire un tirage aléatoire pour déterminer s'il doit se rendre à cette nouvelle destination ou bien rester où il se trouve ou encore retourner à sa position précédente. La principale difficulté de ce modèle est donc de déterminer les valeurs que l'on va stocker dans la matrice P. Lors de la définition de son modèle, Chiang a proposé les valeurs suivantes :

$$\mathbf{P} = \begin{pmatrix} 0 & 0.5 & 0.5 \\ 0.3 & 0.7 & 0 \\ 0.3 & 0 & 0.7 \end{pmatrix}$$

Avec ces valeurs et en considérant que les déplacements s'effectuent à une vitesse constante, un nœud aura une plus forte probabilité de se déplacer dans la même direction que de revenir en arrière, ce qui permet d'avoir un déplacement plus réaliste que dans le cas d'un modèle de *Random Waypoint* classique. La chaîne de Markov à trois états présentée sur la figure 4.2 résume ce processus de changement d'état en tenant compte des probabilités établies par Chiang avec X et Y qui représentent les coordonnées d'un nœud dans un monde en deux dimensions, X' et Y' les coordonnées suivantes que ce nœud va occuper et d la distance qu'il parcourt. La valeur de d est établie en fonction de la vitesse spécifiée par l'utilisateur.

De nombreux autres modèles ont repris ce principe de déplacements probabilistes comme par exemple le modèle de Gauss-Markov présenté dans [50] qui s'appuie sur un processus gaussien pour déterminer à un instant donné la vitesse et la direction d'un nœud mobile. Pour cela il va tenir compte de la position de ce nœud à l'instant $t-1$ ainsi que de la valeur d'un paramètre d'incertitude α prise entre 0 et 1 où 0 permet d'avoir le même comportement qu'un modèle de *Random Waypoint* et 1 un modèle rectiligne. Une description plus précise de ce modèle ainsi que des illustrations détaillant les modèles que nous présentons sont données dans [9].

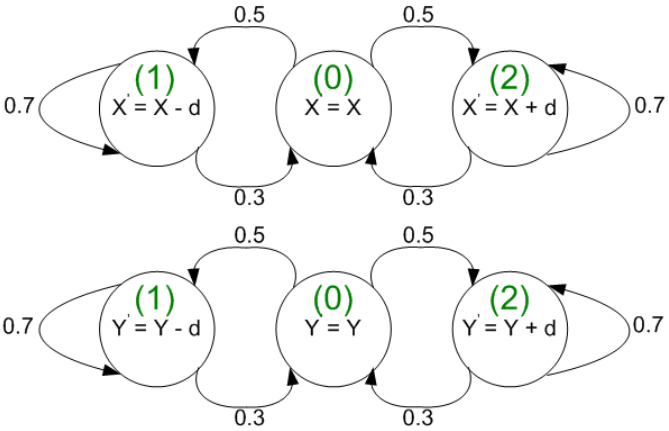


FIG. 4.2: Chaîne de Markov à trois états de la version probabilistique du modèle de Random Waypoint

City Section Mobility Model [21] Ce modèle de mobilité va utiliser des informations relatives au monde qui entoure le nœud mobile pour déterminer son prochain déplacement. Ainsi, pour utiliser ce modèle, le simulateur à besoin d'avoir une description précise de la position des obstacles, de la position des nœuds à l'intérieur du monde pour pouvoir calculer les déplacements d'un nœud mobile. Pour simplifier l'explication, nous parlerons de carte du monde pour faire référence à cette description. Ainsi, lorsqu'un nœud souhaite se déplacer, il commence par tirer une destination aléatoire sur la carte et met ensuite en jeu un algorithme de recherche de plus court chemin pour déterminer la route à suivre pour rejoindre sa destination. Cet algorithme va tenir compte, pour sa recherche, des rues que le nœud va emprunter et de la vitesse qu'il peut atteindre sur ces rues. Par exemple, si un chemin passant par des petites rues avec une vitesse limitées à 30 km/h est plus court en distance qu'un chemin passant par une voie expresse où la vitesse est limitée à 110 km/h, c'est quand même ce dernier chemin qui sera sélectionné si la durée du trajet est plus court. L'algorithme tient aussi compte de la distance minimum qu'il peut y avoir entre deux nœuds (pour éviter les collisions).

Ce modèle de mobilité effectue également un classement entre les nœuds en les répartissant dans différentes catégories (piétons, automobilistes, motocyclistes...). Chaque catégorie empruntera un itinéraire particulier, ainsi, un automobiliste privilégiera la voie expresse pour ses déplacements tandis qu'un piéton devra emprunter les petites rues. Cette technique de répartition et de mise en œuvre d'un algorithme de plus court chemin est beaucoup plus contraignante en terme de puissance de calcul que les solutions que nous avons présentées précédemment. De plus, elle nécessite d'avoir une connaissance approfondie de l'environnement pour pouvoir déterminer les déplacements des nœuds.

4.3.2.2 Mobilité de groupe

Les modèles de mobilité de groupe ont été introduits très tôt pour permettre de tester avec des comportements réalistes des scénarios militaires par exemple ou encore reproduire le déplacement d'un groupe de secouristes sur une zone d'avalanche ou bien pour reproduire de manière réaliste le déplacement d'un groupe d'individus dans une conférence. Dans ce type de modèles, les nœuds ne se déplacent plus aléatoirement mais vont tenir compte des

déplacements des autres nœuds du réseau pour déterminer la position qu'ils vont occuper à l'instant $t+1$.

Modèle de poursuite Dans ce modèle, un groupe de nœuds mobile va prendre pour cible un autre nœud du réseau et essayer de le rattraper. Ce nœud cible peut utiliser indifféremment n'importe quel modèle de mobilité individuel, il n'est en général pas au courant qu'il est poursuivi. Cependant, il est tout à fait envisageable de développer un modèle de mobilité lui donnant connaissance de ses poursuivants et lui permettant de se déplacer de manière à ne pas être rattrapé. Les nœuds poursuivants utilisent en effet un modèle relativement simple décrit selon la formule suivante :

$$p_p(t) = p_p(t-1) + Mouvement(p_c(t) - p_p(t-1)) + \overrightarrow{VA}$$

où p_p représente la position d'un nœud poursuivant, p_c la position du nœud cible, la fonction *Mouvement(...)* permet de calculer le chemin que parcours le poursuivant vers la cible en fonction de la distance qui les sépare et le vecteur aléatoire $\overrightarrow{VA}$ permet d'introduire un certain indéterminisme pour que la poursuite ne soit pas trop simple. Le paramétrage de ce vecteur aléatoire est très important dans la mesure où il ne faut pas non plus que les poursuivants s'écartent trop de leur cible sous peine de ne jamais réussir à la rejoindre.

Pour avoir un maximum de réalisme, ce modèle a donc non seulement besoin de connaître le comportement de sa cible mais doit également pouvoir tenir compte de son environnement. Ainsi, une amélioration simple de ce modèle consisterait à prévoir la destination du nœud poursuivi pour éviter de se retrouver dans un cul-de-sac.

Reference Point Group Mobility Model (RPGM Model) [35] Dans ce modèle, les nœuds se déplacent en tirant une direction et une vitesse aléatoire comme dans le modèle de *Random Waypoint*. En revanche, contrairement à ce dernier, la direction et la vitesse sont partagées entre tous les nœuds ce qui leur permet de toujours garder la même position les uns par rapport aux autres. En fait, un centre logique est calculé pour chaque groupe de nœuds et c'est ce centre logique que l'on fait bouger suivant un vecteur $\overrightarrow{GM}$. Les nœuds peuvent ensuite se déplacer légèrement par rapport à leur position de référence dans le groupe suivant un vecteur aléatoire $\overrightarrow{RM}$. Un exemple de RPGM est donné sur la figure 4.3.

Ce modèle a été utilisé pour reproduire le comportement d'une équipe de secours après une avalanche. Dans ce genre de cas en effet, les membres de l'équipe de secours, s'occupent tous d'une petite section et progressent de concert pour pouvoir couvrir toute la zone d'avalanche. Différentes variations de ce modèle ont ensuite été présentées pour reproduire notamment le comportement que l'on peut retrouver à l'intérieur d'une conférence. Dans cette variation, les nœuds se déplacent en groupe d'une pièce à l'autre et sont répartis en différents groupes : par exemple les personnes qui présentent et les personnes qui assistent. Dans chaque pièce les différents groupes peuvent se déplacer différemment : par exemple le groupe des personnes qui présentent est généralement plus pressé que celui des personnes qui assistent et se déplacera donc plus vite. L'intérieur de chaque pièce est découpée en petites zones dans lesquels les groupes de nœuds continuent à se déplacer à des vitesses différentes.

Community Based Mobility Model [59] Ce modèle a été développé en partant du principe que les modèles précédents proposaient un comportement aléatoire qui ne correspondait pas

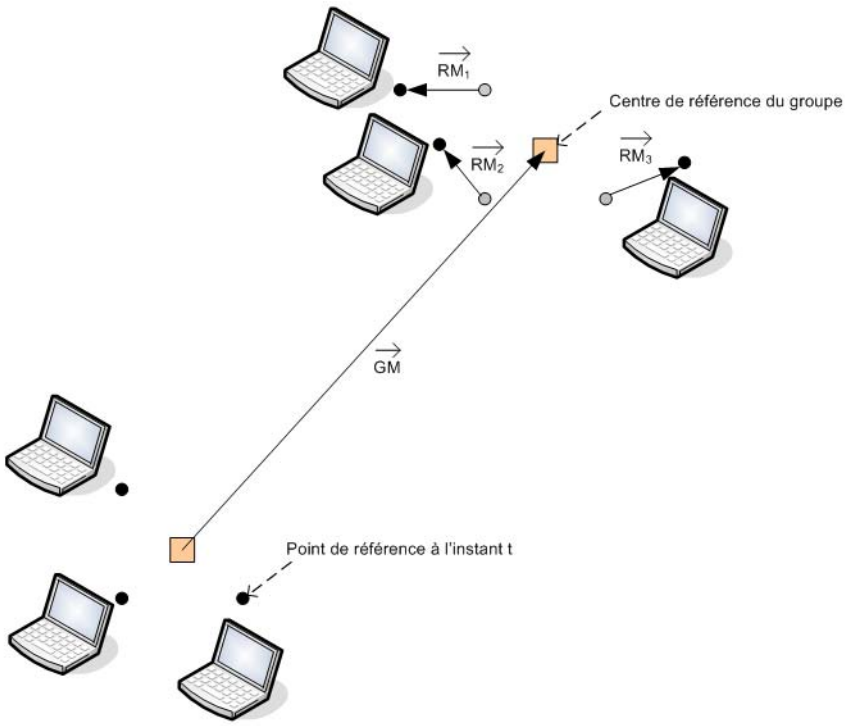


FIG. 4.3: Exemple de mouvements suivant l'algorithme RPGM

au déplacement réaliste d'un humain. Ces modèles ne permettent pas de représenter les interactions qui peuvent exister entre les nœuds mobiles, ce qui n'est pas réaliste pour reproduire un comportement humain. Ainsi, une personne va souvent reproduire les mêmes déplacements au cours d'une journée et va cotoyer en règle générale les même personnes (collègues de travail, amis, famille...). Pour répondre à ce besoin d'interaction entre les nœuds, le modèle basé sur la notion de communauté s'appuie sur la théorie des réseaux sociaux pour décrire un comportement réaliste. Il découpe le réseau en plusieurs ensembles de nœuds en se basant sur les relations sociales entre ces nœuds. Ainsi, les mouvements des nœuds vont être dictés aussi bien par la topologie qui les entoure que par les relations qu'ils entretiennent avec les autres membres de leur groupe. Ces relations peuvent être de plusieurs types et vont avoir différents niveaux d'importance suivant les moments de la journée. Pendant la journée, les interactions avec les collègues de travail seront ainsi plus probables que les interactions avec la famille ou les amis mais ceci n'est plus vrai pour le soir et le week-end. Pour représenter ces relations entre individus, une solution consiste à utiliser un graphe où chaque nœud de ce graphe va représenter un individu. Les poids sur les relations entre les individus vont permettre de montrer les relations sociales existantes entre eux. Le modèle de mobilité basé sur la notion de communauté exploite ce graphe en interprétant le poids comme une probabilité de présence à un endroit donné. Ainsi, en se basant sur le graphe d'interaction il est possible de déterminer pour chaque nœud une matrice d'interactions et de déterminer les différentes communautés. Puis, il est possible de déterminer les positions géographiques des différentes communautés à travers le monde. Le déplacement des nœuds peut ensuite être déterminé en fonction de l'attrait que va avoir chaque communauté à un instant donné de la journée. Cet attrait est caractérisé par un facteur de sociabilité qui varie dans le temps. Ce facteur de sociabilité va être

calculé en additionnant les poids de la matrice d'intéraction dépassant un certain seuil (0.25 dans l'article) et en divisant par le nombre total de nœuds. Ainsi, une personne très sociable aura un taux de sociabilité proche de 1 et une personne solitaire un taux de sociabilité proche de 0. Les déplacements à l'intérieur de la communauté peuvent ensuite suivre un modèle de mobilité différent. Dans le cas du modèle présenté dans [59], le modèle retenu est un modèle de *Random Waypoint* classique.

Ce type de modèle de mobilité de groupe va donc avoir besoin de nombreuses informations pour être mis en œuvre. Il est notamment nécessaire de disposer de traces réelles pour déterminer les interactions entre les individus et construire les communautés. Il faut également avoir une bonne connaissance du monde pour pouvoir placer correctement les communautés dans le monde. De plus, ce modèle qui permet de déterminer les déplacements d'un nœud entre les différentes communautés pourrait tout à fait être étendu en intégrant le modèle de *City Section Mobility* pour améliorer encore le réalisme.

4.3.3 Modèles de propagation

Les modèles de propagation sont utilisés pour déterminer la manière dont le signal radio va se propager dans l'environnement que l'on souhaite émuler. Quand un signal est émis avec une puissance P_t, un modèle de propagation permet d'estimer la puissance P_r reçue en tout point de l'espace. En règle générale, le signal transmis est reçu avec une certaine atténuation, qui dépend de plusieurs facteurs. Le facteur prépondérant est la distance : plus le récepteur est loin de l'émetteur, plus le signal reçu est faible. Un autre facteur important est la présence d'obstacles absorbant une partie du signal entre l'émetteur et le récepteur. Enfin, le signal transmis par la source rayonne dans plusieurs directions (et même dans toutes les directions dans le cas d'une antenne omnidirectionnelle). Le signal peut atteindre le récepteur par plusieurs chemins et se recombiner. Cette recombinaison des signaux reçus par chemins multiples peut donner lieu à des variations importantes de la puissance reçue, et ces variations sont généralement très fréquentes : les chemins multiples sont définis par la position des petites surfaces susceptibles de réfléchir le signal, une petite surface ayant une dimension de l'ordre de la longueur d'onde du signal transmis, soit 12 cm dans le cas WiFi.

Les modèles de propagation à grande échelle tiennent compte principalement de la distance émetteur-récepteur, et s'ils peuvent prendre en compte de manière très grossière la nature de l'environnement de propagation (urbain, intérieur, espace), à distance égale la puissance reçue reste la même. Les modèles de propagation à moyenne échelle prennent mieux en compte les obstacles entre émetteur et récepteur en fournissant une variation de la puissance reçue même à distance égale. Enfin, seuls les modèles de propagation à petite échelle permettent de prendre en compte le phénomène de fading et d'en reproduire les variations rapides de puissance reçue.

La suite de cette section décrit certains des modèles de propagation les plus utilisés [68] dans les trois catégories : grande, moyenne et petite échelle.

4.3.3.1 Modèles de propagation à grande échelle

Les modèles de propagation à grande échelle estiment la puissance du signal reçu au niveau du nœud récepteur en tenant compte uniquement de la variation de la puissance du signal émis par le transmetteur et de la distance qui sépare ces deux nœuds.

Free Space Propagation Model Le modèle de propagation en espace libre (*Free Space Propagation Model*) est utilisé pour déterminer la puissance du signal reçu lorsqu'un émetteur et un récepteur se trouvent dans un environnement totalement dépourvu d'obstacles. Ce modèle est notamment utilisé pour caractériser une communication entre une station terrestre et un satellite.

La puissance reçue au niveau d'un nœud est calculée avec la formule de Friis suivante :

$$P_r(d) = \frac{P_t G_t G_r \lambda^2}{(4\pi)^2 d^2 L}$$

où P_t représente la puissance de l'émetteur, G_t et G_r le gain des antennes de l'émetteur et du récepteur, d la distance entre l'émetteur et le récepteur, λ la longueur d'onde du signal et L le facteur de perte du système. Pour un système sans perte, c'est à dire un système où le matériel ne génère pas de pertes, la valeur de L est égale à 1. Le gain d'une antenne est fourni par son fabriquant dans le manuel d'utilisation, généralement en décibels (dB).

Les pertes de signal (que nous appellerons *path loss* dans la suite) s'expriment sous la forme :

$$PL = \frac{P_t}{P_r} = \frac{(4\pi)^2 d^2 L}{G_t G_r \lambda^2}$$

Cependant, cette formule, dite formule de Friis, n'est pas applicable pour les petites valeurs de d, si bien qu'il est nécessaire de prendre une valeur d_0 de référence pour déterminer la puissance reçue. Cette valeur $P(d_0)$ peut être calculée en fonction d'un d_0 fixé ou bien mesurée à partir d'un réseau réel. La formule de Friis devient alors :

$$P_r(d) = P_r(d_0) \left(\frac{d_0}{d}\right)^2$$

Pour des réseaux sans fil dans la bande des 2.4 Ghz les valeurs de références qui sont généralement retenues pour d_0 sont de 1 m pour une communication en intérieur et de 100 m ou 1 km pour des communications en extérieur.

Il faut également noter que les formules liant P_r ou PL sont souvent données en dB :

$$P_r[\text{dB}] = P_t[\text{dB}] - PL[\text{dB}]$$

avec :

$$x[\text{dB}] = 10 \log_{10} x$$

Two-Ray Ground Reflection Model Le modèle de Friis permet de modéliser la propagation dans un environnement libre d'obstacles, par conséquent, il n'est pas très réaliste pour caractériser une communication entre deux nœuds terrestres. En effet pour caractériser une communication terrestre il faut au moins tenir compte des effets de réflection dus aux rebonds de l'onde sur le sol.

Le modèle *Two-Ray Ground Reflection* considère à la fois le signal reçu en ligne directe et une réflexion du signal sur le sol. Un exemple est présenté sur la figure 4.4.

En tenant compte du rebond de l'onde et de la position de l'émetteur et du récepteur par rapport au sol on obtient la formule suivante :

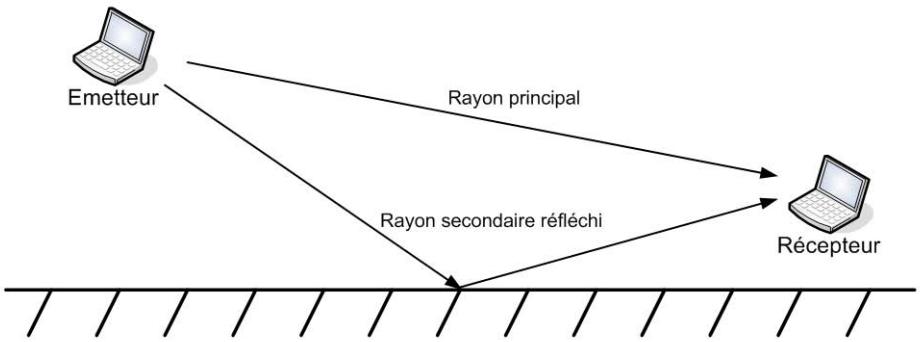


FIG. 4.4: Exemple d'utilisation d'un modèle de *Two-Ray Ground*

$$P_r = \frac{P_t G_t G_r h_t^2 h_r^2}{d^4 L}$$

Nous voyons sur cette formule que la puissance du signal reçu va diminuer beaucoup plus rapidement en fonction de la distance que dans le cas du modèle de *Free Space*. Le *path loss* résultant sera beaucoup plus important que précédemment et il se calcule grâce à la formule :

$$PL = \frac{d^4 L}{G_t G_r h_t^2 h_r^2}$$

Log-distance Path Loss Model Ce modèle de *path loss* s'appuie sur le fait que des résultats théoriques ainsi que des mesures pratiques ont montré que la puissance reçue par un nœud diminue de manière logarithmique en fonction de la distance et ceci aussi bien pour une transmission en intérieur qu'en extérieur. En se basant sur ces résultats la formule de *path loss* retenue pour ce modèle est la suivante :

$$PL[\text{dB}] = \overline{PL(d_0)}[\text{dB}] + 10n \log\left(\frac{d}{d_0}\right) \tag{4.1}$$

où n est un exposant qui va varier en fonction de l'environnement et $\overline{PL(d_0)}$ est une estimation empirique ou théorique du *path loss* en décibels pour une distance d_0. Les différentes valeurs de cet exposant peuvent être obtenues à partir de la littérature et sont résumées dans le tableau 4.1. Un moyen classique pour obtenir la valeur d'un exposant est de capturer des traces sur un réseau sans fil réel et d'effectuer sur celles-ci une régression linaire.

Il faut également faire attention dans ce modèle à bien relier l'exposant utilisé avec la valeur de d_0. Ainsi pour modéliser un environnement en espace libre, on prendra $n = 2$ et $d_0 = 100m$ ou $d_0 = 1km$.

Autres modèles à grande échelle Au delà des modèles présentés dans les sections précédentes, d'autres modèles ont été proposés dans la littérature et permettent de tenir compte de différents aspects supplémentaires pouvant altérer la transmission du signal sur de grandes

Environnement	Path loss exponent, n
En espace libre	2
En zone urbaine	2.7 - 3.5
En zone urbaine avec de la diffraction	3 - 5
Dans un immeuble sans obstacles	1.6 - 1.8
Dans un immeuble avec obstacles	4 - 6
En intérieur avec des obstacles	2 - 3

TAB. 4.1: Valeurs de l'exposant n pour le modèle de Path Loss Exponent

distances comme par exemple la courbure de la terre, la présence de végétation ou d'obstacles de grande taille (bâtiments, montagnes...). Cette prise en compte se fait au moyen de mesures statistiques comme dans le cas du *Log-distance Path Loss Model*. Ces modèles ne sont pas détaillés ici mais sont décrits dans [68].

4.3.3.2 Modèles de propagation à moyenne échelle

Les effets à moyenne échelle tiennent compte de la distance mais également des effets qui peuvent survenir à cause des obstacles présents entre émetteur et récepteur. En effet les modèles à large échelle que nous avons présentés supposent que l'émetteur et le récepteur ne sont séparés par aucun obstacle ou que ces obstacles sont répartis uniformément, ce qui est rarement le cas dans la réalité (surtout dans le cas de communications à l'intérieur d'un bâtiment). Les modèles à moyenne échelle permettent donc de fournir une première réponse pour gérer les obstacles.

Log-normal Shadowing Ce modèle est une extension du *Log-distance Path Loss*, qui fournit une première estimation de la puissance reçue en fonction de la distance d mais aussi de la nature et de la quantité d'obstacles présents dans l'environnement de l'émetteur et du récepteur, à travers l'exposant n. Cependant, le *Log-distance Path Loss Model* ne tient pas compte du fait que pour une même distance source-destination d les effets de l'environnement peuvent changer considérablement d'un endroit à l'autre, par exemple du fait de la disposition des obstacles sur le terrain.

Pour introduire ce type de variation, le modèle de *Log-normal Shadowing* ajoute à un modèle de *path loss* exprimé en dB une variable aléatoire X_σ qui suit une distribution gaussienne de moyenne 0 et de déviation standard σ. Cette variable aléatoire s'exprime elle aussi en dB ; si on ne l'exprime pas en dB elle apparaît comme un facteur dans le rapport P_t/P_r : c'est ce facteur qui suit une distribution log-normale. Ce modèle s'appuie généralement sur le modèle de *Log-distance Path Loss*, mais ce n'est pas une obligation. La formule 4.1 devient dans ce cas :

$$PL(d)\,[dB] = \overline{PL(d_0)}\,[dB] + 10n \log\left(\frac{d}{d_0}\right) + X_\sigma$$

Technique de lancer de rayons L'introduction de la variable aléatoire X_σ permet de faire varier les conditions entre deux environnements possédant des propriétés similaires (même matériaux de construction par exemple). Cependant, cette solution ne permet pas d'exploiter

une connaissance précise des obstacles. Or, la position exacte et le nombre de ces obstacles vont avoir un impact sur l'atténuation qui sera ressentie au niveau du récepteur. Pour améliorer la prise en charge de ces obstacles la technique de lancer de rayons a donc été retenue. Cette technique permet en lançant des rayons en direction de l'émetteur d'avoir une modélisation beaucoup plus précise de l'atténuation qui sera ressentie au niveau du récepteur : les rayons vont prendre différents chemins pour rejoindre la destination. Certains iront en ligne droite s'il n'y a pas d'obstacle tandis que d'autres, au contraire, rebondiront sur les murs, plafonds et planchers. A chaque rebond, une partie du signal peut être absorbée. Au niveau de la destination, l'ensemble des rayons arrivera avec une énergie différente et une phase différente, et c'est en combinant ces signaux reçus que l'on calcule la puissance reçue.

Cette technique permet donc d'avoir des résultats très précis mais par contre, elle est relativement difficile à mettre en œuvre car elle nécessite de modéliser le monde en trois dimensions pour pouvoir effectuer le lancer de rayons. Elle est également très coûteuse en temps de calcul.

4.3.3.3 Modèles de propagation à petite échelle

La technique de lancer de rayons permet donc d'avoir une connaissance précise de son environnement et du chemin que va suivre le signal pour atteindre sa destination. Cependant du fait de sa complexité de mise en œuvre, cette solution n'est que très rarement utilisée. C'est pour cette raison que des modèles probabilistes pouvant tenir compte de manière statistique des chemins multiples que peut prendre le signal radio pour atteindre sa destination ont été proposés. Ces chemins multiples qui sont dus aux rebonds de l'onde sur les obstacles présents dans l'environnement vont interférer entre eux et vont entraîner de fortes variations du signal au niveau du nœud récepteur. Ces variations vont être beaucoup plus importantes et plus fréquentes que celles qui peuvent être observées avec des modèles à grande et moyenne échelle (lancer de rayons mis à part). On parlera ici de modèles de *fading*.

Le fading de Rayleigh L'utilisation d'un modèle de Rayleigh est recommandée lorsque l'émetteur et le récepteur ne sont pas en vue directe. La théorie montre en effet que s'il n'existe pas de chemin direct entre émetteur et récepteur, et si les signaux suivant des chemins multiples arrivent avec des angles d'incidence répartis uniformément entre 0 et 2π, alors l'enveloppe r du signal reçu suit une distribution de Rayleigh de paramètre σ.

Une variable aléatoire r suivant une distribution $Rayleigh(\sigma)$ a pour densité de probabilité :

$$\begin{aligned} p(r) &= \frac{r}{\sigma^2}\ \exp\left(-\frac{r^2}{2\sigma^2}\right) && \text{si } r \geq 0, \\ &= 0 && \text{si } r < 0. \end{aligned}$$

et pour espérance :

$$E[r] = \sigma\sqrt{\frac{\pi}{2}} \tag{4.2}$$

Le *fading* de Rayleigh consiste à estimer une puissance reçue, par exemple en utilisant le *Log-distance Path Loss Model* combiné à un *Log-normal Shadowing Model*, à en déduire une estimation de l'enveloppe du signal reçu, puis à faire varier l'enveloppe suivant une distribution de Rayleigh. On se sert ensuite de l'enveloppe obtenue pour recalculer la puissance reçue. La puissance d'un signal P et son enveloppe r sont en effet liées par la relation suivante :

$$P = \frac{r^2}{2Z_0} \tag{4.3}$$

où $Z_0 = 120\pi \simeq 377\,\Omega$ est l'impédance caractéristique du vide.

Soit P_r^ℓ la puissance calculée avec un modèle de propagation à large et/ou moyenne échelle. On obtient alors une enveloppe moyenne r^ℓ :

$$r^\ell = \sqrt{2Z_0 P_r^\ell}$$

Cette enveloppe moyenne r^ℓ est égale à l'espérance de la v.a. r, ce qui permet de calculer le paramètre σ de la distribution de Rayleigh dans laquelle on va tirer aléatoirement une valeur de r qui prendra en compte les effets du *fading* aussi bien que les effets à grande et moyenne échelle :

$$\sigma = r^\ell \sqrt{\frac{2}{\pi}} = \sqrt{\frac{4Z_0 P_r^\ell}{\pi}}$$

Pour obtenir une v.a. r suivant une distribution $Rayleigh(\sigma)$, on peut utiliser deux v.a. X_σ et Y_σ suivant une loi normale $N(0, \sigma)$:

$$r = \sqrt{X_\sigma^2 + Y_\sigma^2}$$

Une fois obtenu un tel r, on obtient la puissance reçue en tenant compte du *fading* en réutilisant l'équation 4.3.

Le fading de Rice Contrairement au *fading* de Rayleigh qui est utilisé lorsque deux nœuds ne sont pas à portée visuelle l'un de l'autre, l'utilisation du *fading* de Rice est recommandée dans le cas où ils se trouvent en vue directe. Dans ce cas, le signal reçu directement est beaucoup plus fort que le signal reçu par les chemins indirects. Si on note A l'amplitude de crête du signal dominant, et σ le paramètre de Rayleigh caractérisant les signaux reçus par chemins multiples, alors l'enveloppe du signal reçu r suit une distribution de Rice, notée $Rice(A, \sigma)$, ayant pour densité de probabilité :

$$\begin{aligned} p(r) &= \frac{r}{\sigma^2}\, I_0\left(\frac{Ar}{\sigma^2}\right) \exp\left(-\frac{(r^2 + A^2)}{2\sigma^2}\right) && \text{si } r \geq 0, \\ &= 0 && \text{si } r < 0. \end{aligned}$$

où I_0 est la fonction de Bessel modifiée de première espèce et d'ordre 0.

On spécifie généralement à la place de A un paramètre K constant, défini comme le rapport entre la puissance du signal reçu par le chemin dominant et la variance des chemins multiples :

$$K = \frac{A^2}{2\sigma^2}$$

ou en dB :

$$K\,[\text{dB}] = 10 \log\left(\frac{A^2}{2\sigma^2}\right)$$

En introduisant K, l'espérance d'une v.a. r suivant une distribution $Rice(\sigma\sqrt{2K}, \sigma)$ s'écrit :

$$E[r] = \sigma\sqrt{\frac{\pi}{2}}e^{-K/2}\left[(1+K)I_0\left(\frac{K}{2}\right) + KI_1\left(\frac{K}{2}\right)\right] \tag{4.4}$$

où I_0 et I_1 sont les fonctions de Bessel modifiées de première espèce et d'ordre 0 et 1. A partir de K et d'une estimation de la puissance reçue à large échelle P_r^ℓ puis de l'enveloppe r^ℓ, on peut donc déduire les paramètres σ et A d'une distribution de Rice dans laquelle on va tirer les variations de l'enveloppe r dues au *fading* avec vue directe :

$$\sigma = r^\ell \frac{e^{K/2}\sqrt{2/\pi}}{(1+K)\ I_0(K/2) + K\ I_1(K/2)}$$
$$A = \sigma\sqrt{2K}$$

Pour obtenir une v.a. r suivant une distribution $Rice(A, \sigma)$ on peut utiliser deux v.a. X_σ et Y_σ suivant une loi normale $N(0, \sigma)$:

$$r = \sqrt{(X_\sigma + A)^2 + Y_\sigma^2}$$

Il est à noter que le *fading* de Rice peut dégénérer en fading de Rayleigh lorsque $A \to 0$ (et donc $K \to -\infty$), ce qui correspond à une dégradation du signal dominant au profit du bruit induit par les chemins multiples.

Une présentation approfondie de ces formules est donnée dans le livre de Rappaport [68] dont elles sont extraites.

4.3.4 Modèles de communication

Les modèles de mobilité permettent donc de déterminer la façon dont les différents nœuds composant le réseau sans fil vont se déplacer au cours d'une expérience et les modèles de propagation permettent de déterminer la puissance du signal radio reçu au niveau de chaque nœud en fonction de chaque nœud émetteur. En se basant sur cette puissance reçue il est possible de mettre en œuvre un modèle de communication pour déterminer le taux de transmission radio dont le nœud disposera au niveau physique. Grâce à ce taux de transmission, il est ensuite possible de déterminer la Qualité de Service de niveau IP qui sera observée par l'utilisateur. Ces modèles de communication qui vont affecter la qualité de service observée par l'utilisateur dépendent de la position des terminaux et de la qualité du signal reçu (modèles de niveau physique) mais peuvent également dépendre du trafic qui peut circuler sur le réseau. Ainsi, le comportement des modèles de niveau MAC notamment et de manière plus générale le comportement des modèles tenant compte d'une communication effective entre deux nœuds va dépendre du trafic qui circule à un instant donné sur le réseau. Pour présenter les différents modèles de communication, nous nous sommes interessés principalement à la technologie IEEE 802.11b/g mais ceci n'est pas une contrainte forte. Les principes que nous allons présenter peuvent en effet être étendus à n'importe quelle technologie.

4.3.4.1 Modèles de communication indépendants du trafic

Les modèles de communication indépendants du trafic sont principalement des modèles permettant de représenter des comportements de niveau physique. Ils vont notamment être

utilisés pour déterminer le taux de transmission radio disponible au niveau physique entre deux nœuds car celui-ci sera le même qu'une communication ait lieu ou pas. Il n'est pas nécessaire de prendre en compte le partage du canal dû au trafic susceptible de circuler sur le réseau pour pouvoir le déterminer. Seule la distance et/ou la puissance du signal reçue doivent être considérées pour déterminer la modulation correspondant au taux de transmission radio recherché.

Évaluation du taux de transmission en fonction de la distance Une méthode très simple pour pouvoir déterminer le taux de transmission radio est de déterminer la modulation qui est utilisée au niveau physique. Pour cela, il est possible de s'appuyer uniquement sur la distance séparant l'émetteur et le récepteur. En effet, les constructeurs de matériels réseaux sans fil fournissent généralement une table de correspondance semblable à la table 4.2 qui permet de déterminer la modulation utilisée et donc d'évaluer le taux de transmission en fonction de l'environnement dans lequel le réseau sans fil se trouve (intérieur ou extérieur).

Taux de transmission (Mbps)	Modulation	Distance en intérieur (m)	Distance en extérieur(m)
1	DSSS/DBPSK	140	290
2	DSSS/DQPSK	136	287
5.5	DSSS/CCK	130	277
6	OFDM/BPSK	125	274
9	OFDM/BPSK	116	267
11	DSSS/CCK	111	250
12	OFDM/QPSK	108	244
18	OFDM/QPSK	100	229
24	OFDM/16-QAM	87	198
36	OFDM/16-QAM	79	168
48	OFDM/64-QAM	55	107
54	OFDM/64-QAM	32	37

TAB. 4.2: Taux de transmission radio en fonction de la distance pour un point d'accès CISCO pour les modes IEEE 802.11b/g [13].

Cette solution permet d'obtenir très rapidement des résultats et permet de s'affranchir des problèmes de propagation mais elle n'est pas satisfaisante au niveau du réalisme dans la mesure où les effets de transmission radio à petite échelle ne sont pas pris en compte. Ainsi lorsqu'un nœud se déplace, il voit sa bande passante varier avec très peu de variations au cours du temps, ce qui n'est pas un comportement réaliste surtout lorsque ce nœud se trouve éloigné du nœud avec lequel il communique.

Évaluation du taux de transmission en fonction de la puissance reçue Pour améliorer le réalisme et tenir compte des effets de propagation à petite échelle, il est possible de s'appuyer sur les données techniques fournies par les constructeurs indiquant le taux de transmission radio en fonction de la puissance reçue. Pour garder un bon niveau de précision, cette puissance reçue aura été calculée grâce à l'utilisation de modèles de propagation réalistes. En particulier une modélisation correcte du *fading* permet de faire apparaître les variations de la puissance reçue à petite échelle.

Ce taux de transmission radio servira ensuite à déterminer la bande passante théorique qui sera fournie à l'utilisateur au niveau IP. Ces données techniques peuvent être retrouvées dans le tableau 4.3 qui donne les correspondances entre la puissance reçue et le taux de transmission radio de niveau physique que l'on peut utiliser selon que l'on utilise une carte réseau sans fil 802.11b/g CISCO [13] ou une carte réseau sans fil 802.11b/g D-Link [20]. Notez que la diminution de la puissance reçue pour le D-Link entre 5.5 Mbps et 6 Mbps vient du fait que cet appareil tolère un taux d'erreurs paquet (*Packet Error Rate*) de 8% pour le mode 802.11b contre 10% pour le mode 802.11g.

Taux de transmission (Mbps)	$P_r(dBm)$(CISCO Aironet)	$P_r(dBm)$(D-Link DI-624+)
1 (DSSS/DBPSK)	-96	-92
2 (DSSS/DQPSK)	-93	-90
5.5 (DSSS/CCK)	-91	-87
6 (OFDM/QPSK)	-91	-89
9 (OFDM/BPSK)	-85	-87
11 (DSSS/CCK)	-88	-85
12 (OFDM/QPSK)	-83	-86
18 (OFDM/QPSK)	-81	-84
24 (OFDM/16-QAM)	-78	-80
36 (OFDM/16-QAM)	-74	-77
48 (OFDM/64-QAM)	-73	-72
54 (OFDM/64-QAM)	-73	-71

TAB. 4.3: Table de correspondance entre le taux de transmission radio et la puissance reçue

Les pertes au niveau IP Dans un réseau sans fil 802.11 en mode unicast, les pertes sont prises en compte au niveau MAC grâce à un acquittement systématique des paquets. Donc, si une perte est détectée, le paquet est ré-émis par la source et ce, jusqu'à ce que le nombre d'échecs devienne trop grand. Passé un certain nombre d'échecs le mécanisme d'*auto-rate fallback* est mis en œuvre si bien qu'au niveau IP les pertes ne sont ressenties que lorsque l'on atteint un taux de transmission de 1 Mbps.

Cependant, ce mécanisme d'acquittement n'est présent que pour des communications unicast. Dans le cas de communications multicast il est nécessaire de modéliser les pertes de manière plus réaliste y compris pour les taux de transmission supérieur à 1 Mbps. En fonction de la puissance reçue au niveau d'un nœud, il est possible de déterminer les instants t où cette puissance reçue sera inférieure au seuil nécessaire pour avoir le taux de transmission désiré : les paquets envoyés vers ce nœud à cet instant t seront alors perdus.

Le routage Une fois que les différents seuils de transmission radio sont fixés il est possible de mettre en œuvre un algorithme de routage pour déterminer les communications possibles entre les nœuds. En effet comme nous nous proposons de faire de l'émulation de niveau IP, il ne sera pas possible de déployer réellement un protocole de routage sur la plate-forme d'émulation. Il est donc nécessaire de déterminer les effets de ces protocoles de routage au niveau IP de manière à pouvoir les émuler. Le premier effet qu'il faut reproduire est, bien entendu, de déterminer les différentes communications qui seront possibles entre les nœuds. Pour cela, une

solution consiste à mettre en œuvre un algorithme de recherche de plus court chemin. Cet algorithme va prendre en compte les taux de transmission radio qui auront été pré-calculés au préalable et déterminer les chemins possibles entre chaque paire de nœuds du réseau et ce, pour chaque instant t de l'expérience. Éventuellement, il faut également tenir compte de paramètres spécifiques au protocole de routage comme par exemple le niveau d'énergie des nœuds pour le protocole de routage PAR [73] dont nous avons parlé dans la section 2.5.2.

4.3.4.2 Modèles de communication dépendants du trafic

La propagation permet donc de déterminer la modulation et le taux de transmission théorique dont un nœud peut disposer au niveau physique. Cependant, ce taux de transmission ne sera peut être pas celui dont il disposera réellement au moment où la communication aura lieu. En effet, le trafic qui circule sur le réseau va influencer directement le débit utile dont va réellement disposer le nœud. Par exemple, si plusieurs nœuds émettent simultanément, les mécanismes de *backoff* et de *defering* présentés dans la section 2.6 vont entrer en jeu. Ces mécanismes sont importants car ils ont un impact direct sur le débit utile ressenti au niveau IP. De même, si plusieurs nœuds émettent simultanément vers une même destination sans avoir connaissance les uns des autres, des perturbations du signal au niveau du nœud destinataire vont survenir (cas des terminaux cachés). Ce genre de problème est donc dépendant de deux facteurs : de la topologie du réseau à un instant donné et du trafic qui circule sur le réseau à cet instant.

Dans l'approche d'émulation qui nous intéresse, l'un des intérêts par rapport à de la simulation classique est de ne pas avoir besoin de modèle de trafic. Si l'on privilégie la liberté de ne pas disposer de modèle de trafic, il n'est pas toujours possible de prendre en compte ces phénomènes lors d'une simulation partielle hors ligne. Dans certains cas, il est possible de faire des estimations de valeurs comme pour l'évaluation du débit utile au niveau IP mais ceci n'est pas envisageable pour des problèmes directement liés au trafic. En effet, comme nous ne disposons pas de modèle de trafic, il est nécessaire d'attendre la phase d'émulation pour observer le trafic qui circule sur le réseau émulé et pouvoir reproduire le comportement que l'on désire. Dans le cas de terminaux cachés, cela se traduira par une introduction de pertes ou de délais qui correspond aux perturbations du signal ressenties par le nœud récepteur. Cependant, il est possible de pré-calculer certains aspects pour limiter le nombre de calculs à effectuer durant l'émulation comme nous allons le voir sur les quelques exemples suivants : la gestion des terminaux cachés dans un réseau en mode Ad-Hoc et la gestion de la communication dans une cellule dans un réseau en mode infrastructure. Mais dans un premier temps, penchons nous sur le problème de l'évaluation du débit utile au niveau IP.

Le débit maximum théorique au niveau IP En se basant sur le taux de transmission radio obtenu à l'aide d'un des modèles précédents, il est possible de déterminer le débit théorique disponible au niveau IP pour une application répartie ou un protocole de transport. En effet comme l'ont montré les travaux de Jun et al présentés dans [44], et comme nous l'avions déjà signalé dans la section 2.6.2.4, le débit maximum théorique ressenti au niveau IP est différent du taux de transmission radio. Ceci est principalement dû aux protocoles d'accès au canal qui obligent à attendre avant d'émettre un paquet pour éviter les collisions.

Ainsi, avant d'émettre un paquet sur un réseau 802.11b, il va falloir attendre pendant une durée incompressible DIFS puis attendre encore pendant un *backoff* de durée aléatoire. Une fois le paquet émis, il faut attendre l'acquittement renvoyé par la destination après une durée

constante SIFS. Lorsqu'un mécanisme de RTS/CTS est utilisé, ce temps d'attente entre chaque émission de paquet est encore plus long. Ce mécanisme d'accès au canal 802.11, résumé sur la figure 4.5, a été présenté en détail dans la section 2.6.2.4. Il faut cependant remarquer que les délais que nous allons maintenant présenter prennent l'hypothèse qu'il n'y a pas de collisions dans le réseau. L'autre hypothèse est que le réseau 802.11b fonctionne en mode Ad-Hoc ou en mode Infrastructure à la condition qu'il ne communique pas avec un nœud situé dans la même cellule. Nous verrons un peu plus loin que le délai de propagation est beaucoup plus important lorsque la communication a lieu entre deux nœuds situés dans la même cellule.

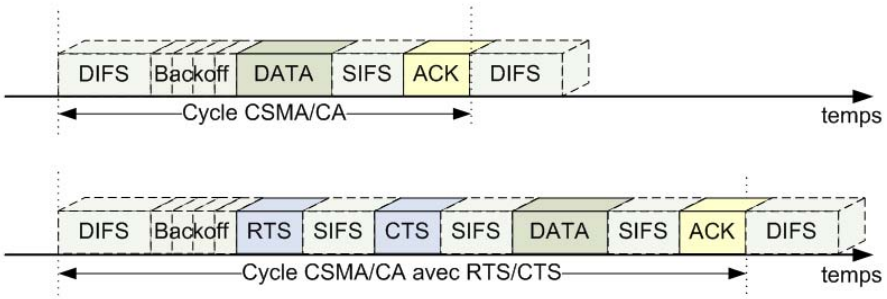


FIG. 4.5: Le cycle temporel CSMA/CA avec et sans RTS/CTS

Chaque paquet émis dans un réseau 802.11 en mode Ad-Hoc n'utilisant pas de mécanisme RTS/CTS, va donc subir un délai d donné grâce à la formule :

$$d = (T_{DIFS} + T_{SIFS} + T_{backoff} + T_{DATA} + T_{ACK}) * 10^{-6}$$

Par contre, lorsque le mécanisme de RTS/CTS est utilisé, le délai ressenti va augmenter. Il est donné par la formule :

$$d = (T_{DIFS} + 3 * T_{SIFS} + T_{backoff} + T_{RTS} + T_{CTS} + T_{DATA} + T_{ACK}) * 10^{-6}$$

où d est exprimé en seconde.

Dans ces deux formules, les durées T_{DIFS}, T_{SIFS}, T_{ACK}, T_{RTS} et T_{CTS} sont constantes. Un rappel de leurs valeurs pour une technologie 802.11b est donné dans la table 4.4. Dans cette table nous avons considéré que les paquets de contrôle étaient émis à un taux de transmission radio de 1 Mbps avec des en-têtes physiques longs.

Le temps de *backoff* dépend du trafic qui circule sur le réseau et varie en fonction des collisions qui y surviennent. Une solution lorsque l'on ne dispose pas d'un modèle de trafic précis pour évaluer ces collisions est de prendre une valeur moyenne de *backoff* c'est-à-dire la valeur moyenne de l'intervalle $[0, CW_{min}]$, soit 310 μs.

En considérant cette valeur moyenne de *backoff*, on peut en déduire que lors de chaque communication il y aura un délai inter-paquets d_i de :

Paramètre	Durée (μs)	Taille
SIFS	10	N/A
DIFS	50	N/A
backoff (slot)	20	N/A
CW_{min}	620	31 slots
En-tête physique long	192	192 bits
En-tête physique court	96	96 bits
RTS	352	160 bits
CTS	304	112 bits
ACK	304	112 bits

TAB. 4.4: Valeurs retenues dans la norme 802.11b avec des messages de contrôle échangés à 1 Mbps

$$\begin{aligned} d_i &= (T_{SIFS} + T_{DIFS} + T_{ACK} + T_{backoff}) \\ &= 10 + 50 + 304 + 310 \\ &= 674\,\mu s \end{aligned}$$

et, lorsque le mécanisme RTS/CTS est utilisé, de :

$$\begin{aligned} d_i &= (3 * T_{SIFS} + T_{DIFS} + T_{ACK} + T_{backoff} + T_{RTS} + T_{CTS}) \\ &= 30 + 50 + 304 + 310 + 352 + 304 \\ &= 1350\,\mu s \end{aligned}$$

Lors de l'émission de chaque paquet, à la taille des données émises au niveau IP est ajouté un en-tête de niveau MAC de 272 bits émis à la même vitesse que celles-ci. En connaissant le taux de transmission et la taille maximum d'un paquet émis, il est donc possible de déterminer le temps que va mettre une trame pour être transmise sur le réseau et de déterminer le débit maximum disponible au niveau IP. La courbe présentée sur la figure 4.6, présente l'évolution de ce débit théorique au niveau IP en fonction d'un taux de transmission radio de 11 Mbps. Sur cette courbe nous avons également considéré un seuil de fragmentation équivalent à celui que l'on retrouve sur un réseau Ethernet (1500 octets).

Jusqu'à présent, nous avons considéré qu'il n'y avait pas de collisions dans le réseau et que le médium était toujours libre au moment de l'émission mais dans le cas d'un réseau 802.11 en mode Infrastructure cela n'est pas possible. En effet, toutes les transmissions ont lieu entre un nœud et le point d'accès chargé de gérer la cellule. Donc, si le nœud destination se trouve dans la même cellule que le nœud émetteur, la communication entre le nœud émetteur et le point d'accès va entrer en concurrence avec la communication entre le point d'accès et le nœud destination.

Lorsqu'un nœud va chercher à émettre il va donc souvent devoir différer son émission et subir encore plus de délais que précédemment. Il devra maintenant attendre un T_{SIFS}, un T_{DIFS}, un T_{ACK} et un T_{DATA} supplémentaire à chaque émission d'un nouveau paquet (le

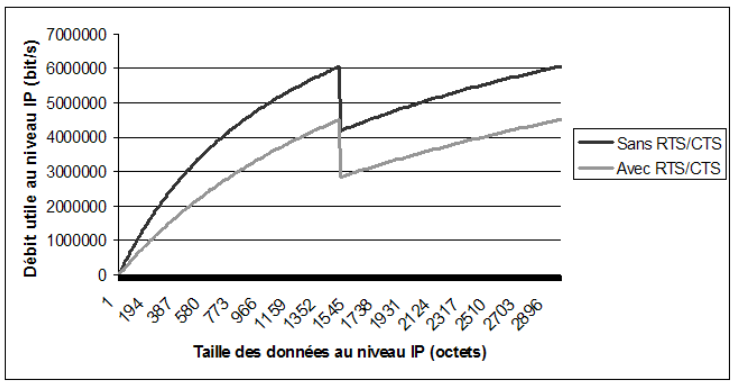


FIG. 4.6: Évolution du débit théorique au niveau IP en fonction de la taille des paquets pour un taux de transmission radio de 11 Mbps

second T_{DATA} correspond à la communication entre le point d'accès et le nœud destination). Dans le cas d'un réseau en mode Infrastructure, le délai de transmission d_{Infra} est donc de :

$$d_{Infra} = (2 * T_{SIFS} + 2 * T_{DIFS} + 2 * T_{ACK} + 2 * T_{DATA} + T_{backoff}) * 10^{-6}$$

le débit utile au niveau IP est donc beaucoup plus faible comme nous pouvons le voir sur la figure 4.7.

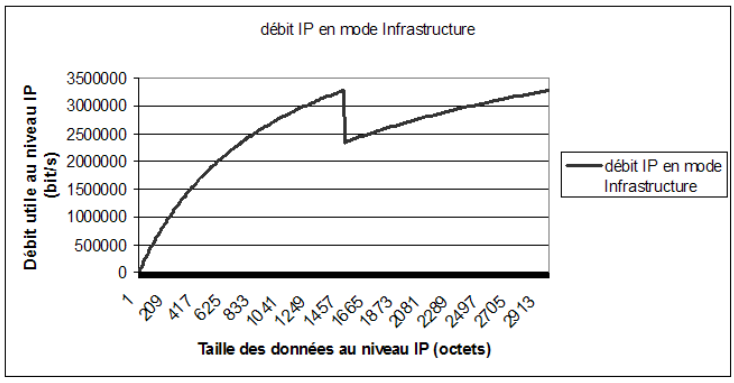


FIG. 4.7: Évolution du débit théorique au niveau IP en fonction de la taille des paquets dans un réseau en mode Infrastructure pour un taux de transmission radio de 11 Mbps

Grâce à ce modèle il est possible de pré-calculer le débit utile qui sera ressenti au niveau IP. Ce modèle dépend du trafic mais il n'a pas besoin d'avoir une connaissance très précise du modèle de trafic. Il lui suffit en effet de connaître la taille des paquets échangés pour déterminer le débit utile. D'autres modèles au contraire sont directement liés au trafic qui circule à un instant donné sur le réseau. Ce sont ces modèles que nous allons maintenant présenter.

Gestion des terminaux cachés Le cas des terminaux cachés, présenté sur la figure 4.8, ne survient que lorsque les nœuds M_1et M_3 émettent simultanément vers M_2. Comme nous ne disposons pas de modèle de trafic nous ne pouvons pas prévoir quand les signaux vont entrer en concurrence et générer du bruit au niveau du nœud récepteur. Cependant, grâce aux modèles

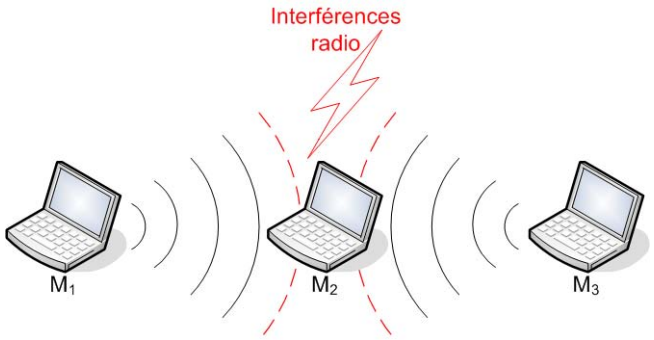


FIG. 4.8: Exemple de cas de terminaux cachés

de mobilité et de propagation il est possible de connaître la topologie du réseau à chaque instant t de l'expérience. En se basant sur cette topologie globale, il est possible de déterminer les triplets de nœuds susceptibles de produire un cas de terminaux cachés si aucun protocole de type RTS/CTS n'est utilisé. Une fois que les triplets à risque sont identifiés il est possible d'appliquer un modèle permettant de reproduire les effets des interférences de collision au niveau du nœud récepteur et de générer les règles de QdS à appliquer pendant l'émulation. Pour chaque triplet on pourra donc générer deux règles à chaque instant t : l'une correspondant au cas où un seul nœud émet vers M_2 et l'autre correspondant au cas où les deux nœuds cachés M_1 et M_3 émettent simultanément vers M_2. Les informations sur les triplets sous surveillance sont ensuite passées à la plate-forme d'émulation qui va surveiller le trafic émis par M_1 et M_3 et choisir les règles à appliquer en fonction de ce trafic.

Gestion de la communication dans une cellule Dans le cas des réseaux en mode infrastructure (ou réseaux cellulaires), les travaux de Heusse *et al* [34] ont mis en avant que, indépendamment du taux de transmission dont peut disposer un nœud, la bande passante effective dont il disposera au niveau IP sera égale à celle du nœud émettant avec le taux de transmission radio le plus faible. Une fois encore il est impossible de déterminer précisément le scénario d'émulation sans disposer d'un modèle de trafic. Il faut donc introduire la notion de cellule dans le simulateur et déterminer les membres de chaque cellule à chaque instant t de l'expérience. Une fois que les cellules sont déterminées, il est possible d'effectuer un classement entre les nœuds en fonction de leur taux de transmission radio. Ainsi, les nœuds ayant le taux de transmission radio le plus faible pourront affecter la bande passante des nœuds disposant à priori d'un meilleur taux de transmission. Les informations sur les cellules doivent ensuite être fournies à la plate-forme d'émulation pour que celle-ci puisse appliquer les règles de QdS correspondant au trafic qui circule au cours d'une expérience.

4.3.5 Conclusion

L'utilisation d'une solution d'émulation reposant sur une phase de simulation préalable va permettre de diminuer le nombre de calculs à effectuer en temps réel sur un pas de temps donné, répondant ainsi aux besoins de précision et de dynamisme que nous avions évoqués. La phase de simulation va ainsi permettre d'utiliser des modèles de mobilité et de propagation suffisamment complexes pour tenir compte de l'environnement et des différents obstacles. Cette prise en compte de la mobilité et de la propagation permet de déterminer les différentes communications

qui peuvent avoir lieu entre les nœuds composant le réseau sans fil. En particulier, on peut déterminer la qualité de service optimale observable à chaque instant. De même, il est possible de prendre partiellement en compte les comportements directement liés au trafic qui circule sur le réseau en déterminant les nœuds à observer durant la phase d'émulation. En générant différentes conditions d'émulation, chacune correspondant à un cas de trafic, on limite le nombre de calcul à effectuer en temps réel. Il ne reste en effet à l'émulateur qu'à observer le trafic qui circule au cours d'une expérience, à déterminer le cas de trafic correspondant et à appliquer les conditions correspondantes.

4.4 Scénarisation des services

4.4.1 Besoin de scénarisation

La solution que nous avons retenue est donc d'avoir une phase de simulation préalablement à la phase d'émulation pour pouvoir utiliser des modèles réalistes qui sont souvent coûteux en temps de calcul. Pour pouvoir passer de cette phase de simulation hors ligne à la phase d'émulation qui va se dérouler en temps réel, il a été nécessaire de définir un format d'échange. Une solution consiste à générer les résultats de la simulation sous forme de fichier binaires et à les relire durant la phase d'émulation. Cependant, les fichiers binaires présentent l'inconvénient de ne pas être lisibles par l'être humain et ils ne peuvent pas être validés de manière simple si bien que le passage à la phase d'émulation peut soulever des problèmes. L'utilisation de fichier texte n'étant pas non plus une solution satisfaisante, nous nous sommes tournés vers *XML (eXtended Markup Language)* le format de description défini par le consortium *W3C (World Wide Web Consortium)* qui s'est maintenant imposé comme un standard de fait.

Le fait d'utiliser XML pour écrire les scénarios d'émulation permet d'avoir des fichiers compréhensibles par un être humain et de pouvoir les valider formellement à l'aide de schémas. Cet aspect de validation est très important car un utilisateur doit être assuré que le scénario qu'il va utiliser lors de la phase d'émulation est sémantiquement juste pour pouvoir placer une confiance minimale dans l'émulation qui va être rendue. Ainsi, lorsqu'un utilisateur souhaite émuler des conditions particulières, il peut écrire le scénario à la main sans passer par le simulateur et le valider grâce à l'utilisation des schémas qui lui sont fournis. De plus cette solution permet d'étendre l'utilisation de la plate-forme d'émulation en ne la limitant pas aux réseaux locaux sans fil. Il est tout à fait envisageable de développer un simulateur produisant des scénarios caractérisant des réseaux de satellites ou des réseaux de capteurs et d'utiliser ces scénarios sur la même plate-forme d'émulation que celle que nous proposons à la condition qu'ils respectent les schémas que nous avons définis.

Dans le même ordre d'idée, la description de l'expérience que l'utilisateur doit fournir au simulateur doit pouvoir être simple à utiliser et à vérifier. La plupart des simulateurs existants tels que NS-2 ou GloMoSim reposent sur des formats de fichier propriétaires. NS-2 par exemple utilise le langage *oTCL (object Tool Command Language)* et une bibliothèque de classes prédéfinies pour décrire l'expérience. Ce script oTCL est directement lié au code développé dans le simulateur ce qui le rend inutilisable en l'état par un autre simulateur. Le langage oTCL n'est pas un langage de description mais bien un langage de programmation si bien qu'il est relativement difficile à valider et, comme tout langage de programmation, il demande une bonne connaissance du langage pour corriger les erreurs. Pour résoudre ce problème de validation et pour permettre à l'utilisateur d'avoir une solution simple pour décrire son expérience, nous avons également choisi le format XML pour décrire une expérience.

4.4.2 Le formalisme RELAX NG compact

Le formalisme *RELAX NG compact* est défini par le consortium *OASIS (Organization for the Advancement of Structured Information Standards)*. Ce formalisme est proche de celui d'une grammaire Backus-Naur, ce qui le rend beaucoup plus lisible qu'un schéma XML classique (au format XSD).

La grammaire utilisée par *RELAX NG compact* est relativement simple et se compose :

- d'éléments (notés *element*) qui vont permettre de définir les balises XML,
- d'attributs (notés *attribute*) qui permettent de définir les attributs que l'on retrouve à l'intérieur d'une balise XML.

```
- <carnet_d_adresses>
  - <contact>
      <nom>Martin</nom>
      <prenom>Pierre</prenom>
      <surnom>Pierro</surnom>
      <tel id="bureau">0561616161</tel>
      <tel id="portable">0660606060</tel>
      <mail>pierro@boulot.com</mail>
    </contact>
  - <contact>
      <nom>Dupont</nom>
      <prenom>Jaques</prenom>
      <tel id="bureau">0561616161</tel>
      <mail id="perso">jacques.dupont@yahoo.com</mail>
      <mail>jdupont@entreprise.com</mail>
    </contact>
</carnet_d_adresses>
```

(a) Exemple de carnet d'adresse en XML

```
start = carnet_d_adresses

carnet_d_adresses =
    element carnet_d_adresses {
        contact+
    }

contact = element contact {
    (
        element nom {xsd:string}&
        element prenom {xsd:string}&
        element surnom {xsd:string}?
    ),
    element tel {
        attribute id { "bureau" | "portable"},
        xsd:integer
    }+,
    element mail {
        attribute id { xsd:string }?,
        text
    }*
}
```

(b) schéma associé

FIG. 4.9: Exemple de schéma RELAX NG compact associé à un fichier XML

Supposons que nous voulions donner le schéma *RELAX NG compact* du fichier XML décrivant un carnet d'adresse qui est présenté sur la figure 4.9. Sans trop rentrer dans les détails, voici quelques éléments de syntaxe nécessaires pour la compréhension de ce schéma ainsi que ceux que nous présenterons dans la suite de ce chapitre.

- (...) : Les parenthèses permettent de grouper différents morceaux du schéma. Le nom et le prénom d'un contact sont indissociables l'un de l'autre.
- * : Ce symbole permet d'exprimer une multiplicité $0..n$. Il ne peut être utilisé que sur des éléments. Dans l'exemple qui nous intéresse cela veut dire que les adresses e-mails (*element mail*) ne sont pas obligatoires mais qu'il peut y en avoir plusieurs.
- + : Ce symbole permet d'exprimer une multiplicité $1..n$. Tout comme le symbole précédent, il ne peut être utilisé que sur des éléments. Le carnet d'adresses doit donc avoir au moins un contact et chaque contact doit avoir au moins un numéro de téléphone pour pouvoir être contacté.
- ? : Ce symbole permet d'exprimer qu'un élément ou un attribut peut être omis dans le fichier XML. Dans le carnet d'adresses, l'élément *surnom* désignant un contact n'est pas obligatoire. L'attribut *id* permettant de différencier des adresses e-mail peut également être omis.
- , : La virgule est un symbole d'association impliquant une idée d'ordre. Ainsi deux éléments ou attributs séparés par une virgule dans le schéma doivent être décrits dans le

même ordre dans le fichier XML (X,Y != Y,X). Sur l'exemple nous voyons qu'il faut donc toujours commencer un contact par le bloc nom, prénom et éventuellement le surnom, puis donner un numéro de téléphone et ensuite éventuellement une adresse e-mail.
- & : Le symbole & permet d'exprimer une association sans notion d'ordre. Ainsi deux éléments ou attributs séparés par un symbole & dans le schéma devront être présent dans le fichier XML mais leur position ne sera pas importante (X&Y = Y&X). Pour un contact il est donc possible de commencer par spécifier le surnom avant de donner le nom et le prénom.
- | : Ce symbole permet d'exprimer un choix entre deux éléments. Le téléphone a ainsi un attribut obligatoire qui peut prendre soit la valeur "bureau" soit la valeur "portable".

Enfin, le formalisme *RELAX NG compact* permet d'utiliser les types XSD pour caractériser le type d'un élément ou d'un attribut. Par exemple, le numéro de téléphone sera dans l'exemple représenté par un type *xsd :integer*. Il est ainsi possible d'utiliser dans les schémas les types xsd :ID et xsd :IDREF qui permettent d'identifier un élément puis de le référencer dans la suite du document. Cela permet par exemple dans le cas d'un modèle de poursuite que la cible est bien un nœud valide. Une présentation plus complète de ce formalisme est disponible dans le tutoriel [14].

4.4.3 Description de haut niveau d'une expérience

La figure 4.10 présente un extrait du schéma permettant de valider le fichier XML de description de l'expérience (que nous appellerons à partir de maintenant fichier de haut niveau). Ce schéma est écrit suivant le formalisme *RELAX NG compact* que nous venons de présenter.

```
Experiment = element experiment {
    # Ad-Hoc by default
    element type_of_network {
        attribute id { "Ad-Hoc" | "Managed" | "Hybrid" }
    }?,
    World,
    element models {
        NetModel+
    }
}

World = element world {
    element time {
        element duration { duration_unit? & real } &
        element step { duration_unit? & real }
    } &
    element origin { distance_unit? & tuple3d } &
    element dimensions { distance_unit? & tuple3d } &
    element obstacles { Obstacle* } &
    element areas { Area+ }? &
    element nodes { Mobile+ & AccessPoint* }
}
```

(a) schéma

```
- <experiment>
    <type_of_network id="Ad-Hoc" />
  - <world>
    - <time>
        <duration unit="s">179.5</duration>
        <step unit="s">0.5</step>
      </time>
    - <origin>
        <x>-2</x>
        <y>0</y>
        <z>-2</z>
      </origin>
    - <dimensions unit="m">
        <x>44</x>
        <y>0</y>
        <z>44</z>
      </dimensions>
    + <obstacles>
    + <nodes>
    </world>
  + <models>
  </experiment>
```

(b) exemple de fichier XML associé

FIG. 4.10: Extrait du fichier de haut niveau décrivant une expérience

Une expérience commence donc par une définition du type de réseau qui va devoir être émulé. Plusieurs possibilités sont proposées : le mode *Ad-Hoc*, le mode *Managed* qui va permettre de simuler un réseau en mode infrastructure (ou cellulaire) 802.11 et un mode *Hybrid* permettant de modéliser un réseau comportant une partie en mode infrastructure et une partie en mode Ad-Hoc. Il faut cependant noter que bien que prévu dans le schéma, ce dernier mode n'a pas

été développé.

Une fois que le type de réseau est défini, il faut décrire le monde proprement dit puis les modèles qui vont être utilisés pour déterminer les paramètres de QdS à reproduire au cours de la phase l'émulation.

Dans la partie *World*, le temps va être défini par la durée de l'expérience ainsi que par le pas de temps à utiliser durant la simulation. Viennent ensuite les dimensions du monde puis les obstacles qui sont définis par leur position, leur forme, leur taille et éventuellement leur capacité d'absorption du signal. Les *Areas* permettent de définir des zones où un *PER (Packet Error Rate)* constant est observé. Elles seront décrites plus en détails lors de la présentation du simulateur dans le chapitre suivant.

La partie *models* permet de définir les modèles qui vont être appliqués à l'ensemble du réseau, c'est-à-dire les modèles de propagation mais également les modèles de niveau MAC permettant de déterminer le débit utile IP ressenti par l'utilisateur. C'est également dans cette section que seront définis les modèles permettant de déterminer les cellules pour un réseau en mode Infrastructure ou de rechercher les cas de terminaux cachés pour un réseau en mode Ad-Hoc n'utilisant pas de mécanisme RTS/CTS. Un exemple de modèle de propagation (ici un modèle de *Path Loss Exponent*) est donné sur la figure 4.11. Dans cet exemple nous pouvons voir qu'il faut déterminer un exposant qui va dépendre de l'environnement où le réseau à émuler est déployé, la distance d_0 de référence et éventuellement la fréquence et le *path loss* en d_0. Si ces deux derniers éléments ne sont pas spécifiés, ce sont des valeurs par défaut qui sont retenues.

Comme on peut le remarquer sur l'exemple de la figure 4.11, un modèle n'est pas décrit à l'aide d'une balise *model* mais à l'aide d'une balise *stage*. En fait un stage va contenir un modèle en référençant la classe Java développée au sein du simulateur qui l'implémente. Les éléments que l'on retrouve ensuite à l'intérieur d'un stage vont correspondre aux paramètres de configuration du modèle. Cette solution basée sur la notion de stage qui consiste à fournir la classe pour pouvoir utiliser un modèle, permet d'ajouter très simplement de nouveaux modèles sans modifier le cœur du simulateur. Nous présenterons plus en détail cette notion de stage dans le chapitre suivant.

```
PathLossExponentModel =
    element stage {
        attribute id { text } &
        attribute class { "swine.models.propagation.PathlossExponentModel" } &
        element exponent { real } &
        element d0 { distance_unit? & real } &
        element frequency { frequency_unit? & real }? &
        element pathloss_d0 { power_unit? & real }?
    }
```

(a) schéma

```
- <stage id="pathloss" class="swine.models.propagation.PathlossExponentModel">
    <exponent>4.6</exponent>
    <d0 unit="m">1</d0>
    <frequency unit="GHz">2.457</frequency>
  </stage>
```

(b) exemple XML associé

FIG. 4.11: Extrait du fichier de haut niveau présentant un modèle de *Path Loss Exponent*

Les nœuds dont le schéma est présenté sur la figure 4.12 vont décrire les terminaux mobiles qui composent le réseau sans fil à émuler.

L'élément *models* inclus dans l'élément *Mobile* permet de définir l'ensemble des modèles qui

```
Mobile = element mobile {
    attribute id { xsd:ID } &
    element models { MobilityModel & EnergyModel? } &
    element gain { power_unit? & real } &
    element tx_power { power_unit? & real } &
    element ip_address { text } &
    element ip_mask { text } &
    element gateway { text } &
    element member_of { text } &
    element mapped_on { text }
}
```

(a) schéma

```
- <mobile id="M1">
  + <models>
    <gain unit="dB">0</gain>
    <tx_power unit="dB">-13</tx_power>
    <ip_address>192.168.106.1</ip_address>
    <ip_mask>255.255.255.0</ip_mask>
    <gateway>192.168.106.100</gateway>
    <member_of>Emulated WiFi Network</member_of>
    <mapped_on>wnine1</mapped_on>
  </mobile>
```

(b) exemple XML associé

FIG. 4.12: Extrait du fichier de haut niveau pour les nœuds

vont être utilisés pour caractériser le comportement dynamique d'un nœud comme les modèles de mobilité et d'énergie. Cette solution permet de pouvoir définir des modèles de mobilité ou d'énergie différents d'un nœud à l'autre. Ceci est notamment utile pour pouvoir utiliser un modèle tel que le modèle de poursuite que nous avons présenté dans la section 4.3.2.2. En effet, nous pouvons définir plusieurs nœuds comme des poursuivants d'une cible qui va, quant à elle, utiliser un autre modèle de mobilité tel que le modèle de *Random Walk* présenté dans la section 4.3.2.1 et dont on peut voir le schéma sur la figure 4.13. Les éléments *gain*, *tx_power*, *ip_address* et *gateway* sont utilisés par les modèles de propagation et de communication sur lesquels nous reviendrons en détails dans le chapitre suivant. Les éléments *member_of* et *mapped_on* ne sont pas directement liés à l'expérience que l'on souhaite émuler mais vont servir pour le passage du réseau virtuel à émuler à la plate-forme d'émulation sur laquelle les mesures vont avoir lieu.

```
RandomWalk =
    element stage {
        attribute id { text } &
        attribute class { "swine.models.mobility.RandomWalk" } &
        element initial_position { distance_unit? & tuple3d } &
        element speed { speed_unit? & real }
    }
```

(a) schéma d'un modèle de RandomWalk

```
- <stage id="mobility" class="swine.models.mobility.RandomWalk">
    <speed unit="m/s">1</speed>
  - <initial_position>
      <x>35</x>
      <y>0</y>
      <z>17</z>
    </initial_position>
  </stage>
```

(b) exemple XML associé

FIG. 4.13: Extrait du fichier de haut niveau pour un modèle de mobilité

Tous les éléments que nous avons présentés dans ce scénario de haut niveau vont donc permettre au simulateur d'initialiser son moteur d'émulation pour fournir les résultats nécessaires à la génération du scénario d'émulation que nous allons maintenant présenter.

4.4.4 Scénario d'émulation

Contrairement à la description de haut niveau, le scénario d'émulation que l'on obtient après la phase de simulation va décrire le réseau à émuler suivant un aspect temporisé. Il va permettre de stocker les paramètres de QdS qui seront reproduits lors de la phase d'émulation. Un extrait du schéma permettant de valider le scénario d'émulation est présenté sur la figure 4.14.

```
emulation_experiment =
    element emulation_experiment {
        element type_of_network { "Ad-Hoc" | "Managed" | "Hybrid"},
        element emulated_platform { emulated_host+ },
        element connectivity { link+ },
        scenario
    }
```

(a) schéma

```
- <emulation_experiment>
    <type_of_network>Ad-Hoc</type_of_network>
  - <emulated_platform>
    - <emulated_host ident="M1">
        <ip_address>192.168.106.1</ip_address>
        <ip_mask>255.255.255.0</ip_mask>
        <gateway>192.168.106.100</gateway>
        <member_of>Emulated WiFi Network</member_of>
        <mapped_on>wnine1</mapped_on>
      </emulated_host>
    + <emulated_host ident="F1">
    </emulated_platform>
  + <connectivity>
  + <scenario>
  </emulation_experiment>
```

(b) exemple XML associé

FIG. 4.14: Extrait d'un scénario d'émulation

Comme dans le fichier de description de haut niveau, la première chose est de définir l'élément *type_ of_ network*. Celui-ci donne des informations sur les éléments présents dans la suite du document XML. Ainsi dans le cas d'un réseau en mode Infrastructure (Managed) il ne sera pas nécessaire de rechercher des indications sur les cas potentiels de terminaux cachés mais plutôt de rechercher des informations sur la composition des cellules au cours du temps.

L'élément *emulated_ platform* permet de faire le lien entre le réseau simulé et la plate-forme physique qui va être utilisée pendant la phase d'émulation. C'est dans cette partie que vont être définis les nœuds qui vont devoir être configurés sur la plate-forme pour réaliser les tests. Comme nous pouvons le voir sur l'exemple présenté sur la figure 4.14 (b), les nœuds émulés caractérisés par l'élément *emulated_ host* sont semblables à ceux qui ont été présentés pour le fichier de haut niveau (figure 4.12 (b)). Seuls les éléments *gain, tx_ power* et *models* ne sont plus présents car les effets des modèles ont déjà été pris en compte.

L'élément *connectivity* permet de spécifier l'ensemble des communications qui peuvent avoir lieu à un instant donné. Cette connectivité se présente comme une suite de liens où chaque lien correspond à une paire de nœuds du réseau suivant le modèle (émetteur, récepteur). Le schéma décrivant un lien est présenté sur la figure 5.7. Cet élément *connectivity* est utilisé au niveau de la plate-forme d'émulation pour construire les canaux de communication qui devront être contraints par le conditionneur de trafic en fonction des règles de QdS spécifiée dans la suite du scénario.

L'élément *scenario* sert à stocker l'ensemble des règles caractérisant les conditions de QdS sous forme de scénario temporisé. Le schéma permettant de valider le *scenario* est décrit sur la

```
link =
    element link {
        attribute id { xsd:ID } &
        element sender { text } &
        element receiver { text }
    }
```

(a) shéma

```
- <connectivity>
  - <link ident="M1 to F1">
      <sender>M1</sender>
      <receiver>F1</receiver>
    </link>
  + <link ident="F1 to M1">
  </connectivity>
```

(b) exemple XML associé

FIG. 4.15: Extrait du scénario d'émulation pour la connectivité

figure 4.16. Un scénario temporisé se découpe en une suite de dates et chaque date se découpe à son tour en une suite de mises à jour de la QdS des liens définis dans la partie *connectivity*. L'utilisation d'un type *xsd :IDREF* dans le schéma permet de s'assurer que le lien sur lequel la mise à jour va avoir lieu a bien été défini. Les paramètres de QdS dynamiques sont spécifiés dans l'élément *link_ update* et sont valables à la date spécifiée dans l'élément englobant *date*. Ces paramètres vont ensuite rester valables jusqu'à la prochaine spécification d'un *link_ update* pour ce lien.

```
scenario =
    element scenario {
        element number_of_event { integer },
        element time_step { duration_unit, real},
        date+
    }

date =
    element date {
        attribute id { xsd:ID },
        attribute start { positiveReal },
        duration_unit,
        link_update+
    }

link_update =
    element link_update {
        attribute hidden { integer }?,
        attribute hidden_id { xsd:string }?,
        element on_link {attribute id { xsd:IDREF}},
        (
            element bandwidth { bandwidth_unit & real}&
            element delay { integer }&
            (
                element plr { positiveReal } |
                element loss_pattern { xsd:string }
            )&
            element queue_size {integer}?
        )
    }
```

(a) schéma

```
- <scenario>
    <number_of_event>360</number_of_event>
    <time_step unit="ms">500.0</time_step>
  - <date ident="t0.0" start="0" unit="ms">
    - <link_update>
        <on_link id="M1 to F1" />
        <bandwidth unit="b/s">4260000.0</bandwidth>
        <plr unit="percent">0</plr>
        <delay>0</delay>
        <queue_size>8</queue_size>
      </link_update>
    + <link_update>
    </date>
  + <date ident="t7.5" start="7500" unit="ms">
  + <date ident="t30.0" start="30000" unit="ms">
```

(b) exemple XML associé

FIG. 4.16: Extrait d'un scénario d'émulation pour la partie scénario

Ce scénario étant utilisé pour faire de l'émulation de niveau IP, les paramètres de QdS à manipuler pour faire évoluer la QdS d'un lien sont : les bandes passantes, les délais et les pertes. Comme nous pouvons le voir sur le schéma présenté sur la figure 4.16, il y a deux gestions possibles des pertes : soit spécifier un taux de pertes fixe, soit utiliser un fichier où sont spécifiées les positions des pertes. Enfin, les attributs *hidden* et *hidden_ id* présents sur ce schéma sont utilisés pour spécifier des alternatives à un scénario. Dans le cas d'une topologie où des cas de terminaux cachés peuvent survenir, un même lien pourra donc être mis à jour de deux manières différentes selon que le cas de terminaux cachés survient ou non. Ainsi que

nous le verrons dans le chapitre suivant le choix de l'alternative à appliquer se fait au niveau de la plate-forme d'émulation en fonction du trafic qui circule pendant une expérience.

4.5 Conclusion

L'utilisation d'une phase de simulation préalablement à une phase d'émulation permet d'utiliser des modèles précis et complexes qui améliorent considérablement le réalisme des conditions d'émulation générée. En utilisant des scénarios d'émulation pour stocker les différentes commandes d'émulations, générées à partir des conditions d'émulation précédemment simulées, il est possible de reproduire plusieurs fois de suite une émulation. De même, l'utilisation d'une description de haut niveau permet à l'utilisateur de se concentrer sur ce qui est le plus important pour lui, à savoir faire des expériences pour mesurer les performances de son protocole (ou de son application répartie). Cette description de haut niveau, écrite en XML est beaucoup plus simple à écrire que les différents fichiers de scripts généralement utilisés par les simulateurs, et elle peut être validée grâce aux différents schémas que nous avons définis. Cette utilisation de deux phases différentes ainsi que de fichiers de description dans un format reconnu permet de pouvoir changer de manière transparente les différentes briques composant notre émulateur. En effet, il est tout à fait possible de changer le simulateur utilisé pour générer les scénarios d'émulation. La seule contrainte ést bien sûr que le nouveau simulateur respecte les schémas que nous avons élaborés pour les fichiers d'entrée, et qu'il puisse générer un scénario respectant le format que nous avons défini. Enfin, l'utilisation des fichiers de haut niveau permet d'ajouter très simplement de nouvelles classes de modèles dans le simulateur dont nous allons maintenant détailler l'architecture.

5 L'émulateur sans fil W-NINE

Dans le chapitre précédent nous avons montré l'intéret d'utiliser deux phases complémentaires, l'une de simulation l'autre d'émulation pour émuler des réseaux sans fil. Une solution reposant sur l'utilisation de fichier XML et permettant la communication entre ces phases (utilisation de scénario) a égalemen été spécifiée. L'objectif de ce chapitre sera de spécifier plus en détail la solution que nous proposons en présentant tout d'abord l'architecture générale de la plate-forme W-NINE (Wireless NINE) puis en insitant sur les solutions que nous avons retenues dans les phases d'émulation et de simulation.

5.1 Architecture de W-NINE

La plate-forme W-NINE, présentée sur la figure 5.1, repose sur l'utilisation de deux outils distincts : SWINE et NINE. Ces deux outils s'utilisent l'un après l'autre au cours de deux phases indépendantes :

- La première phase dite de simulation, va avoir lieu hors-ligne et s'appuie sur l'utilisation d'un simulateur de réseaux sans fil pour l'émulation appelé **SWINE** (Simulator for WIreless Network Emulation) pour produire un scénario d'émulation décrivant les conditions du réseau à émuler.
- La seconde phase dite d'émulation va se dérouler en temps réel et va consister à reproduire en temps réel sur la plate-forme d'émulation **NINE** (NINE Is a Network Emulator) le scénario d'émulation produit par SWINE.

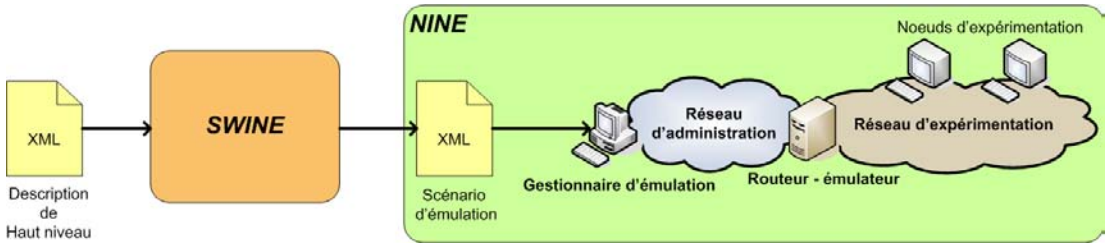


FIG. 5.1: Architecture de W-NINE

Ainsi que nous l'avons expliqué dans le chapitre précédent, l'utilisation d'une phase de simulation préalablement à l'émulation, va permettre d'utiliser des modèles réalistes et ceci même s'ils sont extrêmement coûteux en temps de calcul. En effet, comme cette phase de simulation se déroule hors-ligne, il n'y a pas de contraintes temps réel. Le temps mis pour calculer les effets du modèle n'aura pas d'influence sur la phase d'émulation et donc sur l'évaluation menée par l'utilisateur.

Au cours de cette phase de simulation les effets de la mobilité et de la propagation sur les paramètres de QdS de niveau IP (bandes passantes, délais et pertes) sont pris en compte pour générer le scénario d'émulation. La phase d'émulation doit reproduire les conditions décrites

dans le scénario d'émulation tout en respectant les contraintes temps réel. En effet, la fréquence à laquelle les étapes du scénario vont être lues pour être reproduites, va influer directement sur le réalisme de l'émulation rendue. Si l'intervalle de temps est trop grand, le réalisme ne sera pas bon car des effets comme par exemple des mouvements rapides ou des effets de propagation à petite échelle comme le fading de Rayleigh ne seront pas perçus. Cependant, si l'intervalle de temps est trop petit, le routeur-émulateur n'aura pas le temps de reproduire toutes les conditions si bien que, là encore, l'émulation ne sera pas réaliste. Le choix du pas de temps est donc un facteur important du réalisme de l'émulation rendue.

Le diagramme d'activité UML présenté sur la figure 5.2 résume ce processus d'émulation en deux phases.

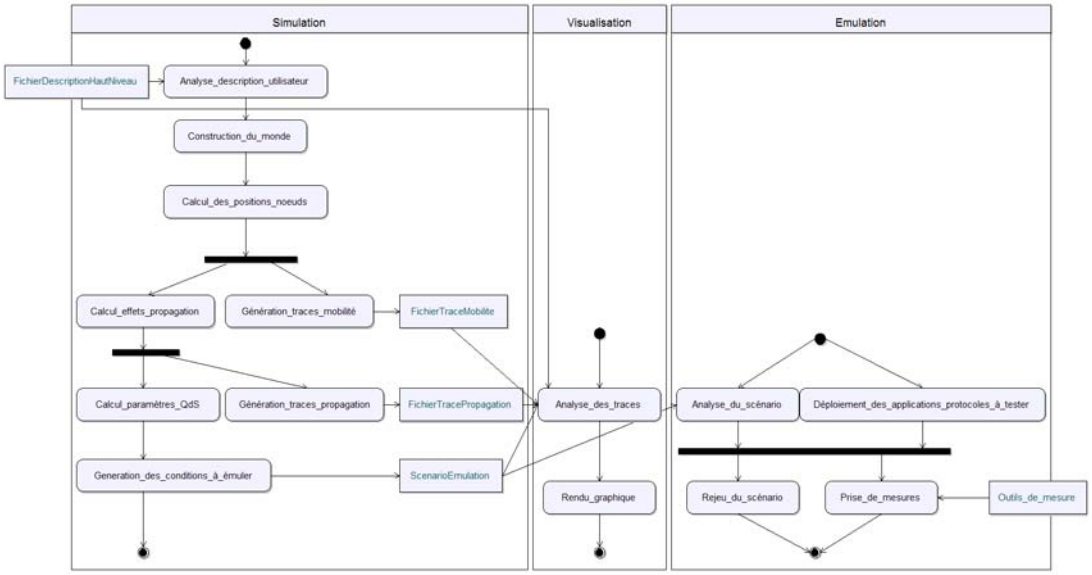


FIG. 5.2: Diagramme d'activité de W-NINE

La partie visualisation représentée sur ce diagramme n'est pas à proprement parler une partie du système d'émulation. Elle a été développée pour permettre à l'utilisateur de vérifier intuitivement les résultats fournis par la simulation en permettant de visualiser les déplacements de terminaux mobiles dans le réseau sans fil émulé ainsi que des courbes présentant l'évolution des conditions de propagation et des conditions de QdS de niveau IP rencontrées dans ce réseau. La vérification proprement dite repose sur l'analyse des mesures faites au cours d'une expérience.

5.2 La plate-forme physique NINE

5.2.1 Architecture de NINE

La plate-forme d'émulation NINE (NINE is a Network Emulator) a été conçue à l'origine dans le but d'émuler des réseaux filaires et des réseaux satellites. Elle se compose principalement de trois types d'éléments :

– Un routeur-émulateur chargé de conditionner le trafic qui sera également chargé de l'observation du trafic ;

– Des éléments terminaux, appelés terminaux ou nœuds d'expérimentation, sur lesquels sont déployés les protocoles de transport (basés sur IP) et les applications réparties à tester ;
– Un gestionnaire d'émulation chargé de faire varier les conditions sur le routeur-émulateur et de configurer les nœuds d'expérimentation de la plate-forme ;

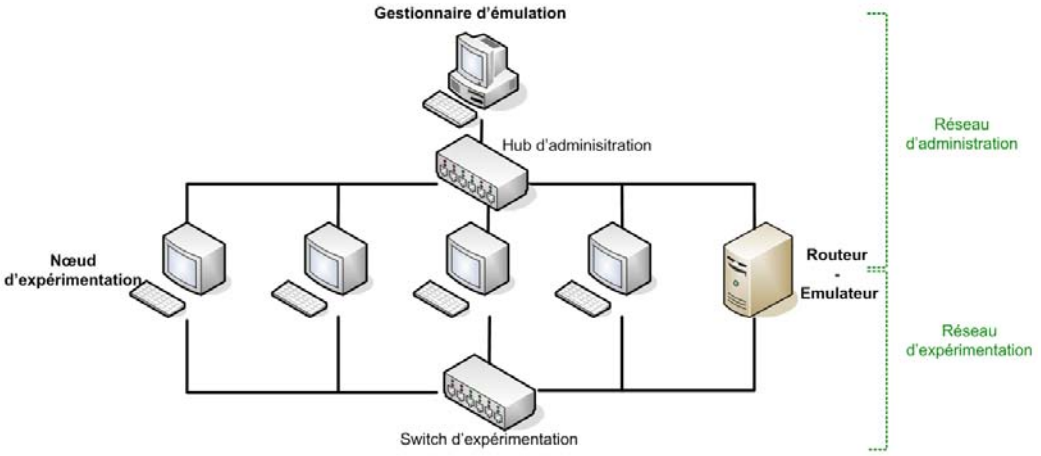


FIG. 5.3: La plate-forme d'émulation NINE

La plate-forme est constituée de deux réseaux, l'un dédié à l'administration et l'autre à l'expérimentation. Cette séparation physique des deux réseaux permet d'assurer que le trafic d'expérimentation n'entrera pas en concurrence avec un quelconque trafic d'administration. De plus, le fait d'utiliser un réseau dédié à l'administration permet d'assurer que les commandes d'administration émises par le gestionnaire d'émulation ne seront pas perdues ou ralenties par un trafic concurrent. Le gestionnaire d'émulation est situé sur le réseau d'administration uniquement, si bien qu'il ne peut pas agir directement sur une expérience en cours. Son seul interlocuteur pendant une expérience d'émulation est donc le routeur-émulateur qui est situé sur les deux réseaux. Les nœuds d'expérimentation sont également situés sur les deux réseaux de manière à pouvoir être configurés à l'aide du gestionnaire d'émulation ***avant*** le début d'une expérience. Il ne recevront plus de trafic d'administration de la part du gestionnaire d'émulation pendant la phase d'émulation. On doit de plus déployer sur ces nœuds d'expérimentation, les applications et protocoles à tester. A part ce déploiement, les nœuds d'expérimentation n'embarquent aucun logiciel ou matériel nécessaire au fonctionnement de l'émulateur si bien qu'il est possible d'utiliser n'importe quel système d'exploitation pour effectuer les tests. Nous avons fait le choix d'être le moins intrusif possible sur ces nœuds d'expérimentation pour ne pas limiter les possibilités de test de l'utilisateur.

5.2.2 Dummynet et ses améliorations

Dummynet a été présenté dans la section 3.2.2.1 mais nous allons revenir plus en détail sur son fonctionnement et sur les techniques qu'il met en œuvre pour contraindre le trafic qui circule dans un routeur-émulateur. Nous montrerons notamment l'intérêt d'utiliser cet émulateur mais également les limitations qui sont les siennes à l'heure actuelle ainsi que les solutions existantes pour y remédier.

5.2.2.1 Fonctionnement classique de Dummynet

Dummynet est un conditionneur de trafic intégré dans le système d'exploitation de FreeBSD. Il fonctionne en partenariat avec le pare-feu de niveau IP *ipfw*. Ce pare-feu intercepte les paquets reçus sur les interfaces réseaux de la machine et les redirige vers Dummynet. Plus précisément, Dummynet fait passer les paquets dans des canaux de communications appelés *pipes* qui vont permettre de reproduire des conditions réseaux prédéfinies en terme de bandes passantes, de délais, de taux de pertes de paquets (PLR) mais également en terme de taille de file d'attente. Ainsi, plusieurs *pipes* peuvent être créés en parallèle, chacun étant chargé de reproduire des conditions réseaux particulières. Un autre point intéressant de Dummynet est la possibilité d'enchaîner ces *pipes* pour permettre à un paquet de subir plusieurs contraintes successives et différentes. Cette particularité peut notamment être utile pour émuler une communication de bout en bout traversant plusieurs réseaux (par exemple une communication de bout en bout traversant un réseau satellite puis un réseau d'accès de type ADSL) comme dans [49, 5].

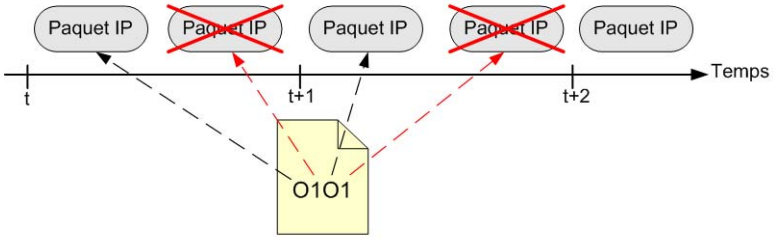
Cependant, Dummynet ne propose pas de fonctionnalité permettant de faire varier dynamiquement les conditions de ces *pipes* au cours du temps. Il n'est par exemple pas possible de demander à un *pipe* de reproduire une bande passante de 2 Mbps pendant 10 secondes puis une bande passante de 1 Mbps pendant les 30 secondes suivantes. Il est donc nécessaire de lui fournir une nouvelle commande de configuration pour pouvoir mettre à jour les conditions d'un canal de communication : pour pouvoir passer de 2 Mbps à 1 Mbps par exemple. Pour remédier à cette limitation, un gestionnaire d'émulation a été développé dans le cadre de NINE. Ce gestionnaire d'émulation est notamment chargé d'envoyer aux dates prévues par un scénario d'émulation ces commandes de mises à jour de Dummynet.

Une autre limitation importante de Dummynet vient de sa gestion des erreurs. Dummynet ne propose qu'une gestion probabiliste des pertes, c'est-à-dire qu'il est uniquement possible de spécifier un PLR pour un pipe ce qui ne permet pas une parfaite reproductibilité d'une expérience. Les résultats obtenus après plusieurs expériences avec les mêmes paramètres seront les mêmes en moyenne mais ce ne seront pas forcément les mêmes paquets qui seront perdus. Pour résoudre ce problème nous avons donc choisi d'utiliser dans le cadre de W-NINE une extension de Dummynet développée par l'Université de Karlstad et appelée KAUnet qui propose un nouveau mode de gestion des pertes de paquet.

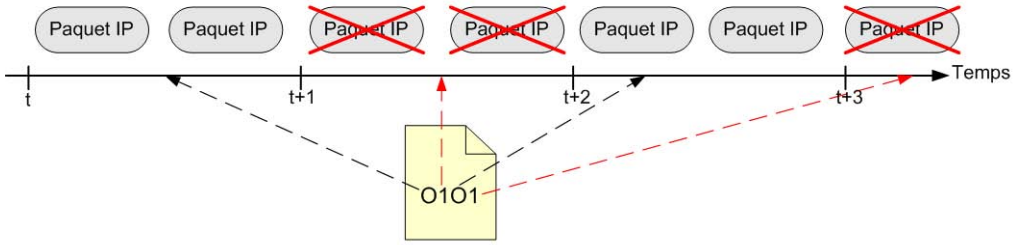
5.2.2.2 L'extension KAUnet

L'extension KAUnet a été développée par l'Université de Karlstad (Suède) pour améliorer la gestion des erreurs dans Dummynet. En effet comme Garcia *et al* l'ont présenté dans [27], la position des pertes dans un flux va avoir une forte influence sur le comportement d'un protocole. Par exemple, le protocole TCP n'aura pas le même comportement suivant que le paquet perdu est un paquet de données (DATA) ou un paquet servant à l'intialisation de la connexion (SYN-ACK). Pour pouvoir comparer deux protocoles, il est nécessaire que les pertes surviennent au même endroit dans le flux de paquet. Pour répondre à ce besoin, KAUnet étend Dummynet en lui adjoignant des fichiers de pertes. Ces fichiers vont représenter sous forme binaire les positions des pertes soit dans un flux soit dans le temps suivant la manière dont ils sont lus. Ainsi, il va être possible d'avancer dans le fichier soit en fonction des paquets qui passent dans le routeur-émulateur, soit en fonction du temps qui passe.

La figure 5.4 montre un exemple de ces deux modes de fonctionnement. Sur la sous-figure



(a) Pertes suivant les données



(b) Pertes suivant le temps

FIG. 5.4: Illustration des modes de fonctionnement de KAUnet pour les pertes de paquets

(a), les pertes sont lues en fonction des paquets qui passent dans le *pipe* : un 0 indique un paquet reçu, un 1 un paquet perdu. Nous pouvons voir qu'avec le fichier de perte 0101, le deuxième et le quatrième paquet du flux doivent être perdus. Sur la sous-figure (b) par contre, les pertes sont lues en fonction du temps qui passe : un 0 correspond à une période de temps où les paquets sont bien reçus et un 1 correspond à une période de temps où les paquets sont tous perdus. Avec un fichier de pertes 0101, les pertes vont donc survenir entre les instants $t+1$ et $t+2$ quel que soit le nombre de paquets circulant pendant cet intervalle de temps. Tous les paquets franchissant le routeur-émulateur au cours de cet intervalle de temps seront perdus. Entre $t+2$ et $t+3$ les paquets sont autorisés à franchir le routeur émulateur, puis ils sont de nouveau perdus à partir de $t+3$. La durée de la période de temps est dépendante de la fréquence du système d'exploitation. Généralement sous FreeBSD celle-ci est de 1 ms.

Le fait de pouvoir positionner précisément les pertes dans un flux est quelque chose de très utile non seulement pour comparer deux protocoles entre eux mais également pour vérifier la cohérence d'une implémentation avec la théorie. Ainsi, l'expérience présentée dans [18] présente le comportement de la pile TCP implémentée sur la version 6 du système d'exploitation FreeBSD. Le but initial était de montrer l'intérêt de placer correctement les pertes dans un flux pour voir leur impact sur le protocole, mais ainsi que nous pouvons le voir sur la figure 5.5, les expériences menées par l'équipe de Karlstad ont montré une inadéquation entre les résultats obtenus et la courbe attendue. Le but de cet exemple était de transférer 25 Ko sur un lien à 40 Kbps avec un délai de transmission de 10 ms. Chaque point de cette courbe présente le temps de transmission complet, obtenu pour une expérience avec une seule perte. L'axe des ordonnés indique la position de cette perte dans le flux de données. Le comportement réaliste aurait voulu que les pertes à partir du vingtième paquet n'aient pas d'effet sur le temps de transmission car l'ensemble des paquets contenant des données a normalement été transmis. Cependant nous pouvons voir sur la figure 5.5 que le délai augmente si bien que l'on peut

conclure que l'implémentation de TCP réalisée sous FreeBSD 6, transmet plus de paquets que nécessaire. Sans un positionnement précis des pertes, ce comportement n'aurait pas pu être mis en évidence. De même pour les positions 13 et 19 les pics de délais observés ne sont pas normaux. Après avoir examiné le code implémenté sous FreeBSD 6, il semble que le problème vienne d'une mauvaise gestion des SYNACK-ACK pour l'estimation du RTT (Round Trip Time).

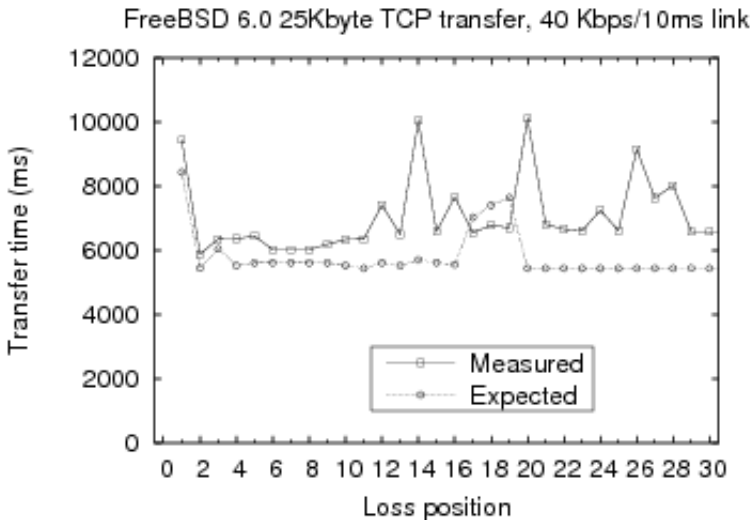


FIG. 5.5: Comportement de TCP sur FreeBSD 6 [18].

L'extension KAUnet permet également d'insérer des erreurs bits à l'intérieur d'un paquet dans un flux. Cette amélioration est très intéressante car elle permet de tester et d'évaluer des protocoles de transport utilisant des codes correcteurs d'erreur de type FEC (Forward Error Correction). En effet, dans un réseau sans fil les paquets corrompus sont généralement rejetés au niveau MAC grâce à l'utilisation du FCS (Frame Check Sequence) qui est ajouté dans chacune des trames transmises sur un canal 802.11. Le nœud récepteur calcule le FCS de chaque paquet qu'il reçoit et le compare à celui calculé avant l'émission et stocké dans la trame. Si les deux FCS sont différents cela signifie que le paquet contient des erreurs et il est donc supprimé par la couche MAC. Cependant, il est possible de désactiver cette vérification pour pouvoir recevoir tout de même ces paquets endommagés qui dans certains cas (images, sons...) sont récupérables grâce à l'utilisation de codes correcteurs (FEC, ARQ ...).

L'utilisation de cette extension de Dummynet comme émulateur dans NINE va donc permettre de tester avec précision une grande variété de protocoles de transport et d'applications réparties. Elle va surtout permettre de répondre au besoin de reproduction que nous avions mis en avant dans le chapitre précédent. En effet, il sera possible d'assurer la reproduction d'une expérience d'un test à l'autre à partir de fichiers de pertes réalistes. Grâce à ces fichiers, les pertes seront toujours positionnées au même instant t dans l'expérience, ce qui assure à l'utilisateur que les conditions seront les mêmes pour qu'il puisse comparer deux protocoles.

```
- <physical_platform>
  - <physical_switch type="ThreeComSuperStack" name="NineExperimentationSwitch" ident="100">
      <ip_address>10.0.1.10</ip_address>
      <login_name>admin</login_name>
      <password />
    </physical_switch>
    <number_of_physical_emulators>1</number_of_physical_emulators>
    <number_of_physical_hosts>2</number_of_physical_hosts>
  - <physical_emulator name="galaga" ident="1">
      <operating_system>freeBSD</operating_system>
    - <administration_interface>
        <interface_name os="windows">Connexion au reseau local</interface_name>
        <interface_name os="linux">eth0</interface_name>
        <interface_name os="freeBSD">em0</interface_name>
        <ip_address>10.0.1.1</ip_address>
        <ip_mask>255.255.255.0</ip_mask>
      </administration_interface>
    + <experimentation_interface>
    + <experimentation_interface>
    + <experimentation_interface>
    + <experimentation_interface>
    + <experimentation_interface>
    </physical_emulator>
  + <physical_host name="wnine1" ident="2">
  + <physical_host name="wnine2" ident="3">
  + <physical_host name="nine1" ident="4">
  </physical_platform>
```

FIG. 5.6: Exemple de fichier XML décrivant la plate-forme physique NINE

5.2.3 Le gestionnaire d'émulation

Le gestionnaire d'émulation sert à configurer l'ensemble de la plate-forme puis à faire varier les conditions d'émulation au cours d'une expérience.

Pour configurer la plate-forme d'émulation sur laquelle les expériences vont avoir lieu, le gestionnaire d'émulation utilise un fichier XML décrivant les différentes machines composant la plate-forme ainsi que les adresses IP de leurs interfaces situées sur le réseau d'administration. Grâce à cela, il va pouvoir contacter chaque machine et la configurer dynamiquement en début d'expérience. C'est également dans ce fichier XML que l'on va indiquer au gestionnaire d'émulation avec quel émulateur il va devoir communiquer au cours d'une expérience. Un extrait de ce fichier décrivant la plate-forme d'émulation est présenté sur la figure 5.6. Une fois la description de la plate-forme physique interprétée, le gestionnaire va charger le scénario d'émulation généré par SWINE.

La première étape à partir de ce scénario va consister à configurer les interfaces des différents nœuds d'expérimentation situés sur le réseau d'expérimentation. Pour cela, le gestionnaire d'émulation va utiliser l'atome *member_ of* défini lors de la description d'un nœud (cf figure 4.12). Ceci permet donc de lier le réseau émulé à la plate-forme physique. Il faut noter que cette phase est réalisée manuellement, c'est-à-dire que l'utilisateur doit définir quel nœud émulé doit être hébergé sur quel terminal d'expérimentation. Nous ne détaillerons pas plus cette problématique de la configuration de la plate-forme et du passage du réseau émulé au réseau physique d'expérimentation mais ces questions ont été approfondies dans notre rapport de DEA [17].

Lorsque les terminaux de la plate-forme physique ont été paramétrés, le gestionnaire d'émulation va devoir configurer le conditionneur de trafic qui sera chargé de reproduire les conditions du réseau émulé au cours d'une expérience. Dans le cadre de W-NINE, notre plate-forme

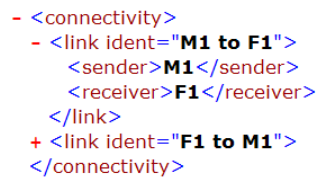


FIG. 5.7: Extrait du scénario d'émulation décrivant la connectivité d'une expérience.

d'émulation sans fil, ce conditionneur de trafic sera KAUnet, l'extension de Dummynet présentée dans la section 5.2.2.2. Préalablement à la phase d'émulation, il est nécessaire d'établir les règles de routage pour rediriger le trafic émis sur la plate-forme vers KAUnet. Pour cela, le gestionnaire d'émulation va utiliser le scénario et plus précisément la partie *connectivity* de ce scénario (cf figure 5.7). Grâce à cette *connectivity,* le gestionnaire d'émulation sera à même de déterminer l'ensemble des communications qui peuvent survenir au cours d'une expérience et donc de construire les règles *ipfw* correspondantes. Chaque lien présenté dans la partie *connectivity* caractérise un canal de communication (*pipe*) Dummynet. Ce sont ces liens qui vont être mis à jour pendant l'expérience pour reproduire les caractéristiques du réseau sans fil. Pour cela, le gestionnaire lit le scénario d'émulation et génère les règles de mise à jour des *pipes* du conditionneur de trafic. Toutes ces règles sont ensuite passées à un ordonnanceur avec la date à laquelle elles doivent être émises en direction du routeur-émulateur (au maximum tous les intervalles de temps). L'expérience d'émulation proprement dite commence donc au moment où l'utilisateur démarre l'ordonnanceur.

Le fait d'utiliser un pipe par communication présente cependant quelques inconvénients. Ainsi, lorsque deux flux sont émis par deux nœuds différents à destination d'un même troisième nœud, ils ne vont pas rentrer en concurrence car ils disposent chacun de leur propre canal de communication. Ceci peut donc amener à des comportements erronés pour l'évaluation de protocoles de transports comme TCP par exemple en diminuant les risques de congestions par rapport à un réseau réel. Pour pouvoir gérer de manière cohérente ces différents flux une solution consiste à simuler une couche MAC en temps réel au niveau du routeur émulateur comme ce qui a été fait dans MobiNet [52]. Sans disposer de modèle de trafic, la prise en compte de ce genre de comportement peut également être envisagée grâce à l'utilisation d'observateurs de réseau. Cependant, cette solution semble relativement complexe à mettre en œuvre car il faudrait un observateur par nœud émulé. De plus, le modèle permettant de déterminer les effets de flux concurrents au niveau d'un nœud est relativement complexe à mettre en œuvre car il dépend à la fois du nombre de nœuds émetteurs mais également de leur taux de transmission radio.

D'un point de vue pratique, le gestionnaire d'émulation se présente sous la forme d'une application Java utilisant CORBA pour ses communications réparties avec le routeur-émulateur et notamment les observateurs. Le choix du langage s'est porté sur Java pour sa robustesse, sa simplicité et surtout son indépendance vis-à-vis du système d'exploitation. Le gestionnaire d'émulation peut donc être utilisé aussi bien sous un environnement de type Windows comme nous le faisons actuellement que sous un environnement Linux. L'utilisation de CORBA répond également à ce besoin d'indépendance vis-à-vis du système d'exploitation et des langages utilisés pour développer les différentes partie du code de l'application répartie formée par le gestionnaire d'émulation et les observateurs. Ainsi, bien que le gestionnaire d'émulation

soit écrit en JAVA et fonctionne en général sous Windows, les observateurs avec lesquels il communique sont écrits en C++ car ils nécessitent plus de réactivité et sont déployés sous FreeBSD. Pour permettre cette inter-opérabilité, CORBA s'appuie sur une notion de contrat. Ces contrats écrits dans le langage IDL (*Interface Definition Language*) vont définir l'ensemble des services fournis par les différents objets répartis. Grâce à l'utilisation de CORBA et en particulier des contrats IDL, il est par exemple possible de changer de manière transparente le code des observateurs (optimisation du code, changement du langage...) ou même de changer le système d'exploitation sur lequel ils sont actuellement utilisés. Il suffit en effet que la nouvelle implémentation des observateurs respecte le contrat IDL établi entre les observateurs et le gestionnaire d'émulation, pour que les communications réparties puissent avoir lieu. Nous allons maintenant voir le type d'informations qui doivent pouvoir être échangées entre les observateurs de réseau et le gestionnaire d'émulation.

5.2.4 Les observateurs de réseau

Comme nous l'avons présenté dans le chapitre précédent (section 4.3.4.2), tout ne peut pas être pré-calculé lors de la phase de simulation car nous ne disposons pas de modèle de trafic. Sans ce modèle de trafic, il n'est pas possible de prendre en compte toutes les perturbations en résultant. Ainsi, on ne peut pas déterminer à l'avance si un cas de terminaux cachés va survenir ou non, ni déterminer la bande passante effective dont un nœud disposera dans une cellule. Pour résoudre ce problème, nous avons cependant montré qu'il était possible d'envisager plusieurs situations de trafic lors de la phase de simulation et de générer plusieurs possibilités dans le scénario d'émulation. Le choix de la possibilité à appliquer se fait ensuite pendant la phase d'émulation en observant le trafic qui circule sur le réseau d'expérimentation pendant une expérience.

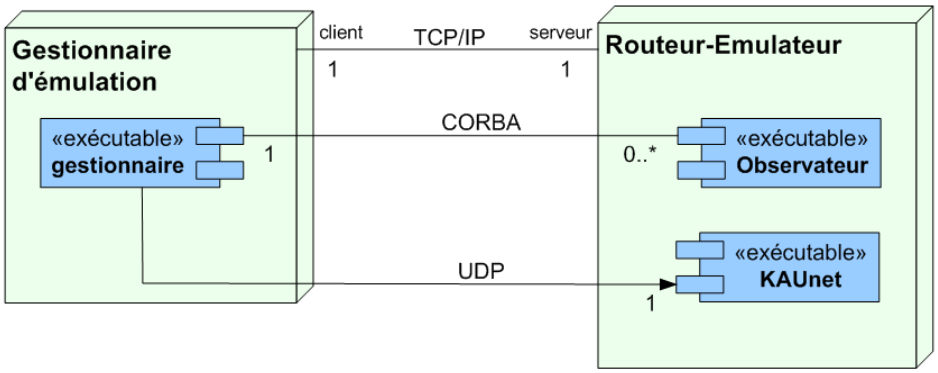


FIG. 5.8: Diagramme de déploiement de W-NINE

Plus précisément, le choix est effectué par le gestionnaire d'émulation en fonction des informations qui lui sont fournies par les observateurs de réseaux que nous allons présenter dans la section 5.2.4.1. Ces observateurs, présents sur le routeur émulateur, partagent les informations avec le gestionnaire d'émulation situé sur une machine dédiée, grâce à la norme de communication CORBA. Les informations partagées sont des types simples (entiers, booléens...) si bien que les performances ne sont pas affectées par le choix de cette norme. Le diagramme de déploiement présenté sur la figure 5.8 résume ce processus de communication et permet de

visualiser la répartition des différents éléments sur le réseau.

5.2.4.1 Support des réseaux Ad-Hoc

Pour pouvoir émuler les réseaux Ad-Hoc, il est nécessaire de prendre en compte des problèmes de communication tels que les cas de terminaux cachés et de terminaux exposés. Le problème de terminaux exposés est directement lié au protocole d'accès au canal radio si bien qu'il peut être partiellement pris en compte hors ligne ainsi que nous le verrons dans la présentation du simulateur SWINE. Le problème des terminaux cachés par contre nécessite de connaître le trafic qui circule sur le réseau émulé pour pouvoir être pris en compte. Etant donné que nous sommes dans une approche d'émulation nous ne disposons pas d'un tel modèle de trafic, il est donc nécessaire d'observer le trafic qui circule pendant une expérience.

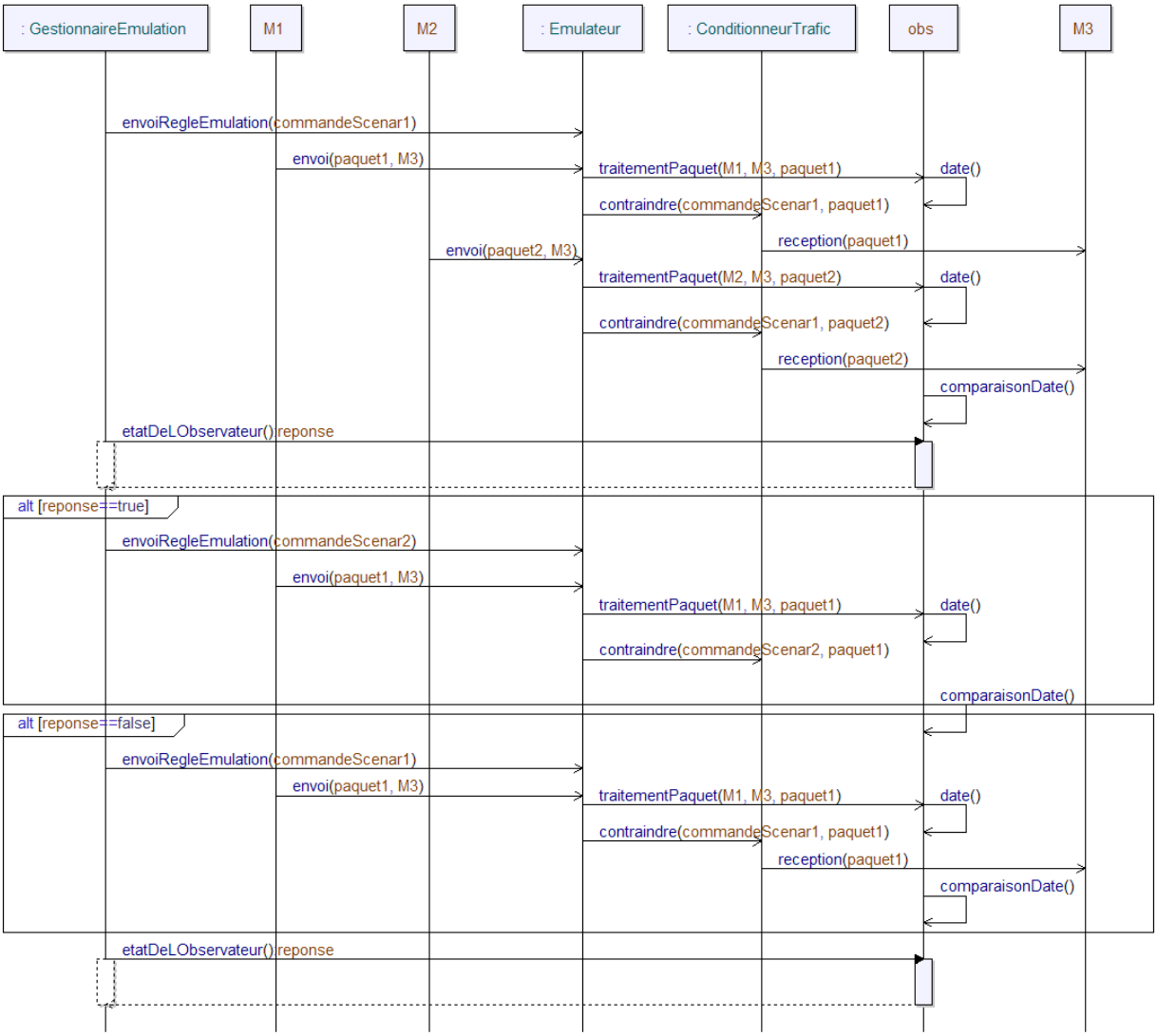


FIG. 5.9: Diagramme de séquence du choix de scénario pour un cas de terminaux cachés

Dans notre approche centralisée, l'ensemble du trafic passe par un nœud central : le routeur-

émulateur. Les observateurs vont donc être déployés sur ce nœud et analyser le trafic qui y transite au cours d'une expérience. Chaque observateur va être chargé de détecter un cas particulier de terminaux cachés. Pour ce faire, il va procéder comme décrit sur le diagramme de séquence présenté sur la figure 5.9 où la valeur de *réponse* caractérise l'état de l'observateur : la valeur VRAI (*true*) correspond à l'occurrence d'un cas de terminaux cachés et la valeur FAUX (*false*) à une occurrence de trafic normal. Ainsi, lorsqu'un nœud M_1 émet vers le nœud intermédiaire M_2, l'observateur détecte le passage de ce paquet dans l'émulateur et mémorise la date. Si M_1 est le seul à émettre, les commandes du scénario 1 correspondant au scénario classique, c'est-à-dire sans interférences dues aux terminaux cachés, sont appliquées. Par contre, si dans un intervalle de temps δ l'observateur détecte un paquet émis par le nœud M_3, il va passer dans l'état *hidden*. Lors de l'envoi des commandes suivant, le gestionnaire d'émulation en interrogeant l'observateur sur son état va détecter l'occurrence du cas de terminaux cachés et changer de scénario. Il va alors appliquer les règles d'émulation correspondant au cas de terminaux cachés qui auront été pré-calculées à l'avance. Ainsi, pour chaque lien susceptible d'appartenir à un cas de terminaux cachés, il aura été nécessaire de générer deux possibilités dans le scénario d'émulation. Un exemple de scénario avec deux choix de scénario à une date pour un lien est donné sur la figure 5.10.

```
link_update =
    element link_update {
        attribute hidden { integer }?,
        attribute hidden_id { xsd:string }?,
        element on_link {attribute id { xsd:IDREF }},
        (
            element bandwidth { bandwidth_unit & real}&
            element delay { integer }&
            (
                element plr { positiveReal } |
                element loss_pattern { xsd:string }
            )&
            element queue_size {integer}?
        )
    }
```

(a) schéma d'une alternative

```
- <link_update hidden="0" hidden_id="F1-M1-F2">
    <on_link id="F2 to M1" />
    <bandwidth unit="b/s">1770000.0</bandwidth>
    <plr unit="percent">0.0</plr>
    <delay>0</delay>
    <queue_size>8</queue_size>
  </link_update>
- <link_update hidden="1" hidden_id="F1-M1-F2">
    <on_link id="F2 to M1" />
    <bandwidth unit="b/s">1770000.0</bandwidth>
    <plr unit="percent">100.0</plr>
    <delay>0</delay>
    <queue_size>8</queue_size>
  </link_update>
```

(b) Extrait d'un scénario d'émulation

FIG. 5.10: Schéma RELAX-NG et extrait du scénario d'émulation présentant une alternative.

5.2.4.2 Support des réseaux en mode infrastructure (cellulaires)

Mis à part dans les solutions d'émulation basées sur la simulation temps réel comme NSE, l'émulation des réseaux 802.11 en mode infrastructure a été très peu étudiée. Ainsi que nous l'avons présenté précédemment dans la section 4.3.4.2, le débit effectif dont va disposer un nœud dans un réseau 802.11 en mode Infrastructure va dépendre directement du taux de transmission radio du nœud qui est en train d'émettre le plus lentement vers le point d'accès gérant la cellule. Ce comportement est donc lié au trafic circulant sur le réseau. Il ne peut donc pas être pris en compte lors de la phase de simulation sans modèle de trafic, c'est pourquoi là encore, nous proposons d'utiliser des observateurs pour pouvoir émuler ce comportement.

Les observateurs utilisés pour supporter le mode infrastructure reposent sur le même principe que les observateurs de réseaux Ad-Hoc, c'est-à-dire qu'ils vont analyser les trafics émis sur le réseau d'expérimentation et déterminer si des trafics entrent en concurrence ou influent les uns sur les autres. Cependant, leur fonctionnement est sensiblement plus complexe que celui que nous venons de voir. En effet, un observateur pour le mode infrastructure va être

chargé d'observé le trafic circulant dans une cellule et ceci pour toute la durée de l'expérience. Il faudra donc utiliser un observateur par cellule (dans le cas des réseaux Ad-Hoc, un observateur était chargé de surveiller un cas de terminaux cachés). En mode Infrastructure, chaque observateur doit donc avoir des informations sur les nœuds composant la cellule à un instant donné. Cela veut dire que la liste des nœuds qu'il va devoir surveiller va changer au cours du temps. De plus, le nombre de nœuds à surveiller devient beaucoup plus important que dans le cas précédent et il est également nécessaire de connaître leur taux de transmission radio de manière à déterminer quel nœud émet le plus lentement et donc quel nœud va ralentir toutes les autres transmissions. Ceci est également important pour déterminer dans quelles proportions ces transmissions seront affectées.

Il y a deux solutions principales pour faire évoluer la liste des nœuds dont l'observateur doit observer le trafic. La première est d'envoyer à l'observateur les changements de composition de la cellule sur l'ensemble de l'expérience avant de démarrer l'émulation. Cette solution nécessite que le routeur-émulateur où sont déployés les observateurs, soit synchronisé temporellement au cours d'une expérience avec le gestionnaire d'émulation pour que les règles soient appliquées au même moment. La seconde solution est que le gestionnaire envoie périodiquement la composition de la cellule à l'observateur. Dans cette solution, le besoin de synchronisation disparaît, c'est donc la solution que nous avons retenue dans l'implémentation actuelle. Par contre, le routeur-émulateur sur lequel les observateurs vont être déployé doit être suffisament puissant pour pouvoir traiter les mises à jours des observateurs en même temps que les commandes d'émulation.

L'observateur va donc connaître à chaque instant la liste des nœuds composant la cellule dont il s'occupe ainsi que leur taux de transmission radio. Il est donc capable de détecter un trafic susceptible d'introduire des délais sur d'autres communications. Lorsque plusieurs nœuds émettent simultanément, l'observateur détecte le taux de transmission le plus faible qui est utilisé par les nœuds en analysant le trafic qui traverse le routeur-émulateur et en informe le gestionnaire d'émulation. Celui-ci peut alors choisir la règle d'émulation correspondant à ce taux de transmission radio pour appliquer les contraintes de délais aux autres communications de la cellule. Cettte règle peut ajouter un délai de transmission ou alors réduire la bande passante disponible au niveau IP en fonction du modèle utilisé.

Dans cette approche, le nombre de possibilités à gérer dans le scénario va être plus important que dans le cas des terminaux cachés, car il va dépendre du nombre de taux de transmission utilisés par les nœuds composant la cellule. Dans le cas d'un réseau 802.11b, si tous les nœuds composant la cellule ont un taux de transmission radio de 11 Mbps, il n'y aura qu'une seule possibilité. Par contre, ce nombre de possibilités peut monter jusqu'à quatre pour un lien si dans la cellule il y a des nœuds avec des taux de transmission à 11 Mbps, 5.5 Mbps, 2 Mbps et 1 Mbps. L'ensemble de ces possibilités doit donc avoir été simulé au préalable par le simulateur SWINE dont nous allons maintenant voir les principales caractéristiques.

5.3 Simulation hors-ligne avec SWINE

5.3.1 Objectifs de SWINE

L'objectif du simulateur SWINE n'est pas de permettre d'évaluer un protocole ou une application comme dans le cas des simulateurs réseaux classiques mais uniquement d'établir le scénario d'émulation qui va devoir être reproduit pendant la phase d'émulation. Ce comportement est donc sensiblement différent de celui que l'on peut retrouver avec des simulateurs tels

que NS-2, GloMoSim ou OPNET pour ne citer qu'eux. Dans ces simulateurs en effet, l'objectif n'est pas de déterminer les conditions du réseau sans fil mais bien d'évaluer le comportement d'une application ou d'un protocole. Pour cela, ces simulateurs reposent sur une architecture en couche et ils ont besoin de faire passer des paquets à travers ces couches pour pouvoir fournir des résultats.

Dans l'approche d'émulation qui nous intéresse, nous souhaitons faire le plus abstraction possible du trafic qui va circuler sur le réseau d'émulation pour pouvoir évaluer le plus grand nombre d'applications et de protocoles. Cette approche est donc à l'opposé de celle retenue dans les autres simulateurs de réseau. Ici, le but est d'obtenir les conditions du réseau sans trafic. On recherche les conditions optimales qu'un nœud peut rencontrer sur le réseau sans fil à un instant donné, c'est-à-dire les conditions qu'il peut observer lorsqu'il est le seul nœud à émettre sur le réseau.

Pour déterminer ces conditions nous avions un temps pensé réutiliser un des simulateurs existant. Cependant, cela revenait à le détourner de sa fonction d'origine. En effet, pour obtenir les conditions optimales qu'un nœud peut observer, il aurait été nécessaire d'effectuer une simulation par paire de nœuds. C'est uniquement à cette condition que l'on pourrait déterminer, en analysant les résultats fournis, les conditions que le nœud a pu observer en étant le seul à émettre.

Le fait de ré-utiliser un simulateur existant n'est donc pas une bonne solution en dépit des nombreux modèles qui ont déjà été développés pour chacun d'eux. Le nombre de simulations à effectuer pour obtenir les informations dont nous avons besoin est beaucoup trop important et ceci même si nous n'avons pas de contrainte de temps. De plus, ces simulateurs ne permettent pas de prévoir les cas de terminaux cachés ni de déterminer la composition d'une cellule. Ils se contentent d'en calculer les effets en fonction du trafic simulé dont ils disposent. Or, ces informations sont nécessaires pour pouvoir utiliser les observateurs au cours de la phase d'émulation.

En nous basant sur ce constat, nous avons donc choisi de développer le simulateur SWINE dédié à la génération de scénarios d'émulation.

5.3.2 Architecture de SWINE

SWINE est un simulateur à événements discrets conduit par le temps. Le processus de simulation est découpé en trois étapes indépendantes mais néanmoins complémentaires :

- Une étape de mobilité, au cours de laquelle l'ensemble des positions qu'un nœud va occuper au cours de l'expérience est calculé ;
- Une étape de propagation permettant de calculer la puissance reçue au niveau de chaque nœud pour chaque paire de nœuds composant le réseau et ceci pour chaque instant t de l'expérience ;
- Une étape de communication qui va déterminer les caractéristiques de niveau IP (débits utiles, délais, pertes) mais également rechercher les cas de terminaux cachés pour le mode Ad-Hoc, construire les cellules et calculer les différentes possibilités du scénario pour le mode infrastructure.

Ce processus d'émulation est présenté sur la figure 5.11.

SWINE est un simulateur orienté objet, développé en Java qui manipule principalement deux types d'objets :

- les *objets du domaine* qui vont représenter les éléments physiques du réseau (nœuds mobiles, obstacles ...)

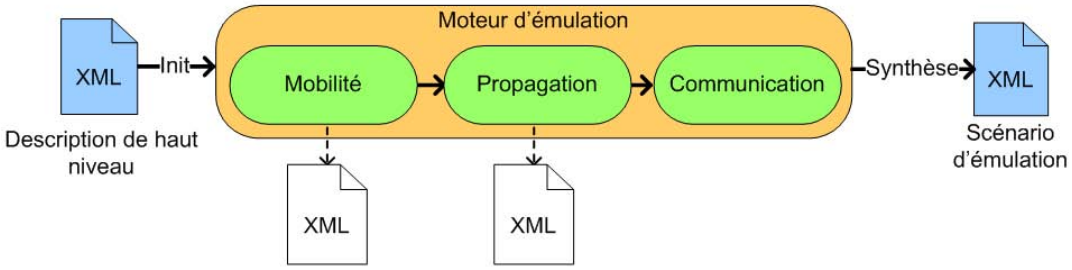


FIG. 5.11: Architecture du simulateur SWINE

– les *objets modèles* qui vont représenter les équations des modèles qui sont utilisés pour calculer les conditions du réseau. Ce sont les objets modèles qui vont mettre en œuvre les équations.

Ces deux types d'objets sont initialisés à partir du fichier de description de haut niveau fourni par l'utilisateur.

SWINE propose une architecture ouverte fonctionnant essentiellement autour des objets modèles. De manière à permettre d'ajouter simplement de nouveaux modèles, nous avons fait le choix d'utiliser le chargeur de classe (class loader) de Java. Cette solution évite à l'utilisateur de devoir modifier différents fichiers pour utiliser son nouveau modèle. Dans le simulateur NS-2 par exemple, une fois que le modèle est compilé il faut encore le faire reconnaître par le simulateur en spécifiant son nom ainsi que l'emplacement où se trouve l'objet compilé puis refaire l'édition de liens. L'utilisation du chargeur de classe permet donc à l'utilisateur de se concentrer uniquement sur la compilation de son modèle. Une fois le modèle compilé, il suffit juste de spécifier le chemin de la classe dans le fichier de description de haut niveau pour pouvoir l'utiliser. Lors de l'ajout d'un nouveau modèle, il est également nécessaire de mettre à jour le schéma RELAX-NG du fichier de haut niveau afin de prendre en compte les paramètres d'entrée du modèle.

5.3.3 Les objets du domaine

Les objets du domaine vont permettre de représenter les éléments physiques du réseau à émuler tels que les nœuds mobiles, les point d'accès pour le mode Infrastructure et les obstacles, mais également des éléments plus abstraits comme les cellules dans un réseau en mode Infrastructure ou les liens représentant la qualité d'une communication entre deux nœuds.

Tous ces objets du domaine sont regroupés sous forme d'attributs dans un objet *World*. Cet objet *World* va également contenir les objets modèles de niveau global (voir section 5.3.4) utilisés au cours d'une expérience ainsi que la matrice de liens qui va contenir les résultats de la simulation en dehors des résultats de mobilité. Un diagramme de classes partiel de *World* peut être retrouvé sur la figure 5.12.

La matrice de liens est l'élément clé de SWINE. C'est dans cette matrice que sont stockées les conditions d'une communication unidirectionnelle et ceci pour chaque paire de nœuds du réseau et à chaque instant t de l'expérience. Ces conditions vont regrouper les informations de niveau IP tels que le débit utile (*ipThroughput*), les délais (*delay*) et les pertes (*plr* et *packetLossFile* suivant que l'on utilise ou non l'extension KAUnet) mais également les informations de propagation comme la puissance reçue au niveau du nœud destination (*rx_power*), le rapport signal bruit (*snr*), le *fading*, le *shadowing* et le *path loss* et des informations de

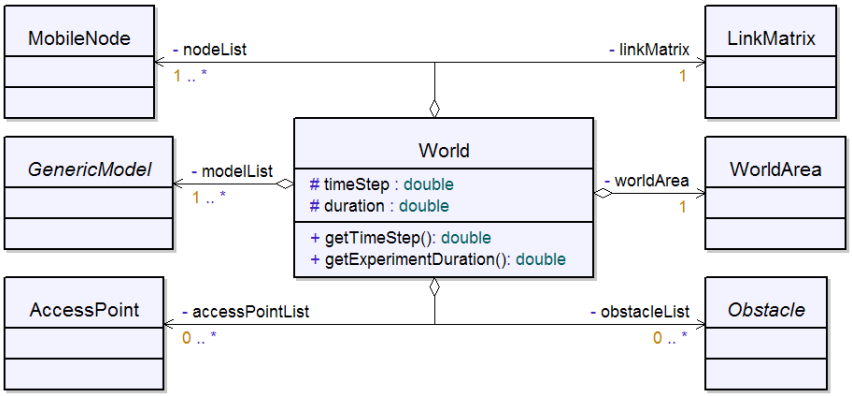


FIG. 5.12: Diagramme de classes UML simplifié de la classe World

communication comme par exemple celles liées à la gestion des terminaux cachés (*hiddenTerminalName*). Le diagramme de classes UML présenté sur la figure 5.13 illustre ces relations ainsi que les différents paramètres qui sont pris en compte par un objet de type *Conditions*.

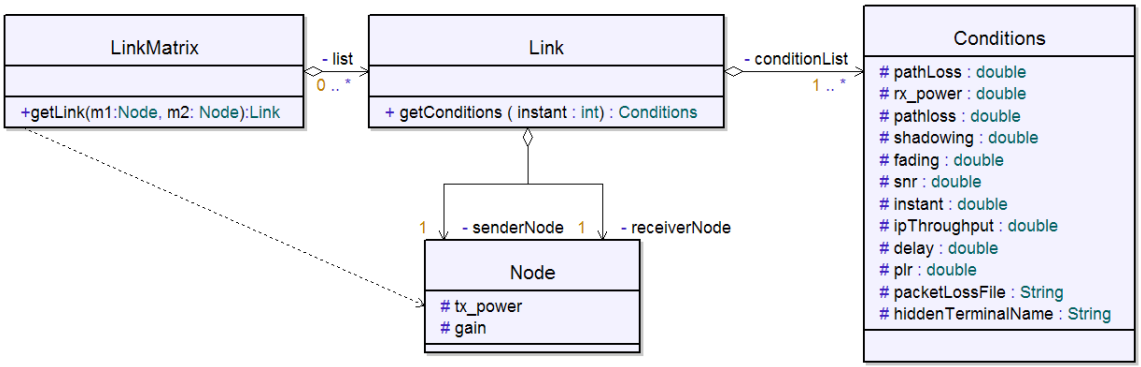


FIG. 5.13: Diagramme de classes UML simplifié de la matrice de liens

Un lien contient donc l'ensemble des informations concernant la communication entre deux nœuds. Il se caractérise par une liste d'objets *Conditions*, chaque objet *Conditions* correspondant aux conditions du lien à un instant t de l'expérience. Les seules informations à ne pas être contenues dans le lien sont les informations de mobilité.

Celles-ci étant déterminées à partir des modèles de mobilité qui sont propres à chaque nœud, elles peuvent être retrouvées en attributs de l'objet *Node* dont le diagramme de classes UML est présenté sur la figure 5.14. Un objet *Node* va ainsi contenir en attribut le nom qui le caractérise (*name*) la liste des nœuds voisins qu'il va avoir à travers toute la durée de l'expérience (*neighbours*), sa position initiale (*initialPosition*), la liste des modèles qu'il va utiliser (*modelsList*) et la puissance de transmission qu'il va utiliser (*tx_power*) et le gain de son antenne. De cet objet *Node* va notamment hériter l'objet *MobileNode* qui représente un nœud mobile caractérisé notamment pas ses niveaux d'énergie au cours de l'expérience (*energieLevelList*) mais surtout par la trajectoire qu'il va suivre (*trajectory*). Cette trajectoire est représentée à l'aide d'une liste d'objet *Position*. L'autre objet qui hérite de l'ojet *Node* est l'objet *AccessPoint* qui représente un point d'accès dans un réseau en mode Infrastructure.

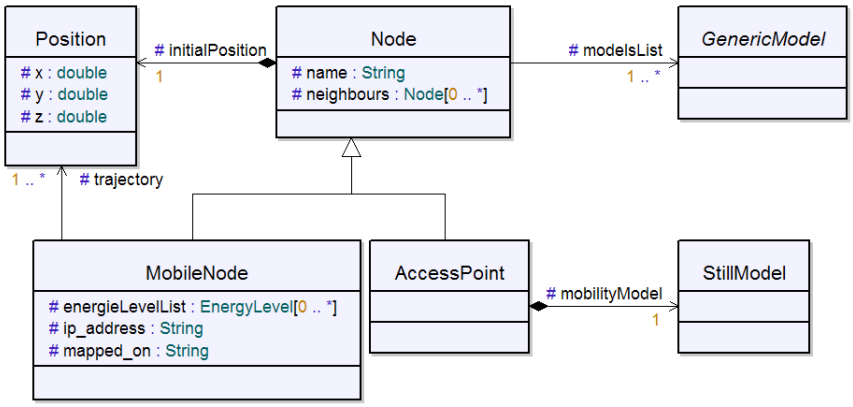


FIG. 5.14: Diagramme de classes UML simplifié d'un nœud

Nous avons considéré qu'un point d'accès était toujours fixe.

Les nœuds sont stockés sous forme de liste dans l'objet *World* et peuvent donc être retrouvés par son intermédiaire. Une autre solution est d'utiliser les objets *Link* contenus dans la matrice de lien qui possèdent en attribut une référence vers le nœud émetteur et le nœud récepteur qu'ils relient. La matrice de liens qui est également un attribut de l'objet *World* constitue également une troisième solution pour retrouver un nœud : elle permet de retrouver les liens en fonction d'un nœud source et d'un nœud destination. Une représentation informelle de cette matrice est donnée sur la figure 5.15.

Identifiant du nœud / Identifiant du nœud	M_1	M_2	M_3	M_4
M_1		Lien M_1-M_2	Lien M_1-M_3	Lien M_1-M_4
M_2	Lien M_2-M_1		Lien M_2-M_3	Lien M_2-M_4
M_3	Lien M_3-M_1	Lien M_3-M_2		Lien M_3-M_4
M_4	Lien M_4-M_1	Lien M_4-M_2	Lien M_4-M_3	

FIG. 5.15: Représentation informelle de la matrice de liens

Une remarque tout de même sur cette matrice : elle n'est pas symétrique. Dans un réseau sans fil les communications sont potentiellement unidirectionnelles si bien que le lien reliant le nœud M_1 au nœud M_2 est différent du lien reliant le nœud M_2 au nœud M_1. En effet, les effets de la propagation ne seront pas obligatoirement les mêmes ou alors il peut également arriver que le nœud M_1 utilise une puissance d'émission différente de celle utilisée par le nœud M_2. De

```
- <stage id="pathloss" class="swine.models.propagation.PathlossExponentModel">
    <exponent>4.6</exponent>
    <d0 unit="m">1</d0>
    <frequency unit="GHz">2.457</frequency>
  </stage>
```

FIG. 5.16: Extrait du fichier de description de haut niveau pour un modèle de Path Loss Exponent

même l'énergie résiduelle de chaque nœud peut avoir un impact sur sa puissance d'émission.

5.3.4 Les objets modèles

Les objets modèles vont représenter les différentes équations et algorithmes qui vont être utilisés pour déterminer la mobilité des nœuds et les conditions du réseau à émuler. Ces objets modèles vont utiliser les différents objets du domaine que nous venons de présenter et vont principalement servir à remplir la matrice de liens. Dans cette section nous allons présenter la conception que nous avons retenue pour utiliser les modèles et présenter les différents modèles que nous avons introduits. Dans la mesure où les différents modèles de mobilité et de propagation ont été présentés dans le chapitre précédent, nous insisterons principalement sur la partie communication et notamment sur les choix algorithmiques que nous avons été amenés à faire. Dans un premier temps nous présenterons le moteur de simulation qui va être chargé d'exécuter ces différents modèles puis nous illustrerons le fonctionnement de ces modèles à travers les phases de mobilité, de propagation et de communication.

5.3.4.1 Le moteur de simulation

Le moteur de simulation va devoir jouer plusieurs rôles : le premier est d'analyser le fichier de description de haut niveau fourni par l'utilisateur pour pouvoir construire les différents modèles qui devront être utilisés pendant la simulation ; le second sera d'exécuter ces différents modèles pour obtenir les résultats de la simulation.

La première étape consiste donc à analyser le fichier de description de haut niveau fourni par l'utilisateur pour déterminer d'un coté les modèles utilisés par chaque nœud puis les modèles régissant la propagation et les communications à l'intérieur du monde. La description XML de haut niveau va donc contenir le nom des modèles à utiliser mais également les paramètres d'entrée nécessaires à leur bon fonctionnement. Par exemple si nous prenons le modèle décrit sur la figure 5.16, nous pouvons voir que c'est un modèle de *Path Loss Exponent* avec comme paramètres d'entrée la fréquence, la valeur de l'exposant ainsi que la valeur de d_0. Nous pouvons également remarquer que cette description contient le nom complet de la classe à utiliser (ici *swine.models.propagation.PathLossExponentModel*). Ceci est une facilité de conception autorisée par le langage Java qui permet à SWINE de construire un objet sans connaître a priori sa classe par l'intermédiaire du "*class loader*". Cette technique permet d'ajouter simplement de nouveaux modèles sans avoir besoin de modifier le code du moteur de simulation.

Chaque fois qu'un nouveau modèle est construit, il est stocké en attribut d'un objet *stage*[1] qui sera ensuite chargé de le mettre en œuvre. Un objet stage comprend un nom (ici *pathloss*) que l'on retrouve dans le fichier de description de haut niveau par l'intermédiaire de l'attribut

[1] Pour éviter de confondre avec les étapes de mobilité, de propagation et de communication, nous utiliserons le terme anglais *stage* dans la suite de ce chapitre.

id et comprend également une suite de modèles. Comme nous utilisons un système de chargement de classe, il est nécessaire que les modèles fournis au simulateur respectent des règles d'héritage pour pouvoir être utilisés par l'objet stage qui le contient. Ainsi, chaque modèle va devoir hériter de la classe abstraite *GenericModel* présentée sur la figure 5.17 ou d'une de ces classes filles. Pour ajouter un modèle de mobilité il faudra que celui-ci hérite de la classe MobilityModel et implémente la méthode *update(node : Node, instant : Integer)* qui permet de déterminer la position d'un nœud à un instant donné. Un modèle de propagation implémentera quand à lui la méthode *update(link : Link, instant : Integer)* pour déterminer les conditions de propagation sur un lien à un instant donné. Un modèle de communication implémentera la méthode *update(linkMatrix : LinkMatrix, instant : Integer)* pour déterminer les conditions de communications (routage, terminaux cachés, communication en mode Infrastructure...) à un instant donné en tenant compte de tous les liens du réseau. Pour permettre à l'utilisateur de définir des modèles ne pouvant pas entrer dans l'une des trois premières catégories, nous avons introduit la classe *CrossStagesModel* qui oblige à implémenter les trois méthodes précédemment citées. De cette classe hériteront notamment les modèles de communication travaillant au niveau lien comme ceux permettant de passer de la puissance reçue à la bande passante de niveau IP.

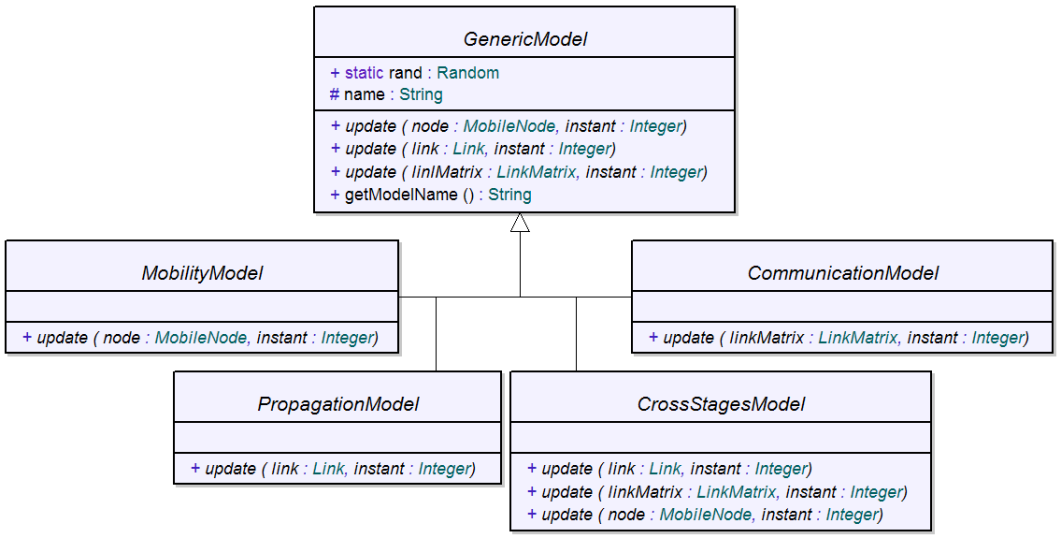


FIG. 5.17: Diagramme de classes UML simplifié présentant le Framework des modèles

Une fois les objets stage construits, ils sont mis en œuvre par le moteur de simulation dans l'ordre fourni par l'utilisateur. L'ordre est donc très important car il ne faudra pas par exemple simuler une étape de propagation avant d'avoir déterminé les positions que les nœuds vont occuper dans l'espace sous peine d'obtenir des résultats totalement incohérents.

Grâce à l'utilisation de l'héritage et des trois méthode abstraites *update(...)* de l'objet *GenericModel*, un objet stage est capable de mettre en œuvre n'importe quel modèle qui lui est fourni. Pour chaque modèle, il va déterminer si le modèle doit s'appliquer à un nœud ou au monde dans son ensemble en vérifiant le type du modèle. Si le modèle est une instance de *MobilityModel,* le stage va parcourir la liste des nœuds et appliquer la méthode *update(node :MobileNode, instant :Integer)* sur chaque nœud ayant ce modèle en attribut. Par contre, si le modèle est une instance de *PropagationModel,* le stage applique la méthode update

travaillant sur le lien et enfin, si c'est une instance de *CommunicationModel,* il applique celle travaillant avec la matrice. Si l'objet est de type *CrossStagesModel,* les trois méthodes sont alors exécutées. Ces méthodes sont appelées à chaque instant t de l'expérience ce qui permet à un stage de fournir des résultats couvrant toute la durée de l'expérience. La mise en œuvre des modèles par les stages est résumée sur le diagramme d'activité présenté sur la figure 5.18.

Sur ce diagramme, les booléens *iMobility*, *iCross*, *iPropagation* et *iCommunication* sont "vrai" lorsqu'un modèle est, respectivement, une instance de *MobilityModel*, *CrossStagesModel*, *PropagationModel* et *CommunicationModel*. Le booléen *modelOfNode* est lui "vrai" lorsqu'un modèle est bien stocké en attribut du nœud considéré.

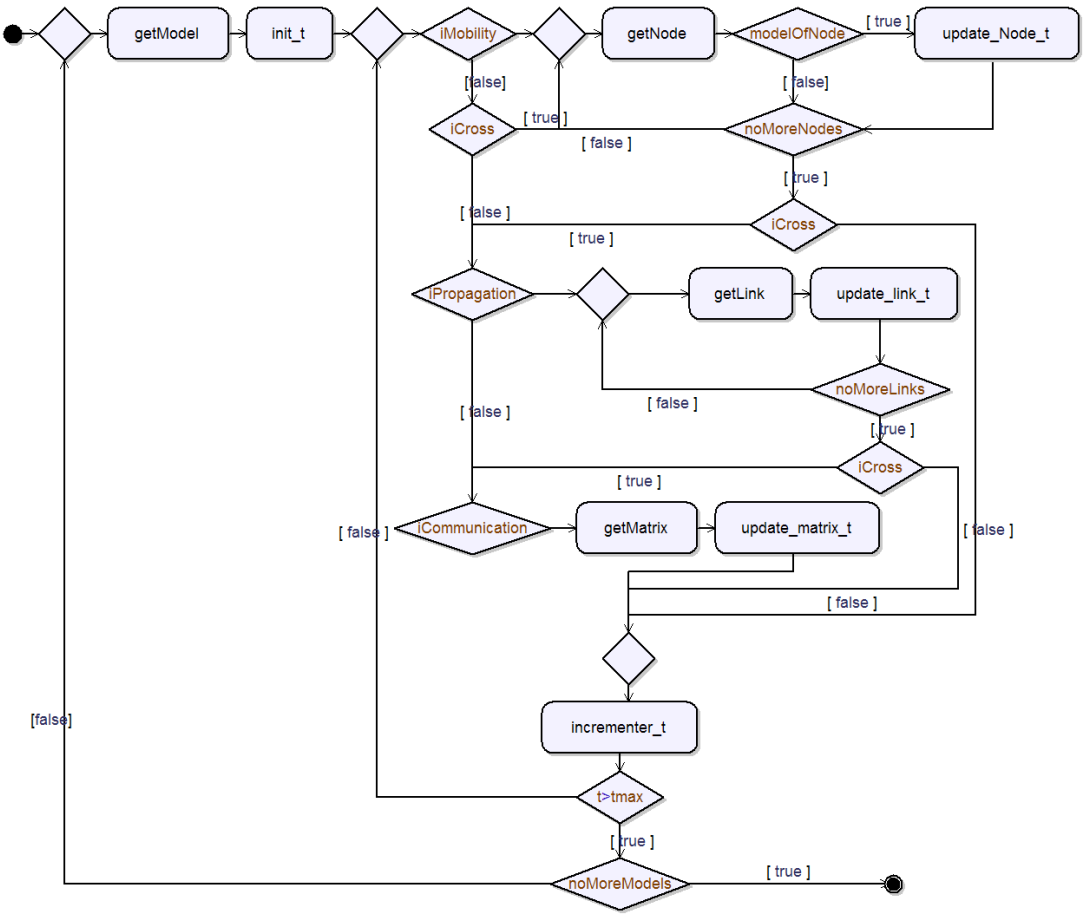


FIG. 5.18: Diagramme d'activité décrivant le fonctionnement d'un objet *Stage*.

Le diagramme de classe décrivant les relations entre les différents objets représentant les stages, les modèles et le moteur de simulation est présenté sur la figure 5.19

Nous allons maintenant entrer plus en détail dans le fonctionnement des étapes de mobilité, de propagation et de communication pour illustrer le déroulement d'une simulation avec SWINE. La première chose pour pouvoir déterminer les conditions d'un réseau sans fil à émuler est de prendre en compte la mobilité des nœuds.

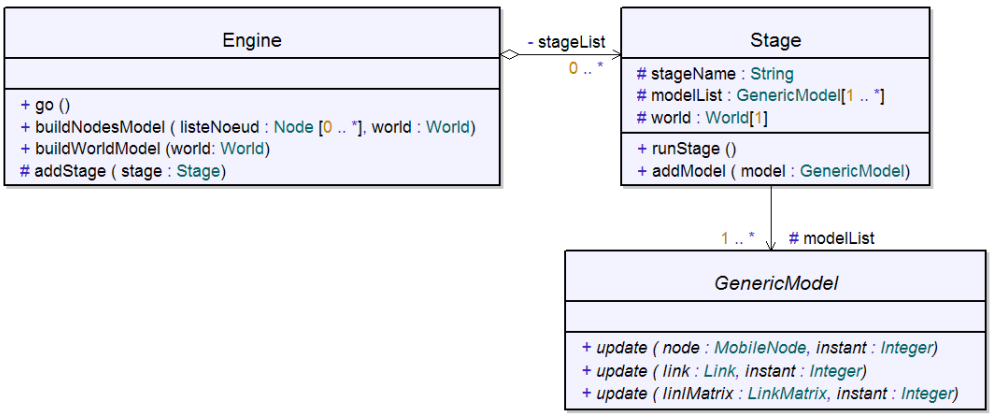


FIG. 5.19: Diagramme de classes UML simplifié du moteur d'émulation

5.3.4.2 L'étape mobilité

Dans cette étape, le moteur de simulation doit calculer les déplacements des nœuds à l'intérieur du monde virtuel en fonction de leur modèle de mobilité et des obstacles. Pour cela nous avons implémenté les différents modèles que l'on peut retrouver sur le diagramme de classes UML présenté sur la figure 5.20. Tous ces modèles vont suivre le même modèle de conception. Chaque modèle possède en attribut les informations dont il va avoir besoin pour pouvoir déterminer les positions. Le modèle de *RandomWalk* présenté sur la figure 5.20 va par exemple posséder en attribut la vitesse maximum qu'un nœud peut atteindre (*maxSpeed*). Le modèle de *RandomWaypoint* contiendra également le temps de pause maximum pendant lequel un nœud doit patienter lorsqu'il atteint une destination (*pauseTime*) avant d'en choisir une nouvelle et de se remettre en mouvement. Il n'y a pas de relation d'héritage entre ces deux modèle car mis à part la vitesse maximum leurs attributs sont différents et surtout les méthodes utilisées pour déterminer la prochaine position du nœud diffèrent.

Nous ne reviendrons pas sur les modèles de *RandomWaypoint*, de *RandomWalk* et de poursuite qui ont été présentés dans le chapitre précédent.

Le modèle *Rectilinear* permet à un nœud de se déplacer en ligne droite à travers le monde suivant un vecteur fixé par l'utilisateur. Lorsque ce nœud rencontre un obstacle ou la limite du monde, il rebondit sur celui-ci et repart dans la direction opposée. Ce modèle a été développé pour effectuer des tests simples et prédictibles.

Le modèle *PredefinedPath* permet à l'utilisateur de spécifier le chemin que va suivre le nœud. Ce chemin est décrit sous forme de balises que le nœud va rejoindre en se déplaçant à une vitesse constante, fixée par l'utilisateur. Ce modèle est particulièrement utile pour décrire le comportement d'un individu se déplaçant à l'intérieur d'un bâtiment. Bien qu'il soit plus contraignant à définir pour l'utilisateur qu'un modèle de *RandomWalk*, il permet d'avoir un comportement beaucoup plus réaliste et surtout beaucoup plus contrôlable. Les paramètres d'entré de ce modèle peuvent également être obtenus à partir d'un moteur de prédiction de mobilité externe. Il est en effet possible d'utiliser un tel moteur pour déterminer les différentes balises constituant le chemin que va devoir suivre un nœud mobile. Ces balises doivent ensuite être exprimées sous forme XML pour pouvoir être utilisées par le modèle *PredefinedPath*.

Pour permettre d'ajouter simplement un nouveau modèle de mobilité, nous avons choisi une approche basée sur la notion d'héritage. Ainsi, lorsqu'un utilisateur souhaite ajouter un

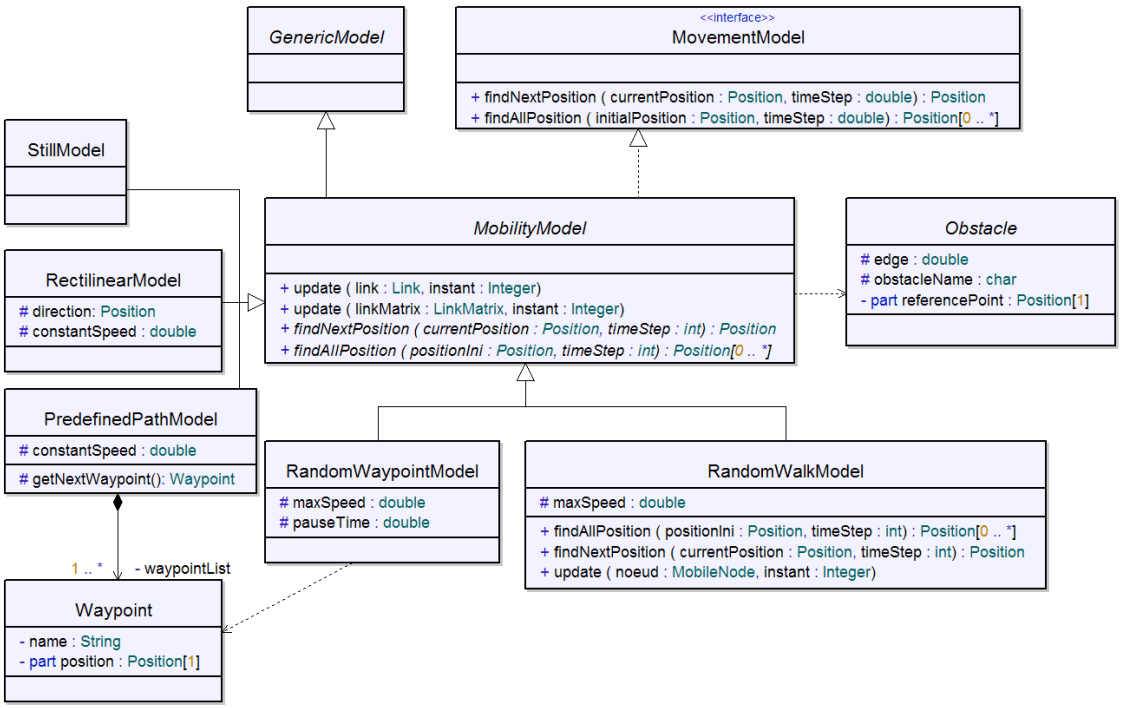


FIG. 5.20: Diagramme de classes des modèles de mobilité

modèle de mobilité dans SWINE, il peut choisir soit de spécialiser un modèle existant soit de spécialiser la classe mère abstraite *MobilityModel*. Pour que ce nouveau modèle puisse ensuite être utilisé par le moteur de simulation, il doit impérativement instancier la méthode *update(node :MobileNode, instant :Integer)* qui va permettre de calculer l'ensemble des positions d'un nœud au cours de l'expérience. Il est également nécessaire de définir le schéma XML de ce modèle pour qu'il puisse être utilisé dans la spécification de haut niveau. Enfin, les méthodes définies dans l'interface *MovementModel* doivent également être instanciées et servent à faciliter la conception d'un nouveau modèle à l'utilisateur.

5.3.4.3 L'étape propagation

Le composant propagation va travailler avec chaque paire de nœuds pour déterminer la puissance reçue au niveau du nœud récepteur en fonction du modèle de propagation défini par l'utilisateur dans le fichier de description de haut niveau. Dans la première version de SWINE nous avons implémenté plusieurs des modèles que nous avions présentés dans le chapitre précédent tels que les modèles de *Rayleigh*, de *Path Loss Exponent*, de *Two-Ray* etc. L'ensemble de ces modèles va instancier la méthode *update(lien :Link, instant :Integer)* qui permet de mettre à jours un lien reliant une paire de nœuds. Certains modèles comme le modèle de *Two-Ray* vont spécifier l'atténuation du signal en fonction de la distance et vont hériter d'un modèle abstrait de *pathloss*. Ils vont déterminer le *path loss* observable sur une communication entre deux nœuds. D'autres, qui vont hériter du modèle de *shadowing*, vont prendre en compte les effets à moyenne échelle et déterminer les effets des obstacles sur la communication. Enfin les modèles héritant du modèle de *fading* vont prendre en compte les effets à petite échelle sur la communication. Les différents modèles implémentés sont présentés sur la figure 5.21.

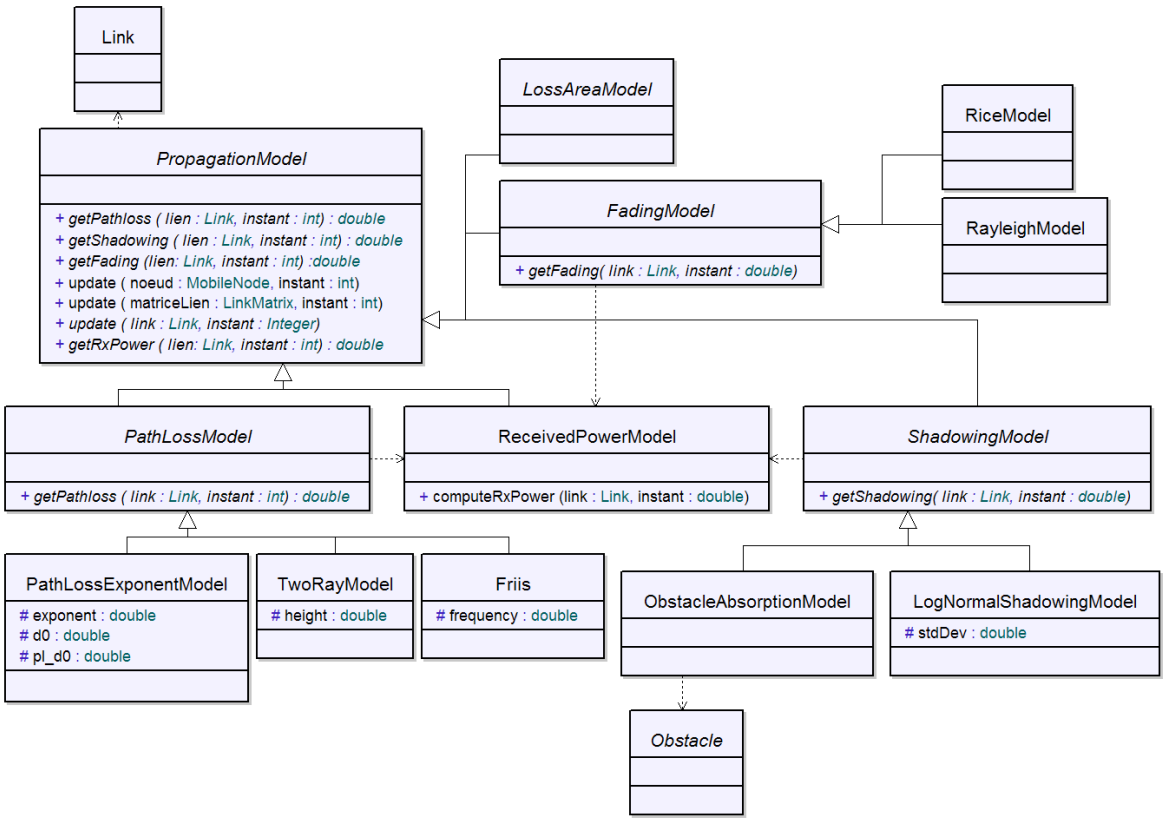


FIG. 5.21: Diagramme de classes des modèles de propagation

En dehors des modèles classiques de la littérature, nous avons développé plusieurs modèles pour simplifier les expériences de l'utilisateur et dont voici une description succinte.

Le modèle *ObstacleAbsorption* est un modèle de *shadowing* qui permet de spécifier pour chaque obstacle un facteur d'absorption du signal. Ce modèle est particulièrement utile pour isoler certains mobiles les uns des autres. Ainsi, un obstacle avec un taux d'absorption de 100% va bloquer totalement les transmissions radio si bien que deux nœuds séparés par un tel obstacle ne pourront jamais communiquer l'un avec l'autre de manière directe. Ils devront passer obligatoirement par un nœud intermédiaire. Ce modèle peut donc être utilisé pour facilement mettre en œuvre un topologie susceptible de conduire à des cas de terminaux cachés.

Le modèle *ReceivedPowerModel* doit par convention toujours être utilisé au début d'une étape de propagation. C'est ce modèle qui va calculer la puissance reçue au niveau du nœud et ceci pour chaque lien en s'appuyant sur les informations fournies par l'utilisateur (gain d'antenne, puissance de transmission). Ce modèle permet de déterminer la puissance reçue sans tenir compte d'autres informations que la distance séparant l'émetteur du récepteur. Les autres modèles de propagation vont ensuite se servir de cette puissance reçue pour déterminer les effets de la propagation. Une fois ces effets calculés, ils mettent à jour la valeur de la puissance reçue.

Toute l'étape de propagation se construit comme une arbre où la branche la plus basse se sert des calculs effectués par les branches de niveau supérieur pour effectuer ses calculs. Par

exemple, un modèle de *fading* de Rayleigh se servira de la puissance calculée par un modèle de *shadowing*. Ce modèle de *shadowing* se sera quant à lui servi de la valeur calculée par un modèle de *pathloss* qui aura été déterminée à partir de la puissance reçue calculée par le modèle *ReceivedPowerModel*. Notons cependant qu'il n'est pas obligatoire d'avoir un modèle de *shadowing* ou de *pathloss* pour utiliser un modèle de fading. Celui-ci peut directement utiliser la valeur obtenue par *ReceivedPowerModel* mais le réalisme s'en ressentira. Le diagramme d'activité de l'étape de propagation est présenté sur la figure 5.22. Sur celui-ci, les booléens *pathLossModel*, *shadowingModel* et *fadingModel* sont à "vrai" lorsqu'il y a, respectivement, une instance de *PathlossModel*, de *ShadowingModel* ou de *FadingModel* qui est utilisée dans une simulation.

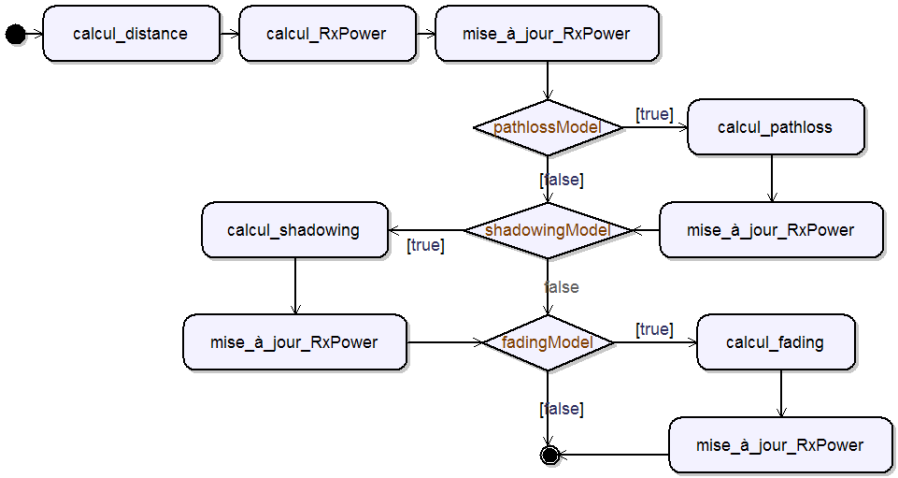


FIG. 5.22: Diagramme d'activité de l'étape de propagation

Les modèles héritant de *LossArea,* présentés sur la figure 5.23, ont été développés pour permettre de simplifier la gestion de la propagation. Pour cela, ces modèles utilisent les objets du domaine *Area* qui permettent de spécifier un taux de pertes de paquets pour une zone géographique donnée. Ils vont donc exploiter ces zones géographiques pour déterminer le PLR qui sera observé du coté du nœud récepteur. Si le nœud est à l'intérieur d'une zone, plusieurs choix s'offrent à nous en fonction du modèle sélectionné.

Le modèle *DDLossArea* permet de générer un fichier de pertes respectant ce PLR qui sera ensuite lu par l'extension KAUnet suivant le mode conduit par les donnés (DD=data driven). Une fois généré, le nom du fichier de pertes est mémorisé dans le champ *packetLossFile* de l'objet *Conditions* du lien cible à l'instant t. Il est ainsi possible de le retrouver lors de la génération du scénario d'émulation.

Le modèle *TDLossArea* va également servir à générer un fichier de pertes mais celui-ci devra par contre être lu par l'extension KAUnet en fonction du temps qui passe (TD=time driven).

Enfin, le modèle *PLRLossArea* permet d'utiliser la gestion classique des pertes de Dummynet en stockant la valeur du PLR observé dans la zone, au niveau du lien. Ce dernier modèle est en général déconseillé pour la comparaison de protocoles car les résultats qu'il fournit ne sont pas reproductibles d'une expérience à l'autre. Il peut cependant être utile lors de l'évaluation d'une application répartie pour avoir une idée de son comportement sur un réseau sans fil.

A la fin de cette étape, les conditions radios ont été calculées et les effets des modèles de niveau physique ont été pris en compte, il reste ensuite à traiter les effets des modèles de

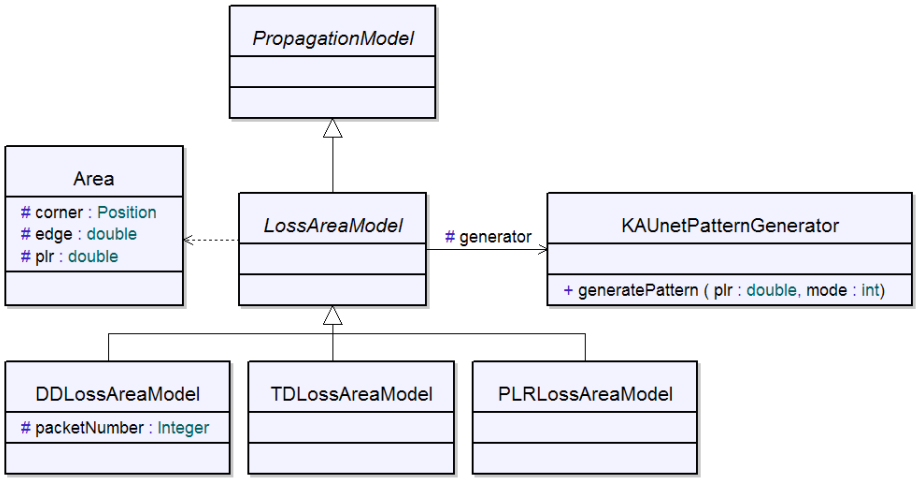


FIG. 5.23: Diagramme de classes présentant les modèles *LossArea*.

niveau MAC et supérieur. Ce sera le rôle de l'étape de communication.

5.3.4.4 L'étape de communication

Lors de l'étape de communication, seules les informations de niveau physique et les informations concernant la mobilité sont disponibles. Pour obtenir les conditions de QdS de niveau IP correspondantes, plusieurs solutions sont envisageables comme nous avons pu le voir dans la section 4.3.4.1, certaines étant basées sur la distance entre les nœuds, d'autres sur la puissance reçue. Pour répondre à ce besoin, différents modèles permettant de déterminer le débit utile au niveau IP ont donc été implémentés. Ces modèles, présentés sur le diagramme de classes UML de la figure 5.24, vont tous hériter de l'objet *CrossStages* car ils vont travailler au niveau de chaque lien.

Le modèle *DistanceBasedIPModel* permet de déterminer le débit utile au niveau IP sur un lien en se basant sur la distance séparant un nœud émetteur du nœud destination. Ce modèle utilise des tables, fournies par l'utilisateur dans le fichier de description de haut niveau, qui associent un débit utile de niveau IP à une distance. Ce débit utile de niveau IP peut être retrouvé à partir du taux de transmission radio en utilisant un backoff moyen, soit un délai inter-paquet de 674 μs pour un réseau 802.11 sans mécanisme de RTS/CTS. Dans le cas d'un réseau utilisant le RTS/CTS on calculera ce débit en tenant compte d'un délai inter-paquet de 1350 μs. Lors de l'étape de communication, la distance séparant l'émetteur du récepteur est comparée avec les seuils spécifiés dans la table. Si la distance est supérieure à la plus grande distance stockée dans la table (le seuil le plus élevé) on considèrera que la communication n'est pas possible et un taux de pertes de 100% sera appliqué sur le lien.

Le modèle *PowerBasedIPModel* permet de déterminer le débit utile dont peut disposer un nœud au niveau IP dans un réseau de type 802.11 en s'appuyant sur une table fournie par l'utilisateur. Cette table dont un exemple est donnée sur la figure 5.25, va associer un débit utile de niveau IP à une puissance reçue. Après l'étape de propagation, il est possible de comparer la puissance reçue au niveau de chaque nœud pour chaque lien et de chercher dans la table le seuil correspondant. Si la puissance calculée est supérieure à tous les seuils spécifiés dans la table, la communication n'est pas possible et on applique comme précédemment un

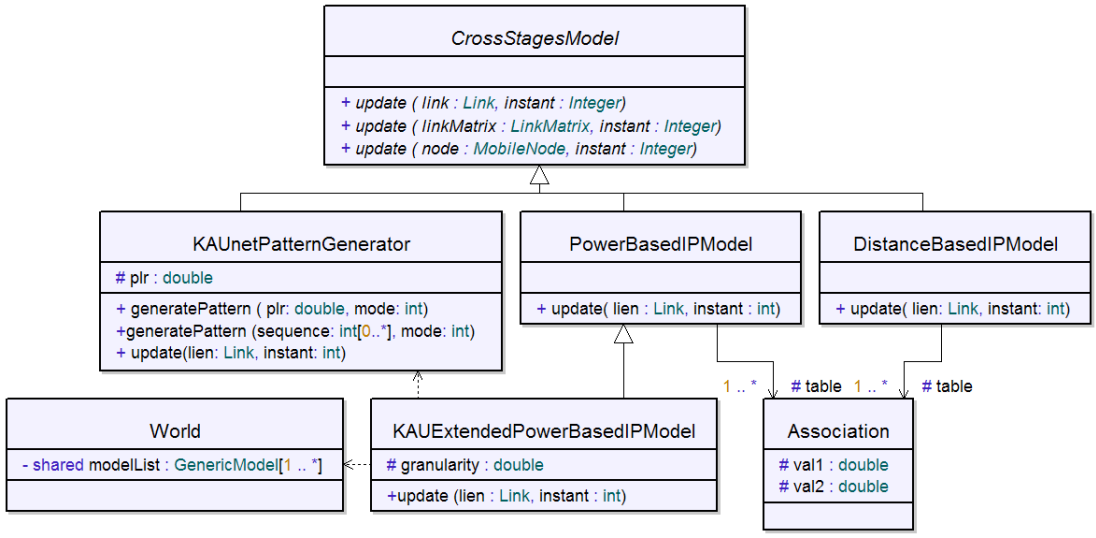


FIG. 5.24: Diagramme de classes des modèles de communication travaillant au niveau d'un lien

taux de perte de 100% sur le lien.

```
- <stage id="ip" class="swine.models.crosslayers.PowerBasedIPModel">
  - <power_table>
    - <entry>
        <received_power unit="dBm">-82</received_power>
        <ip_throughput unit="Mbit/s">6.87</ip_throughput>
      </entry>
    - <entry>
        <received_power unit="dBm">-87</received_power>
        <ip_throughput unit="Mbit/s">4.19</ip_throughput>
      </entry>
    - <entry>
        <received_power unit="dBm">-91</received_power>
        <ip_throughput unit="Mbit/s">1.77</ip_throughput>
      </entry>
    - <entry>
        <received_power unit="dBm">-94</received_power>
        <ip_throughput unit="Mbit/s">0.93</ip_throughput>
      </entry>
    </power_table>
  </stage>
```

FIG. 5.25: Extrait du fichier de haut niveau présentant une table de puissance.

Le modèle *KAUExtendedPowerBasedIPModel* est une extension du modèle précédent qui permet de générer des fichiers de pertes compatibles avec l'extension KAUnet lorsque la bande passante atteint le taux de transmission minimum de communication (1 Mbps pour un réseau IEEE 802.11) au lieu de spécifier un taux de perte de 100% comme dans les modèles précédents. Ces pertes sont calculées avec une granularité de 1 ms ce qui évite d'avoir des communications coupées pendant trop longtemps lorsque l'utilisateur a spécifié un intervalle de temps trop grand entre deux mises à jour des conditions. En effet un nœud qui se trouve assez loin de sa destination peut très bien voir sa communication rompue à un instant donné et la voir rétablie quelques dizaines de millisecondes plus tard grâce aux effets de propagation à petite échelle ou bien parce qu'il se sera déplacé de quelques centimètres.

Le modèle *KAUnetPatternGenerator* permet quant à lui de générer des fichiers de pertes

compatibles avec l'extension KAUnet en fonction d'un PLR constant spécifié par l'utilisateur. Ce modèle s'utilise en complément du modèle *DistanceBasedModel* ou du modèle *PowerBasedIPModel* car il ne permet pas de déterminer les débits utiles au niveau IP. Il permet d'avoir un taux de pertes constant dans un réseau, ce qui peut s'avérer utile pour évaluer une application répartie ou un protocole dans un environnement fortement bruité. Il faut également noter que bien que le PLR soit constant d'un lien à l'autre, les pertes ne surviendront pas forcément au même instant car chaque lien va avoir un fichier de pertes associé qui lui est propre. De manière générale, ce modèle est utilisé par tous les autres modèles utilisant des fichiers de pertes compatibles avec l'extension KAUnet . Plus précisément ils font appel aux méthodes generatePattern() de ce modèle pour obtenir les fichiers de pertes correspondants.

Les modèles héritant de l'objet *CommunicationModel* présentés sur la figure 5.26 permettent de prendre en compte les particularités liées au mode dans lequel le réseau sans fil va fonctionner (par exemple le routage pour un réseau en mode Ad-Hoc ou la construction des cellules pour un réseau en mode Infrastructure). Ces modèles ne vont donc plus travailler au niveau d'un lien mais au niveau de la matrice de lien dans son ensemble. Ils vont avoir besoin d'une vision plus étendue du réseau pour affiner les conditions à émuler.

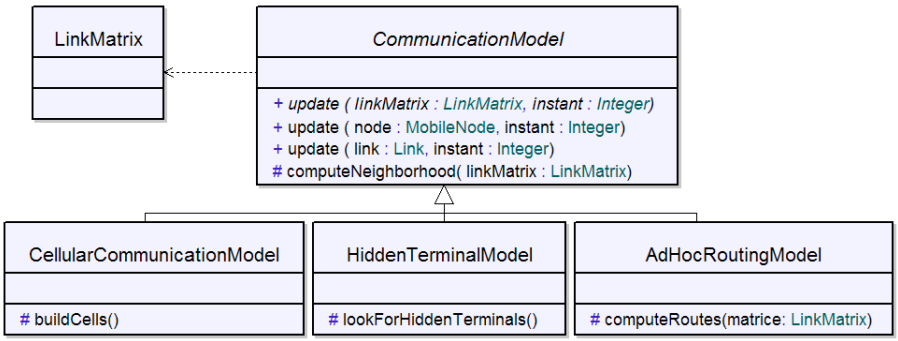


FIG. 5.26: Diagramme de classes pour la partie communication

Le modèle *CellularCommunicationModel* va commencer par construire les cellules au cours du temps. Cet algorithme va regarder pour chaque nœud avec quel point d'accès il a la meilleure connectivité. Le nœud est ensuite ajouté dans la cellule gérée par ce point d'accès. Pour déterminer les éventuels changements de cellules provoqués soit par la mobilité soit par la propagation nous avons utilisé l'algorithme suivant : tant qu'un nœud peut communiquer avec le point d'accès dont il dépendait à l'instant $t-1$, il reste connecté avec lui. On ne relance la procédure de recherche de meilleur point d'accès que lorsque la communication entre un nœud et le point d'accès avec lequel il communiquait est rompue. Dès que les cellules sont construites, il est possible de mettre à jour la matrice de liens en fonction de tables spécifiées par l'utilisateur. Pour le mode Infrastructure les tables sont au nombre de deux :

- La première table permet de déterminer les conditions de communication à l'intérieur de la cellule, c'est-à-dire les communications où le trafic émis par un nœud vers sa destination va entrer en concurrence avec lui même. En effet dans un réseau en mode Infrastructure lorsqu'un nœud communique avec un nœud présent dans la même cellule, il envoie son trafic vers le point d'accès qui se charge de l'acheminer à destination. Cela signifie donc que la communication entre la source et le point d'accès va se retrouver en concurrence

avec la communication entre le point d'accès et la destination lors de l'accès au canal.
- La seconde table va permettre de déterminer les communications entre deux nœuds situés dans des cellules différentes. Les délais observés sur ce type de communication sont donc identiques à ceux que l'on peut trouver dans une communication radio en ligne directe si l'on considère que les deux points d'accès sont reliés par un réseau de type Ethernet.

En regardant la composition des cellules il est donc possible de déterminer si une communication sera intra-cellulaire ou inter-cellulaire et mettre à jour les débits utiles de chaque lien.

Le modèle *HiddenTerminalModel* parcourt la matrice de liens et cherche les nœuds pouvant être à l'origine de cas de terminaux cachés. L'algorithme utilisé consiste à déterminer les nœuds situés dans le voisinage de chaque nœud pour chaque instant de l'expérience. Une fois les listes de voisins établies, l'algorithme va rechercher les voisins d'un nœud qui ne sont pas voisins entre eux. En effet, cela signifie que ces deux nœuds ont un voisin en commun mais n'ont pas connaissance l'un de l'autre ce qui peut entraîner des cas de terminaux cachés. Une fois les triplets repérés, les conditions du lien sont mises à jour (le champ hiddenTerminalName) pour que lors de la génération du scénario d'émulation deux choix soient produits. Lors de cette génération le modèle de HiddenTerminal sera réutilisé pour générer l'alternative au scénario classique.

Enfin, le modèle *AdHocRoutingModel* va mettre en œuvre un algorithme de Dijkstra pour permettre d'avoir des communications multi-bonds dans un réseau en mode Ad-Hoc. La métrique utilisée pour ce modèle est le nombre de sauts séparant la source de sa destination. Une fois que la route est établie, il est possible de déterminer les conditions rencontrées en utilisant les métriques suivantes :
- Le débit utile de la route est obtenu en déterminant le lien ayant le plus petit débit sur la route
- Le délai observé sera la somme des délais sur chaque lien composant la route
- Les pertes seront une combinaison des pertes de chaque lien lorsqu'un PLR est utilisé. Pour les fichiers de pertes, la métrique est plus complexe car il faut générer un nouveau fichier de pertes tenant compte des pertes observées sur chaque lien à un intervalle de temps t.

Bien que nous ayons prévu ce modèle dans le diagramme de classes UML, il n'a pas été intégré dans le prototype de SWINE.

Une fois l'ensemble des objets modèles exécutés, la matrice de lien va contenir l'ensemble des conditions du réseau au cours du temps. Cette matrice est donc parcourue puis les informations en sont extraites et synthétisées en un scénario d'émulation qui sera ensuite lu en temps réel au cours de la phase d'émulation. Cependant, avant cette phase d'émulation, il est possible pour l'utilisateur de visualiser les résultats grâce à un outil nommé Jackobi

5.3.5 L'outil de visualisation de scénario Jackobi

L'outil Jackobi a été initialement développé par des étudiants de première année de l'ENSICA dans le cadre d'un projet d'été et a évolué depuis lors pour intégrer les différentes améliorations apportées à SWINE. Cet outil s'appuie sur le scénario de haut niveau décrivant l'expérience pour construire l'environnement en trois dimensions et sur les traces générées aux étapes de mobilité, de propagation et de communication pour permettre ensuite au nœud de se mouvoir dans le monde. Les différentes communications possibles sont également représentées

et il est possible d'obtenir des informations sur un lien (bande passante, délais, pertes...) ou sur un nœud (nom, adresse IP...).Un exemple de visualisation est présenté sur la figure 5.27.

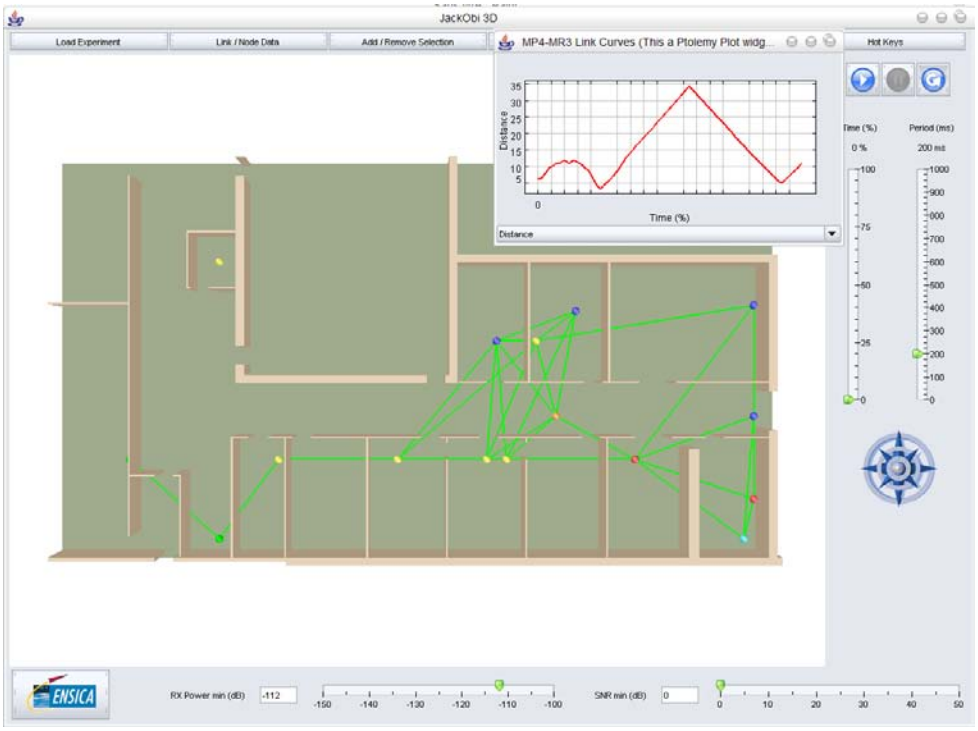


FIG. 5.27: Exemple de visualisation avec SWINE

Les différentes couleurs utilisées par les nœuds et que l'on peut retrouver sur la figure 5.27, correspondent en fait à différents modèles de mobilité et permettent ainsi à l'utilisateur de visualiser si le comportement des nœuds est bien celui qu'il attendait. Les différentes lignes correspondent aux communications qui peuvent avoir lieu à un instant donné. Celle-ci vont donc apparaître et disparaître au cours du temps en fonction de la propagation et de la mobilité des nœuds. En sélectionnant deux nœuds, il est ainsi possible d'obtenir des informations sur le lien qui les relie comme la bande passante, la puissance reçue, le *pathloss* etc. Un exemple de visualisation de ces conditions est également donné sur la figure 5.27 et montre l'évolution de la distance entre deux nœuds. Grâce à cet outil, l'utilisateur peut avoir une représentation intuitive de l'expérience et donc mieux interpréter les résultats qu'il obtiendra lors de la phase d'émulation. De même cette visualisation peut permettre de repérer des erreurs dans l'expérience ou d'aménager le scénario. Il suffit alors de corriger le fichier de haut niveau (en repositionnant les nœuds par exemple) et de re-simuler le scénario d'émulation.

5.3.6 Conclusion sur SWINE

Le simulateur de réseaux sans fil SWINE est un simulateur dédié à l'émulation. Il permet, en se basant sur une description de haut niveau fournie par l'utilisateur, de déterminer les conditions qui devront être reproduites au cours de la phase d'émulation. Pour cela, il repose sur une architecture découpée en trois étapes :

– Une étape de mobilité permettant de déterminer les différentes positions des nœuds au

cours du temps ;
- Une étape de propagation chargée de déterminer les conditions de propagation à l'intérieur du réseau à émuler et de déterminer la puissance potentiellement reçue au niveau de chaque nœud après transmission d'un message par un autre nœud ;
- Une étape de communication permettant de déterminer les conditions que l'émulateur devra reproduire au niveau IP et d'éventuellement générer les différents choix de scénarios qui peuvent survenir au cours d'une expérience.

Ces trois étapes génèrent des traces qui peuvent ensuite être visualisées à l'aide de l'outil Jackobi pour estimer de manière intuitive (et donc rapide) les résultats obtenus et synthétisés dans le scénario d'émulation.

5.4 Conclusion sur W-NINE

La plate-forme d'émulation de réseaux sans fil W-NINE repose sur l'utilisation successive de deux phases : l'une de simulation et l'autre d'émulation. Pour cela, W-NINE s'appuie au cours la phase de simulation sur SWINE, un simulateur de réseau sans fil pour l'émulation, qui va être chargé de générer un scénario d'émulation en se basant sur une description de haut niveau fournie par l'utilisateur. Lors de la phase d'émulation, ce scénario est rejoué en temps réel sur la plate-forme d'émulation filaire NINE par un gestionnaire d'émulation chargé également d'effectuer le choix du scénario à rejouer en fonction des informations sur le trafic qui circule au cours d'une expérience. Ces informations lui sont fournies par les observateurs de réseaux présent sur le routeur-émulateur. En fonction de ces informations, le gestionnaire d'émulation va déterminer quelles conditions doivent être reproduites par KAUnet, une extension de Dummynet supportant les pertes reproductibles.

La plate-forme d'émulation W-NINE répond donc aux différents besoins que nous avions énoncés à savoir la précision des conditions, le dynamisme de ces conditions et la possibilité de reproduire fidèlement une expérience. Le besoin de précision est résolu par l'utilisation de la phase de simulation hors ligne qui permet d'utiliser des modèles précis et ceci même s'ils sont coûteux en temps de calcul. Le dynamisme des conditions est quant à lui assuré par le scénario d'émulation qui va spécifier les changements de conditions du réseau à travers le temps et par le gestionnaire d'émulation qui est chargé de le lire. Enfin, les expériences d'émulation sont pleinement reproductibles d'une part grâce à l'utilisation d'un scénario et d'autre part grâce à l'utilisation de l'extension KAUnet développée par l'équipe de Karlstad qui permet de lever l'indéterminisme présent dans Dummynet en utilisant un placement précis des pertes. Dans le chapitre suivant nous allons illustrer les différents mécanismes que nous avons présentés dans ce chapitre en insistant surtout sur la cohérence entre la phase de simulation et la phase d'émulation et sur l'utilisation des observateurs pour prendre en compte des problèmes comme celui des terminaux cachés.

6 Mise en œuvre de W-NINE sur des études de cas

6.1 Introduction

Dans le chapitre précédent nous avons présenté l'architecture de W-NINE, une plateforme d'émulation de réseau sans fil reposant sur l'utilisation d'une phase de simulation hors ligne préalablement à une phase d'émulation. Avec cette architecture, le passage de la phase de simulation à la phase d'émulation doit être le plus précis possible de manière à ne pas perdre l'avantage de la simulation hors ligne. En effet, si la phase d'émulation n'est pas capable de reproduire fidèlement le scénario d'émulation en temps réel, l'utilisation de modèles précis lors de la phase de simulation perd de son intérêt. L'aspect dynamique des scénarios d'émulation produits par SWINE doit également pouvoir être retrouvé au niveau de l'émulation. Le premier cas d'utilisation illustrera et discutera ce passage de la phase de simulation à la phase d'émulation.

Pour pouvoir gérer les problèmes inhérents au trafic circulant sur un réseau sans fil au cours d'une expérience, nous avons proposé l'utilisation d'observateurs de réseau. Comme nous l'avons expliqué dans le chapitre précédent, le gestionnaire d'émulation va s'appuyer sur ces observateurs pour choisir les règles qu'il doit envoyer au routeur émulateur. Par conséquent, le délai entre cette phase d'observation et de détection des événements de trafic et la prise de décision correspondante doit être réduite au minimum pour que l'utilisateur obtienne des résultats réalistes.

Enfin, nous montrerons dans le troisième cas d'utilisation l'intérêt d'utiliser des fichiers de pertes compatibles avec l'extension KAUnet pour émuler un réseau sans fil. Nous discuterons notamment l'évolution du débit IP obtenu à l'aide de pertes caractérisant les conditions de propagation simulées.

6.2 Passage de la phase de simulation à la phase d'émulation

Dans ce cas d'utilisation, nous illustrerons le passage de la phase de simulation à la phase d'émulation en discutant les résultats obtenus lors de chaque phase. L'objectif est ici de montrer que la précision des modèles lors de la phase de simulation est bien conservée lors de la phase d'émulation. De même nous pourrons illustrer avec cette expérience l'aspect dynamique des scénarios.

6.2.1 Description de l'expérience

Pour illustrer le passage de la phase de simulation à la phase d'émulation nous avons choisi un cas d'utilisation relativement simple. Deux nœuds en mode Ad-Hoc 802.11 simple bond vont communiquer à l'intérieur d'un bâtiment représentant le Département de Mathématiques et

d'Informatique (DMI) de l'École Nationale Supérieure d'Ingénieurs en Construction Aéronautique (ENSICA). Une représentation graphique du DMI peut être retrouvée sur la figure 6.1. Le nœud $F1$ restera à la même position pendant toute la durée de l'expérience tandis que le nœud $M1$ suivra un chemin prédéfini ($1 \rightarrow 2 \rightarrow 3 \rightarrow 2 \rightarrow 4 \rightarrow 5 \rightarrow 6$) à travers le département en se déplaçant à une vitesse constante de 1.5 m/s.

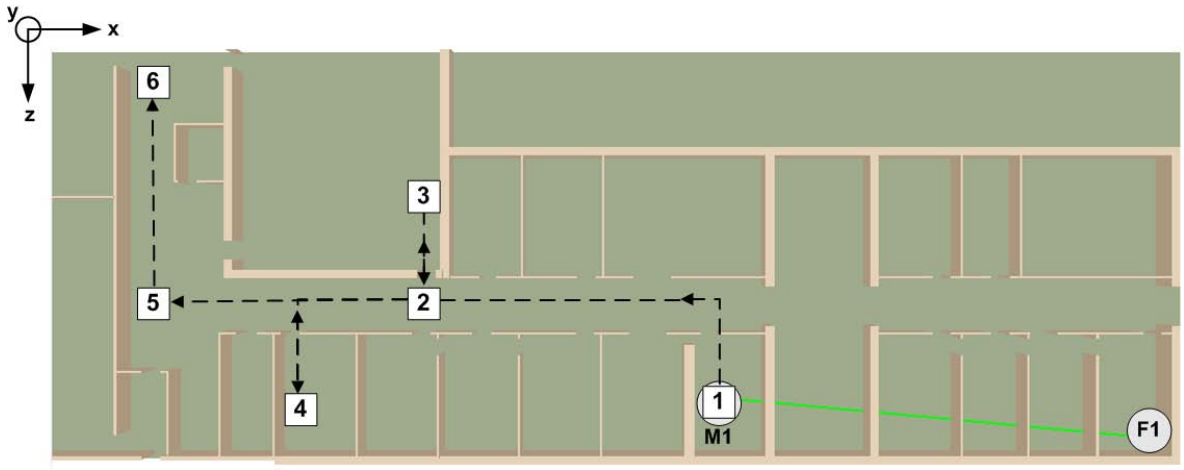


FIG. 6.1: Visualisation graphique de l'expérience à l'aide de Jackobi

Au cours de cette expérience, le nœud $F1$ va émettre un flux CBR (*Constant Bit Rate*) de datagrammes UDP de 1472 octets à destination du nœud mobile $M1$ et ce, avec un débit de 10 Mb/s au niveau applicatif. Un extrait du fichier de haut niveau décrivant les modèles de mobilité ainsi que les principales caractéristiques de ces nœuds est présenté sur la figure 6.2.

```
- <mobile id="F1">
  - <models>
    - <stage id="mobility" class="swine.models.mobility.Still">
      - <initial_position>
          <x>54.5</x>
          <y>0</y>
          <z>19</z>
        </initial_position>
      </stage>
    </models>
    <gain unit="dB">0</gain>
    <tx_power unit="dB">-15</tx_power>
    <ip_address>192.168.3.101</ip_address>
    <ip_mask>255.255.255.0</ip_mask>
    <gateway>192.168.3.100</gateway>
    <member_of>Emulated WiFi Network</member_of>
    <mapped_on>nine1</mapped_on>
  </mobile>
```

```
- <mobile id="M1">
  - <models>
    - <stage id="mobility" class="swine.models.mobility.SimplePath">
      - <waypoint>
          <x>33</x>
          <y>0</y>
          <z>17</z>
        </waypoint>
      + <waypoint>
      + <waypoint>
      + <waypoint>
      + <waypoint>
      + <waypoint>
      + <waypoint>
      + <waypoint>
      + <waypoint>
      + <waypoint>
        <speed unit="m/s">1.5</speed>
      </stage>
    </models>
    <gain unit="dB">0</gain>
    <tx_power unit="dB">-15</tx_power>
    <ip_address>192.168.4.101</ip_address>
    <ip_mask>255.255.255.0</ip_mask>
    <gateway>192.168.4.100</gateway>
    <member_of>Emulated WiFi Network</member_of>
    <mapped_on>wnine1</mapped_on>
  </mobile>
```

FIG. 6.2: Extrait du fichier de haut niveau décrivant les nœuds F1 et M1.

Nous pouvons remarquer sur cet extrait que les nœuds ne font pas partie du même sous-

réseau IP. En effet, le routeur-émulateur que nous allons utiliser lors de la phase d'émulation est configuré en mode passerelle si bien qu'il est nécessaire que les deux nœuds soient situés dans des sous-réseaux IP différents. Ceci est donc une facilité de déploiement lors de la phase d'émulation qui ne remet pas en cause les résultats que nous allons obtenir. Dans le cas où des tests nécessiteraient d'avoir absolument le même sous-réseau IP pour chaque nœud, il est possible de configurer le routeur-émulateur en mode pont.

Pour décrire l'environnement, nous avons utilisé dans un premier temps, un modèle de propagation à grande échelle : le *Path Loss Exponent*. Les paramètres que nous avons utilisés pour cette expérience ont été obtenus après une campagne de mesures réalisée à l'intérieur du département. Au cours de cette campagne, un nœud mobile se déplaçait dans le couloir du DMI. A partir des différentes traces que nous avons collectées grâce au logiciel fourni avec les cartes sans fils utilisées (ORINOCO Silver), nous avons effectué une régression linéaire pour obtenir les paramètres d'un modèle de *Path Loss Exponent*. Conformément à ce que l'on peut retrouver dans la littérature où la valeur d'un exposant pour une communication en intérieur est comprise entre 4 et 6, nous avons obtenu un $n = 5.68$. Cette valeur relativement élevée peut s'expliquer par la présence de nombreuses sources d'interférences électriques à l'intérieur du département. L'utilisation de souris et de clavier bluetooth par certains membres du département a également pu jouer sur cette valeur. Toujours en utilisant cette technique de régression linéaire nous avons obtenu une valeur $PL(d_0) = 19.97\,\mathrm{dB}$.

Ce modèle de propagation à grande échelle va s'appuyer sur un modèle *Received Power* qui va déterminer la puissance reçue en fonction des gains d'antennes et de la puissance de transmission qui ont été spécifiés dans la description des nœuds $F1$ et $M1$. Le modèle de *Path Loss Exponent* propose un comportement linéaire qui est assez éloigné de ce que l'on peut observer dans la réalité car les chemins multiples de l'onde ne sont pas pris en compte. Pour améliorer le réalisme nous avons donc effectué une seconde expérience en ajoutant cette fois un modèle de *Rayleigh* qui va permettre de prendre en compte les variations à petite échelle. Dans la mesure où nous gardons les même paramètres pour le modèle de *Path Loss Exponent*, la puissance reçue au niveau de $M1$ que nous allons obtenir avec cette combinaison de modèles sera moins bonne que celle observée dans le réseau réel. Cependant, cet ajout est intéressant pour voir le comportement de l'émulateur avec des conditions changeant de manière brusque.

Enfin, le modèle de communication utilisé est un modèle à base de table permettant de déterminer le débit utile au niveau IP en fonction de la puissance reçue au niveau de chaque nœud. Les modèles utilisés au cours de l'expérience sans *fading* de *Rayleigh* ainsi que leurs paramètres sont présentées sur la figure 6.3.

Pour finir, il faut préciser que l'expérience va durer 40 s et qu'elle a été réalisée une première fois avec des conditions calculées toutes les 10 ms et une seconde fois avec des conditions calculées toutes les 100 ms. La graine utilisée pour la génération des nombres aléatoires *(seed)* au cours de ces deux expériences n'est pas la même si bien que l'on pourra observer des différences de comportement entre les courbes utilisant le modèle de *Rayleigh*.

6.2.2 La phase de simulation

En utilisant un modèle de *Path Loss Exponent*, la puissance reçue (en dB) au niveau du nœud $M1$ devrait être inversement proportionnelle à la distance qui le sépare du nœud émetteur $F1$. Les résultats obtenus par SWINE après les étapes de mobilité et de propagation sont présentés sur la figure 6.4 a) pour l'expérience avec un pas de temps de 10 ms et sur la figure 6.4 b) pour l'expérience avec un pas de temps de 100 ms.

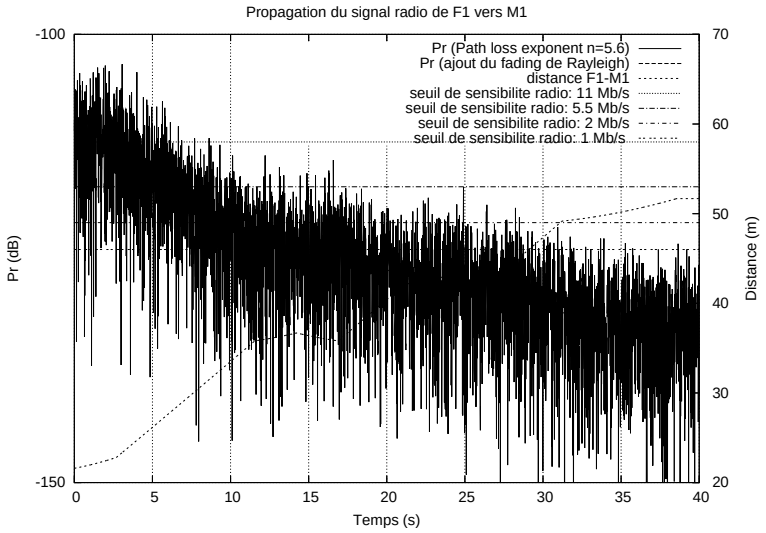
```
- <models>
    <stage id="distance" class="swine.models.propagation.EuclidianDistance" />
    <stage id="rxPower" class="swine.models.propagation.ReceivedPowerModel" />
  - <stage id="pathloss" class="swine.models.propagation.PathlossExponentModel">
      <exponent>5.68</exponent>
      <d0 unit="m">1</d0>
      <frequency unit="GHz">2.457</frequency>
      <pathloss_d0 unit="dB">19.97</pathloss_d0>
    </stage>
  - <stage id="ip" class="swine.models.crosslayers.PowerBasedIPModel">
    - <power_table>
      - <entry>
          <received_power unit="dBm">-82</received_power>
          <ip_throughput unit="Mbit/s">6.87</ip_throughput>
        </entry>
      - <entry>
          <received_power unit="dBm">-87</received_power>
          <ip_throughput unit="Mbit/s">4.19</ip_throughput>
        </entry>
      - <entry>
          <received_power unit="dBm">-91</received_power>
          <ip_throughput unit="Mbit/s">1.77</ip_throughput>
        </entry>
      - <entry>
          <received_power unit="dBm">-94</received_power>
          <ip_throughput unit="Mbit/s">0.93</ip_throughput>
        </entry>
      </power_table>
    </stage>
  </models>
```

Fig. 6.3: Extrait du fichier de haut niveau présentant les modèles utilisés.

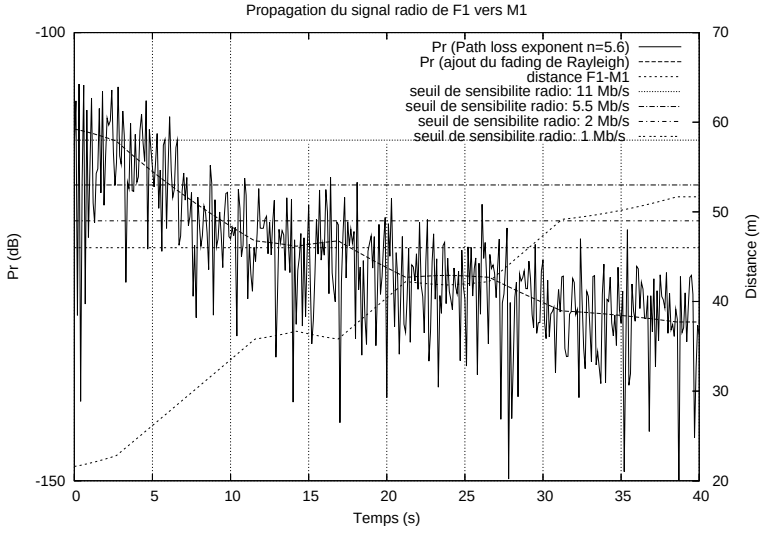
Conformément à ce que nous attendions, la puissance reçue obtenue avec le modèle de *Path Loss Exponent* est bien inversement proportionnelle à la distance séparant les nœuds. Ainsi, nous pouvons voir que plus la distance augmente entre $M1$ et $F1$ et plus la puissance reçue diminue. Le modèle de *Rayleigh* se comporte lui aussi comme espéré en faisant varier la puissance reçue de manière significative par rapport au modèle de *Path Loss Exponent*. Il est d'ailleurs possible de remarquer que, dans certains cas, ce modèle va permettre d'améliorer les communications.

Une fois l'étape de propagation terminée, il est possible d'utiliser le modèle de communication à base de table pour déterminer le débit au niveau IP dont va disposer le nœud $M1$. Dans la figure 6.5, si nous regardons la courbe de puissance reçue n'utilisant pas de *fading* de Rayleigh, entre 5 et 10 secondes, nous voyons que le signal diminue et franchit le seuil correspondant à un taux de transmission de 5.5 Mb/s (soit un débit utile IP maximum de 4.19 Mb/s) à $t = 6.13$ s. Par conséquent, le débit correspondant au niveau IP va passer de 4.19 Mb/s à 1.77 Mb/s. Les variations de débit obtenues après l'application de ce modèle avec un pas de temps de 10 ms sont donc bien celles qui étaient espérées.

Lorsque l'on ajoute un *fading* de Rayleigh, le principe est le même mais cette fois-ci les variations sont beaucoup plus importantes et fréquentes que dans le modèle précédent où le débit diminuait de manière régulière jusqu'à atteindre un débit nul à $t = 17.55$ s. Les résultats présentés sur la figure 6.6 montrent les variations du débit avec un *fading* de Rayleigh pour des pas de temps de 10 ms et de 100 ms. Les résultats avec un pas de temps de 10 ms présentent des variations importantes. Cette grande variation de la puissance reçue va donc se traduire par une grande évolutivité du débit au niveau IP. Ainsi, celui-ci peut passer de 6.87 Mb/s à 0.93 Mb/s entre deux pas de temps de 10 ms puis remonter à 6.87 Mb/s lors de l'intervalle



(a) pas de temps de 10 ms



(b) pas de temps de 100 ms

FIG. 6.4: Résultat de simulation pour les étapes de mobilité et de propagation.

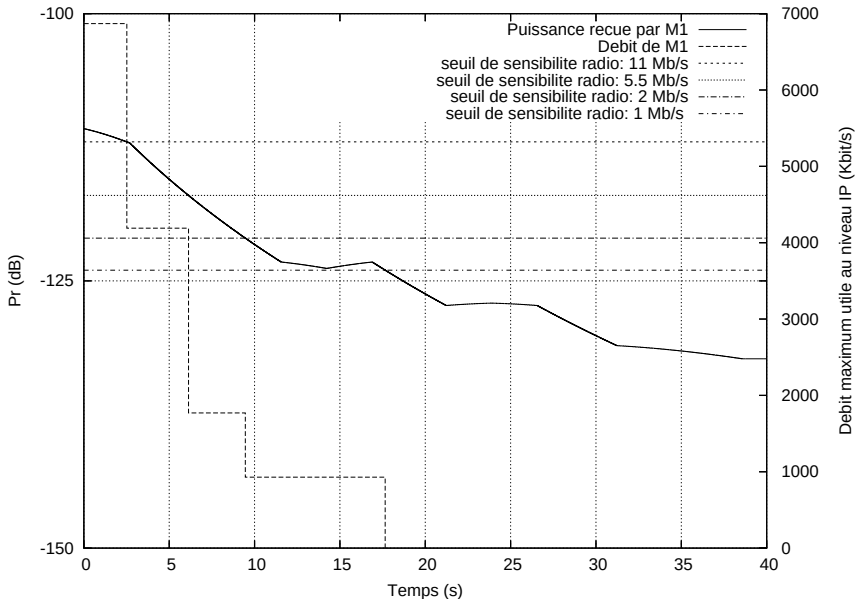


FIG. 6.5: Calcul du débit au niveau IP en fonction de la puissance reçue (sans fading de Rayleigh)

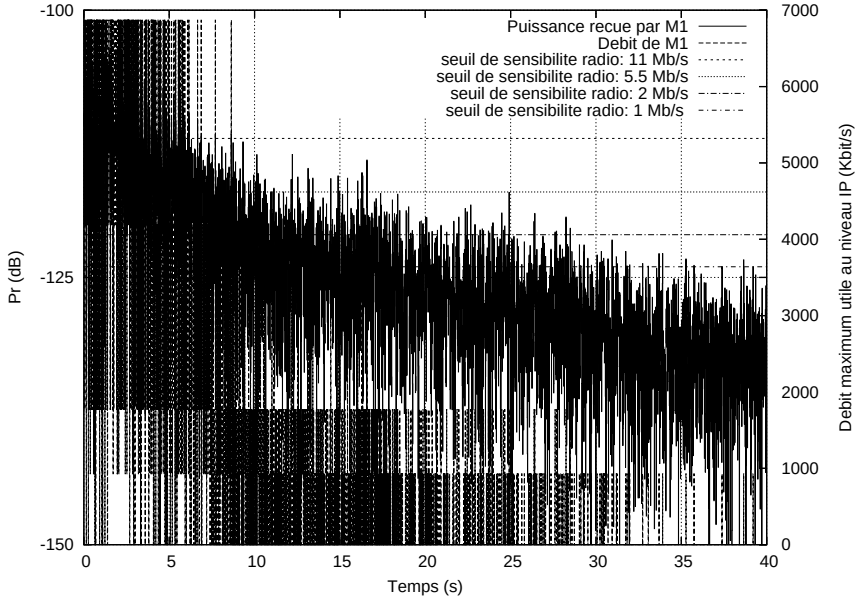
suivant. L'expérience avec un pas de temps de 100 ms présente également de nombreuses variations mais le pas de temps plus grand les rend plus simples à interpréter que dans le cas précédent. Il est ainsi possible de voir sur la figure 6.6 b) que les variations du débit IP sont bien directement liées au franchissement des seuils de transmission radio par la puissance reçue.

6.2.3 La phase d'émulation

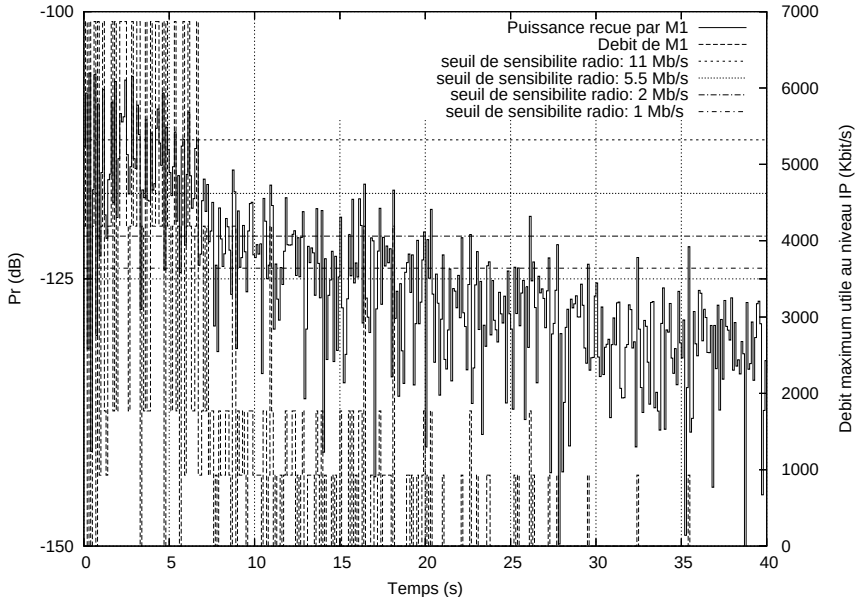
Pour réaliser les mesures de débit au cours de l'émulation, nous avons utilisé le générateur de trafic *Multi-Generator* (MGEN) développé par le *Naval Research Laboratory* (NRL). Cet outil permet de générer du trafic UDP/IP en mode *unicast* et en mode *multicast*. Dans l'expérience qui nous intéresse, nous avons généré un flux CBR de datagrammes UDP de taille 1472 octets à une vitesse de 8 Mb/s. Ce débit est supérieur à celui que nous voulons émuler ce qui va nous permettre de vérifier que les contraintes appliquées respectent bien les règles d'émulation obtenues par simulation.

Les machines jouant le rôle de $M1$ et de $F1$ sont synchronisées avec le gestionnaire d'émulation à l'aide du protocole *Network Time Protocol* (NTP) pour pouvoir comparer les résultats obtenus lors de la phase de simulation et lors de la phase d'émulation. Cette synchronisation n'est pas obligatoire pour évaluer un protocole ou une application mais va nous permettre de déterminer si un délai est introduit entre l'envoi des règles par le gestionnaire d'émulation et leur application par le conditionneur de trafic. En effet, avec cette solution, si un délai est observé entre les résultats de simulation et d'émulation, il ne peut être causé que par le conditionneur de trafic.

Enfin, les traces obtenues ont été analysées grâce à l'outil *TRace Plot Real-time* (TRPR) également développé au NRL. Celui-ci permet notamment d'extraire des traces obtenues par l'outil MGEN, des informations telles que le débit IP, les délais, les taux de pertes, le *jitter*



(a) pas de temps de 10ms



(b) pas de temps de 100 ms

FIG. 6.6: Calcul du débit au niveau IP en fonction de la puissance reçue (ajout du fading de Rayleigh)

etc. et de fournir les courbes correspondantes.

Les résultats obtenus lors de l'expérience sans *fading* de *Rayleigh*, présentés sur la figure 6.7, sont parfaitement synchronisés avec le scénario simulé. Le débit IP mesuré lors de l'expérience est bien strictement inférieur aux limites fixées par le scénario. Ainsi pour un taux de transmission radio de 11 Mb/s, le débit IP mesuré ne dépasse jamais les 6.87 Mb/s spécifiés dans le scénario d'émulation. Le fait que les valeurs du débit IP soient légèrement inférieures aux valeurs attendues est dû au conditionneur de trafic Dummynet. En effet, il a tendance fournir un débit légèrement inférieur au débit attendu lorsque ses files d'attente sont surchargées. Comme nous émettons à 8 Mb/s au niveau applicatif, soit 1.13 Mb/s au dessus du débit émulé, les files d'attentes du routeur ont tendance à s'engorger et provoquent des congestions qui influent sur le comportement de Dummynet.

L'aspect crénelé que l'on peut observer sur cette courbe est causé par l'outil TRPR qui ne fournit qu'une valeur moyenne du débit de niveau IP. Cette moyenne a été calculée dans un intervalle de temps très court (100 ms) pour pouvoir comparer les variations entre l'émulation et la simulation si bien que le nombre de datagrammes pris en compte pour déterminer ce débit est relativement faible entraînant ainsi des variations.

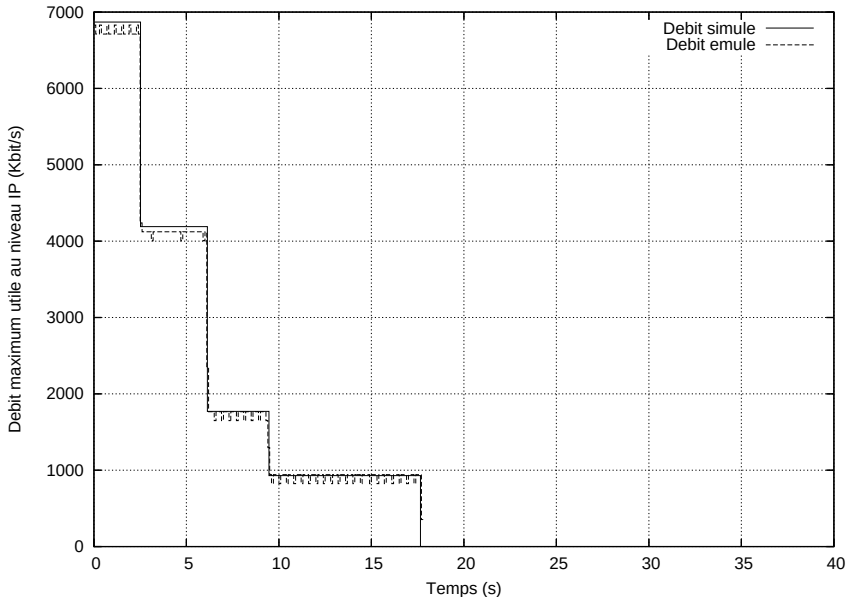
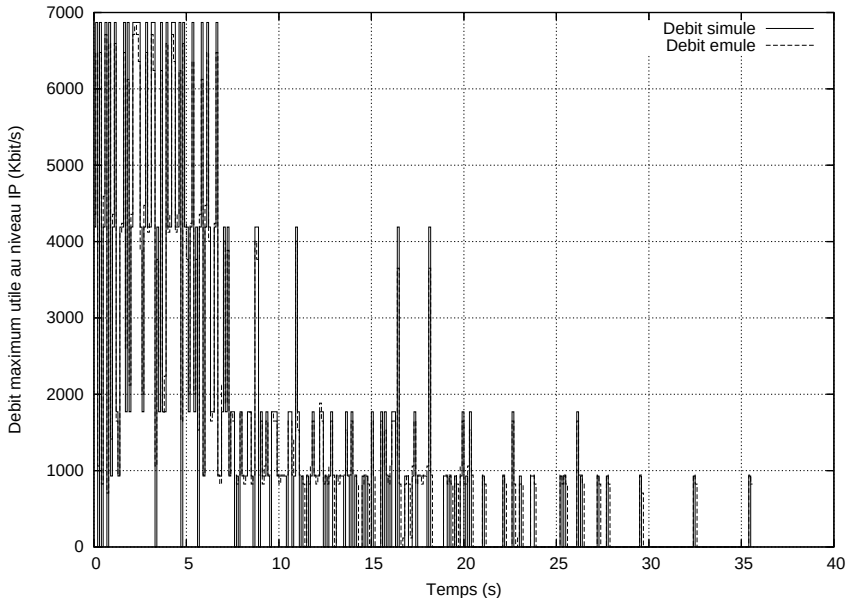
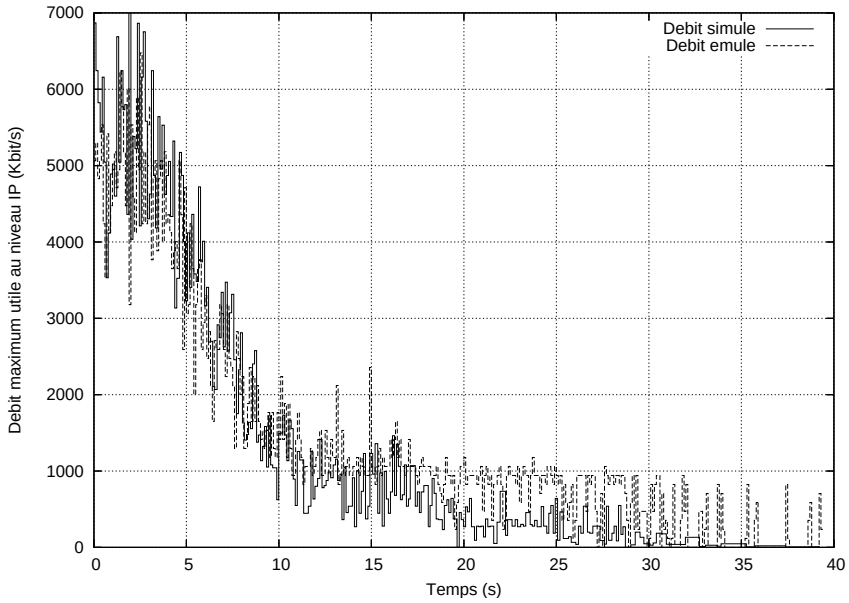


FIG. 6.7: Résultat d'émulation sans fading de Rayleigh

Les résultats obtenus avec l'ajout du *fading* de Rayleigh qui sont présentés sur la figure 6.8 offrent les mêmes conclusions. Comme le montre la figure 6.8 a) qui présente les résultats obtenus avec une granularité de 100 ms, les changements de règles au niveau de l'émulateur s'effectuent bien à l'instant prévu par le scénario d'émulation qui a été simulé. De même ces deux figures mettent en évidence que les contraintes de débit sont bien respectées dans la mesure où il n'y a jamais de débit IP calculé qui soit supérieur au débit IP obtenu lors de la phase de simulation. On peut cependant constater que les différences entre le débit IP attendu et le débit IP mesuré sont plus importantes que dans l'exemple précédent. Une fois encore, ceci peut s'expliquer par le calcul en moyenne de ces débits et par le faible nombre de datagrammes présents au cours d'un intervalle de temps.



(a) pas de temps de 100ms



(b) pas de temps de 10ms (valeurs moyennes)

FIG. 6.8: Résultat d'émulation avec un fading de Rayleigh

Pour tracer ces différentes courbes, l'outil TRPR a été configuré pour déterminer le débit IP sur un intervalle de temps de 100 ms, les variations pour un intervalle de temps de 10 ms étant trop importantes pour être comparées. La figure 6.8 b) présente donc les résultats mesurés avec un pas de temps de 100 ms. Ceux-ci sont comparés avec le débit IP simulé moyen calculé sur le même intervalle de temps par le simulateur SWINE. Les deux courbes sont extrêmement proches l'une de l'autre, en particulier au début de l'expérience, ce qui nous permet de conclure que les variations s'effectuent bien aux instants prévus par le scénario d'émulation. Les différences observées en fin d'expérience peuvent s'expliquer par le calcul de moyenne de l'outil TRPR qui dépend du nombre de paquets franchissant le routeur-émulateur au cours d'un intervalle de temps, contrairement au débit moyen simulé qui dépend uniquement du débit IP instantané. Lorsque ce nombre de paquet est trop faible, on peut mesurer une valeur supérieure à celle attendue. Ainsi, avec des paquets de 1472 octets et un débit émulé de 0.93 Mb/s, il y a moins de 0.79 paquets qui peuvent être reçus au cours d'un intervalle de temps de 10 ms si bien qu'à chaque fois qu'un paquet franchit le routeur émulateur au cours d'un intervalle de temps, la valeur obtenue est sur-évaluée.

6.2.4 Discussion

Cette expérience montre qu'il est possible de contrôler de manière précise un émulateur en se basant sur un scénario pré-calculé lors d'une phase de simulation. Nous avons pu voir à travers ce cas d'utilisation que le conditionneur de trafic réagissait bien aux instants précis où les commandes d'émulation lui sont envoyées, fournissant ainsi un niveau de réalisme satisfaisant. La précision n'est donc pas pénalisée par le passage de la phase de simulation à la phase d'émulation. De même, la dynamicité des conditions lors de la phase d'émulation correspond bien à celle définie dans le scénario.

6.3 Mise en œuvre des observateurs

Dans ce cas d'utilisation, nous allons utiliser des observateurs pour détecter un cas de terminaux cachés et illustrer ainsi le processus de choix qui est mis en œuvre par le gestionnaire d'émulation pour déterminer la règle d'émulation à appliquer.

6.3.1 Description de l'expérience

Dans cet exemple nous allons utiliser les deux mêmes nœuds que dans le cas d'utilisation précédent puis ajouter un second nœud fixe $F2$. Le nœud $F1$ reste au même endroit que lors de l'expérience précédente. Le nœud $M1$, par contre, va parcourir le couloir du DMI en partant du point de passage numéro 1, situé à coté de $F1$, pour s'arrêter au point de passage numéro 3, situé au bout du couloir. Par rapport au cas d'utilisation précédent, il ne va plus entrer dans les différents bureaux. Le nœud $F2$ sera situé entre les points de passage numéro 2 et 3. La figure 6.9 présente les différentes positions des nœuds ainsi que le parcours de $M1$ dans ce cas d'utilisation ($1 \rightarrow 2 \rightarrow 3$).

Dans cette topologie et pour toute la durée de l'expérience, les nœuds $F1$ et $F2$ seront dans l'impossibilité de communiquer et n'auront pas connaissance l'un de l'autre si bien que nous allons nous retrouver dans une situation classique où des cas de terminaux cachés peuvent survenir. Pour avoir un comportement réaliste des conditions de propagation, nous avons utilisé un modèle de *fading* de Rayleigh en plus d'un modèle de *Path Loss Exponent*. Les

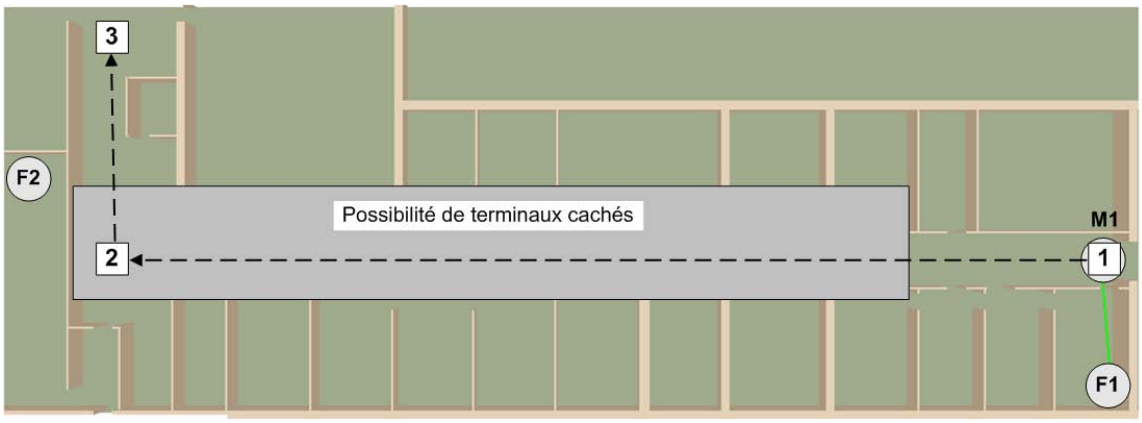


FIG. 6.9: Visualisation graphique d'une expérience avec des terminaux cachés.

paramètres utilisés pour le modèle de *Path Loss Exponent* sont les mêmes que ceux utilisés dans les expériences précédentes à savoir $n = 5.68$ et $PL(d_0) = 19.97\,\text{dB}$.

Pour déterminer ces cas de terminaux cachés et générer les différents choix de règles d'émulation, nous avons utilisé le modèle présenté sur la figure 6.10. Ce modèle est un modèle ON/OFF, c'est-à-dire que nous allons considérer que lorsque deux signaux sont reçus simultanément par un même nœud destination, tous les paquets sont perdus et ce, pour toute la durée du pas de temps de calcul de SWINE (ici 100 ms).

```
- <stage id="hiddenTerminal" class="swine.models.communication.HiddenTerminalModel">
    <plr unit="percent">100</plr>
  </stage>
</models>
```

FIG. 6.10: Extrait du fichier de haut niveau présentant le modèle de gestion des terminaux cachés

6.3.2 La phase de simulation

La phase de simulation génère plusieurs choix de scénario chaque fois que la topologie du réseau peut favoriser l'apparition de cas de terminaux cachés. Le scénario d'émulation généré, dont un extrait est présenté sur la figure 6.11, va intégrer ces différents choix pour informer le gestionnaire d'émulation que différentes règles d'émulation peuvent être appliquées à un instant donné et qu'il doit donc tenir compte des informations de trafic que les observateurs vont lui faire parvenir pour appliquer le choix adéquat.

En se basant sur les informations de topologie et sur les différentes communications possibles entre les nœuds, le simulateur génère deux choix dans le scénario lorsqu'un cas de terminaux cachés peut survenir : soit le cas de terminaux cachés ne se produit pas (un seul nœud émet par exemple) et le premier choix de scénario est appliqué (PLR=0%), soit il survient et c'est alors le second choix (PLR=100%) qui est appliqué. Concrètement, cela se traduit par la perte de tous les paquets émis par les deux nœuds cachés (ici $F1$ et $F2$) à destination d'un même troisième nœud (ici $M1$). La figure 6.12 présente les différences en termes de débit maximum utile de niveau IP pour les communications $F1 \rightarrow M1$ et $F2 \rightarrow M1$ lorsqu'une seule de ces communications a lieu. Le débit IP maximum pour la communication $F1 \rightarrow M1$ diminue

```
- <date id="t8.4" start="8400" unit="ms">
  - <link_update hidden="0" hidden_id="F1-M1-F2">
      <on_link id="F1 to M1" />
      <bandwidth unit="b/s">6870000.0</bandwidth>
      <plr unit="percent">0.0</plr>
      <delay>0</delay>
      <queue_size>8</queue_size>
    </link_update>
  - <link_update hidden="1" hidden_id="F1-M1-F2">
      <on_link id="F1 to M1" />
      <bandwidth unit="b/s">6870000.0</bandwidth>
      <plr unit="percent">100.0</plr>
      <delay>0</delay>
      <queue_size>8</queue_size>
    </link_update>
  + <link_update hidden="0" hidden_id="F1-M1-F2">
  + <link_update hidden="1" hidden_id="F1-M1-F2">
  </date>
- <date id="t8.5" start="8500" unit="ms">
  - <link_update>
      <on_link id="F1 to M1" />
      <bandwidth unit="b/s">6870000.0</bandwidth>
      <plr unit="percent">0</plr>
      <delay>0</delay>
      <queue_size>8</queue_size>
    </link_update>
  + <link_update>
  </date>
```

FIG. 6.11: Extrait du scénario d'émulation pour un cas de terminaux cachés.

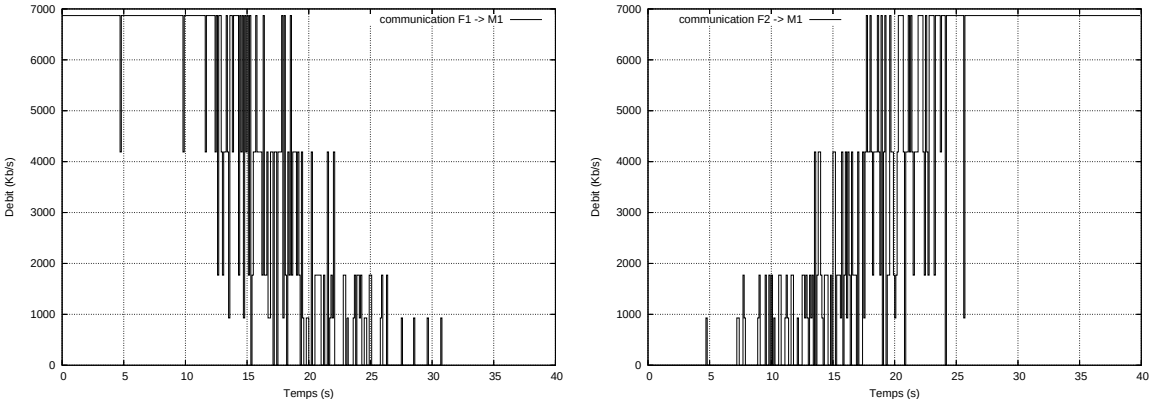
progressivement jusqu'à être rompu à partir de 26.2 s. De même la communication $F2 \rightarrow M1$ n'est possible qu'à partir de 6.6 s et voit son débit IP maximum augmenter progressivement à partir de cette date au fur et à mesure que le nœud $M1$ se rapproche du nœud $F2$.

La figure 6.13 présente l'autre scénario qui prend en compte le fait que les deux communications ont lieu simultanément. On constate qu'entre 15 s et 20 s les communications $F1 \rightarrow M1$ et $F2 \rightarrow M1$ sont simultanées et subissent une forte dégradation.

6.3.3 La phase d'émulation

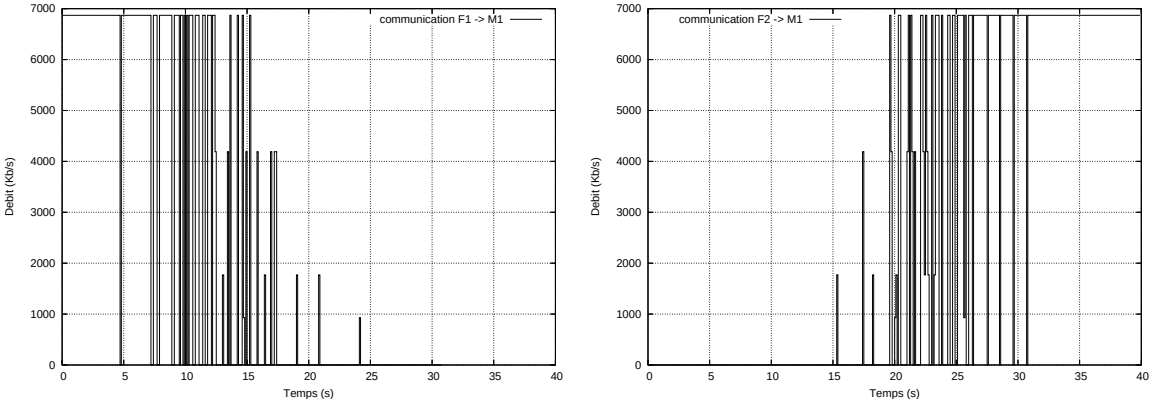
Pour illustrer de manière plus visible les choix dans le scénario, nous avons décidé de ne pas faire durer les communications $F1 \rightarrow M1$ et $F2 \rightarrow M1$ tout au long de l'expérience. La communication $F1 \rightarrow M1$ commencera dès le début de l'expérience puis s'arrêtera après 20 s. La communication $F2 \rightarrow M1$ commencera quant à elle à partir de $t = 5s$ et s'arrêtera à la fin de l'expérience soit $t = 40s$. Ainsi, le choix de scénario sans terminaux cachés sera appliqué pendant les cinq premières secondes puis il y aura un changement de choix au niveau du gestionnaire d'émulation si des paquets circulent sur le réseau. Le choix de scénario sans terminaux cachés devrait égalementêtre appliqué pour la communication $F2 \rightarrow M1$ à partir de $t = 20s$ puisque le signal émis par $F2$ ne sera plus perturbé. Le schéma présenté sur la figure 6.14 illustre le comportement des deux flux et signale l'intervalle de temps au cours duquel des terminaux cachés peuvent survenir et donc les dates auxquelles le gestionnaire d'émulation devra changer de choix de scénario.

Comme nous pouvons le voir sur la figure 6.15, les résultats obtenus sont bien ceux qui étaient attendus. Jusqu'à $t = 5s$ la communication $F1 \rightarrow M1$ n'est jamais perturbée. Par contre, à partir de $t = 5s$, les chutes de débit deviennent beaucoup plus importantes que celles



(a) Communication de F1 vers M1 (b) Communication de F2 vers M1

FIG. 6.12: Scénario sans occurrence de terminaux cachés.



(a) Communication de F1 vers M1 (b) Communication de F2 vers M1

FIG. 6.13: Scénario avec occurrence de terminaux cachés.

prévues dans le cas du scénario sans occurrence de terminaux cachés. On peut donc en déduire que le gestionnaire d'émulation a bien appliqué l'alternative de scénario correspondant à une occurrence de terminaux cachés.

A partir de $t = 20s$, il est possible de constater que les variations de débit vont en diminuant ce qui correspond bien au cas où les communications ne se perturbent plus. Les variations de débit que l'on observe à partir de cette date sur la communication $F2 \rightarrow M1$ correspondent au scénario simulé sans occurence de terminaux cachés. Les observateurs ont donc été en mesure de détecter que seul le nœud $F2$ était en train d'émettre si bien que le gestionnaire d'émulation a pu de nouveau changer de choix de scénario et appliquer les règles correspondant à une communication sans interférence.

6.3.4 Discussion

L'introduction des observateurs permet d'avoir un comportement plus fin de l'émulation rendue et ce sans avoir besoin d'un modèle de trafic au niveau du simulateur. En contrepartie, la taille du scénario généré va augmenter sensiblement ce qui peut conduire à des

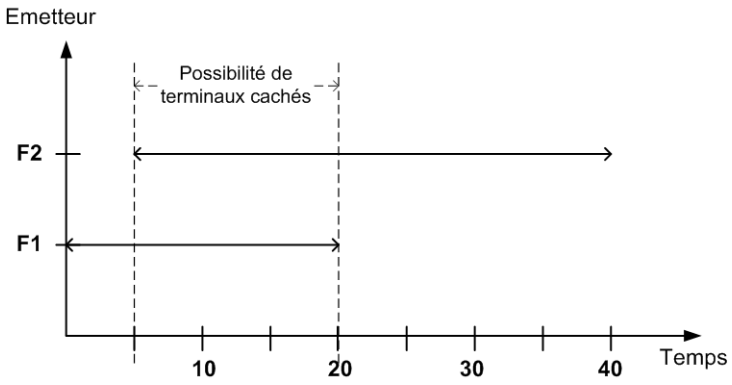


FIG. 6.14: Illustration des communications.

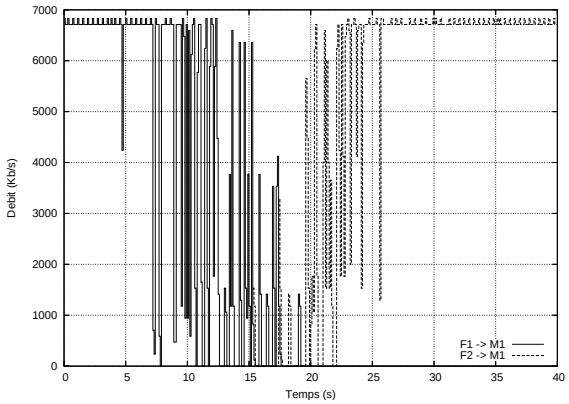


FIG. 6.15: Résultats d'émulation pour un cas de terminaux cachés.

problèmes d'occupation mémoire au niveau du gestionnaire d'émulation lorsque le nombre de cas de terminaux cachés à surveiller ou, dans le cas d'un réseau en mode infrastructure, lorsque le nombre de cellules et de nœuds dans chaque cellule devient trop important. Cependant, ce problème peut être résolu en introduisant des techniques d'optimisation dans la lecture de ce scénario par le gestionnaire d'émulation.

La technique d'observation et de décision soulève un autre problème. En effet, la détection du trafic qui circule sur la plate-forme d'émulation et le changement d'alternative par le gestionnaire d'émulation ne sont pas synchronisés si bien qu'avant que l'alternative ne soit appliquée, des paquets qui seraient perdus dans un réseau réel seront ici bien reçus par la destination. Plus l'intervalle entre deux mises à jours des règles d'émulation est grand, plus le nombre de ces paquets devient important c'est pourquoi il est recommandé de ne pas utiliser d'intervalle trop grand pour garder un bon niveau de réalisme. Dans le prototype actuel, le gestionnaire repose sur une technique de *polling*, c'est-à-dire qu'il interroge les observateurs de manière périodique pour obtenir des informations sur le trafic qui circule. En permettant aux observateurs de prévenir le gestionnaire d'émulation pour que celui-ci réagisse immédiatement, le nombre de paquets qui arrivent au niveau du destinataire sans être contraints devrait être réduit. Cependant, même cette solution ne permet pas de les éliminer complètement ce qui peut être gênant pour une évaluation de protocole par exemple. Pour supprimer complètement

ces cas, il serait nécessaire d'intégrer le processus de décision dans le conditionneur de trafic. En effet seule une prise en compte en temps réel des paquets qui circulent sur le réseau permet d'assurer que tous les paquets seront bien soumis aux pertes simulées.

6.4 Utilisation de l'extension KAUnet

Dans ce cas d'utilisation, nous illustrerons l'utilisation des fichiers de pertes compatibles avec l'extension KAUnet, qui vont permettre d'émuler des conditions réalistes et reproductibles de propagation, et donc de jouer sur le débit utile observé au niveau IP par le nœud récepteur. En effet, celui-ci ne pourra calculer le débit utile de niveau IP qu'en fonction des paquets qui auront effectivement été reçus. Les pertes au niveau du routeur émulateur se traduiront donc par une diminution du débit IP mesuré au niveau du récepteur.

6.4.1 Description de l'expérience

Dans cette expérience, deux nœuds $F1$ et $M1$ vont communiquer à l'intérieur du DMI. Comme dans le premier cas d'utilisation, le nœud $F1$ sera immobile pour toute la durée de l'expérience tandis que le nœud $M1$ suivra un chemin prédéfini lui permettant de se déplacer le long du couloir principal en entrant dans différents bureaux du département (voir figure 6.1 page 124).

Les modèles de propagation seront les mêmes que ceux que nous avions présentés dans la section 6.2 à savoir un modèle de *Path Loss Exponent* configuré avec les paramètres suivants : $n = 5.68$ et $PL(d_0) = 19.97$ et un modèle de *fading* de Rayleigh pour obtenir des variations plus importantes du signal reçu au niveau de $M1$.

La principale différence avec le cas d'utilisation présenté dans la section 6.2 vient du modèle de communication utilisé. En effet, le modèle présenté dans la section 6.2 permet de faire évoluer la débit utile IP en fonction de la puissance reçue mais n'introduit des pertes de paquet que lorsque la puissance reçue au niveau de $M1$ franchit le seuil correspondant au taux de transmission radio minimum (1 Mb/s dans les expériences précédentes). De plus, lorsque ces pertes surviennent nous avons considéré que tous les paquets étaient perdus (PLR=100%). Or, dans un réseau sans fil réel, ces pertes sont en général beaucoup moins brutales.

Dans cette expérience nous avons utilisé un modèle de communication, présenté sur la figure 6.16, qui est chargé de générer des fichiers de pertes compatibles avec l'extension KAUnet présentée dans la section 5.2.2.2. Ce modèle va permettre d'obtenir une meilleure gestion des pertes que le modèle précédent. Les pertes sont obtenues en simulant les conditions de propagation avec une granularité de 1 ms. Dans l'expérience présente cela signifie que des pertes ne seront introduites que si la puissance reçue au niveau du nœud $M1$ passe en dessous de -91 dBm qui correspond à un taux de transmission radio de 2 Mb/s soit un débit IP maximum de 1.77 Mb/s pour des paquets de 1472 octets. Toutes ces pertes sont ensuites stockées dans des fichiers de pertes compatibles avec l'extension KAUnet et qui seront lus en fonction du temps qui passe (*Time-driven mode*). Chaque fichier de pertes couvrira un pas de temps de calcul de SWINE, spécifié dans le fichier de description de haut niveau, si bien que tous les pas de temps, un nouveau fichier de pertes sera chargé.

Comme le modèle utilisé ne présente qu'une seule ligne dans sa table de communication, aucun mécanisme d'adaptation du débit ne sera simulé, si bien que l'évolution du débit mesuré ne pourra être causée que par les pertes introduites par le routeur-émulateur. Ceci correspond

```
- <stage id="KAU_loss_generation" class="swine.models.crosslayers.KAUExtendedPowerBasedIPModel">
  - <power_table>
    - <entry>
        <received_power unit="dBm">-91</received_power>
        <ip_throughput unit="Mbit/s">1.77</ip_throughput>
      </entry>
    </power_table>
  </stage>
```

FIG. 6.16: Extrait du fichier de haut niveau décrivant un modèle de communication basé sur l'extension KAUnet.

au comportement que l'on peut observer lorsque des terminaux sans fil sont configurés pour utiliser un taux de transmission fixe.

6.4.2 Résultats de simulation et d'émulation

Ainsi que le montre la figure 6.17, plus la distance entre les nœuds augmente et plus le pourcentage de pertes observé sur le lien $F1 \rightarrow M1$ augmente. Avec une configuration wifi usuelle, cette augmentation des pertes se traduirait au niveau IP par une diminution du débit car un mécanisme d'acquittement systématique serait utilisé au niveau MAC, ainsi que le mécanisme d'adaptation du débit (*auto-rate fallback*). SWINE a été configurée avec un pas de temps de 100 ms. A chaque pas de temps, un fichier KAUnet est généré avec une granularité de 1 ms. La figure 6.17 présente l'évolution du PLR moyen calculé sur un pas de temps pour chaque fichier KAUnet.

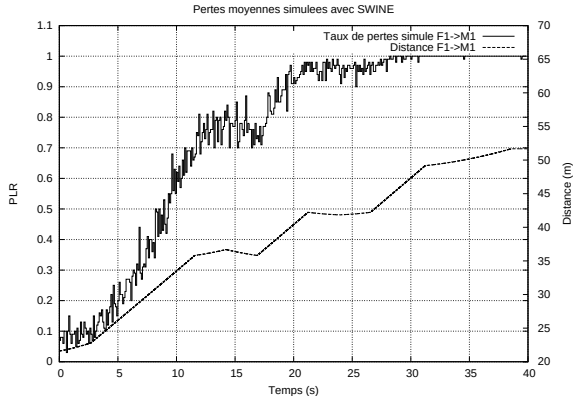


FIG. 6.17: Taux de pertes simulé entre $F1$ et $M1$ pour un pas de temps de 100 ms.

Lors de l'expérience d'émulation proprement dite, nous avons émis un trafic CBR/UDP à 1.5 Mb/s pour éviter les pertes par congestion au niveau du lien. La courbe présentée sur la figure 6.18 montre l'évolution du débit mesuré. Comme prévu celui-ci décroit progressivement jusqu'à se rapprocher de zéro lorsque la distance devient trop importante. En théorie ce débit devrait devenir nul lorsque la distance devient importante mais on peut constater que le débit mesuré à partir de $t = 30$ s est proche de 100 Kb/s au lieu du zéro attendu. Ceci signifie donc que des paquets sont reçus au niveau du récepteur alors qu'ils devraient être perdus.

Pour diagnostiquer ce problème, le PLR réel a lui aussi été mesuré (cf. figure 6.19). On peut ainsi remarquer que le taux de pertes moyen mesuré à l'aide de l'outil TRPR est légérement

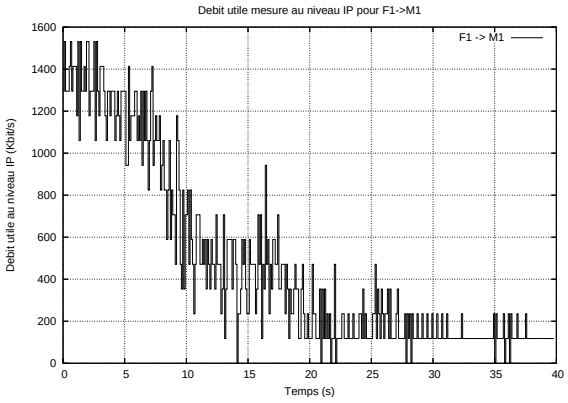


FIG. 6.18: Débit utile mesuré au niveau du nœud $M1$.

inférieur à celui qui était attendu, notamment lorsque le taux de pertes doit être à 1. Ce PLR légèrement inférieur à 1 confirme que des paquets ont été reçus du coté du nœud récepteur. Comme le PLR mesuré est très proche de la valeur attendue le reste du temps, nous pensons que ceci est dû au changement de fichier de pertes dans le conditionneur de trafic. Au moment de l'application de la règle d'émulation, le conditionneur de trafic doit charger le nouveau fichier de pertes en mémoire ce qui introduit un petit délai entre l'application de la règle et le début de lecture du fichier si bien que certains paquets doivent pouvoir passer à travers le routeur-émulateur.

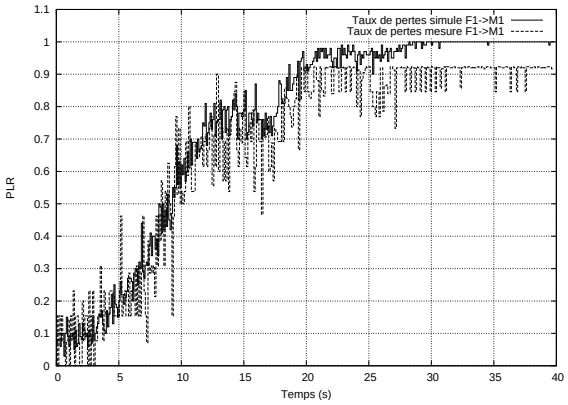


FIG. 6.19: Taux de pertes mesuré lors de l'émulation, entre $F1$ et $M1$, pour un pas de temps de 100 ms.

Pour vérifier cette hypothèse, nous avons simulé de nouveau la même expérience en utilisant cette fois-ci un pas de temps de 500 ms. Lors de la phase d'émulation, nous pouvons observer sur la figure 6.20 que nous obtenons de nouveau un PLR légèrement inférieur à 1 en fin d'expérience, par contre, celui-ci est plus proche de 1 que lorsque le pas de temps était de 100 ms, ce qui laisse penser qu'il y a eu moins de paquets indûment reçus au niveau du nœud récepteur. Ceci semble donc bien confirmer que le passage des paquets survient lors la lecture du nouveau fichier de pertes à appliquer. Une modification de la manière dont le fichiers sont lus est actuellement à l'étude à l'Université de Karlstad pour y remédier.

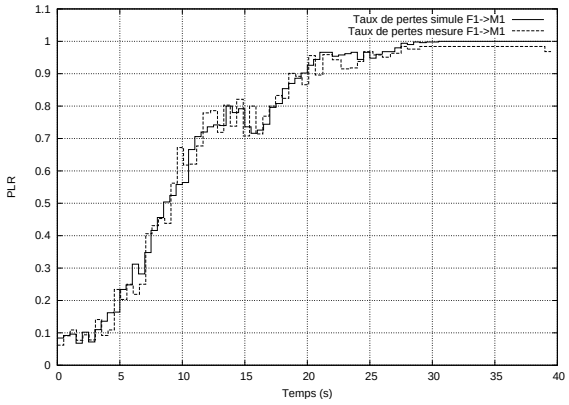


FIG. 6.20: Taux de pertes mesuré lors de l'émulation, entre $F1$ et $M1$, pour un pas de temps de 500 ms.

6.4.3 Discussion

La génération et l'utilisation des fichiers de pertes permettent d'avoir une évolution du débit utile IP au cours du temps en jouant cette fois-ci sur les erreurs pouvant survenir sur une transmission. Nous avons ainsi pu générer des pertes de telle manière que le taux de perte augmente en fonction de la distance séparant l'émetteur du récepteur ce qui se traduit par une diminution du débit utile IP du côté du récepteur. L'utilisation de fichiers de pertes assure que ces pertes surviendront bien aux mêmes instants d'une émulation à l'autre. Grâce à cette possibilité de repoduire fidèlement une expérience plusieurs fois de suite, il est possible d'évaluer avec W-NINE des protocoles de transports dans un environnement sans fil n'utilisant pas de mécanisme d'adaptation du débit ni de mécanismes d'acquittements systématiques et de retransmissions au niveau MAC. Le principe des fichiers peut être étendu aux autres paramètres de QdS comme par exemple les délais ce qui permettra notamment de mieux émuler le mécanisme d'adaptation du débit en reproduisant les délais causés par les retransmissions, avant de diminuer le débit maximum IP.

6.5 Conclusion

Nous avons pu voir à travers ces différents cas d'utilisation que le niveau de réalisme obtenu lors de la phase de simulation et bien conservé lors du passage à la phase d'émulation ce qui valide l'utilisation de scénarios et l'utilisation d'une phase de simulation préalable. Ainsi, les commandes sont bien traitées par le conditionneur de trafic aux instants spécifiés dans le scénario d'émulation et ceci sans introduction de délais superflus. L'utilisation d'observateurs de réseaux permet également de prendre en compte les événements liés au trafic réel. Cependant, nous avons constaté que certains paquets atteignaient leur destination au lieu d'être perdu. La solution actuelle qui ne prend pas de décision en temps réel ne permet pas de résoudre complètement ce problème. Il s'écoule un temps de latence égal à la durée d'un intervalle de temps entre deux mises à jours avant que les nouvelles règles d'émulation ne soient appliquées. Cependant, cette solution reste tout de même intéressante pour l'évaluation d'applications réparties. De plus, lorsque l'intervalle de temps entre deux mises à jours est suffisamment faible, le comportement moyen observé se rapproche de ce que l'on peut retrouver dans la réalité

si bien que cette solution peut également être retenue pour évaluer des protocoles de niveau transport. De même, l'utilisation de pertes reproductibles permet d'évaluer avec précision ce type de protocoles ou d'applications. Les futures extensions de KAUnet (fichier de délais...) permettront de mieux émuler d'autres comportements de niveau MAC comme l'adaptation du débit.

7 Conclusion

7.1 Bilan

Dans cette thèse nous avons présenté W-NINE, une nouvelle solution d'émulation de niveau IP pour les réseaux sans fil.

Le premier chapitre présente l'environnement sans fil en pointant les différences qui existent avec les réseaux filaires. En présentant la norme IEEE 802.11, ce chapitre a permis de mettre en évidence les problèmes qui existent tant au niveau de l'accès au canal qu'au niveau d'une communication (terminaux cachés, terminaux exposés, communications dans une cellule en mode Infrastructure...). A travers ce chapitre, il a donc été possible de mettre en évidence les nombreuses différences qui existent entre les réseaux filaires et les réseaux sans fil et qui rendent difficile l'émulation de ces derniers.

Le deuxième chapitre présente les différentes solutions d'émulation sans fil existantes et met en évidence leurs principales limitations. Pour les solutions d'émulation de niveau IP notamment, on constate que les plates-formes proposant un bon niveau de réalisme de l'émulation rendue ne peuvent assurer un traitement du trafic en temps réel lorsque le nombre de liens ou la quantité de trafic deviennent trop importants. Pour permettre de respecter ces contraintes de temps, d'autres solutions ont fait un sacrifice au niveau du réalisme des modèles utilisés pour diminuer la quantité de calcul à effectuer en temps réel. D'autres encore ont proposé de simuler la partie mobilité qui existe dans les réseaux sans fil avant la phase d'émulation. Dans W-NINE, nous avons généralisé cette approche en l'étendant au pré-calcul de la propagation et au pré-calcul de certains aspects de communication liés au trafic.

Le troisième chapitre présente quelques modèles de la littérature permettant de représenter correctement un réseau sans fil. Ces modèles se classent en trois grandes catégories : les modèles de mobilité qui permettent de déterminer les positions des nœuds au cours de l'expérience, les modèles de propagation qui déterminent les conditions de transmission du signal radio à l'intérieur du réseau et les modèles de communications chargés de déterminer les conditions de qualité de service du réseau au niveau IP en terme de débits, de délais et de pertes. Ce chapitre propose également des solutions originales quant à la manière de prendre en compte le trafic lors de la phase de simulation sans en donner de modèle. Finalement, nous avons introduit l'architecture générale de la plate-forme W-NINE qui repose sur l'utilisation de deux phases successives : une phase de simulation se déroulant hors ligne et une phase d'émulation se déroulant en temps réel. Pour assurer l'interface entre l'utilisateur et la phase de simulation d'une part et entre la phase de simulation et la phase d'émulation d'autre part, nous avons défini des fichiers au format XML ainsi que leurs schémas au format RELAX NG compact.

Le quatrième chapitre s'attache aux détails de conception de W-NINE. La phase de simulation utilise SWINE, un simulateur de réseau développé pour produire des scénarios d'émulation qui seront ensuite utilisés lors de la phase d'émulation. SWINE est un simulateur à événements discrets conduit par le temps. En se basant sur un fichier de haut niveau qui décrit les nœuds composant le réseau à émuler, les obstacles présents dans l'environnement émulé et les modèles de mobilité, de propagation et de communication à utiliser, SWINE va déterminer l'évolution

dans le temps des conditions de niveau IP (débits, pertes et délais) caractérisant ce réseau. Pour permettre à l'utilisateur d'ajouter simplement des modèles, l'architecture de SWINE a été conçue comme une architecture ouverte où l'héritage joue un rôle clef. Ainsi, il est possible d'ajouter un modèle simplement en héritant d'un des modèles abstraits que nous avons défini et en mettant à jours le schémas du fichier servant d'interface entre l'utilisateur et le simulateur. Ces modèles sont au nombre de quatre (*MobilityModel*, *PropagationModel*, *CommunicationModel*, *CrossStagesModel*) et permettent de répondre à la plupart des besoins de l'utilisateur.

Une autre particularité de SWINE par rapport à des simulateurs de réseau comme NS-2 est qu'il n'utilise pas de modèle de trafic pour déterminer les conditions du réseau à émuler. En effet, l'émulation ne nécessite pas de fournir un tel modèle. Ceci pose cependant certains problèmes pour prendre en compte les effets directement liés au trafic qui circule sur le réseau à un instant donné comme, par exemple, les terminaux cachés ou encore les collisions au niveau de l'accès au canal. Pour résoudre le problème de l'accès au canal, nous utilisons une valeur moyenne de *backoff*. Pour pouvoir prendre en compte les événements liés au trafic comme les terminaux cachés, une nouvelle technique a été proposée. Elle se base sur l'observation du trafic en temps réel et sur l'utilisation de boucles-retour au niveau de la phase d'émulation.

Au cours de la phase d'émulation, le gestionnaire d'émulation qui est chargé d'envoyer périodiquement des commandes de mise à jour au conditionneur de trafic, s'appuie sur des observateurs de trafic pour déterminer la règle d'émulation qu'il doit envoyer. En effet, lors de la phase de simulation, plusieurs choix sont produits dans le scénario pour permettre de prendre en compte le trafic. Le rôle des observateurs est donc d'informer le gestionnaire d'émulation du choix qu'il va devoir appliquer en fonction du trafic qui circule sur le réseau d'expérimentation.

Pour assurer la bonne reproduction de conditions d'une expérience à l'autre lorsque le même scénario d'émulation est utilisé, l'extension KAUnet basée sur Dummynet a été intégrée dans W-NINE. Cette extension permet une meilleure gestion de pertes que celle offerte par Dummynet grâce à l'utilisation de fichiers de pertes. Ces fichiers permettent un placement précis des pertes dans un flux ou dans le temps. Le positionnement des pertes dans le temps est le mode de fonctionnement privilégié dans W-NINE car il ne nécessite pas de modèle de trafic. En outre, ces fichiers de pertes permettent de reproduire les effets de la propagation au niveau IP.

Le chapitre cinq illustre et discute les différents résultats obtenus avec W-NINE. Le premier cas d'utilisation permet de montrer la cohérence des résultats obtenus entre les deux phases et de conclure que l'utilisation d'une phase de simulation reposant sur des modèles réalistes mais coûteux en temps de calcul est intéressante pour les aspects indépendants du trafic. Pour la prise en compte du trafic nous pouvons voir sur le second cas d'utilisation que les observateurs permettent d'obtenir des résultats proches de la réalité. Cependant, le fait d'observer puis de réagir ne permet pas de reproduire ces effets de manière immédiate. La solution actuelle réagit avec un temps de latence équivalent à un intervalle de temps entre deux mises à jour.

De manière générale, le trafic doit être pris en compte en temps réel pour émuler de manière précise l'accès au canal et les événements liés au trafic. L'utilisation d'un simulateur peut cependant permettre de calculer à l'avance les effets nécessitant beaucoup de calcul mais la détection et la décision doivent être réalisées en temps réel pour éviter les effets de bords que nous obtenons actuellement.

7.2 Perspectives

Dans un premier temps, nous souhaitons développer un support des trafics autre que CBR. Lors de l'évaluation d'une application ou d'un protocole utilisant du trafic VBR, il n'est pas possible de déterminer précisément le débit au niveau IP pendant la phase de simulation, sans utiliser de modèle de trafic. Par contre, il est possible de déterminer le délai inter-paquets qui permet d'obtenir ce débit. Les scénarios d'émulation et le conditionneur de trafic utilisés dans W-NINE sont en cours de modification pour pouvoir supporter ce type de délais et permettre une émulation réaliste de l'accès au canal indépendamment du type de trafic utilisé.

Dans le cadre du réseau d'excellence européen NEWCOM, le conditionneur de trafic KAUnet est en cours d'extension par nos partenaire de l'Université de Karlstad pour mieux supporter les réseaux sans fil. La gestion des délais va être améliorée pour supporter les délais inter-paquets que nous venons d'évoquer d'une part, et des fichiers de délais ou de débits semblables aux fichiers de pertes utilisés actuellement d'autre part. Ces fichiers de délais vont permettre de modéliser plus fidèlement les délais de retransmission qui sont observés lors de l'utilisation du mécanisme d'adaptation du débit par exemple. SWINE est actuellement en cours d'extension pour supporter ces nouveaux formats de fichiers et de nouveaux modèles permettant de déterminer les délais de retransmission sont à l'étude. De plus, KAUnet ne permet pas à l'heure actuelle de tests réalistes utilisant plusieurs flux. Dans le cas de TCP par exemple, il n'est pas possible d'observer un comportement réaliste lorsque des flux sont émis par des nœuds différents vers une même destination (sans terminaux cachés). Les deux flux n'entreront pas en concurrence dans la solution actuelle. Dans le cadre de W-NINE, une solution basée sur des observateurs est envisageable pour résoudre ce problème mais là encore des effets de bords devraient survenir si le processus de décision n'est pas effectué en temps réel.

L'émulation du routage dans les réseaux Ad-Hoc est également à l'étude. En effet, bien qu'il soit possible de déterminer les routes à l'avance, les délais introduits par la recherche de route sont impossibles à déterminer hors ligne sans utiliser de modèle de trafic. Le support de ces délais peut cependant être réalisé en temps réel à l'aide d'une couche virtuelle située au niveau du routeur-émulateur et chargée de déterminer lors de l'arrivée de chaque paquet, les délais dûs à l'algorithme de routage. De même, si le protocole de routage utilise des messages de contrôle (maintenance, rupture de route...), il est nécessaire d'émuler les effets de ce trafic sur le débit de niveau IP. Si ces messages de contrôle sont de taille fixe, ces effets peuvent être pré-calculés lors de la phase de simulation. Une fois de plus, l'utilisation d'une couche virtuelle au niveau du routeur émulateur permet d'appliquer en temps réel ces effets en fonction du trafic qui circule sur le réseau d'expérimentation. Cette solution peut donc être vue comme une version allégée de celle retenue dans MobiNet où l'établissement et la maintenance des routes sont effectuées en temps réel. Cependant, une solution d'émulation décentralisée de niveau 2 semble mieux adaptée pour reproduire de manière réaliste le comportement d'un réseau sans fil en mode Ad-Hoc, une solution centralisée ne pouvant qu'approcher ce comportement.

Malgré ces quelques bémols, ce travail a montré que l'utilisation d'une phase de simulation préalable à l'émulation proprement dite permet d'utiliser une solution centralisée pour faire de l'émulation de réseaux sans fil avec un bon niveau de réalisme. Nous préconisons donc la généralisation de cette technique de pré-calcul des conditions pour améliorer les performances des émulateurs existants en diminuant la charge de calcul à effectuer en temps réel tout en préservant leurs particularités. Par exemple, NSE peut être modifié de manière à pré-calculer à l'avance la mobilité et les conditions de propagation entre les nœuds. NSE n'aura donc plus

que les accès au canal et les effets liés au trafic à prendre en compte en temps réel.

L'utilisation de l'extension KAUnet et des observateurs dans W-NINE permet d'obtenir un comportement proche de la réalité, le niveau de réalisme dépendant du réalisme des modèles utilisés lors de la phase de simulation. W-NINE répond au principal besoin que nous avions énoncé : la possibilité d'utiliser des modèles précis pour obtenir un comportement dynamique réaliste et reproductible d'une expérience à l'autre. La solution obtenue permet d'évaluer des applications réparties ou des protocoles de haut niveau, en particulier ceux de la couche transport, dans le cadre des réseaux sans fil.

8 Liste des publications

Conférences internationales avec comité de lecture

1. E. Conchon, T. Pérennou, M. Diaz, ***A Feedback Based Solution to Emulate Hidden Terminals in Wireless Networks.*** Dans les actes de IEEE Symposium on Future Wireless Systems of the International Conference on Software, Telecommunications and Computer Networks (SoftCom 2005) en 2005
2. E. Conchon, J. Garcia, ***Increasing the Determinism of Network Emulation to Evaluate Communication Protocols.*** Dans les actes de Student Workshop at CONEXT'05 en 2005
3. T. Pérennou, E. Conchon, L. Dairaine, M. Diaz, ***Two-Stage Wireless Network Emulation.*** Dans les actes de IFIP World Computer Congress - Workshop on Challenges of Mobility (WCM 2004) en 2004

Conférences nationales avec comité de lecture

1. E. Conchon, T. Pérennou, L. Dairaine, M. Diaz, ***W-NINE : Une plate-forme d'émulation de réseaux sans fil.*** Dans les actes des 6èmes Journées Doctorales Informatique et Réseau (JDIR'04) en 2004

Conférences nationales sans comité de lecture

1. E. Conchon, ***Emulation de Réseaux sans fils.*** Dans les actes du Colloque de l'EDIT en 2004

Rapports de recherche

1. E. Conchon, T. Pérennou, J. Garcia, ***Improved IP-level Emulation for Mobile and Wireless System.*** Rapport LAAS, Septembre 2006
2. E. Conchon, T. Pérennou, J. Garcia, ***Integrating KAUnet and SWINE.*** Dans les actes de NEWCOM Department 6 Fourth Technical Workshop en Septembre 2006
3. E. Conchon, T. Pérennou, J. Garcia, ***Wireless Network Emulation.*** Dans les actes de NEWCOM Department 6 Third Technical Workshop en Février 2005
4. E. Conchon, T. Pérennou, J. Garcia, ***Wireless Network Emulation***. Dans les actes de NEWCOM Department 6 Second Technical Workshop en Septembre 2004, rapport numéro 507325

Bibliographie

[1] *P 802.11 Draft Standard for Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specification.* New York, 1997.

[2] *Standard for Information technology–Telecommunications and information exchange between systems– Local and metropolitan area networks–Specific requirements Part 15.1 : Wireless medium access control (MAC) and physical layer (PHY) specifications for wireless personal area networks (WPANs).* New York, 2002.

[3] *IEEE Standard 802.16-2004 : IEEE Standard for Local and Metropolitan Area Networks - Part 16 : Air Interface for Fixed Broadband Wireless Access Systems.* New York, 2004.

[4] M. Allman and S. Ostermann. ONE : The Ohio Network Emulator. Technical Report TR-19972, Computer Science, Ohio University, 1997.

[5] L. Apvrille, L. Dairaine, P. Senac, and M. Diaz. Dynamic Reconfiguration Architecture of Satellite Network Software Services. In *Proceedings of AIAA 2001*, 2001.

[6] S. Basagni, . Chlamtac, V. R. Syrotiuk, and B. A. Woodward. A Distance Routing Effect Algorithm for Mobility (DREAM). In *Proceedings of the 4th Annual ACM/IEEE International Conference on Mobile Computing and Networking (MOBICOM-98)*, pages 76–84, New York, October 25–30 1998. ACM Press.

[7] N. Boulicault, G. Chelius, and E. Fleury. Ana4 : a 2.5 Framework for Deploying Real Multi-hop Ad hoc and Mesh Networks. *Ad Hoc & Sensor Wireless Networks : an International Journal (AHSWN)*, to be published, 2005.

[8] L. Breslau, D. Estrin, K. Fall, S. Floyd, J. Heidemann, A. Helmy, P. Huang, S. McCanne, K. Varadhan, Y. Xu, and H. Yu. Advances in Network Simulation. *IEEE Computer*, 33(5), May 2000.

[9] T. Camp, J. Boleng, and V. Davies. A Survey of Mobility Models for Ad Hoc Network Research. *Wireless Communications and Mobile Computing*, 2(5) :483–502, 2002.

[10] M. Carson and D. Santay. NIST Net : A Linux-based Network Emulation Tool. *ACM SIGCOMM Computer Commununications Review*, 33(3) :111–126, 2003.

[11] G. Chelius. *Architecture et Communications Dans les Réseaux Spontanés Sans Fil.* PhD thesis, INSA de Lyon, INRIA Rhône Alpes, France, 2004.

[12] C. Chiang. Wireless Network Multicasting. In *PhD thesis, Univertity of California, Colorado School of Mines*, 2000.

[13] Cisco. Data sheet : Cisco aironet 1240ag series a/b/g access point.

[14] J. Clark, J. Cowan, and M. Murata. RELAX NG Compact Syntax Tutorial, 2003.

[15] T. Clausen, , and P. Jacquet. Optimized Link State Routing Protocol (OLSR). Internet Request for Comments RFC 3626, 2003.

[16] IEEE Computer Society LAN MAN Standards Committee. *IEEE 802.11 : Wireless LAN Medium Access Control and Physical Layer Specifications*, 1999.

[17] E. Conchon. Système d'émulation de niveau IP : Principes, modélisation, mise en oeuvre et expériences. In *Rapport de DEA*, 2002.

[18] E. Conchon, J. Garcia, and T. Perennou. Improved IP-level Emulation for Mobile and Wireless Systems. Technical report, 2006.

[19] S. Corson and J. Macker. Mobile Ad hoc Networking (MANET) : Routing Protocol Performance Issues and Evaluation Considerations. Internet Request for Comments RFC 2501, Internet Engineering Task Force (IETF), january 1999.

[20] D-Link. Di-624+ : Spécifications techniques.

[21] V. Davies. Evaluating Mobility Models Within an Ad Hoc Network. Master's thesis, Colorado School of Mines, Dept. of Mathematical and Computer Sciences, 2000.

[22] European Telecommunication Standard ETS. *ETSI TC-RES. Radio Equipment and Systems(RES) ; HIgh PErformance Radio Local Area Network(HIPERLAN) Type 1 ; Functional specification. European Telecommunication Standard ETS 300 652*, October 1996.

[23] European Telecommunication Standard ETS. *ETSI TS 101 475 : Broadband Radio Access Networks (BRAN) ; HIPERLAN Type 2 ; Physical (PHY) layer*, 1999.

[24] K. Fall. Network Emulation in the VINT/NS Simulator. In *Proceedings of the fourth IEEE Symposium on Computers and Communications*, 1999.

[25] J. Flynn, H. Tewari, and D. O'Mahony. JEmu : A Real Time Emulation System for Mobile Ad Hoc Networks. In *Proceedings of the First Joint IEI/IEE Symposium on Telecommunications Systems Research*, November 2001.

[26] S. Ganu, H. Kremo, R. E. Howard, and I. Seskar. Addressing Repeatability in Wireless Experiments using ORBIT Testbed. In *Proceedings of TRIDENTCOM 2005*, pages 153–160. IEEE Computer Society, 2005.

[27] J. Garcia, S. Alfredson, and A. Brunstrom. The Impact of Loss Generation on Emulation-based Protocol Evaluation. In *Proceedings of the IASTED International Conference on Parallel and Distributed Computing and Networks (PDCN 2006)*, 2006.

[28] Z. J. Haas, M. R. Pearlman, and P. Samar. The Zone Routing Protocol (ZRP) for Ad Hoc Networks. Internet-draft, IETF MANET Working Group, July 2002. Expiration : January, 2003.

[29] C. Hedrick. Routing Information Protocol. Internet Request for Comments RFC 1058, 1988.

[30] S. Hemminger. `http://developer.osdl.org/shemminger/netem/`.

[31] D. Herrscher, A. Leonardi, and K. Rothermel. Modeling Computer Networks for Emulation. In *Proceedings of PDPTA'02*, pages 1725–1731, June 2002.

[32] D. Herrscher, S. Maier, J. Tian, and K. Rothermel. A Novel Approach to Evaluating Implementations of Location-Based Software. In *Proceedings of the 2004 International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS 2004)*, pages 484–490. SCS, July 2004.

[33] D. Herrscher and K. Rothermel. A Dynamic Network Scenario Emulation Tool. In *Proceedings of ICCCN 2002*, pages 262–267, October 2002.

[34] M. Heusse, F. Rousseau, G. Berger-Sabbatel, and A. Duda. Performance Anomaly in 802.11b. In *Proceedings of IEEE INFOCOM 2003*, 2003.

[35] X. Hong, M. Gerla, G. Pei, and C. Chiang. A Group Mobility Model for Ad Hoc Wireless Networks. In *Proceedings of the ACM/IEEE International Workshop on Modeling and Simulation of Wireless and Mobile Systems (MSWiM)*, 1999.

[36] IEEE. *IEEE Std 802.11a-1999, Part 11 : Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications : High-speed Physical Layer in the 5 GHz Band*, 1999. statut : « Supplement to IEEE Std 802.11-1999, Adopted by ISO/IEC and redesignated as ISO/IEC 8802-11 :1999/Amd 1 :2000(E) », http ://a957.g.akamai.net/7/957/3680/v0001/standards.ieee.org/reading/ieee/std/lanman/802.11a-1999.pdf.

[37] IEEE. *IEEE Std 802.11b-1999, Part 11 : Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications : Higher-Speed Physical Layer Extension in the 2.4 GHz Band*, 1999. statut : « Supplement to IEEE Std 802.11-1999 », http ://a957.g.akamai.net/7/957/3680/v0001/standards.ieee.org/reading/ieee/std/lanman/802.11b-1999.pdf.

[38] D.B. Ingham and G.D. Parrington. Delayline : A Wide-Area Network Emulation Tool. *USENIX, Computing Systems*, 7(3) :313–332, 1994.

[39] *Basic Reference Model for Open System Interconnection.* Int. Standards Org., 1984. ISO 7498.

[40] A. Iwata, C. C. Chiang, G. Pei, M. Gerla, and T. W. Chen. Scalable Routing Strategies for Ad hoc Wireless Networks. *IEEE Journal on Selected Areas in Communications*, 17(8) :1369–1379, August 1999.

[41] A. Jardosh, E. M. Belding-Royer, K. C. Almeroth, and S. Suri. Towards Realistic Mobility Models for Mobile Ad hoc Networks. In *Proceedings of MobiCom'03*, September 2003.

[42] D. Johnson, T. Stack, R. Fish, D. M. Flickinger, L. Stoller, R. Ricci, and J. Lepreau. Mobile Emulab : A Robotic Wireless and Sensor Network Testbed. In *Proceedings of the 25th Conference on Computer Communications (IEEE INFOCOM 2006)*, 2006.

[43] D.B. Johnson and D.A. Maltz. Dynamic source routing in ad-hoc wireless networks. *Kluwer Academic Publishers*, pages 153–181, 1996.

[44] J. Jun, P. Peddachagari, and M. Sichitiu. Theoretical Maximum Throughput of IEEE 802.11 and its Applications. In *Proceedings of the 2nd IEEE International Symposium on Network Computing and Applications (NCA-03)*, 2003.

[45] Q. Ke, D. Maltz, and D.B. Johnson. Emulation of Multi-Hop Wireless Ad Hoc Networks. In *Proceedings of the 7th International Workshop on Mobile Multimedia Communications (MoMuC 2000)*, October 2000.

[46] M. Kojo, A. Gurtov, J. Mannner, P. Sarolahti, T. Alanko, and K. Raatikainen. Seawind : a Wireless Network Emulator. In *Proceedings of the 11th GI/ITG Conference on Measuring, Modelling and Evaluation of Computer and Communication Systems (MMB 2001)*, 2001.

[47] S. Kurkowski, T. Camp, and M. Colagrosso. MANET Simulation Studies : The Incredibles. *SIGMOBILE Mob. Comput. Commun. Rev.*, 9(4) :50–61, 2005.

[48] J. Lacan and T. Pérennou. Evaluation of Error Control Mechanisms for 802.11b Multicast Transmissions. In *Proceedings of the Second International Workshop On Wireless Network Measurement (WINMee 2006)*, 2006.

[49] L. Lancerica, L. Dairaine, F. de Belleville, H. Thalmensy, and C. Fraboul. MITV, A Solution for Interactive TV Based on IP Multicast over Satellite. In *Proceedings of the IEEE International Conference on Multimedia and Expo (ICME)*, June 2004.

[50] B. Liang and Z. J. Haas. Predictive Distance-Based Mobility Management for PCS Networks. In *Proceedings of INFOCOM'99*, pages 1377–1384, 1999.

[51] P. Mahadevan, A. Rodriguez, D. Becker, and A. Vahdat. MobiNet : A Scalable Emulation Infrastructure for Ad-Hoc and Wireless. Technical report, June 2004.

[52] P. Mahadevan, A. Rodriguez, D. Becker, and A. Vahdat. MobiNet : A Scalable Emulation Infrastructure for Ad-Hoc and Wireless Networks. In *WiTMeMo '05 : Papers presented at the 2005 workshop on Wireless traffic measurements and modeling*, pages 7–12, Berkeley, CA, USA, 2005. USENIX Association.

[53] D. Mahrenholz and S. Ivanov. Real-Time Network Emulation with NS-2. In *DS-RT*, pages 29–36. IEEE Computer Society, 2004.

[54] D. Mahrenholz and S. Ivanov. Adjusting the NS-2 Emulation Mode to a Live Network. In Paul Müller, Reinhard Gotzhein, and Jens B. Schmitt, editors, *Proceedings of KiVS 2005*, Informatik Aktuell, pages 205–217. Springer, 2005.

[55] S. Mann. Smart Clothing : The Shift to Wearable Computing. *Commun. ACM*, 39(8) :23–24, 1996.

[56] M. Matthes, H. Biehl, M. Lauer, and O. Drobnik. MASSIVE : An Emulation Environment for Mobile Ad-Hoc Networks. In *Proceedings of WONS 2005*, pages 54–59. IEEE Computer Society, 2005.

[57] P. Mühlethaler. *802.11 et les réseaux sans fil.* Eyrolles, 2002.

[58] J. Moy. OSPF Version 2. Internet Request for Comments 2328, 1998.

[59] M. Musolesi, S. Hailes, and C. Mascolo. An Ad Hoc Mobility Model Founded On Social Network Theory. In *Proceedings of the 7th International Symposium on Modeling Analysis and Simulation of Wireless and Mobile Systems (MSWiM 2004)*, pages 20–24. ACM, 2004.

[60] B.D. Noble, M. Satyanarayanan, G.T. Nguyen, and R.H. Katz. Trace-Based Mobile Network Emulation. In *Proceedings of ACM SIGCOMM'97*, September 1997.

[61] OPNET. OPNET Technologies. `http://www.opnet.com`, July 2002.

[62] K. Pawlikowski, H.D.J. Jeong, and J.S.R. Lee. On Credibility of Simulation Studies of Telecommunication Networks. *IEEE Communications Magazine*, 40 :132–139, 2002.

[63] C. E. Perkins, E. M. Belding-Royer, and S. Das. Ad Hoc On Demand Distance Vector (AODV) Routing. Internet Request for Comments RFC 3561, 2002.

[64] C. E. Perkins and P. Bhagwat. Highly Dynamic Destination-Sequenced Distance-Vector Routing (DSDV) for mobile computers. In *SIGCOMM*, pages 234–244, 1994.

[65] M. Puzar and T. Plagemann. NEMAN : A Network Emulator For Mobile Ad-Hoc Networks. In *Proceedings of the 8th International Conference on Telecommunications (ConTEL 2005)*, pages 155–161, 2005.

[66] QualNet. Scalable Network Technologies. `http://www.scalable-networks.com/`, 2001.

[67] K. Ramachandran, S. Kaul, S. Mathur, M. Gruteser, and I. Seskar. Towards Large-Scale Mobile Network Emulation Through Spatial Switching on a Wireless Grid. In *Proceeding of the 2005 ACM SIGCOMM workshop on Experimental approaches to wireless network design and analysis (E-WIND'05)*, pages 46–51, New York, NY, USA, 2005. ACM Press.

[68] T. S. Rappaport. *Wireless Communications Principles and Practice*. Prentice Hall, Upper Saddle River, 1996.

[69] D. Raychaudhuri, I. Seskar, M. Ott, S. Ganu, K. Ramachandran, H. Kremo, R. Siracusa, H. Liu, and M. Singh. Overview of the ORBIT Radio Grid Testbed for Evaluation of Next-Generation Wireless Network Protocols. In *Proceedings of the IEEE Wireless Communications and Networking Conference (WCNC 2005)*, 2005.

[70] L. Rizzo. Dummynet : A Simple Approach to the Evaluation of Network Protocols. *ACM Computer Communication Review*, 27(1), January 1997.

[71] S. Sanghani, T. Brown, S. Bhandare, and S. Doshi. EWANT : Emulated Wireless Ad Hoc Network Testbed. In *Proceedings of IEEE Wireless Communications and Networking Conference (WCNC)*, March 2003.

[72] A. Schoonen. Designing Wireless Indoor Radio Systems with Ray Tracing Simulators. Technical report, Eindhoven University of Technology, December 2003. Available at `http://people.spacelabs.nl/~admar/SimpleCSD/report.pdf`.

[73] S. Singh, M. Woo, and C. S. Raghavendra. Power-aware routing in mobile ad hoc networks. In *Proceedings of the 4th annual ACM/IEEE international conference on Mobile Computing and Networking (MobiCom'98)*, pages 181–190, New York, NY, USA, 1998. ACM Press.

[74] T. Stockhammer, G. Liebl, H. Jenkac, P. Strasser, D. Pfeifer, and J. Hagenauer. WiNe2 Wireless Network Demonstration Platform for IP-based Real-Time Multimedia Transmission. In *Proceedings of the International Packet Video Workshop*, 2003.

[75] C.-K. Toh. *Ad Hoc Mobile Wireless Networks : Protocols and Systems*. Prentice Hall, 2002.

[76] A. Vahdat, K. Yocum, K. Walsh, P. Mahadevan, D. Kostic, J. Chase, and D. Becker. Scalability and Accuracy in a Large-Scale Network Emulator. In *Proceedings of 5th Symposium on Operating Systems Design and Implementation (OSDI)*, december 2002.

[77] S.Y. Wang, C.L. Chou, C.H. Huang, C.C. Hwang, Z.M. Yang, C.C. Chiou, and C.C. Lin. The Design and Implementation of the NCTUns 1.0 Network Simulator. *Computer Networks*, 42(2) :175–197, 2003.

[78] S.Y. Wang and Y.B. Lin. NCTUns Network Simulation and Emulation for Wireless Resource Management. *Wireless Communications and Mobile Computing*, 5(8) :899–916, December 2005.

[79] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar. An Integrated Experimental Environment for Distributed Systems and Networks. In *Proceedings of OSDI02*, 2002.

[80] M. Zec. Implementing A Clonable Network Stack in the FreeBSD Kernel. In *USENIX Annual Technical Conference, FREENIX Track*, pages 137–150. USENIX, 2003.

[81] X. Zeng, R. Bagrodia, and M. Gerla. GloMoSim : A Library for Parallel Simulation of Large-scale Wireless Networks. In *Proceedings of PADS '98*, May 1998.

[82] Y. Zhang and W. Li. An Integrated Environment for Testing Mobile Ad-Hoc Networks. In *Proceedings of the Third ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc'02)*, 2002.

[83] P. Zheng and L.M. Ni. EMWin : Emulating a Mobile Wireless Network using a Wired Network. In *Proceedings of the 5th ACM international workshop on Wireless mobile multimedia*, 2002.

[84] P. Zheng and L.M. Ni. EMPOWER : A Network Emulator for Wireless and Wireline Networks. In *Proceedings of IEEE INFOCOM 2003*, 2003.

[85] J. Zhou, Z. Ji, and R. Bagrodia. TWINE : A Hybrid Emulation Testbed for Wireless Networks and Applications. In *Proceedings of IEEE INFOCOM 2006*, April 2006.