Les réseaux sans fil spontanés pour l'Internet Ambiant

Vincent Untz

INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

N° attribué par la bibliothèque

|_|_|_|_|_|_|_|_|_|_|

THÈSE

pour obtenir le grade de

DOCTEUR DE L'INP Grenoble

Spécialité : « Informatique : Systèmes et Logiciels »

Préparée au Laboratoire d'Informatique de Grenoble

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

préparée et soutenue publiquement par

Vincent UNTZ

Le 5 décembre 2007

TITRE :

Les réseaux sans fil spontanés pour l'Internet Ambiant

Directeurs de thèse :
***Andrzej* DUDA**
***Martin* HEUSSE**

JURY

Pr. Denis TRYSTRAM	*Président*
Pr. Christian BONNET	*Rapporteur*
Pr. Isabelle GUÉRIN LASSOUS	*Rapporteur*
Pr. Andrzej DUDA	*Directeur de thèse*

Il m'est impossible de ne pas exprimer ma gratitude envers Andrzej Duda et Martin Heusse qui ont fait plus que simplement encadrer mes travaux puisque j'ai été accueilli avec chaleur et bonne humeur. Les discussions techniques que nous avons partagées, bien que lumineuses, resteront éclipsées par toutes les autres discussions tout aussi passionnantes. Dziękuję !

Christian Bonnet et Isabelle Guérin Lassous, mes deux rapporteurs, ont eu la patience et l'énergie nécessaire à la lecture de ce manuscrit et je les en remercie vivement. Leurs remarques et le recul ainsi apporté m'ont été bénéfiques. J'ai en outre été honoré d'avoir Denis Trystram pour président dans mon jury.

Ces années de travail n'auraient pas été les mêmes sans la présence de tant de personnes agréables à côtoyer au laboratoire, à commencer par tous les membres de l'équipe Drakkar, mais aussi les membres des autres équipes et le personnel administratif. Totalement en vrac, merci donc à vous tous, et notamment à Franck, Gilles, Mohammad, Abdelmalik, Cristina, Fabrice, Pawel, Aurélie, Dia, Christiane, Pascale, Carol, Vianney, Jean-Luc.

Il serait malheureux pour moi de ne pas adresser un mot plus particulier pour ceux qui sont devenus des amis proches : Yan, complice et plein d'idées aussi folles que les miennes ; Céline, grâce à qui manigancer et comploter devient un réel plaisir ; Marius, qui cache beaucoup de bonnes choses, sans qu'il ressente le besoin de les exhiber ; Nicolas, dont l'excellente humeur n'aurait d'égal que son appétit ; Alph, à la fois sérieux et intenable, mais toujours à l'affût, sauf quand il dort ; Fred, qui malgré un passage éclair aura instauré de longues traditions.

Durant ma thèse, j'ai pu avoir le plaisir de découvrir les joies de l'enseignement au sein de l'INPG, et plus particulièrement de l'ESISAR et de l'ENSIMAG. Les équipes pédagogiques, mais aussi les étudiants m'ont ouvert l'esprit et offert de très bons moments. De plus, l'enseignement m'a permis de m'échapper de la recherche lorsque j'en ressentais le besoin. Il s'agit donc d'une expérience sans prix.

J'adresse bien évidemment mes remerciements à ma famille, mes parents, mes frères, mais aussi à mes amis qui ont tous su me soutenir sans insister et s'informer de l'état d'avancement de mes travaux sans être trop curieux. Aurélie a suivi cette étape de ma vie et m'a encouragé durant toutes ces années. Je me dois en outre d'adresser une mention particulière à Olivia, Élodie, Laurent et plus généralement à tous mes amis de ma promotion regroupés dans le groupuscule du chat.

Naturellement, je tiens à ne pas oublier et à remercier mes amis de la communauté GNOME qui, durant toute cette période de thèse, m'ont permis de passer de si bons moments, m'ont fait découvrir tant de choses et ont su me soutenir jusqu'au bout, tout en me taquinant sur la fin. Il est impossible de tous les nommer, mais j'ai une pensée particulière pour Daniel Veillard qui a su me remettre sur le droit chemin lorsque j'ai eu des idées de jeune fou.

Enfin, la rédaction d'une thèse marque les esprits, et elle sera pour moi toujours associée à certains morceaux de musique et à des moments de détente télévisuels. La bande originale de *Kundun* écrite par Philip Glass m'aura ainsi accompagné pendant mes débuts, avant que les thèmes entraînants que Klaus Badelt a composés pour *Pirates des Caraïbes* me permettent d'atteindre ma vitesse de croisière. Impossible aussi de ne pas penser à Danny Elfman, dont la musique est depuis longtemps ancrée en moi. Pour une pause entre deux paragraphes, rien ne remplacera jamais Peter Falk dans son éternel rôle de *Columbo* et la bonne humeur de Nagui.

Et merci à tout ceux que je n'ai pas pu citer. S'il est compréhensible de penser qu'il s'agit d'un oubli, je préfère avancer l'excuse du manque de place ;-).

Vincent

Expliquer, c'est oublier.
Kalîla et Dimna

Table des matières

Table des figures

Liste des tableaux

Chapitre 1

Introduction

> Oh, this is gonna be fun ! We can stay up late, swaping manly stories, and in the morning, I'm making waffles !
>
> Donkey, *Shrek*

Tout a commencé avec le surf. C'est sa passion pour ce sport qui poussa Norm Abramson, alors professeur à Stanford, à demander à l'Université de Hawaii si celle-ci souhaitait recruter un professeur. Arrivé à son nouveau poste en 1970, il travailla alors sur une architecture réseau permettant la communication entre les différentes îles de l'archipel. En effet, pour des raisons financières, il n'était pas possible de déployer des lignes de transmission entre chaque île et la piste de la communication radio fut donc développée.

Après quelques mois, les travaux d'Abramson aboutirent au système ALOHA [1], qui fut l'une des premières architectures réseau et surtout la première architecture sans fil basée sur la commutation de paquets. L'originalité d'ALOHA consiste dans le fait que, au lieu d'attribuer une fréquence à chaque transmission afin d'éviter le problème des collisions, le choix de partager un unique medium a été réalisé. Le réseau ALOHAnet [2] fut déployé dès la fin de l'année 1970 et fut une source d'inspiration importante pour d'autres chercheurs, tels que les créateurs d'Ethernet [3, 4]. En 1972, ALOHAnet devint le premier réseau à être connecté à un autre réseau particulièrement connu dans l'histoire informatique : ARPANET [5, 6].

Évidemment, d'autres travaux réalisés à cette période ainsi que des années plus tard sont tout aussi, voire plus importants que ce qu'a réalisé Abramson : Baran et Kleinrock ont tous deux introduit les principes de la commutation de paquets au début des années 1960 [7, 8] et Kleinrock a en outre énormément contribué à la création d'ARPANET en 1969 [9] ; Postel a joué un énorme rôle pour la documentation des protocoles dès 1969 et a continué cette tâche pendant de très longues

années [10] ; Cerf et Kahn ont développé TCP/IP à partir de 1973 [11] ; Berners-Lee a inventé le *World Wide Web* en 1990 [12], ce qui a été le déclencheur de la popularité d'Internet.

S'il n'est pas l'unique pionnier, Abramson fait néanmoins partie de ce groupe de chercheurs qui a révolutionné les réseaux. Alors même que les réseaux constituaient un nouveau domaine de recherche, il s'est intéressé au cas des réseaux sans fil avec toutes les contraintes qu'ils impliquent. Et trente ans plus tard, le Wi-Fi [13] est apparu et a permis la démocratisation des réseaux sans fil. Le domaine de recherche des réseaux sans fil est toujours aussi actif.

1.1 Motivations et problématique

La société actuelle s'est petit à petit adaptée aux nouvelles technologies qui deviennent toujours plus omniprésentes. Ainsi, le taux de pénétration de la téléphonie mobile en France est de plus de 82% [14] ; de même, les connexions à Internet haut-débit deviennent la norme et non plus l'exception avec plus de 14 millions d'abonnements haut-débit [15]. Ces changements reflètent un changement de mode de vie plus général : les utilisateurs veulent pouvoir accéder à tout moment à différentes informations et pouvoir communiquer entre eux dès qu'ils en ressentent le besoin. Certains parlent même d'une dépendance envers ces technologies.

En outre, l'évolution du matériel reflète une tendance à la mobilité, facilitant cette omniprésence technologique. De nombreux appareils, comme les téléphones portables et les assistants personnels, se voient doter d'une interface 802.11 ; la création de nouveaux types d'appareils, telles que les tablettes Internet, visent même directement ce marché naissant. Une conséquence de cette tendance à la mobilité est donc l'utilisation, presque pervasive, des technologies sans fil pour le réseau.

Ces changements s'accompagnent aussi de difficultés, principalement pour les utilisateurs, car les réseaux actuels ne sont pas adaptés à de telles utilisations. La quasi-nécessité de disposer d'une infrastructure limite de fait les possibilités : à titre d'exemple, les connexions directes entre appareils restent généralement une exception. En outre, la configuration des terminaux reste pour le moins délicate dans de nombreux cas puisque le public ne dispose pas — et ne souhaite pas nécessairement disposer — de compétences réseaux avancées.

Parallèlement à cela, l'informatique pervasive [16] commence à apparaître dans les habitations, avec la mise en réseau de différents appareils. Cette mise en réseau peut être aussi bien réalisée grâce à des technologies filaires que sans fil. Il semble donc naturel de distinguer ces nouveaux réseaux, composés de l'ensemble des terminaux dont disposent une personne chez elle, ou à son bureau. Il s'agit en effet d'un contexte particulier nécessitant des solutions dont les priorités doivent

être l'autoconfiguration et les performances [17], ces dernières étant évaluées selon des critères qui ne sont pas nécessairement les mêmes que pour un réseau de cœur. Ces réseaux de bordure peuvent être élargis à l'échelle d'un bâtiment en gardant les mêmes caractéristiques.

Toutes ces évolutions créent un nouveau cadre d'étude pour les réseaux, et c'est dans celui-ci que se situent les travaux effectués pendant cette thèse. On suppose qu'IP reste le protocole de base des communications dans ces nouvelles catégories de réseau.

1.2 Organisation du manuscrit

Le manuscrit de cette thèse est organisé en neuf chapitres. Cette introduction a permis d'illustrer la problématique de la connectivité dans des réseaux qui commencent à apparaître et d'expliquer les motivations qui ont poussé à la conception de nouvelles solutions.

Les différentes technologies bas-niveau auxquelles peut être confronté un utilisateur sont brièvement présentées dans le chapitre 2, avec notamment les problèmes qui leur sont spécifiques.

Une étude approfondie des algorithmes et protocoles de routage est réalisée dans le chapitre 3. Trois méthodes traditionnelles de routage servent d'introduction car, malgré leur relative ancienneté, elles se révèlent être toujours d'actualité. Par la suite, une attention particulière est portée au routage dans les réseaux sans fil ad hoc, qui est un domaine de recherche particulièrement actif ces dernières années. Enfin, une nouvelle catégorie d'architectures se plaçant au niveau 2.5 est détaillée et certains points communs intéressants de ces différentes architectures sont dégagés.

Le chapitre 4 consiste en une présentation du cadre considéré pour les travaux avec l'introduction des réseaux de bordure spontanés. Leurs principales caractéristiques y sont décrites, et des choix architecturaux servant de base à l'élaboration de la solution y sont justifiés.

Le cahier des charges étant établi, une architecture d'interconnexion pour les réseaux de bordure spontanés est proposée et décrite en détail dans le chapitre 5. Les notions présentes dans cette architecture y sont introduites et le fonctionnement global y est par la suite analysé par modules, en particulier en ce qui concerne les étapes du routage et de la maintenance des connexions. Des optimisations optionnelles sont en outre exposées.

Le travail d'implémentation est présenté dans le chapitre 6. Les outils utilisés sont abordés en premier lieu afin de pouvoir comprendre le fonctionnement interne de l'implémentation du prototype pour l'architecture proposée, Lilith. Les raisons

qui ont amené à créer une nouvelle implémentation de MPLS y sont aussi expliquées.

Dans le chapitre 7, une confrontation avec la réalité est effectuée : le prototype est mis à l'épreuve avec des expérimentations dont le but est de valider l'architecture. En outre, des problèmes rencontrés de manière concrète lors de différents tests sont détaillés, certains étant liés au prototype mais d'autres dépendant de l'architecture même.

La suite possible des travaux est envisagée dans le chapitre 8, avec des modifications potentielles de l'architecture proposée, mais aussi des thèmes de recherches plus larges qui s'appliquent à d'autres contextes. Il y est de plus montré que l'expérience acquise permet de s'intéresser différemment à des réseaux étendus.

Enfin, nous présentons les conclusions que permet de tirer cette thèse. Un recul est pris par rapport à la recherche dans le monde des réseaux afin d'envisager des perspectives encore plus larges que la simple continuation des travaux effectués.

Première partie

État de l’art

Chapitre 2

Les technologies bas-niveau pour réseaux locaux

> I don't make things difficult. That's the way they get, all by themselves.
>
> Martin Riggs, *Lethal Weapon*

Dans sa vie quotidienne, une personne n'utilise pas de technologies bas-niveau particulièrement variées. Ethernet, 802.11 et Bluetooth sont même généralement les seules technologies mises en œuvre dans le cadre d'un réseau local. Ce faible nombre s'explique par les standardisations qui ont eu lieu au cours du temps afin de garantir l'interopérabilité des systèmes.

Cependant, il peut arriver qu'une autre technologie apparaisse et soit utile dans des conditions pour lesquelles il n'y a pas d'autres solutions. Les couches hautes, avec notamment IP, permettent alors l'interconnexion de manière transparente des différents réseaux entrant en jeu.

Les trois principales technologies présentées ici, Ethernet, 802.11 et Bluetooth, appartiennent à la famille IEEE 802 [18] : sous cette appellation sont regroupées des normes qui ont trait aux réseaux locaux et métropolitains. Le développement de Bluetooth continue néanmoins au sein du Bluetooth SIG [19].

2.1 Ethernet

Ethernet est le nom courant de la norme 802.3. Historiquement, ses créateurs ont tiré leur inspiration d'ALOHA [1], et certaines ressemblances entre les deux

spécifications permettent d'en attester. Il s'agit désormais de la norme de réseau filaire sans conteste la plus répandue, et ce depuis plus de vingt ans.

Le câble sur lequel est branché une interface Ethernet constitue un medium partagé par toutes les interfaces présentes sur ce câble. Il convient donc de disposer d'un protocole d'accès au canal, qui est ici CSMA/CD (*Carrier Sense Multiple Access / Collision Detection*). CSMA/CD permet de détecter une collision sur le medium ; lorsque c'est le cas, un mécanisme de résolution de la collision utilisant un algorithme d'attente exponentiel par calcul binaire (*binary exponential backoff*) est utilisé. Ni CSMA/CD ni cet algorithme ne seront détaillés ici, pour des raisons de concision.

Alors qu'initialement Ethernet supportait un débit de 10 Mb/s, la norme a été modifiée pour permettre du 100 Mb/s avec Fast Ethernet, puis du 1000 Mb/s avec Gigabit Ethernet. Les câblages utilisés ont eux aussi évolué dans le temps.

Le succès rencontré par Ethernet s'explique relativement naturellement : il s'agit d'une technologie simple, facile à mettre en place, fiable et peu coûteuse. Aucune configuration particulière ne s'avère nécessaire. En outre, la norme a évolué afin de répondre aux attentes grandissantes en terme de débit, tout en gardant une compatibilité descendante, permettant de conserver le matériel déjà déployé.

Ethernet a néanmoins un inconvénient assez visible : la nécessité d'un câble pour relier deux stations. Or, le câble rend toute mobilité extrêmement complexe. Cet inconvénient n'était pas majeur il y a quelques années, mais l'apparition des réseaux sans fil a changé le mode de vie des utilisateurs et de nombreuses personnes considèrent désormais qu'un câble est une contrainte importante.

2.2 802.11

Lorsqu'il est apparu qu'il existait une demande pour les réseaux sans fil locaux, de nombreuses solutions ont été mises sur le marché, chacune étant différente des autres. Les problèmes de compatibilité ne pouvant qu'empêcher une démocratisation des technologies sans fil, un processus de standardisation a eu lieu et a abouti à la norme 802.11 [20], connue également sous le nom de Wi-Fi. Les débits atteignables vont de 1 Mb/s à 54 Mb/s, et bientôt 248 Mb/s.

802.11 est une norme en constante évolution. Plus d'une dizaine d'amendements ont été approuvés ou sont sur le point de l'être ; les différents amendements varient entre améliorations en terme de débit (802.11a [21], 802.11b [22], 802.11g [23], 802.11n [24]) et meilleurs mécanismes de sécurité (802.11i [25]), en passant par des spécificités liées aux régulations de différents pays (802.11d [26], 802.11h [27], 802.11j [28]) ou encore l'intégration de mécanismes de qualité de service (802.11e [29]). La fréquence très importante de ces amendements a cependant tendance à créer une certaine confusion.

Cette technologie est désormais extrêmement répandue car la majorité des ordinateurs portables en sont équipés. En outre, tous les appareils un tant soit peu mobiles commencent à disposer d'interfaces 802.11 ; on peut notamment constater cette tendance sur les téléphones portables.

Deux modes de fonctionnement sont disponibles en 802.11 :
- le mode infrastructure : dans ce mode, toute communication passe par un point d'accès. Celui-ci joue aussi souvent le rôle de passerelle vers un réseau filaire, voire vers Internet. Le point d'accès joue ici le rôle d'une infrastructure servant au bon fonctionnement du réseau ;
- le mode ad hoc : toute communication entre deux stations a dans ce cas lieu directement, sans intermédiaire. L'intérêt de ce mode réside dans la possibilité de communiquer en l'absence d'infrastructure.

C'est majoritairement le mode infrastructure qui est utilisé en raison du fait que les utilisateurs souhaitent généralement avoir accès à Internet, et donc ont besoin d'une passerelle que le point d'accès fournit.

D'un point de vue technique, 802.11 est vu par les couches hautes de la pile réseau de la même façon qu'Ethernet. Néanmoins, la couche physique, qui ne sera pas détaillée ici, est extrêmement complexe et la couche MAC fonctionne différemment de celle d'Ethernet. Cette dernière peut utiliser deux modes : DCF (*Distributed Coordinated Function*, fonction de coordination distribuée) ou PCF (*Point Coordinated Function*, fonction de coordination par point d'accès). PCF étant optionnel, c'est généralement DCF qui est mis en œuvre. DCF se base sur CSMA/CA (*Carrier Sense Multiple Access / Collision Avoidance*) [30]. L'amendement 802.11e [29] introduit un nouveau mode HCF (*Hybrid Coordinated Function*), avec deux méthodes : EDCA (*Enhanced Distributed Channel Access*) et HCCA (HCF *Controlled Channel Access*), cette seconde méthode étant optionnelle ; ces deux méthodes définissent des classes de trafic, ce qui permet de modifier le comportement de l'accès au medium en fonction de la priorité du trafic. Toujours pour des raisons de concision, ces fonctions ne seront pas étudiées plus en détail ici.

2.2.1 WDS

Une configuration spéciale des points d'accès rend la création d'une infrastructure étendue 802.11 possible. WDS (*Wireless Distribution System*, système de distribution sans fil) permet d'interconnecter les points d'accès entre eux tout en leur laissant la capacité de jouer le rôle de point d'accès auprès des clients sans fil. Le protocole WDS n'est pas défini dans la norme 802.11 et il n'existe pas de spécification détaillée à l'heure actuelle : il est fréquent de rencontrer des problèmes d'interopérabilité pour WDS entre des matériels de constructeurs différents.

WDS tire profit du fait que l'en-tête d'une trame 802.11 possède quatre champs d'adresse :

– adresse source ;
– adresse destination ;
– adresse émetteur ;
– adresse récepteur.

Habituellement, les quatre adresses ne sont pas utilisées simultanément. En effet, dans le cas d'une connexion directe entre deux stations, seules les adresses source et destination sont renseignées. De même, lorsque la communication passe par un point d'accès, seule s'ajoute une troisième adresse : l'adresse récepteur permet au point d'accès de recevoir la trame émise par la source, qu'il renverra alors à la destination en renseignant l'adresse émetteur — l'adresse récepteur devenant inutile pour cette seconde étape.

En remplissant les adresses émetteur et récepteur, il devient possible d'acheminer une trame sur plusieurs points d'accès afin d'atteindre une destination. L'utilisation de ces deux adresses peut être assimilée à une forme de routage au niveau de la couche MAC. C'est sur cette fonctionnalité que se base WDS.

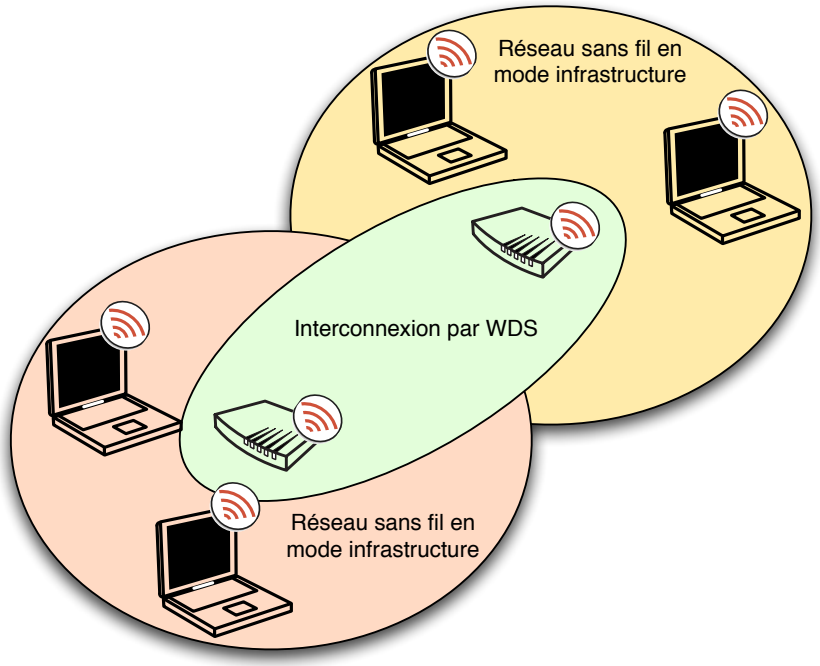

FIG. 2.1 – Utilisation de WDS pour interconnecter deux réseaux sans fil

Un protocole au niveau MAC est utilisé afin de distribuer les adresses MAC des stations connectées aux points d'accès à chaque point d'accès. C'est grâce à cela que, lorsqu'un point d'accès d'un réseau utilisant WDS reçoit une trame, il détermine si cette trame doit être retransmise et, si c'est le cas, vers quel point d'accès elle doit l'être.

Néanmoins, un inconvénient majeur de WDS est qu'il faut configurer manuellement chaque point d'accès pour lui indiquer quels autres points d'accès voisins participent à ce réseau étendu. Tous les points d'accès participant doivent partager le même canal. Il s'agit donc d'un fonctionnement adapté à un réseau statique, avec des points d'accès jouant le rôle d'infrastructure, mais peu adapté dans le cadre de réseaux avec une mobilité potentielle pour chaque station. En outre, l'utilisation de WDS a un impact important sur le débit car chaque point d'accès retransmettant une trame bloque les points d'accès voisins.

2.2.2 Difficultés liées aux technologies sans fil

La technologie 802.11 souffre de problèmes qui ne lui sont pas spécifiques, mais qui touchent toute technologie sans fil dans une mesure plus ou moins importante.

Consommation d'énergie

L'émission et la réception de données en sans fil sont des opérations coûteuses en énergie. Or, dans le cadre d'utilisation d'une technologie sans fil, il est habituel que la station soit mobile et donc sur batterie. L'énergie devient alors une ressource précieuse qu'il faut préserver lorsque c'est possible.

Si des stations jouent des rôles particuliers dans un réseau sans fil, comme cela peut être le cas à cause du routage, alors leur consommation d'énergie est plus importante et leur espérance de vie diminue, ce qui peut au final avoir un impact sur le bon fonctionnement du réseau lui-même. Une gestion efficace de l'énergie se révèle donc être un élément à prendre en compte.

Le simple fait d'écouter sur une fréquence radio consomme de l'énergie. Une solution aidant à limiter la consommation d'énergie consiste donc à mettre en veille régulièrement le périphérique radio afin de ne pas écouter en permanence.

Atténuation du signal

Les signaux radio ont une puissance limitée, qui diminue avec la distance mais aussi avec les obstacles de l'environnement tels que les murs. Des réglementations gouvernementales participent en outre à la limitation des puissances utilisées. Il en résulte une portée relativement courte des données émises.

Interférences

Deux communications sans fil simultanées peuvent interférer entre elles et donc rendre leurs signaux mutuels incompréhensibles. Il est donc nécessaire d'éviter dans la mesure du possible toute communication simultanée, et c'est ce que réalise 802.11 avec CSMA/CA. Néanmoins, des facteurs extérieurs tels que d'autres technologies sans fil ou des appareils électriques peuvent aussi contribuer des interférences dans le cas où la bande de fréquences n'est pas réservée à cet usage. Ainsi, 802.11 et Bluetooth partagent tout deux la bande de fréquence de 2.4 GHz et l'utilisation des deux technologies peut poser problème [31].

Sécurité

Par la nature même du medium utilisé, il n'est pas possible de contrôler qui peut « écouter » les communications. Il faut donc supposer que toute donnée transmise peut être interceptée, et un chiffrement peut s'avérer nécessaire. En outre, sans authentification, on ne peut s'assurer que les données reçues n'ont pas été émises par une station tierce.

Un aspect plus problématique de la sécurité des réseaux sans fil est qu'il est extrêmement simple pour une station de bloquer totalement le réseau en émettant en permanence des données afin de brouiller toute communication.

2.2.3 Problèmes spécifiques à 802.11

Au delà des difficultés inhérentes à toute technologie sans fil, 802.11 connait aussi des problèmes spécifiques liés à la norme elle-même. Ceux-ci peuvent être dévoilés dans le cas de topologies particulières, généralement dans un réseau ad hoc multi-sauts, ou causés par la possibilité de changer le taux de transfert utilisé par les interfaces 802.11.

Anomalie de performances

En raison des algorithmes utilisés dans la couche MAC, une anomalie dans le comportement de 802.11 existe et peut avoir un impact important sur les performances en terme de débit des stations [32]. En effet, des stations configurées avec un taux de transfert élevé voient leur débit diminuer lorsqu'une station utilise un taux de transfert plus bas. Cela s'explique par le fait que la répartition du temps d'occupation du canal n'est dans ce cas pas équitable : toutes les stations ont autant de chance d'accéder au canal pour l'émission d'une trame, mais une station avec un faible taux de transfert occupe le canal plus longtemps, créant ainsi un déséquilibre.

Les stations cachées

Le problème des stations cachées [33] est lié au fait que la portée du signal n'est pas toujours suffisante pour permettre à toutes les stations pouvant interférer avec une émission d'être informées de celle-ci. Une illustration très simple de ce problème est possible avec seulement trois stations A, B et C positionnées sur une chaîne comme dans la figure 2.2. A ne peut pas entendre C lorsqu'elle émet vers B, et donc A considère pouvoir utiliser le canal. En cas d'émission de A, des collisions au niveau de B seront donc causées.

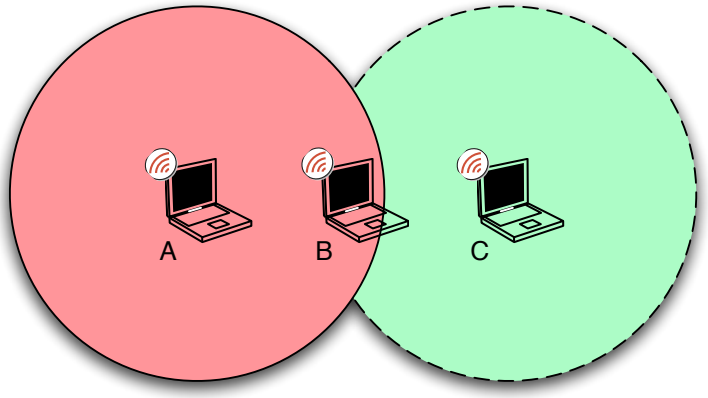

FIG. 2.2 – Problème des stations cachées

En pratique, dans le cas d'un réseau ad hoc multi-sauts, cette configuration peut ne pas relever de l'exception.

Le mécanisme de demande de permission d'émettre RTS/CTS (*Request to Send / Clear to Send*) a donc été introduit afin d'éviter tout problème de ce type [34]. La station ayant une trame à émettre envoie une trame RTS à la destination, cette trame incluant la durée nécessaire pour l'émission des données ; la destination répond avec une trame CTS, qui inclue elle aussi la durée que contenait le message RTS. Toute autre station que la source recevant la trame CTS s'abstient alors d'émettre pendant la durée spécifiée dans la trame CTS.

Les stations exposées

Le problème des stations exposées [35] est, en quelque sorte, l'inverse du problème des stations cachées. Dans le cas des stations exposées, l'émission d'une station peut bloquer l'émission d'une autre station alors que les deux destinations auraient pu recevoir les deux trames sans difficulté.

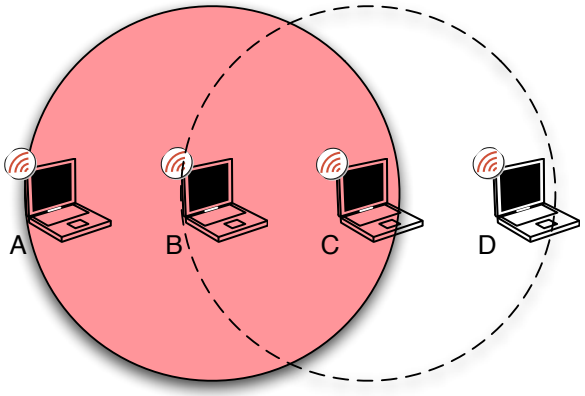

FIG. 2.3 – Problème des stations exposées

Par exemple, avec le réseau illustré par la figure 2.3, si B émet une trame vers A et que C souhaite émettre une trame vers D, C sera bloquée parce qu'elle détectera que le canal est utilisé. Or une émission par C aurait été reçue correctement par D, sans perturber la réception par A de la trame qui lui est destinée. Il y a donc sous-utilisation du canal.

La zone grise

L'envoi de trames en diffusion dans 802.11 est réalisé à un taux de transfert plus bas que celui utilisé pour les autres trames. Or, plus un taux de transfert est bas, plus la portée est importante. Il existe donc un écart de portée entre une trame envoyée en diffusion et une trame envoyée à une station précise : la différence entre les deux zones de portée est appelée la zone grise [36]. La figure 2.4 illustre ce phénomène.

Ce phénomène pose de réels problèmes, notamment pour les protocoles de routage. Ces derniers ont ainsi tendance à découvrir les routes avec des paquets émis en diffusion, et il est donc possible que les routes découvertes ne soient pas utilisables par la suite pour une émission normale.

2.3 Bluetooth

Bluetooth [37] est une technologie pour les communications sans fil à courte distance. La version 1.1 de la spécification a été standardisée par l'IEEE en 802.15.1, mais le Bluetooth SIG [19] a continué à développer cette technologie et la version actuelle est la version 2.1. De très nombreux types d'appareil peuvent disposer d'une

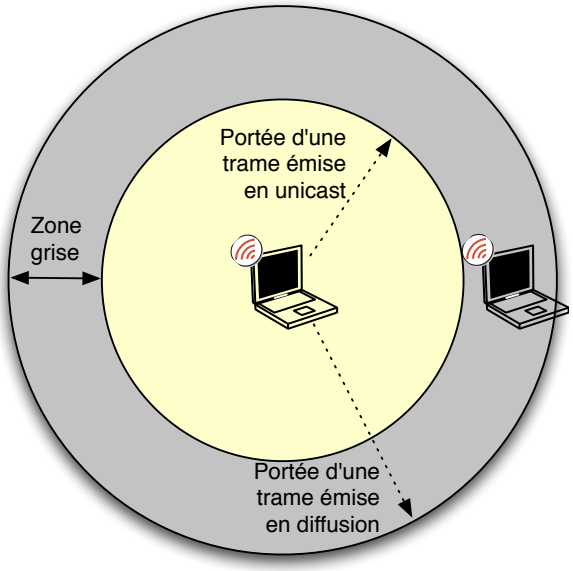

FIG. 2.4 – Zone grise d'une station utilisant 802.11

connectivité Bluetooth : téléphones portables, ordinateurs, imprimantes, appareils photo numériques, récepteurs GPS, claviers et souris, consoles de jeu, etc. Les puces Bluetooth coûtent peu à la construction et cet aspect économique a contribué à la large adoption de la technologie.

De par son cahier des charges, cette technologie est peu consommatrice d'énergie. Le revers de cette faible consommation est la courte portée des communications — trois classes existent, avec pour portée respectivement 10 m, 20 m et 100 m, la première classe étant généralement celle utilisée pour les puces — et le faible débit disponible — jusqu'à 3 Mb/s. Une nouvelle version de Bluetooth est prévue, promettant un débit pouvant atteindre 100 Mb/s. Les transmissions radio ont lieu sur la bande de fréquence de 2.4 GHz, qui a l'avantage d'être mondialement disponible.

2.3.1 Piconet et scatternet

Toute communication entre deux périphériques Bluetooth a lieu dans un *piconet*. Il s'agit d'un réseau qui est créé automatiquement lorsque plusieurs périphériques se situent à portée l'un de l'autre ; un piconet est donc un réseau ad hoc. L'espace d'adressage dans un piconet est sur trois bits et il ne peut donc pas y avoir plus de huit membres dans un tel réseau. Les communications dans un piconet ont lieu directement sur la couche 2 et IP n'est pas utilisé.

Dans un piconet, un des périphériques joue le rôle de *maître*, contrôlant le réseau, et les autres deviennent donc des *esclaves*. Tous les périphériques ont une

horloge synchronisée sur celle du maître ; en outre, deux esclaves ne peuvent pas communiquer directement entre eux et donc toute communication a lieu entre le maître et un esclave. Deux esclaves peuvent éventuellemenet communiquer par l'intermédiaire du maître.

Plusieurs piconets peuvent être connectés et former ainsi un *scatternet*, ce qui permet de s'affranchir de la limite de taille des piconets. L'interconnexion de deux piconets est possible grâce à l'intermédiaire d'un membre des deux réseaux qui joue le rôle de pont ; celui-ci ne peut néanmoins être maître que dans un piconet. L'utilisation d'un scatternet permet à deux périphériques Bluetooth de communiquer malgré une distance importante rendant impossible toute communication directe entre eux. Les scatternets restent cependant limités au cadre de la recherche pour l'instant car Bluetooth ne spécifie pas comment le routage est effectué dans un scatternet, et donc peu d'implémentations existent.

2.3.2 Réseau local personnel

BNEP [38] (*Bluetooth Network Encapsulation Protocol Specification*) est une spécification définissant un format de paquet Bluetooth permettant d'encapsuler les protocoles réseaux habituels, tels qu'IP. D'un point de vue application, BNEP joue le rôle d'une couche MAC habituelle. BNEP est utilisé comme base du profil Bluetooth PAN [39] (*Personal Area Network*). Ce profil définit le fonctionnement d'un piconet pour qu'il forme un réseau IP, le cas des scatternets étant ignoré. Dans ce contexte, les couches basses de Bluetooth, nécessaires au fonctionnement du piconet, sont cachées aux couches hautes.

PAN définit trois *scenarii* de fonctionnement :
- point d'accès à un réseau : dans ce scenario, un des membres du piconet est connecté à un réseau utilisant une autre technologie que Bluetooth — typiquement, un réseau local filaire. Ce membre, appelé le point d'accès, joue le rôle de pont ou de routeur entre le réseau Bluetooth et le second réseau, permettant aux autres membres du piconet d'accéder au second réseau.
- réseau ad hoc : des utilisateurs PAN peuvent former spontanément un réseau ad hoc, sans aucune infrastructure extérieure. Ils peuvent communiquer librement entre eux — une communication entre deux esclaves passe toujours par le maître —, mais n'ont accès à aucun autre réseau. Le réseau ad hoc est donc limité au piconet.
- connexion entre deux utilisateurs PAN : une connexion point-à-point entre deux périphériques est réalisée pour permettre une communication directe eux uniquement. Il s'agit en réalité de simuler l'utilisation d'un câble pour relier directement deux stations.

En pratique, il est relativement rare d'avoir plus de deux périphériques Bluetooth communiquant par IP dans une même zone. La principale utilisation des PAN

consiste donc en une connexion entre deux périphériques afin qu'ils communiquent par IP, l'un des deux périphériques jouant généralement le rôle de point d'accès vers un autre réseau.

Chapitre 3

Algorithmes et architectures de routage

> You must unlearn what you have learned.
>
> Yoda, *The Empire Strikes Back*

Dans la plupart des réseaux, les paquets émis doivent traverser des nœuds — les routeurs — avant de parvenir jusqu'à leur destination. À l'origine, les réseaux étaient tous de taille très limitée et il était donc possible de configurer manuellement la route pour chaque destination. Mais très vite, la taille des réseaux n'allant qu'en s'accroissant, il a fallu automatiser cette tâche à l'aide d'algorithmes de routage.

Il n'existe pas d'unique solution pour le problème du routage dans les réseaux car une solution valide dans une situation donnée ne l'est pas forcément dans un autre cas ; c'est pourquoi plusieurs algorithmes et protocoles sont utilisés dans des cadres différents. Différents critères d'évaluation existent pour les algorithmes de routage : l'exactitude des routes obtenues est essentielle, mais la stabilité des routes, les délais causés par le routage ou encore le niveau d'optimisation des routes sont autant de critères à prendre en compte. Le routage étant un élément de base dans le fonctionnement d'un réseau, il est nécessaire pour un algorithme de routage d'être solide et et de ne pas causer d'effondrement du réseau à cause d'erreurs.

Les notions de routage et d'acheminement sont différentes, bien que fortement liées : le routage consiste à trouver une route pour une destination tandis que l'acheminement consiste à transmettre le paquet d'un routeur à l'autre sur la route trouvée. L'acheminement est donc une phase postérieure au routage, mais certains algorithmes l'utilisent néanmoins de manière rétroactive sur le routage. Par exemple, l'acheminement peut permettre de valider une route, ou en cas de problème, de redéclencher l'algorithme de routage.

3.1 Méthodes de routage traditionnelles

Certains algorithmes de routage ont été définis très tôt mais restent des méthodes solides et éprouvées qui sont encore utilisées aujourd'hui, et qui gardent une influence très importante sur les travaux réalisés dans ce domaine.

3.1.1 Le routage par inondation

Le routage par inondation est probablement la méthode de routage la plus triviale : chaque routeur recevant un paquet le réémet sur toutes les interfaces si le routeur n'est pas la destination du paquet. Afin que la fin de l'inondation soit garantie, plusieurs techniques peuvent être employées.

Une première solution est d'utiliser un champ de durée de vie (*time to live*) dans les paquets transmis, champ qui est décrémenté à chaque retransmission du paquet. Lorsque la valeur du champ est nulle, alors le paquet n'est pas retransmis. Une difficulté pour l'émetteur est alors d'établir quelle valeur initiale choisir pour atteindre la destination sans surcharger le réseau ; une valeur arbitraire est généralement choisie. Le protocole IP possède un tel champ qui a pour valeur maximale 255.

Une seconde approche, qui peut complémenter la première, consiste à éviter d'envoyer une seconde fois les paquets déjà retransmis. Chaque paquet doit donc disposer d'un identifiant — il s'agit généralement de l'adresse de la source couplée avec un numéro de séquence incrémenté à chaque paquet émis par la source — que les routeurs utilisent afin de déterminer si le paquet a déjà été reçus et retransmis. Un paquet avec un identifiant connu sera donc ignoré, et il n'y aura pas de duplication inutile des paquets dans le réseau.

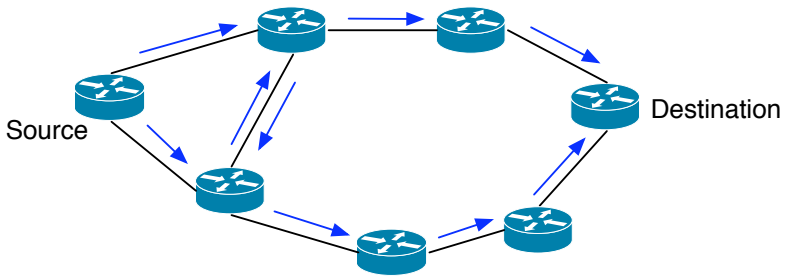

FIG. 3.1 – Émission d'un paquet dans le cas du routage par inondation

Une propriété du routage par inondation intéressante à noter est qu'il garantit que le meilleur chemin est utilisé pour atteindre la destination. En effet, par définition, tous les chemins sont utilisés de manière parallèle et cela inclut entre

autres le meilleur d'entre eux, quelle que soit la métrique servant de base pour la comparaison. Il faut cependant prendre en compte la surcharge induite par l'important trafic généré. Cette surcharge peut avoir un impact non négligeable sur les performances, comme c'est le cas dans un réseau sans fil.

Le routage par inondation a en outre pour particularité une grande robustesse en raison de la redondance liée à l'inondation. Si le paquet vers une destination est perdu sur un chemin, il suffit qu'un autre chemin disjoint du premier existe pour que le paquet parvienne tout de même à la destination.

S'il n'est pas adapté au routage de paquets à diffusion individuelle, le principe derrière le routage par inondation ne reste pas moins une option simple à mettre en œuvre et relativement peu coûteuse pour l'acheminement des paquets de diffusion.

3.1.2 Le routage par vecteur de distance

Une des premières approches non triviales qui a été développée pour le routage dans les réseaux informatiques est celle du routage par vecteur de distance. Elle se base sur l'algorithme de Bellman-Ford distribué.

Chaque routeur possède une table de routage qui consiste en un couple de données pour chaque destination : le routeur par lequel passer pour atteindre cette destination et le coût associé selon une métrique définie. Ces informations sont transmises périodiquement à tous les voisins, et donc chaque routeur reçoit ces informations de ses voisins. Il en résulte que, pour un routeur, toute destination existant dans la table de routage d'un voisin devient connue et donc accessible par ce routeur. Le tableau 3.1 illustre l'évolution des tables de routage des routeurs pour le réseau de la figure 3.2.

FIG. 3.2 – Réseau utilisant le routage par vecteur de distance

On peut noter que l'information relatant l'apparition d'un routeur est propagée assez rapidement, ce qui rend cet algorithme de routage réactif à l'apparition de nouveaux routeurs.

Généralement, la métrique utilisée pour cet algorithme est le nombre de sauts, mais d'autres métriques peuvent être utilisées. L'avantage de la métrique basée sur le nombre de sauts est qu'elle ne nécessite aucun traitement supplémentaire, la distance entre deux routeurs voisins restant toujours 1.

TAB. 3.1 – Évolution des distances dans les tables de routage de A, B, C et D dans le cas de la figure 3.2

Routeur	A	B	C	D
État initial	Vers A : 0	Vers B : 0	Vers C : 0	Vers D : 0
Après un échange	Vers A : 0 Vers B : 1	Vers A : 1 Vers B : 0 Vers C : 1	Vers B : 1 Vers C : 0 Vers D : 1	Vers C : 1 Vers D : 0
Après deux échanges	Vers A : 0 Vers B : 1 Vers C : 2	Vers A : 1 Vers B : 0 Vers C : 1 Vers D : 2	Vers A : 2 Vers B : 1 Vers C : 0 Vers D : 1	Vers B : 2 Vers C : 1 Vers D : 0
Après trois échanges	Vers A : 0 Vers B : 1 Vers C : 2 Vers D : 3	Vers A : 1 Vers B : 0 Vers C : 1 Vers D : 2	Vers A : 2 Vers B : 1 Vers C : 0 Vers D : 1	Vers A : 3 Vers B : 2 Vers C : 1 Vers D : 0

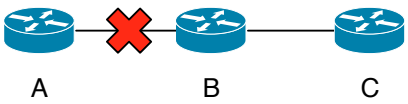

FIG. 3.3 – Problème de la rupture de lien dans le routage par vecteur de distance

Un des problèmes de cette méthode est que l'information d'une rupture de lien se propage très lentement. La figure 3.3 et le tableau 3.2 illustrent ce problème : on considère que les tables de routage sont correctement remplies et que le routeur A disparait brusquement. Les routeurs B et C s'échangent alors tous les deux une information erronée en supposant que l'autre routeur possède une route vers A, ce qui n'est pas le cas. À chaque étape, la distance vers A s'incrémente donc doucement. La notion d'horizon coupé a été introduite afin de limiter cette incrémentation et de la considérer comme une rupture de lien : lorsque la distance est supérieure à une certaine valeur, alors elle est considérée comme infinie, et la route est donc supprimée. La valeur utilisée pour l'horizon coupé ne peut cependant pas être trop faible car elle limite de fait le diamètre maximal du réseau. La prise en compte de la disparition d'un routeur nécessite donc une phase de convergence lente.

Cet algorithme forme le cœur du protocole RIP [40, 41], qui fut utilisé au début d'Internet. En raison du problème majeur qu'est la convergence des informations de routage en cas de rupture de lien et parce qu'il était plus adapté à des réseaux de taille limitée, le routage par vecteur de distance a été assez rapidement abandonné au profit du routage par état de lien [42].

TAB. 3.2 – Évolution des distances dans les tables de routage de B et de C dans le cas d'une rupture du lien entre A et B

Routeur	B	C
État initial	Vers A : 1 Vers B : 0 Vers C : 1	Vers A : 2 Vers B : 1 Vers C : 0
Après un échange	Vers A : 3 Vers B : 0 Vers C : 1	Vers A : 2 Vers B : 1 Vers C : 0
Après deux échanges	Vers A : 3 Vers B : 0 Vers C : 1	Vers A : 4 Vers B : 1 Vers C : 0
Après trois échanges	Vers A : 5 Vers B : 0 Vers C : 1	Vers A : 4 Vers B : 1 Vers C : 0
...	...	...

3.1.3 Le routage par état de lien

Le principe de base du routage par état de lien est que la connaissance de la topologie complète du réseau permet de calculer aisément une route de toute source vers toute destination. Il convient donc de faire en sorte que chaque routeur connaisse la topologie du réseau. Cette construction d'une représentation de la topologie se fait en deux étapes : chaque routeur effectue une découverte de ses voisins et informe l'ensemble du réseau des informations qu'il a collectées. On peut remarquer qu'il y a tout d'abord un processus local visant à obtenir une vision locale du réseau, et ensuite un processus global consistant à partager toutes les visions locales du réseau pour créer une vision globale, et donc la topologie du réseau.

Pour découvrir ses voisins, un routeur envoie un message HELLO sur chacune de ses interfaces. Chaque routeur recevant un tel message envoie une réponse, ce qui permet au routeur initial de savoir qui a reçu le message, et incidemment quels sont ses voisins. Cette étape est répétée régulièrement afin de détecter l'apparition de nouveaux voisins ou la disparition d'anciens voisins.

La mesure du coût d'un lien entre deux routeurs fonctionne comme pour le routage par vecteur de distance : il peut s'agir du simple nombre de sauts, ou d'une autre métrique prenant en compte, par exemple, le débit atteignable sur les liens.

Ensuite, chaque routeur rassemble l'ensemble des informations qu'il possède sur ses voisins et sur les coûts des liens qui le relient à ceux-ci dans un paquet d'état de lien, et transmet ce paquet en diffusion à tout le réseau. Ce processus est répété périodiquement, mais il est possible d'utiliser une fréquence faible et de

générer un tel paquet sans attendre la fin d'un intervalle lorsqu'un évènement tel que l'apparition ou la disparition d'un voisin a lieu. Cela permet une réactivité plus forte au changement de topologie. La diffusion des paquets d'état de lien s'effectue par inondation, comme décrit dans la partie 3.1.1.

Une fois la topologie du réseau connue, et celle-ci étant couplée avec l'ensemble des coûts des liens existant dans la topologie, un routeur dispose de toutes les informations nécessaires pour le routage. Il met en application l'algorithme de Dijkstra [43] et détermine le plus court chemin vers tous les autres routeurs existant dans le réseau ; ce plus court chemin définit la route à utiliser.

Par rapport au routage par vecteur de distance, le routage par état de lien nécessite donc des ressources en mémoire et en puissance de calcul plus importantes, ainsi qu'un trafic réseau plus soutenu — mais stable même dans le cas d'un changement de topologie — afin de garder une image valide de la topologie dans chaque routeur. Cette complexité offre évidemment certains avantages : chaque routeur connaissant la topologie du réseau, une analyse plus fine peut être réalisée de manière individuelle. Il est par exemple possible de connaître d'autres chemins que le meilleur chemin, ce qui peut s'avérer utile lorsqu'il y a dysfonctionnement sur une route. Une utilisation avancée peut consister à connaître la topologie couplée avec deux métriques distinctes, permettant de disposer de deux meilleurs chemins, chacun plus adapté à un type de trafic.

Le routage par état de lien est utilisé dans le protocole de routage interne le plus répandu actuellement, OSPF [44, 45]. Un autre protocole de routage par état de lien est IS-IS [46].

3.2 Le routage dans les réseaux sans fil ad hoc

Les méthodes de routage traditionnelles sont des méthodes éprouvées, qui fonctionnent parfaitement. Elle ne sont néanmoins pas adaptées à tous les réseaux, notamment aux réseaux sans fil ad hoc : de tels réseaux possèdent des contraintes plus fortes qu'un réseau filaire. Ces méthodes ont par exemple tendance à utiliser un trafic assez important, sans considérer l'accès au medium comme une ressource rare. En outre, elles ne sont pas adaptées aux cas de topologies fortement variables, ce cas étant suffisamment rare dans le cadre des réseaux filaires pour être ignoré ; or, dans le cadre d'un réseau sans fil, il n'est pas possible de supposer que les nœuds sont fixes.

Le problème du routage dans les réseaux sans fil a donc été très étudié ces dernières années, et ce largement en raison du groupe de travail de l'IETF *Mobile Ad hoc Networking* [47] (MANET).

3.2.1 Cadre des réseaux MANET

La RFC 2501 [48] a été publiée afin de motiver le travail effectué dans le cadre du groupe MANET et constitue en quelque sorte le document fondateur du groupe. En effet, le cadre des études est défini, ainsi que les caractéristiques souhaitées pour les solutions et des critères de performances pour évaluer ces dernières.

Les réseaux MANET sont constitués d'un ensemble de nœuds mobiles communiquant entre eux par des interfaces sans fil. La mobilité des nœuds résulte en une topologie qui peut varier au cours de temps, sans schéma prédéfini, et aussi dans la nécessaire prise en compte de la gestion de l'énergie ; les nœuds mobiles ne disposent en effet généralement pas d'une source d'énergie, et doivent donc optimiser leur fonctionnement. L'utilisation d'une technologie sans fil, le plus souvent 802.11, a pour conséquence une bande passante limitée, qui n'est pas comparable à ce qui existe dans un réseau filaire ; le fait que le medium est partagé ne contribue pas à améliorer cette situation et les congestions sont donc considérées plus fréquentes que dans le cas filaire. Enfin, l'accès au medium étant facilité, des problèmes de sécurité se posent et ne doivent pas être oubliés.

Des protocoles de routage ont été proposés pour prendre en compte les spécificités des réseaux MANET. Une contrainte forte imposée par le groupe de travail a été l'utilisation d'IP de manière traditionnelle, sans connexion. Les protocoles de routage doivent en outre être distribués en raison de l'absence d'infrastructure, et sans boucle de routage afin d'éviter tout problème majeur dans le réseau. La bidirectionnalité des liens est considérée comme la norme pour faciliter l'élaboration des protocoles.

Deux approches sont possibles pour un protocole de routage MANET : l'approche réactive, afin d'adapter le protocole au trafic existant pour minimiser les ressources consommées, ou l'approche proactive dans le but de minimiser le délai de mise en place des communications. D'éventuelles approches hybrides sont aussi envisageables.

De nombreux critères d'évaluation existent, tels que le débit de bout-en-bout, le temps d'acheminement de la source à la destination, le temps de découverte d'une route ou encore l'impact des messages du protocole de routage sur les performances du réseau.

Une méthodologie complète d'évaluation a donc été développée afin de juger de la qualité des différentes propositions. Parmi les très nombreuses propositions de protocoles envoyées au groupe de travail, l'attention s'est vite portée sur un nombre limité d'entre elles afin de les standardiser. Dans la suite de cette section, les principaux protocoles MANET seront donc présentés.

3.2.2 DSR

DSR [49] (*Dynamic Source Routing*) est un des premiers protocoles de routage pour les réseaux sans fil ad hoc qui a été proposé. Après de nombreuses années, il a été normalisé sous la forme de la RFC 4728 [50]. Il s'agit d'un protocole réactif qui a la particularité de s'appuyer sur un routage par la source. En effet, lorsqu'un paquet est émis, celui-ci contient toutes les informations qui sont nécessaires à son acheminement jusqu'à la destination. Le protocole se décompose en deux grandes phases : la découverte de route et la maintenance de route.

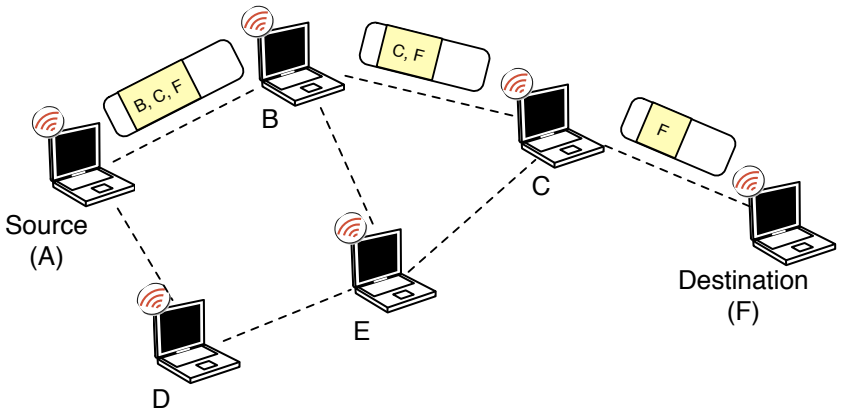

FIG. 3.4 – Émission d'un paquet dans le cas du routage par la source

Le routage par la source fonctionne en incluant dans chaque paquet la liste des nœuds par lesquels doit passer le paquet pour atteindre la destination. Une optimisation visant à réduire la surcharge de données transmises consiste à utiliser un routage par la source implicite [51].

Découverte de route

Lorsqu'un nœud cherche à émettre un paquet vers une destination pour laquelle il n'a pas de route en cache, le nœud initie une découverte de route vers la destination, appelée alors la cible, et met le paquet dans un tampon. Ce dernier sera automatiquement vidé après un délai sans réponse.

Un message ROUTE REQUEST est envoyé en diffusion à l'aide d'un mécanisme d'inondation. Ce message contient les informations nécessaires au bon fonctionnement de la découverte de route, à savoir : l'adresse du nœud initiateur, l'adresse de la cible, un identifiant unique de la requête — l'unicité est locale au nœud initiateur —, ainsi qu'une liste de tous les nœuds parcourus par le message. Cette liste est évidemment différente pour chaque instance du message.

Lorsqu'un nœud reçoit un message ROUTE REQUEST, s'il n'est pas la cible de cette requête, alors il détermine s'il doit retransmettre la requête. Celle-ci ne sera retransmise que si aucune requête du même nœud initiateur avec le même identifiant n'a été reçue et si le nœud n'apparait pas dans la liste des nœuds parcourus par le message. Avant l'éventuelle retransmission, le nœud s'ajoute à la liste des nœuds parcourus.

Si le nœud recevant le message ROUTE REQUEST est la cible de la requête, alors une réponse ROUTE REPLY est envoyée au nœud initiateur. Cette réponse contient la liste des nœuds parcourus par le message ROUTE REQUEST reçu. Si la cible possède une route vers le nœud initiateur, alors cette route est utilisée pour l'envoi de réponse. Dans le cas contraire, deux solutions sont possibles :

- le mécanisme de découverte de route peut être utilisé pour obtenir une route de la cible au nœud initiateur de la première requête. Si cette approche est utilisée ; la réponse ROUTE REPLY doit être incluse « sur le dos » du nouveau message ROUTE REQUEST (*piggyback*) afin d'éviter toute boucle ;
- la liste des nœuds parcourus est inversée et utilisée comme route pour la réponse ROUTE REPLY. Cette approche est requise si le support du réseau est 802.11 afin de vérifier que toute la route fonctionne de manière bidirectionnelle.

Lorsque le nœud initiateur reçoit une réponse ROUTE REPLY, la route fournie est mise en cache afin de pouvoir être réutilisée ultérieurement. Les paquets mis en tampon pour la cible sont finalement émis.

Différentes optimisations sont possibles. La plus importante est probablement la possibilité pour un nœud recevant un message ROUTE REQUEST pour une cible pour laquelle il possède une route en cache d'envoyer directement au nœud initiateur une réponse ROUTE REPLY contenant la route en cache concaténée à la liste des nœuds parcourus par le message ROUTE REQUEST.

Une seconde optimisation consiste à mettre en cache toute route qui est apprise de manière inopinée afin d'éviter une éventuelle découverte de route pour la destination. Ainsi, tout nœud intermédiaire acheminant une réponse ROUTE REPLY peut disposer gratuitement d'une route vers la cible ayant émis cette réponse, mais aussi vers les nœuds entre lui-même et la cible.

Maintenance de route

Lors de l'acheminement d'un paquet d'une source à une destination, la retransmission du paquet pour chaque saut est acquittée par le nœud intermédiaire recevant le paquet. Cet acquittement peut avoir plusieurs formes, actives ou passives :

- si la couche MAC fournit un acquittement pour la réception des trames, cet acquittement est automatiquement utilisé comme acquittement du paquet ;

- si le nœud qui a retransmis le paquet « entend » le nœud qui l'a reçu envoyer le paquet vers le nœud suivant sur la route, alors le paquet est considéré comme acquitté ;
- une approche plus directe consiste à demander explicitement un acquittement par l'envoi d'un simple message. Dans un tel cas, il est possible de considérer l'acquittement comme valide pour un temps limité et de ne pas requérir d'acquittement pour les prochains paquets émis dans cet intervalle de temps.

Si un paquet n'est pas acquitté lors d'un saut, alors toute route dans le cache passant par le nœud suivant est invalidée et un message ROUTE ERROR est émis à destination des nœuds utilisant le nœud suivant dans une de leurs routes afin de les prévenir que celles-ci ne sont plus valides.

Une opération appelée « sauvetage des paquets » peut aussi être effectuée dans le cas où le nœud détectant un lien brisé dispose d'une route alternative vers la destination du paquet. L'opération consiste à utiliser cette route en remplacement de la route spécifiée par l'émetteur du paquet. Le message ROUTE ERROR est toujours transmis.

Évaluation

DSR est un protocole qui a l'avantage d'être relativement simple, tout en fournissant de bons résultats. L'approche réactive et l'absence de messages périodiques liés au routage permettent de ne pas avoir d'impact majeur en terme de charge sur le réseau, mais aussi d'énergie consommée. Les nœuds par lesquels aucune route ne passe ne consomment pas non plus d'énergie pour le bon fonctionnement du réseau, sauf durant la phase de découverte de route. Un dernier aspect intéressant de DSR est lié au routage par la source : chaque nœud intermédiaire ne fait alors que rentransmettre « bêtement » le paquet sur une route choisie dans son ensemble par la source. La source peut donc effectuer différents choix de routage qui seront respectés.

Le protocole n'est pas pour autant exempt de défauts. L'approche réactive crée évidemment un certain délai avant toute communication. En outre, l'utilisation intensive de caches pour les routes pose des questions quant à la validité dans le temps de ces caches, notamment par les nœuds qui obtiennent les routes de manière indirecte et qui peuvent donc ne pas être informés d'erreurs lors de la phase de maintenance des chemins. Enfin, le routage par la source crée évidemment une surcharge au niveau de chaque paquet transmis en raison des informations supplémentaires qui sont nécessaires.

3.2.3 AODV

Un second protocole de routage pour les réseaux MANET est AODV [52] (*Ad-hoc On-demand Distance Vector*), qui a été normalisé avec la RFC 3561 [53]. AODV a fait

l'object de nombreux travaux. Comme DSR, il s'agit d'un protocole réactif, et donc il existe des similitudes importantes entre les deux protocoles. Néanmoins, AODV n'utilise pas de routage par la source, et utilise des numéros de séquence afin de déterminer si un message est plus récent ou ancien que l'information déjà connue. En outre, une métrique est utilisée afin de pouvoir utiliser une meilleure route si elle devient disponible ; il s'agit d'une métrique comptant simplement le nombre de sauts. Ce protocole est donc proche du routage par vecteur de distance vu dans la partie 3.1.2.

Découverte de route

Un nœud souhaitant communiquer avec une destination pour laquelle il n'a pas de route dans sa table de routage émet une requête RREQ en inondation. Cette requête contient un identifiant de la requête, le nombre de sauts parcourus, ainsi que le numéro de séquence de la source et le dernier numéro de séquence connu de la destination.

Un nœud recevant une requête RREQ vérifie d'abord que celle-ci n'a pas déjà été reçue : si l'identifiant de la requête est connue, alors elle est ignorée. Si le nœud ne connait pas de route pour la destination demandée, alors les informations liées à la requête, dont le numéro séquence de la source, sont enregistrées localement de manière temporaire afin de pouvoir faire suivre une éventuelle réponse RREP vers l'émetteur de la requête sur la route inverse. En particulier, si un message est reçu avec un numéro de séquence correspondant au nœud initiateur plus récent que le numéro de séquence enregistrées dans ce cache, alors les informations sont mises à jour pour disposer de la route inverse plus récente.

Lorsque la requête RREQ parvient à un nœud disposant d'une route vers la destination, ou à la destination elle-même, une réponse RREP est envoyée sur la route inverse. Si le nœud recevant la requête n'est pas la destination, il vérifie d'abord que le numéro de séquence de sa route vers la destination est supérieur ou égal au numéro de séquence destination inclus dans la requête : si ce n'est pas le cas, cela signifie que le nœud initiateur de la requête ignorera la réponse et donc celle-ci ne doit pas être émise, et la requête doit être retransmise. Une réponse RREP contient l'adresse du nœud initiateur de la requête, l'adresse de la destination de la requête, le numéro de séquence de la destination de la requête, le nombre de sauts parcourus par la requête. La réponse RREP est émise sur la route inverse de celle parcourue par la requête, à destination de l'initiateur de la requête.

Chaque nœud recevant une réponse RREP met à jour sa table de routage vers la destination s'il n'y avait pas d'entrée, si le numéro de séquence destination de la réponse est plus grand que le numéro de séquence enregistré dans la table de routage, ou si la métrique est meilleure dans le cas où les deux numéros des séquences sont égaux. S'il s'agit de la première réponse reçue pour cette destination — donc si

la table de routage n'avait pas d'entrée pour cette destination —, alors la réponse est retransmise sur les chemins inverses qui ont été gardés en mémoire pour cette destination lors de la réception des requêtes RREQ.

Le nœud initiateur finit par recevoir une réponse RREP, et peut ainsi émettre les paquets vers la destination.

Maintenance de route

De même que pour DSR, une maintence de route est effectuée, notamment pour détecter la rupture de liens. Le fonctionnement est néanmoins différent.

Un premier but est de garantir qu'un lien est toujours bidirectionnel. Si cela est possible, la couche MAC est utilisée à cette fin grâce aux acquittements MAC. Dans le cas contraire, un mécanisme simple de messages HELLO périodiques est mis en œuvre : ces messages sont envoyés à tous les voisins par un nœud pour signaler son existence. Un message HELLO contient la liste de tous les nœud connus par l'émetteur ; en vérifiant qu'il est présent dans un message HELLO reçu de son voisin, un nœud peut ainsi vérifier que le lien entre lui et le voisin est bidirectionnel.

Les messages HELLO sont aussi utilisés afin de découvrir une rupture de lien. Les non-réceptions consécutives de plusieurs messages de ce type sont interprétées ainsi. Quand une rupture de lien apparait, le nœud l'ayant découverte envoie une réponse RREP non sollicitée avec une métrique infinie et un numéro de séquence destination incrémenté à tous ses voisins qui utilisaient ce lien. Cette réponse sera alors retransmise de nœud en nœud pour informer tous les nœuds utilisant le lien dans une route. Si un nœud source reçoit une telle réponse RREP, il peut décider de lancer à nouveau une découverte de route dans l'éventualité où il a toujours du trafic à envoyer.

Un autre aspect de la maintenance est la suppression des routes inutilisées. Pour cela, chaque paquet retransmis en utilisant une route revalide la route, ce qui la garde active. Après un délai d'inactivité sans paquet, la route est supprimée.

Enfin, si une réponse RREP est reçue pour une destination déjà présente dans la table de routage, alors celle-ci est utilisée pour changer la route seulement si le numéro de séquence destination est plus grand que celui présent dans la table de routage ou, dans le cas où les deux numéros de séquence sont égaux, si la métrique de la route offerte par la réponse RREP est meilleure.

Gestion des numéros de séquence

Il n'y a pas de numéro de séquence unique pour le réseau car il serait impossible de déterminer en permanence sa valeur de manière distribuée. Chaque nœud

possède donc son propre numéro de séquence permettant de dater les informations provenant de lui et de lui seul.

Un numéro de séquence est incrémenté dans les cas suivants :
- avant de commencer une découverte de route, un nœud incrémente son numéro de séquence ;
- avant d'envoyer une réponse RREP, le nœud met à jour son numéro de séquence en utilisant le maximum du numéro de séquence actuel et de celui indiqué comme numéro de séquence destination dans la requête RREQ reçue ;
- en cas de rupture d'un lien, pour chaque route passant par le lien, le numéro de séquence associé à la destination de la route est incrémenté avant d'envoyer la réponse RREP informant de la rupture du lien.

Évaluation

Comme tout protocole réactif, AODV souffre d'un délai lors de l'envoi des premiers paquets vers une destination non connue. L'utilisation des numéros de séquence crée aussi une certaine complexité, mais a l'avantage de permettre de fortement limiter les retransmissions inutiles. Ajouté au fait que l'approche réactive du protocole ne pèse que peu sur la charge du réseau, il en résulte qu'AODV n'a que peu d'impact sur celle-ci. Les messages HELLO périodiques restent cependant nécessaires.

Une différence majeure d'AODV par rapport à DSR est le fait qu'un nœud intermédiaire sur une route peut modifier la route d'une source à une destination. C'est notamment le cas si un lien est rompu et que le nœud intermédiaire parvient à trouver une route alternative ou si une meilleure route devient disponible entre le nœud intermédiaire et la destination. On peut parler de réparation locale du lien et d'optimisation locale de la route car ces informations n'ont pas à être remontées jusqu'à la source. Cette différence fait qu'AODV est plus adapté que DSR dans le cas d'une mobilité des nœuds importante [54]. Le routage par la source de DSR reste néanmoins intéressant de par le fait qu'il permet à la source de contrôler exactement quelle route est utilisée ; cela permet notamment à chaque source de choisir une route en fonction de critères qui lui sont propres, comme une métrique particulière ou encore le choix d'éviter certains nœuds ou liens.

3.2.4 OLSR

À la différence de DSR et d'AODV, OLSR [55] (*Optimized Link State Routing*) est un protocole de routage pour les réseaux MANET proactif. Il se base sur du routage par état de lien comme décrit dans la partie 3.1.3, mais optimise celui-ci en minimisant le trafic nécessaire pour que chaque nœud connaisse la topologie du réseau grâce aux *relais multi-points* [56]. OLSR a lui aussi été formalisé sous la forme de la RFC 3626 [57].

Relai multi-points

L'objectif des relais multi-points est de limiter localement le nombre de retransmissions lors d'une inondation : chaque nœud dispose d'un ensemble de relais multi-points choisis parmi ses voisins et seuls ces relais multi-points peuvent retransmettre les paquets émis par le nœud — les paquets sont toutefois reçus par tous les voisins. Si l'ensemble des relais multi-points est plus petit que l'ensemble des voisins, il en résulte immédiatement une réduction du trafic retransmis ; en outre, plus cet ensemble est petit, plus la réduction du nombre de retransmissions est efficace.

Pour savoir s'il peut retransmettre un paquet reçu, chaque nœud doit donc maintenir la liste des nœuds qui l'ont choisi comme relai multi-points. Ces derniers sont les *sélectionneurs multi-points* du nœud.

Les relais multi-points d'un nœud sont choisis parmi ses voisins directs, et de telle sorte que l'ensemble des relais multi-points permette l'accès à tous les nœuds voisins à deux sauts. En outre, tous les voisins à deux sauts du nœud doivent avoir au moins un lien bidirectionnel avec un des relais multi-points, et le nœud doit lui aussi avoir un lien bidirectionnel avec chacun de ses relais multi-points, ceci afin d'éviter tout problème lié aux liens unidirectionnels.

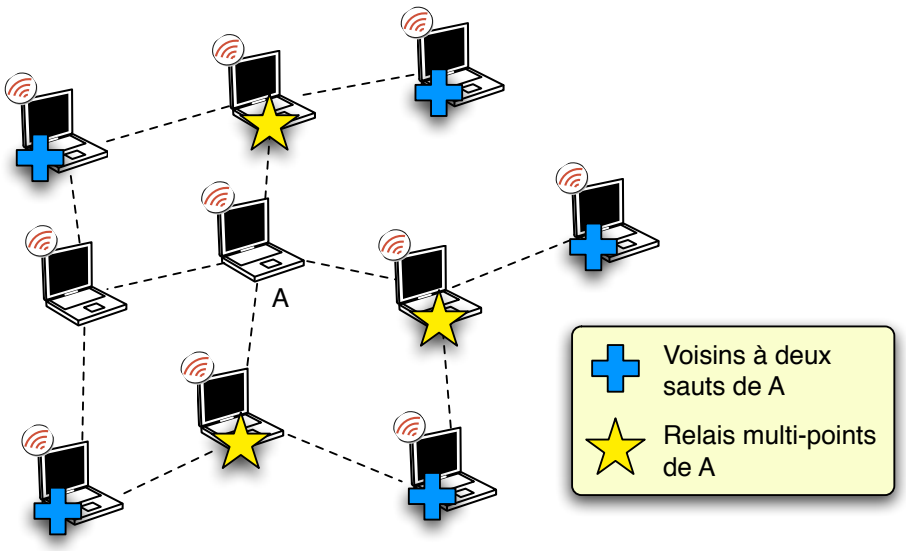

FIG. 3.5 – Relais multi-points d'un nœud

Sélection des relais multi-points

De manière périodique, chaque nœud envoie un message HELLO à tous ses voisins. Ce message contient trois listes de nœuds :

– la liste des adresses des voisins pour lesquels le nœud a reçu un paquet HELLO ;
– la liste des adresses des voisins qui sont accessibles par un lien bidirectionnel ;
– la liste des adresses des voisins que le nœud a choisis comme relais multi-points.

Lorsqu'un nœud reçoit un message HELLO d'un voisin, alors il place le voisin dans la première liste. Si en outre, son adresse est dans la première liste du message reçu, alors il considère que le lien entre lui et le voisin est bidirectionnel ; il place donc le voisin dans la seconde liste. Enfin, si son adresse est dans la troisième liste, cela signifie que le voisin est un de ses sélectionneurs multi-points ; le nœud sait ainsi qu'il doit retransmettre les paquets en diffusion de ce voisin.

Avec l'aide des messages HELLO de ses voisins, un nœud peut calculer la liste de tous ses voisins à deux sauts : il s'agit de l'ensemble des voisins de ses voisins qu'il ne connait pas directement. Il possède alors toutes les informations requises pour sélectionner un ensemble de voisins qui formeront ses relais multi-points. Lors de l'envoi du prochain message HELLO, cette liste sera transmise aux voisins et donc aux relais multi-points qui seront ainsi informés de leur rôle. L'ensemble des relai multi-points doit être recalculés lorsque l'ensemble des voisins change ainsi que lorsque l'ensemble des voisins à deux sauts change.

Le processus de sélection des relais multi-points fait que les liens entre un nœud et ses relais multi-points sont tous bidirectionnels. Cette caractéristique permet de ne pas souffrir de problèmes liés à des liens unidirectionnels dans le routage.

Calcul des routes

Chaque nœud envoie régulièrement en diffusion un message TOPOLOGY CONTROL à destination de l'ensemble du réseau afin de déclarer l'ensemble de ses sélectionneurs multi-relais. L'utilisation des relais multi-points pour l'envoi par inondation d'un tel message permet de limiter le nombre de retransmissions inutiles du message, tout en garantissant que tous les nœuds du réseau reçoivent le message.

La réception de ces messages permet à chaque nœud de construire une topologie du réseau basée sur les relais multi-points. L'algorithme de Dijkstra [43] est ensuite mis en œuvre pour trouver une route pour chaque nœud.

Une conséquence du fait que seuls les sélectionneurs multi-points sont transmis dans les messages TOPOLOGY CONTROL est que les routes sont toutes une suite de relais multi-points, de la source à la destination. Une démonstration formelle montre que ces routes formées uniquement de relais multi-points sont les plus courts chemins et sont donc optimales [55].

Évaluation

OLSR est un protocole de routage efficace car il garantit l'optimalité des routes en terme de sauts, et ne souffre pas de problème de délai pour l'émission des premiers paquets comme c'est la cas pour les protocoles réactifs. Indépendamment du protocole lui-même, la notion de relai multi-points est particulièrement intéressante car elle permet d'optimiser le mécanisme de diffusion en effectuant une inondation dans le réseau avec un impact moins important qu'une inondation normale. Néanmoins, si la sélection des relais multi-points minimise le nombre de relais multi-points, la redondance est aussi minimisée dans le réseau et une perte des paquets de diffusion peut avoir lieu dans le cas de topologies très variables [58].

Comme tout protocole proactif, OLSR nécessite des ressources plus importantes qu'un protocole réactif : la connaissance de la topologie du réseau a un coût, notamment en terme d'énergie. De plus, le calcul des routes avec l'algorithme de Dijkstra implique la nécessité d'une puissance de calcul un peu plus importante. Ces aspects rendent OLSR peu utilisables dans les réseaux de capteurs, mais ne posent pas réellement de problèmes pour des appareils un peu plus évolués.

OLSR devient plus efficace qu'un protocole réactif dans un contexte où chaque nœud communique avec un nombre important d'autres nœuds : par exemple, pour un réseau de 40 nœuds, des connexions vers 12 autres nœuds doivent être maintenus pour qu'OLSR soit efficace [59]. Le coût de la connaissance de la topologie devient alors moins important que le coût de nombreux établissements de route.

Enfin, le fait que les routes utilisées ne sont formées que par des relais multi-points est une limitation arbitraire de la diversité disponible dans le réseau. Cela peut avoir plusieurs conséquences, notamment un engorgement au niveau des liens entre des relais multi-points si la sélection globale des relais multi-points dans le réseau n'est pas assez variée. Cet aspect rend impossible l'utilisation de toute la capacité disponible dans le réseau.

3.2.5 DYMO

Le protocole DYMO [60] (*DYnamic Manet On-demand*) est une des plus récentes propositions en matière de routage pour les réseaux MANET. Il s'agit en quelque sorte du successeur et de la fusion d'AODV et de DSR. Ce protocole étant toujours en développement, il reste encore peu étudié.

DYMO est un protocole de routage réactif, se fondant sur le routage par vecteur de distance. Le fonctionnement de base pour la découverte de route et la maintenance de route est extrêmement similaire à celui d'AODV, décrit dans la partie 3.2.3, et ne sera donc pas rappelé. DYMO est prévu pour être en partie extensible, ce qui explique l'optionnalité de certaines opérations. Les principales différences avec AODV sont les suivantes :

– dans le protocole de base, uniquement la cible d'une requête RREQ peut répondre à celle-ci ;
– la gestion des numéros de séquence a été améliorée de telle sorte que les numéros de séquence sont incrémentés moins souvent. Par exemple, seul le nœud auquel appartient le numéro de séquence peut l'incrémenter ;
– la liste des nœuds parcourus par les requêtes RREQ peut être incluse dans les requêtes, comme c'est le cas pour DSR. Cette information peut être utilisée afin de connaître de manière partielle la topologie du réseau, et donc de disposer gratuitement de certaines routes comme dans un routage proactif, mais sans souffrir de l'impact des messages périodiques qu'un routage proactif peut connaître.

En outre, il a été envisagé d'utiliser un mécanisme comme les relais multi-points afin de limiter l'impact de l'inondation, ou encore de calculer un ensemble dominant connecté plus petit à partir des relais multi-points [61].

Ce nouveau protocole cherche donc à garder le meilleur des protocoles de la génération précédente, et a pour principale base l'expérience acquise durant ces dernières années. Il n'apporte néanmoins aucun changement radical dans l'approche du problème du routage dans les réseaux MANET.

3.3 De nouvelles approches au niveau 2.5

Alors que les méthodes de routage traditionnelles sont relativement figées et que beaucoup de travaux ont été réalisés sur les protocoles de routage dans le cadre des réseaux MANET, certaines propositions sont récemment apparues avec des solutions innovantes pour répondre au problème du routage. Ces approches considèrent souvent le problème du routage comme partie d'un cadre plus grand, et définissent une architecture d'interconnexion dépassant le simple protocole de routage. Un point commun de ces approches réside dans le fait qu'une partie de la solution est placée au niveau 2.5, entre le niveau 2 et le niveau 3, alors qu'auparavant tout était généralement réalisé soit au niveau 2, soit au niveau 3.

3.3.1 Motivations pour un niveau 2.5

La principale motivation pour l'utilisation d'un niveau 2.5 est la volonté de cacher la différence entre le réseau tel qu'il existe physiquement et le réseau tel qu'il devrait fonctionner : un réseau peut être considéré comme un unique réseau local alors qu'il contient des nœuds à plusieurs sauts de distance l'un de l'autre. Ainsi, tous les nœuds sont virtuellement voisins et la diffusion au niveau 3 doit atteindre tous les nœuds, alors que la diffusion au niveau 2 ne permet que d'atteindre les nœuds qu'à un saut de distance.

En outre, les difficultés qu'impliquent un réseau multi-sauts passent alors du niveau 3 au niveau 2.5. Cela simplifie l'utilisation des protocoles fonctionnant en étroite collaboration avec le niveau 3, tels que DHCP qui nécessite l'utilisation de relais lorsque le réseau est composé de sauts multiples.

Dans ce contexte, le routage et l'acheminement des paquets a lieu non pas au niveau 3, mais au niveau 2.5. Une encapsulation des paquets est généralement mise en œuvre pour l'acheminement, avec l'ajout d'un nouvel en-tête.

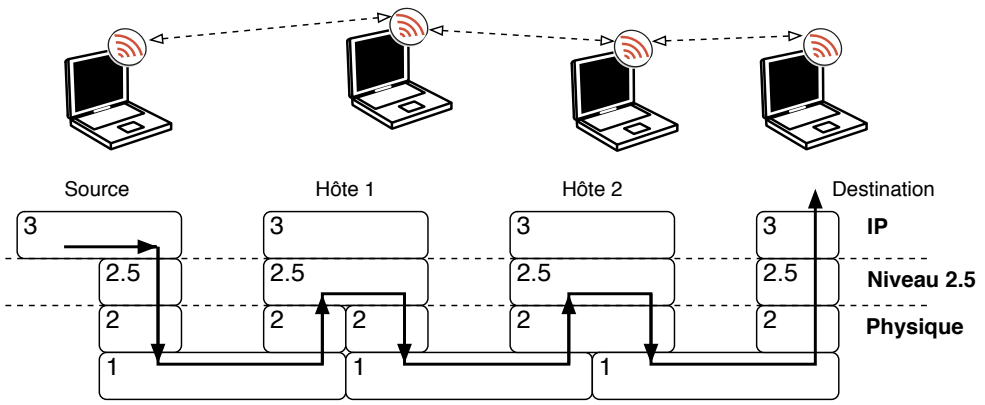

FIG. 3.6 – Utilisation d'un niveau 2.5 pour l'acheminement de données

Une autre motivation est la volonté de ne pas changer la sémantique du niveau 3. Ainsi ; les protocoles de routage réactifs MANET se placent au niveau 3 ; pourtant, leur fonctionnement va au delà du niveau 3 tel qu'il est habituellement défini. Par exemple, les paquets doivent être interceptés afin de déclencher le mécanisme de découverte de route puis réémis lorsqu'une route est disponible : cette étape, similaire au mécanisme employé pour ARP, interfère avec le fonctionnement de la pile réseau au niveau 3 car une telle interception des paquets n'est pas prévue. En outre, les paquets de données acheminés peuvent aussi avoir un sens en terme de routage : dans AODV, l'existence d'un trafic le long d'une route revalide cette route et permet de la garder active. Cette utilisation des données comme sonde attribue un rôle aux paquets de données, surchargeant la sémantique du niveau 3.

3.3.2 Rbridges

Dans le cadre d'un réseau filaire de taille importante, comme sur un campus, un ensemble de ponts et de routeurs est utilisé. Les ponts ont différents problèmes liés à l'utilisation du protocole de l'arbre couvrant [62] (*Spanning Tree Protocol*, STP) : les routes mises en œuvre sont non optimales, l'apparition d'un nouveau pont nécessite un délai avant sa prise en compte, ou encore des boucles temporaires peuvent

exister en raison des différentes optimisations de STP [63]. RSTP [64] (*Rapid Spanning Tree Protocol*) permet de réduire le délai d'activation d'un nouveau pont, mais n'est adapté qu'à des réseaux avec une profondeur maximale de sept ponts [65].

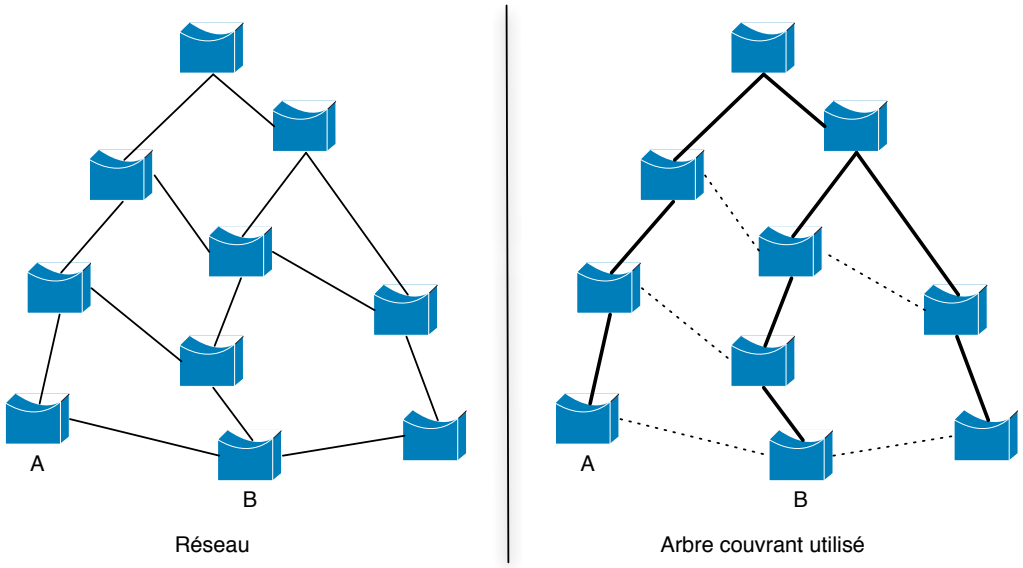

FIG. 3.7 – Non-optimalité de la route entre des stations branchées sur les ponts A et B lorsque STP est utilisé dans un réseau

Néanmoins, les ponts restent parfois préférés à des routeurs notamment en raison de leur simplicité d'utilisation et de la possibilité de disposer d'un unique réseau local. Un autre avantage non négligeable d'un réseau basé sur des ponts par rapport à un réseau basé sur des routeurs est qu'un nœud peut se déplacer dans le réseau et changer de point de connexion sans avoir à modifier son adresse.

Rbridges [63] (*Routing bridges*) est une proposition d'architecture visant à garder le meilleur des ponts et des routeurs sous la forme de ponts-routeurs. Un pont-routeur reste compatible avec les ponts et les routeurs, mais permet comme un pont de relier deux réseaux locaux comme s'ils n'en formaient qu'un, tout en garantissant comme les routeurs un chemin optimal entre deux nœuds. En outre aucune configuration particulière n'est nécessaire, et l'ajout d'un nouveau pont-routeur est rapidement pris en compte.

Fonctionnement global

Sur chaque lien peuvent se trouver plusieurs ponts-routeurs. Si c'est le cas, un pont-routeur est élu DR (*Designated Rbridge*) pour ce lien. De la même manière qu'un pont, le DR apprend quels nœuds sont présents sur lien en observant le trafic.

Tous les ponts-routeurs utilisent le protocole de routage à état de lien IS-IS afin de connaître un chemin optimal vers tous les autres ponts-routeurs. En outre, les DR transmettent dans les informations d'état de lien la liste des nœuds présents sur leur lien. Ainsi, chaque pont-routeur connait une route vers tous les nœuds. Les informations d'état de lien obtenues par ce protocole permettent en outre de calculer un arbre couvrant des ponts-routeurs. Cet arbre est utilisé pour l'envoi de trames de diffusion au niveau MAC et pour l'envoi de paquets vers une destination pour laquelle aucune route n'est connue. Ce dernier point permet de ne pas souffrir d'un délai lié à la mise à jour des bases de données d'état de lien.

Lorsqu'un nœud doit transmettre un paquet vers une destination, il effectue une requête ARP. Si la destination est sur un autre lien, le DR répond à cette requête avec sa propre adresse MAC. Le paquet sera donc envoyé au DR, qui l'encapsulera dans un paquet spécifique. Le nouveau paquet encapsulé est ensuite routé sur les différents ponts-routeurs afin de parvenir jusqu'au lien de la destination. Le dernier pont-routeur sur la route décapsule alors le paquet avant de l'émettre sur le lien de la destination. L'encapsulation permet de distinguer un paquet émis normalement par un nœud d'un paquet qui est acheminé de pont-routeur en pont-routeur jusqu'à la destination.

Évaluation

Rbridges propose donc une architecture d'interconnexion qui est plus évoluée que l'architecture généralement employée sur un réseau ; il en résulte un fonctionnement qui est plus efficace en raison de la qualité des routes choisies et de l'absence de délais. En outre, le réseau résultant peut être considéré comme un unique réseau local, ce qui permet de tirer profit de protocoles de découverte de services qui ne fonctionnent que sur un réseau local, mais aussi de faciliter l'utilisation du réseau en supprimant une partie importante de la configuration nécessaire.

La communication entre deux ponts-routeurs est particulière, notamment pour l'acheminement des paquets IP qui sont encapsulés. On assiste dans ce cas à une communication au niveau 2.5. Le fait que le réseau agrégé est, au final, considéré comme un unique réseau local par les nœuds montre aussi que la structure réelle du réseau est virtualisée à travers un niveau 2.5.

Rbridges n'est malheureusement pas applicable à tout type de réseau. L'exemple typique utilisé dans la proposition est celui du réseau filaire d'un campus. Il s'agit d'un réseau traditionnellement fixe, et sur lequel il n'y a pas d'importantes contraintes en ce qui concerne l'utilisation de la bande passante. Si on s'intéresse à des réseaux sans fil, Rbridges sera moins adapté en raison de la topologie fortement variable et de son utilisation de IS-IS. Si certains travaux [66] montrent que les délais liés aux changements de topologie dans un réseau utilisant IS-IS peuvent être réduits, les contraintes de bande passante et la faible fiabilité des liens qui

existent dans les réseaux sans fil font de IS-IS un protocole peu efficace dans ce contexte.

3.3.3 LUNAR

LUNAR [67] (*Lightweight Underlay Network Ad-hoc Routing*) propose une approche inhabituelle pour résoudre le problème du routage dans les réseaux sans fil ad hoc. Le but initial de LUNAR est d'explorer de nouvelles stratégies, et pour cela il a été préféré de garder un protocole relativement simple. Ce protocole est basé sur SelNet [68, 69], qui fournit une abstraction permettant l'acheminement au niveau 2.5. L'utilisation de SelNet permet de faire apparaître tout nœud du réseau comme un voisin, recréant ainsi la vision d'un réseau local pour les couches hautes.

Une restriction non négligeable est placée sur la taille des réseaux ad hoc considérés : à l'origine, LUNAR se limite à des réseaux de trois sauts de largeur maximum car les auteurs estiment que ce cas atteint déjà les limites des cartes 802.11 en terme de connectivité, que la qualité des informations disponibles dans un réseau multi-sauts plus grand n'est pas suffisante en raison de l'âge que le temps de transport sur toute la largeur du réseau implique, et que la découverte et la maintenance de routes ont un impact négatif trop important par rapport à l'intérêt d'un bon fonctionnement purement local. Cette restriction est cependant purement arbitraire, et LUNAR peut fonctionner sur des réseaux plus grands.

SelNet

SelNet (*Selector Network*) est une nouvelle abstraction qui se place entre la couche IP et la couche MAC. SelNet introduit la notion de *sélecteur* pour un paquet : lorsqu'un nœud reçoit un paquet marqué par un sélecteur, une opération définie par le sélecteur est effectuée. L'opération peut consister en l'envoi du paquet vers un autre nœud avec un autre sélecteur ou encore en l'émission du paquet sans sélecteur pour le réintroduire dans la pile réseau standard.

L'utilisation de SelNet nécessite que des sélecteurs soient mis en place sur les différents nœuds, et connus par les nœuds voulant communiquer entre eux à l'aide de ces sélecteurs. Des messages de type XRP (*eXtensible Resolution Protocol*) peuvent être utilisés à cette fin. SelNet en tant que tel ne définit pas comment sont utilisés ces messages, mais permet à une couche plus haute de les utiliser afin de décider de la mise en place des sélecteurs. LUNAR est un exemple d'une telle couche placée au-dessus de SelNet et utilisant les messages XRP.

Pour l'acheminement des paquets à l'aide des sélecteurs, un second type de message SAPF (*Simple Active Packet Format*) est défini. Un message SAPF est plus

spécifique que les messages XRP car il ne possède qu'un en-tête réduit au strict nécessaire pour l'acheminement, à savoir le sélecteur. SelNet traite donc directement un tel messages et effectue automatiquement l'opération associée au sélecteur que contient le message.

D'une certaine façon, SelNet est très proche de MPLS, qui sera présenté dans la partie 5.1. La principale différence est que SelNet intègre directement des messages XRP pour la communication entre les nœuds au niveau 2.5, et ne dépend donc pas d'un protocole extérieur.

Routage

LUNAR utilise un protocole de routage similaire à celui d'AODV, décrit dans la partie 3.2.3. Ainsi, des requêtes RREQ sont envoyées en inondation sur le réseau et les nœuds intermédiaires gardent les informations nécessaires pour acheminer une éventuelle réponse RREP. Les messages RREQ et RREP sont transmis à l'aide de SelNet sous la forme de messages XRP.

Une différence est l'intégration avec la pile réseau. La découverte de route est ici lancée lorsqu'une requête ARP est émise pour une destination par la couche IP. D'une certaine façon, la découverte de route se traduit donc par un mécanisme d'extension d'ARP aux multi-sauts.

Les messages RREP ont, outre le rôle d'informer de la route, celui de créer un chemin sur la route en utilisant les sélecteurs SelNet. À chaque saut, un nouveau sélecteur est créé localement et est liée à l'opération consistant à faire suivre le paquet vers le nœud suivant dans la route avec le sélecteur défini dans la réponse RREP reçue. Ce nouveau sélecteur est placé dans la réponse RREP qui est envoyée au nœud précédent dans la route. Ainsi, le nœud source pourra dès la réception de la réponse RREP utiliser ce chemin pour l'acheminement au niveau 2.5 des paquets vers la destination.

La gestion des paquets de diffusion est plus surprenante puisqu'un mécanisme différent de l'inondation est utilisé : un arbre de diffusion est en effet construit par l'émetteur. Pour cela, un mécanisme similaire à celui utilisé pour le découverte de route est utilisé :

- le nœud émetteur envoie une requête RREQ dans un message XRP ;
- chaque nœud recevant une telle requête réemet la requête RREQ s'il ne l'a pas déjà reçue et envoye une réponse après un délai ; le délai permet de mettre en place la suite de l'arbre ;
- la réception des réponses permet de connaître à quels nœuds le paquet devra être retransmis ;
- après avoir reçu les réponses de ses voisins, le nœud émetteur peut envoyer le paquet de diffusion dans l'arbre ainsi construit.

Le coût initial de construction de l'arbre est important, mais permet d'envoyer plus efficacement un nombre important de paquets de diffusion. Le nœud émetteur ne connait pas l'arbre dans son ensemble.

Afin de prendre en compte tout éventuel changement de topologie, les chemins d'acheminement construits sont rendus invalides après six secondes. Pour éviter les interruptions de connexion, chaque chemin est donc reconstruit toutes les trois secondes. Cette approche permet de se passer de toute signalisation pour la maintenance des chemins.

Autoconfiguration et intégration avec DHCP et DNS

Une autoconfiguration des adresses IP est proposée grâce à une intégration avec DHCP. Ainsi, toute requête DHCP est interceptée par LUNAR qui implémente un pseudo-serveur DHCP transformant les requêtes pour utiliser un autre mécanisme que DHCP. Si le client demande une adresse IP particulière, une résolution d'adresse avec des messages XRP est effectuée ; si cette résolution n'aboutit pas, alors le pseudo-serveur DHCP répond favorablement à la requête. Si la résolution d'adresse aboutit — donc si l'adresse est déjà utilisée —, ou si le client n'avait pas spécifié d'adresse IP particulière, une adresse IP aléatoire est choisie et testée de la même manière. Ce mécanisme est similaire à la sélection d'une adresse lien local avec Zeroconf [70].

De plus, LUNAR implémente un pseudo-serveur DNS qui intercepte les requêtes DNS [71]. Le domaine *net.lunar* est alors utilisé comme domaine pour tous les nœuds appartenant au réseau. Une requête pour un nom de domaine complètement qualifié n'appartenant pas à ce domaine sera transférée à un serveur DNS classique pour résolution si un tel serveur est disponible. Dans le cas d'une requête pour un nom de domaine non complètement qualifié ou appartenant au domaine *net.lunar*, la requête DNS est transformée en découverte de route qui utilise le nom, au lieu d'une adresse particulière, comme cible. Un nœud répondra à cette découverte de route si ce nom correspond au sien ; la réception d'une réponse pour la découverte de route permet au pseudo-serveur DNS d'envoyer une réponse DNS avec l'adresse IP obtenue.

Évaluation

L'approche proposée avec LUNAR n'est pas particulièrement performante, et le fait de reconstruire les chemins toutes les trois secondes est sous-optimal. De plus, l'utilisation d'un arbre de diffusion pour l'envoi des paquets de diffusion est discutable : il serait possible de compenser le coût important en utilisant les informations qu'offrent la connaissance de l'arbre, mais celles-ci ne sont pas mises à profit.

Cependant, l'architecture proposée est intéressante car elle sépare clairement les problèmes usuels liés au routage de la couche 3 avec les problèmes spécifiques des réseaux ad hoc en isolant ces problèmes dans une couche 2.5. En outre, la volonté de garantir le bon fonctionnement des applications de manière transparente, tout en offrant une autoconfiguration et une résolution DNS intégrées, distingue LUNAR. Le but est clairement de pouvoir utiliser un ordinateur sans efforts particuliers pour l'utilisateur, comme sur tout autre réseau. Un dernier aspect intéressant est qu'un prototype de LUNAR a été très rapidement développé afin de pouvoir tester l'ensemble dans des conditions réelles, et pas uniquement effectuer des mesures par simulation.

3.3.4 Ananas

Ananas [72, 73] est une proposition d'architecture qui a pour but de faciliter le fonctionnement d'un réseau ad hoc en créant l'illusion, pour la couche IP de chaque nœud, que tous les nœuds sont sur le même réseau local, cachant ainsi l'aspect multi-sauts du réseau et toute les difficultés que cela implique. Pour cela une interface virtuelle est créée sur chaque station, qui a pour but d'être une abstraction de toutes les interfaces physiques présentes sur le nœud ; c'est par cette interface virtuelle que tous les paquets IP passent pour être émis sur le réseau. Chaque nœud est identifé dans le réseau par l'adresse ad hoc de son interface virtuelle. Le réseau ad hoc multi-sauts est ainsi relégué au niveau de cette interface virtuelle pendant que pour la couche IP, le réseau apparait simplement comme un réseau local. L'interface virtuelle se place entre la couche IP et la couche MAC et représente logiquement une couche 2.5.

D'un point de vue architectural, il existe donc trois niveaux d'abstraction :
- le niveau physique : c'est l'ensemble des interfaces physiques du réseau pouvant communiquer entre elles ; les communications à ce niveau ont lieu avec des trames, sur un seul saut ;
- le niveau ad hoc : c'est à ce niveau qu'existe le réseau ad hoc, composé de tous les nœuds du réseau. À ce niveau, un nœud ne possède plus plusieurs interfaces physiques, mais seulement une interface virtuelle configurée avec une adresse ad hoc unique dans le réseau. S'agissant d'un réseau ad hoc habituel, les communications à ce niveau peuvent être multi-sauts ;
- le niveau IP : le réseau n'est, à ce niveau, plus qu'un réseau local dans lequel tous les nœuds sont directement accessibles à travers une unique interface : l'interface virtuelle. Une communication IP ne semble donc consister qu'en un seul saut pour le paquet.

L'insertion du niveau ad hoc comme couche 2.5 permet le fonctionnement normal d'IP sans modification, notamment pour des outils habituellement difficiles à mettre en œuvre sur un réseau ad hoc. Par exemple, dans le cas de DHCP, même si un nœud ne possède pas encore d'adresse IP, il peut recevoir la réponse grâce à son

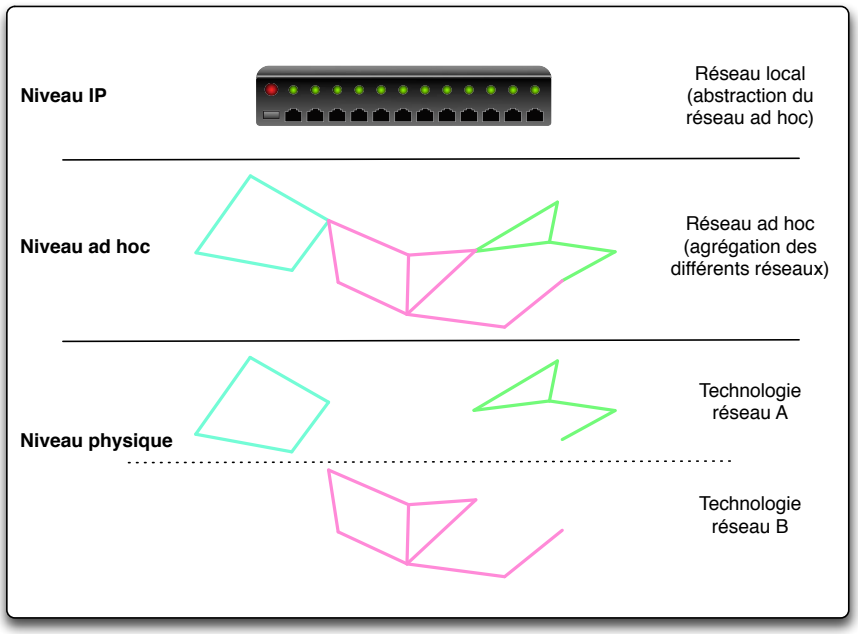

FIG. 3.8 – Niveaux d'abstraction d'Ananas

adresse ad hoc qui joue un rôle équivalent à celui de l'adresse MAC dans un réseau local[1].

Protocole de routage

Aucun protocole de routage n'est défini par Ananas car cet aspect est indépendant de l'architecture générale. Donc des protocoles tels que DSR, AODV ou OLSR peuvent être utilisés après une légère adaptation. En effet, le protocole de routage est utilisé pour faire le routage sur les adresses ad hoc, et non sur les adresses IP. Si l'on se place du point de vue de la couche IP, avec l'illusion créée par l'interface virtuelle, aucun routage n'entre en jeu car tous les nœuds sont directement accessibles sur le réseau local. Le routage a donc lieu au niveau 2.5 et c'est également à ce niveau qu'existe la table de routage.

L'acheminement des paquets étant purement basé sur les adresses ad hoc, et notamment celle de la destination, il en résulte que l'adresse IP de la destination doit être convertie en adresse ad hoc destination avant l'émission du paquet. Il est

[1]Bien que cela ne soit pas précisé dans la proposition, le bon fonctionnement de DHCP implique que l'adresse ad hoc ait le format d'une adresse MAC. En effet, un serveur DHCP émet directement une réponse vers l'adresse MAC du nœud ayant fait une requête.

donc nécessaire d'utiliser une table de traduction d'adresses qui fonctionne exactement comme une table ARP dans la pile réseau habituelle. Le remplissage de cette table est couplé avec le mécanisme de découverte de route du protocole de routage.

Évaluation

L'architecture d'Ananas est relativement originale, et l'utilisation d'une interface virtuelle présente de nombreux intérêts. Elle permet par exemple de considérer l'ensemble des nœuds d'un réseau comme membres d'un même réseau local et de garantir le bon fonctionnement de toutes les opérations TCP/IP sur ce réseau local virtuel, et incidemment de garantir la compatibilité avec tous les logiciels existant et les mécanismes d'autoconfiguration habituels. En outre, l'interface virtuelle créant une abstraction des différentes interfaces physiques, il est possible de déplacer les connexions établies d'une interface physique à une autre sans la moindre coupure ; ceci permet d'effectuer des opérations de mobilité d'une interface à une autre dans le cas d'un changement de topologie.

Néanmoins, l'architecture reste trop peu définie et beaucoup de choix sont délégués à un protocole de routage. Par exemple, aucun choix n'est effectué sur l'approche à utiliser pour le routage, la maintenance des routes ou encore la gestion des paquets de diffusion. Or, une intégration plus fine entre l'architecture et le protocole de routage aurait pu apporter de nouveaux bénéfices.

Tout comme LUNAR, Ananas a été implémenté afin de valider l'architecture proposée et quatre déploiements permettant des tests ont été effectués [73].

3.4 Conclusion

Le routage est une des fonctions de base essentielles au bon fonctionnement des réseaux. Il s'agit donc d'un sujet qui a fait l'objet de très nombreuses recherches, et seule une infime partie de ces recherches a été présentée ici.

On peut observer que les méthodes de routage traditionnelles ont aujourd'hui encore une influence importante, et ceci est particulièrement visible quand on s'intéresse plus particulièrement au problème du routage dans les réseaux MANET. Cela s'explique par le fait que ces méthodes sont bien connues pour leur qualités et la première voie qui est apparue pour résoudre le problème du routage dans les réseaux MANET a donc été d'adapter ces méthodes en les optimisant. Certaines nouvelles notions sont néanmoins apparues, comme la notion de relai multi-points.

L'émergence de propositions basées sur un niveau 2.5 est un phénomène relativement récent. Celles-ci ont pour point commun de chercher à présenter à la couche IP un réseau qui s'étend sur plusieurs liens comme un unique réseau local.

En outre, l'importance du bon fonctionnement des applications est gardée à l'esprit, et des protocoles complexes à gérer d'un point de vue routage, comme DHCP, servent souvent d'exemple pour illustrer ce point. Cependant, l'utilisation d'un niveau 2.5 n'est pas revendiquée dans le cas de Rbridges, alors qu'elle est mise en avant dans les propositions de LUNAR et d'Ananas : une certaine différence de philosophie s'exprime ici, peut-être liée au fait que ces deux architectures se placent dans le cadre des réseaux sans fil ad hoc alors que Rbridges vise un réseau filaire tel qu'on en trouve sur les campus.

LUNAR et Ananas ont un autre point commun : il s'agit du fait que les deux architectures ont été construites de manière couplée avec l'implémentation d'un prototype. On peut se demander s'il existe un lien de causalité entre le niveau 2.5 et la volonté de prototypage car cette démarche n'est plus usuelle. En effet, alors qu'historiquement, spécification et implémentation des premiers protocoles de routage étaient fortement liées, les protocoles de routage MANET ont principalement été développés à l'aide de simulations. Même si quelques implémentations de protocoles tels qu'AODV et OLSR sont apparues, celles-ci ne sont pas devenues le moteur de l'évolution de ces protocoles et n'ont eu qu'un impact très limité, les simulations restant préférées aux expérimentations. Ce manque d'ancrage dans le monde réel pour la recherche réalisée par la communauté MANET est critiqué dans [74].

Deuxième partie

Contribution

Chapitre 4

Cadre des travaux

> Are you driving with your eyes open ? Or you like using "the force" ?
>
> Axel Foley, *Beverly Hills Cop II*

Résoudre tous les problèmes qu'implique le fonctionnement d'un réseau quelconque est un projet si ambitieux qu'il est probablement impossible de le réaliser complètement. Comme tout problème complexe, il y a différents cas qui peuvent aussi bien être résolus d'une même façon qu'être totalement en opposition pour leur solution. Il est donc important de savoir délimiter le cadre des travaux à effectuer afin de choisir les critères qui importent et de pouvoir déterminer ce qui définit une bonne solution.

Ainsi, lorsqu'on s'intéresse à la problématique du routage, il s'avère nécessaire d'expliciter :

- la taille des réseaux : l'architecture ne peut être la même pour un réseau simple d'à peine 5 ou 6 nœuds et pour un réseau qui couvrirait un pays entier comme la France, voire un continent ;
- la nature des nœuds : alors que des ordinateurs portables disposent d'une batterie et peuvent être branchés sur le secteur, des capteurs disposent d'une quantité d'énergie fortement limitée. D'autres contraintes matérielles et logicielles peuvent encore varier selon la nature des nœuds ;
- le type d'utilisation visée des réseaux, qui peut varier du tout au tout.

Définir un cadre précis ne consiste pas à limiter les travaux à ce cadre et à rendre les travaux inapplicables en dehors de celui-ci, mais à favoriser des choix. Cela n'empêche en rien l'utilisation ultérieure des travaux dans un contexte autre que celui qui aura servi de référence initiale. La généralisation de l'utilisation des travaux reste toujours un objectif à atteindre par la suite.

Une parfaite illustration de ce principe d'ouverture à l'inconnu est l'architecture actuelle d'Internet [75] et notamment la pile TCP/IP[1] [76, 77]. Cet ensemble de protocoles avait été créé dans les années 1970 pour un réseau d'une taille réduite par rapport aux réseaux actuels, et testé en situation réelle sur seulement plusieurs centaines d'ordinateurs. Néanmoins, le passage à l'échelle au cours du temps n'a nécessité que des ajustements [78, 79, 80, 81, 82, 83] qui n'ont pas remis en cause l'architecture globale. C'est cette solidité et stabilité qui a permis à TCP/IP de rester omniprésent et de s'imposer comme la pile réseau de référence. Bien que les créateurs de TCP/IP eussent probablement un certain nombre d'usages comme objectif, il était impossible pour eux de prévoir tous les développements qui ont eu lieu et toutes les applications qui se sont basées sur cette pile de protocoles.

4.1 Caractéristiques des réseaux étudiés

Nous nous intéressons à des *réseaux de bordure spontanés*. Un réseau de bordure spontané est un réseau à sauts multiples de taille limitée, qui ne fait pas partie du cœur d'un réseau tel qu'Internet et dont les nœuds apparaissent ou disparaissent sans schéma pré-établi. Un tel réseau peut donc être créé spontanément avec l'apparition de nœuds, ainsi que totalement disparaitre lorsqu'il ne reste plus aucun nœud.

Un réseau de bordure spontané est proche d'un réseau ad hoc : la similitude sémantique des termes « spontané » et « ad hoc » n'est pas purement accidentelle. Néanmoins, parce que les réseaux ad hoc ont été très étudiés, certaines caractéristiques leur sont désormais automatiquement associées et créent le besoin de différencier ces deux types de réseau. La principale différence réside dans le fait qu'un réseau ad hoc est toujours, dans la littérature, un réseau sans fil et le terme « ad hoc » est ainsi fortement lié à cette caractéristique, et particulièrement à la technologie 802.11.

4.1.1 Hétérogénéité des technologies

À la différence d'un réseau ad hoc, un réseau de bordure spontané ne dépend pas que d'une technologie sans fil : il est d'ores et déjà courant d'observer différents réseaux utilisant différentes technologies interconnectés entre eux. Disposer d'un réseau 802.11 en mode infrastructure, adossé à un réseau filaire, ne constitue par exemple plus une exception. Certains utilisateurs connectent en outre leur téléphone portable ou autre périphérique à leur ordinateur grâce au Bluetooth ou à un

[1] On peut cependant remarquer que TCP/IP n'était au départ que TCP, avant une scission en deux protocoles.

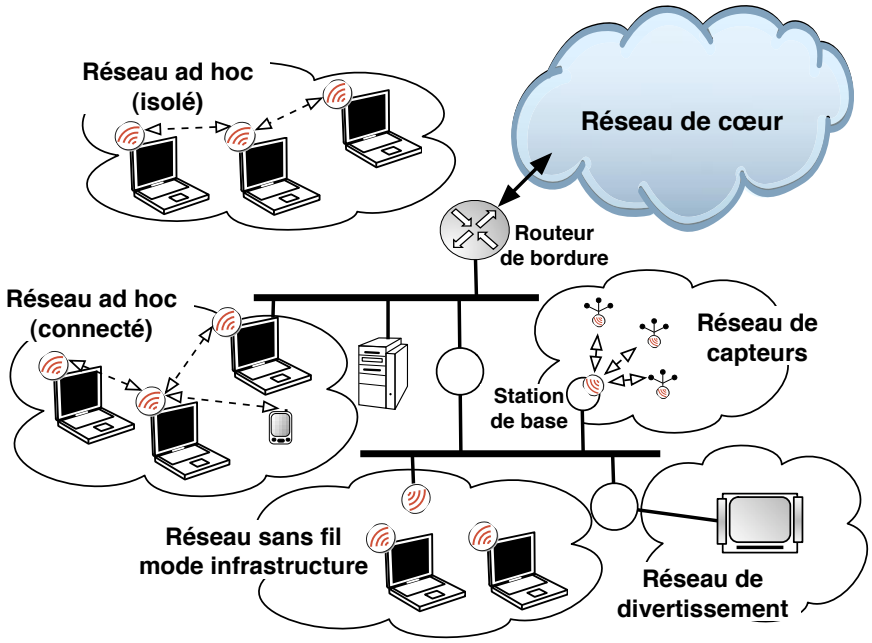

FIG. 4.1 – Réseau de bordure spontané

câble USB, et une rencontre entre différentes personnes peut pousser à la création d'un réseau 802.11 en mode ad hoc.

Malgré l'hétérogénéité de toutes les technologies en jeu, la connectivité entre tous les appareils reste primordiale. Celle-ci doit être effective et les différences entre les technologies présentes ne doivent pas limiter les possibilités. Au contraire, il est même possible de tirer profit de cette hétérogénéité : séparer deux trafics sur deux technologies distinctes pour limiter leurs interférences mutuelles peut ainsi être bénéfique.

Il est intéressant de remarquer que des parties d'un réseau de bordure spontané peuvent consister chacune en un réseau ad hoc multi-sauts. De ce point de vue, le réseau de bordure spontané est donc un sur-ensemble du réseau ad hoc.

4.1.2 Un réseau local

De manière intuitive, on a tendance à penser que tous les noeuds dans un réseau de bordure spontané peuvent vouloir communiquer entre eux. En effet, le simple fait de les considérer comme étant dans le même réseau tend à pousser dans cette direction. La caractéristique des réseaux de bordure spontané qui est exprimée ainsi est que le réseau n'est qu'un unique réseau local. Au niveau logique, il n'y a aucune différence entre les nœuds utilisant différentes technologies.

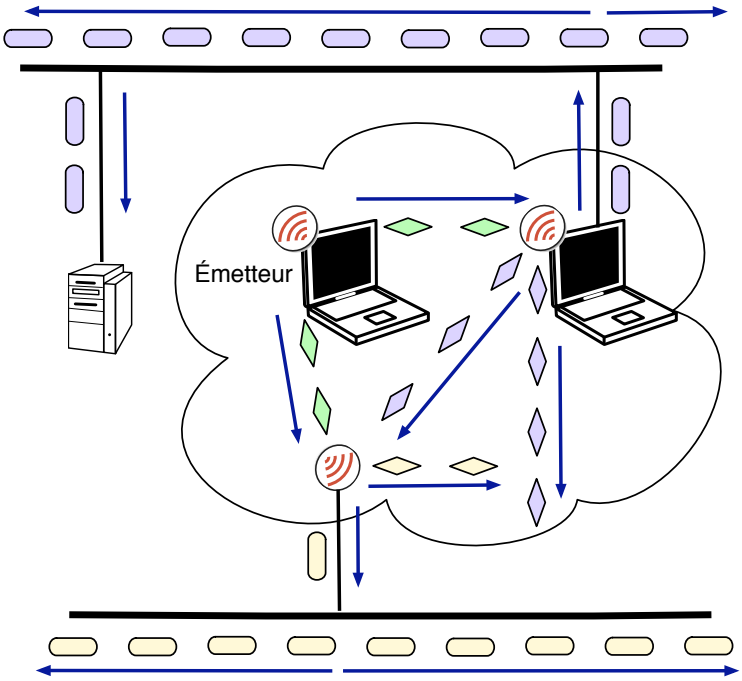

FIG. 4.2 – Émission d'un paquet en diffusion dans un réseau de bordure spontané

La réalité diffère néanmoins de cette vision logique : un paquet envoyé en diffusion au niveau de la couche réseau ne correspond pas à un paquet envoyé en diffusion au niveau de la couche MAC. Ce décalage s'explique tout d'abord par la topologie du réseau ; en effet, l'existence de plusieurs sauts entre deux nœuds rend impossible l'acheminement au niveau MAC des paquets de diffusion. Une seconde cause de ce décalage est l'utilisation simultanée de plusieurs technologies ne pouvant pas communiquer directement entre elles. Il existe ainsi un besoin de lien entre les deux couches que l'architecture devra combler. C'est ce besoin qui a amené des propositions comme Rbridges [63] et Ananas [72]

Le fait de considérer un réseau de bordure spontané comme un réseau local permet aussi de résoudre naturellement certaines problématiques déjà résolues dans ce cadre. Par exemple, il est possible d'installer un serveur DHCP qui permettra d'obtenir une adresse IP et les informations nécessaires pour la connectivité à Internet.

4.1.3 Autonomie du réseau

Une dernière caractéristique importante d'un réseau de bordure spontané est son autonomie. Celle-ci se traduit de deux façons distinctes :

– en terme d'administration : étant un réseau de bordure, le réseau n'est pas nécessairement administré par une personne avec des compétences réseau avancées, et il est ainsi préférable que le réseau fonctionne de manière autonome, sans réel besoin d'administration ;
– en terme de connectivité : il est possible qu'un réseau de bordure spontané soit déconnecté d'Internet et donc qu'il devienne un réseau autonome. Le réseau étant connecté ou déconnecté, les nœuds doivent pouvoir communiquer entre eux et le réseau doit donc fonctionner seul.

Ces deux aspects de l'autonomie se rejoignent pour illustrer le besoin d'autoconfiguration qui existe : un nouveau nœud doit pouvoir rejoindre le réseau sans configuration préalable, et les différents services offerts par d'autres nœuds doivent pouvoir être découverts. Des protocoles tels que DHCP [84], Zeroconf [70] et DNS-SD [85] doivent pouvoir être mis en œuvre sans difficulté. Cela implique notamment un support des paquets de diffusion, mais peut-être aussi le développement de protocoles supplémentaires : l'obtention d'adresses lien local par Zeroconf est rendu possible grâce à ARP [86], mais uniquement dans un réseau non multi-sauts. Or un réseau de bordure spontané peut être multi-sauts.

L'autoconfiguration se traduit aussi par la découverte automatique d'une passerelle vers l'extérieur lorsqu'elle existe. DHCP peut permettre cela, mais d'autres solutions sont à envisager lorsqu'aucun serveur DHCP n'existe.

4.1.4 Taille des réseaux

La limitation de la taille que imposée au réseau est probablement la caractéristique la plus visible de prime abord. Il s'agit d'une limitation aussi bien d'un point de vue géographique que d'un point de vue « nombre maximum de sauts entre deux nœuds ». Nous nous intéressons donc, concrètement, à un réseau à l'échelle d'un bâtiment, voire d'un petit quartier, ou encore à un réseau d'une dizaine de sauts de diamètre maximum.

Cette approche peut apparaître comme une simplification extrême du problème. Cependant, nous pensons qu'un réseau principalement sans fil de grande échelle ne peut fonctionner correctement que si l'architecture distingue le réseau global du réseau local : un simple routage proactif ou réactif ne peut pas passer à l'échelle d'un réseau contenant des dizaines de milliers de nœuds, ou bien plus encore, tout en restant un réseau plat, non hierarchisé. Certains travaux [87] ont montré qu'il est préférable de travailler sur des réseaux avec un nombre de nœuds relativement faible ; il a été aussi montré [88] que des protocoles de routage tels qu'AODV sont moins efficaces que des protocoles de routage géographique à partir de 400 nœuds. Cette limitation de la taille est donc une approche raisonnable, que LUNAR [67] a aussi choisi avec une limite de trois sauts.

Il reste donc important de distinguer le fonctionnement local du réseau du fonctionnement global, et dans ce cadre, une architecture pour un réseau de taille limitée peut participer à la résolution du problème du fonctionnement local d'un réseau de grande échelle. La définition de ce qu'est un réseau local au sein d'un réseau global n'entre pas dans nos travaux et nous admettons que cette définition en elle-même peut ne pas être triviale car il faut choisir comment découper des réseaux locaux voisins — découpage sans intersection, découpage avec une intersection minimale, ou alors découpage où la taille de l'intersection n'importe peu, par exemple.

Il faut aussi prendre en compte la densité du réseau. Pour l'existence même d'un réseau spontané, il est nécessaire d'avoir une densité minimale de nœuds : il est en effet difficile de parler de réseau lorsque les nœuds ne peuvent même pas communiquer entre eux. À l'opposé, il est facile d'imaginer que, dans le futur, de nombreux appareils disposeront d'une interface sans fil[2], et seront donc tous connectés à un réseau. Dans une telle situation, la densité des appareils sans fil sera très importante — souvent plusieurs appareils par mètre carré — et il en découlera des problèmes de congestion du réseau *a priori* plus importants que dans un réseau actuel. Nous pensons cependant qu'il faudra attendre encore plusieurs années avant qu'un réseau de bordure spontané ait une densité de plus d'un nœud par mètre carré.

4.2 Choix fondamentaux pour l'architecture

La nature même d'un réseau de bordure spontané permet de faciliter certains choix pour la conception d'une architecture adaptée à ce cadre. Il n'en reste néanmoins qu'il n'existe pas une unique approche et certains autres choix importants entrent alors en jeu pours la définition d'une telle architecture. Même s'il existe des raisons expliquant ces choix, ceux-ci restent des choix arbitraires. Ce sont cependant eux qui définissent la philosophie derrière l'architecture en résultant.

4.2.1 Nécessité de la réactivité aux évènements

En raison du caractère spontané du réseau considéré, de fortes variations de topologie peuvent avoir lieu. Concrètement, ces variations peuvent correspondre à des apparitions ou disparitions de nœuds, à une mobilité de certains nœuds, ou encore à la disparition d'une interface réseau sur un nœud pour des raisons énergétiques. D'autres variations, notamment en ce qui concernent la qualité des liens

[2]On prend assez souvent l'exemple du grille-pain ou de la machine à café pour décrire ce phénomène futur. La pertinence de cet exemple ne réside pas dans l'intérêt qu'auraient ces deux appareils à être connectés au réseau, mais plus dans le fait qu'une énorme majorité des foyers disposent de ces appareils chez eux.

ou leur charge, entrent naturellement en jeu. Pour garantir le meilleur fonctionnement possible dans ce cadre, il est primordial que toute solution proposée soit réactive pour s'adapter à un maximum de ces variations.

Pour obtenir une bonne réactivité, il est essentiel de disposer d'informations techniques décrivant la situation à un instant donné. Ces informations peuvent à la fois être purement logicielles, comme des informations sur une quantité de trafic envoyé, ou au contraire physiques, telles que des statistiques sur les collisions ayant lieu sur un lien particulier. Faire le lien entre ces deux types d'informations s'avère donc particulièrement intéressant et une approche inter-couches apparait comme désirable.

La réactivité peut être anticipée, et se transformer en proactivité afin de prévenir un problème particulier. Par exemple, lorsqu'un lien s'affaiblit progressivement sur une longue période, il peut être déduit que ce lien sera coupé dans le futur et une alternative peut être recherchée ; de manière plus générale, une connexion entre deux nœuds doit être préservée et donc une protection proactive de celle-ci devient utile. Réactivité et proactivité ne sont donc pas deux approches radicalement opposées : au contraire, il s'agit généralement d'approches similaires mais développées à un degré différent. Elles peuvent donc être parfaitement complémentaires dans une architecture.

Lorsqu'on se limite à une réactivité non proactive, celle-ci implique certains délais puisque, par définition, l'effet n'étant pas anticipé, la mise en œuvre d'une solution limitant l'effet en question a lieu après celui-ci. Il faut donc faire attention à ce que les délais ainsi impliqués restent minimes.

4.2.2 Découverte des routes réactive

Si la proactivité face aux évènements semble plus intéressante qu'une approche réactive, une approche proactive n'en est pas pour autant souhaitable dans tous les domaines. Le choix de la réactivité ou de la proactivité est ainsi un problème typique lorsqu'on considère le routage.

Or, dans le cadre des réseaux de bordure spontanés, il est très peu probable que chaque nœud communique avec tous les autres, ou du moins avec un nombre important d'autres nœuds. La surcharge impliquée par un routage proactif n'est alors pas compensée [59], et les délais induits par la découverte des routes réactives ne sont que peu subis en raison du faible nombre de découvertes de route ayant lieu. Un routage réactif est donc plus adapté dans ce contexte.

4.2.3 Mode connecté

Un des principaux objectifs choisis est de garantir le bon fonctionnement des communications existantes. Lorsqu'un nœud commence à communiquer avec un

autre nœud, il ne s'agit pas, dans la majorité des cas, d'un envoi de données isolé : ces données entrent dans le cadre d'une communication qui dure au-delà du premier paquet. Nous considérons donc la communication entre une source et une destination comme une connexion d'un nœud à un autre et l'objectif devient donc la garantie du bon fonctionnement des connexions existantes : toute connexion établie doit être maintenue afin qu'elle continue à fonctionner malgré les changements survenant dans le réseau.

Une connexion est une représentation abstraite de ce qui lie deux nœuds, de bout-en-bout. Les nœuds intermédiaires ne sont plus des entités indépendantes qui font des choix en ce qui concerne la connexion, mais simplement des étapes sur le chemin qui relie les deux nœuds et seuls ces derniers sont en position d'effectuer un choix sur leur connexion. Cette vision orientée connexion va à l'encontre de la philosophie du protocole IP, dans laquelle les parcours de deux paquets peuvent être totalement distincts l'un de l'autre.

Une connexion peut être unidirectionnelle ou bidirectionnelle. Considérer la connexion comme bidirectionnelle peut faciliter le fonctionnement de celle-ci car cela automatise le cas où deux nœuds s'échangent des données ; néanmoins, cela implique une surcharge de travail dans le cas où le trafic n'existe que dans un sens. En outre, alors qu'une connexion bidirectionnelle implique que le chemin est le même dans les deux sens, il n'en va pas de même pour deux connexions unidirectionnelles pour lesquelles il est possible d'utiliser deux chemins distincts, offrant ainsi de plus nombreuses possibilités en terme de qualité de service. Le choix de considérer une connexion comme unidirectionnelle se révèle donc plus souple pour la mise en œuvre.

Un autre choix en ce qui concerne les connexions est de savoir s'il faut considérer une connexion unique pour tout flot entre deux nœuds ou si une connexion pour chaque flot est plus adaptée. Cette seconde possibilité est encore une fois plus souple car elle permet de contrôler le comportement en fonction du flot : un flot nécessitant un débit important et un flot requérant une faible latence n'ont pas les mêmes besoins, et deux chemins distincts peuvent donc mieux répondre aux besoins des deux flots. En outre, l'utilisation de la diversité des chemins existant permet d'éviter de concentrer les flots sur certains liens et donc aussi d'éviter une congestion locale. L'ensemble du trafic ne sera donc pas agrégé dans une unique connexion.

Il est à noter que le fait de considérer une connexion comme unidirectionnelle et dépendante d'un flot en particulier est probablement la définition la plus restreinte possible pour une connexion. Il est donc théoriquement possible d'utiliser la même architecture, avec quelques simplifications, pour utiliser des connexions bidirectionnelles ou gérant tout le trafic entre deux nœuds.

4.2.4 Garantie de performances

Les performances d'un réseau restent un élement essentiel qu'il ne faut pas négliger. Elles peuvent être mesurées de différentes façons : débit, latence, temps de connexion d'une source à une destination (incluant la découverte de la route) ou encore consommation d'énergie. Une architecture qui a un fort impact négatif sur les performances n'a évidemment que très peu d'intérêt. Il n'existe malheureusemenet aucune solution miracle et on ne trouve généralement que des compromis entre les différents critères entrant en jeu.

L'architecture proposée sera orientée connexion. Cet aspect primordial entre aussi en compte pour les performances car il devient alors logique de faire en sorte que toute connexion établie reste de bonne qualité et donc que ses performances ne diminuent pas dans le temps. Une approche évaluant la qualité d'une connexion et cherchant à l'optimiser en arrière-plan est donc appropriée. La vision bout-en-bout de la connexion permet de faire abstraction du chemin utilisé et d'éventuellement en chercher un plus adapté aux nouvelles conditions.

Il reste important de garantir que l'architecture n'aura qu'un impact minimal sur le débit et la latence par rapport à un fonctionnement simple tel que qu'un acheminement IP statique. En ce qui concerne les critères liés à chaque nœud, tels que la gestion de l'énergie, différentes stratégies sont envisageables. Mais à la différence des réseaux de capteurs où tous les nœuds partagent le même objectif, dicter une stratégie à chaque nœud n'est ici pas approprié. La stratégie à mettre en œuvre est donc évaluée de manière individuelle et ne dépend pas de l'architecture.

Même s'il ne s'agit pas d'une priorité, garder la possibilité d'assurer la qualité de service et de permettre de l'ingéniérie du trafic est intéressant. Il s'agit cependant d'un travail qui dépasse le cadre de cette architecture ; par exemple, la création d'une métrique indiquant le meilleur chemin entre deux nœuds dépend fortement de l'utilisation faite du réseau, et un réseau de bordure spontané reste un cadre trop large pour déterminer de manière unique une telle métrique.

4.2.5 Utilisation sans barrière

Le fait qu'un réseau de bordure spontané soit autonome, comme vu dans la partie 4.1.3, a pour conséquence que l'architecture utilisée doit être facile à mettre en œuvre. Si l'on se place du point de vue d'un utilisateur, cela implique qu'il s'agisse, par exemple, d'un programme démarré automatiquement ou facile à démarrer. Dans le cadre d'un prototype, cette contrainte est évidemment moins forte, mais il est important de la garder à l'esprit.

Un autre aspect lié à l'absence de barrières réside dans le fait que la compatibilité avec les applications existantes doit exister. Étant donnés les déploiements

actuels, le nombre de logiciels existants et la prédominance de TCP/IP, toute solution doit prendre en compte ces protocoles, et donc la grande majorité des logiciels qui s'appuient dessus. Ignorer cela n'est en aucun cas acceptable car personne n'acceptera d'utiliser une solution rendant impossible l'utilisation des applications habituelles.

4.2.6 Création d'une nouvelle couche 2.5

Un dernier choix, qui n'est pas sans impact, est que l'architecture proposée doit s'insérer proprement dans toute la pile des protocoles d'ores et déjà existante. Ce choix est fortement lié à la nécessité de compatibilité avec les applications déployées : sans cette obligation de compatibilité, oublier la pile des protocoles actuelle aurait pu être une voie à explorer. La partie 3.3.1 explique les motivations pour une couche 2.5.

Sans couche 2.5, une approche peut consister à répondre aux différents problèmes en plaçant une solution purement dans la couche réseau ou purement dans la couche MAC. Or, certaines contraintes placent la couche réseau trop haute pour cette architecture : la vue d'un réseau de bordure spontané comme un réseau local implique théoriquement que, du point de vue de la couche réseau, tous les nœuds sont voisins. De même, le besoin d'un routage IP multi-sauts place la couche MAC trop basse : celle-ci n'a pas pour rôle de gérer un acheminement sur des sauts multiples puisque c'est le rôle de la couche réseau. De fait, toute solution ayant pour objectif de virtualiser un réseau multi-sauts sous forme de réseau local sera donc nécessairement en partie sous la couche réseau et au dessus de la couche MAC.

Il resterait possible d'étendre le rôle d'une de ces deux couches comme base d'une solution — en particulier la couche réseau, grâce à la flexibilité d'IP —, mais une séparation reste préférable afin de ne pas créer de confusion entre les différents rôles des couches. En outre, en ce qui concerne l'implémentation en vue d'un prototype, l'insertion d'une nouvelle couche est une approche naturelle car elle permet de s'appuyer sans modification sur la plate-forme existante. Ainsi, alors que les principales propositions dans le contexte des réseaux ad hoc se situent à la couche réseau, une architecture au niveau 2.5 prend ici tout son sens. Ce niveau 2.5 a pour rôle l'acheminement des données dans un réseau multi-sauts formant un réseau local.

4.3 Conclusion

Dans le cadre des nos travaux, nous nous restreignons volontairement au cas des réseaux de bordure spontanés. Ces réseaux ont des caractéristiques particulières en termes de taille, de fonctionnement ou encore d'utilisation.

L'étude de ces caractéristiques, à laquelle s'ajoute une certaine philosophie, résulte en un ensemble de règles à respecter et de choix effectués pour l'élaboration d'une solution au problème du routage dans les réseaux de bordure spontanés, ainsi qu'en des critères d'évaluation. Néanmoins, nous avons toujours gardé à l'esprit l'importance de ne pas nous attacher à résoudre une liste de cas précis et de garder les travaux suffisamment génériques pour pouvoir éventuellement étendre les résultats en dehors du cadré étudié.

Chapitre 5

Architecture pour les réseaux de bordure spontanés

> I suppose you think that was terribly clever.
>
> Gandalf, *The Fellowship of the Ring*

Le cadre défini dans le chapitre précédent a permis de créer une liste de caractéristiques que la solution à développer doit respecter : il s'agit en quelque sorte d'un cahier des charges pour une architecture pour les réseaux de bordure spontanés.

Nous proposons donc une telle architecture qui se base sur MPLS pour l'acheminement des données et sur un protocole de routage réactif. Le protocole de routage réactif permet de créer à la demande des chemins de manière unidirectionelle d'une source vers une destination. Ces chemins seront ensuite utilisés *via* le protocole MPLS pour acheminer les données, mais seront aussi maintenus pour garantir autant que possible le bon acheminement des données futures malgré les changements qui peuvent intervenir. Ainsi, un changement mineur de topologie n'aura pas d'impact sur les trafics en cours, et il en va de même si de nouveaux trafics apparaissent et créent une congestion.

Grâce au couplage du protocole de routage réactif et de MPLS et grâce à la maintenance des connexions, l'architecture dispose de fonctionnalités qui correspondent au cahier des charges qui a été défini.

5.1 MPLS

Le protocole MPLS (commutation multiprotocole par étiquettes, *MultiProtocol Label Switching*) est un protocole standardisé par l'IETF dans la RFC 3031 [89], et d'autres RFC annexes [90, 91, 92, 93].

MPLS a été développé à l'origine afin d'optimiser les performances des routeurs. En effet, pour router un paquet IP, un routeur cherche à quel sous-réseau existant dans sa table de routage l'adresse IP de destination du paquet correspond. Pour cela, il faut effectuer, pour chaque sous-réseau existant dans la table de routage, une opération binaire `et` entre l'adresse destination et le masque de sous-réseau et comparer le résultat au sous-réseau en question. Ce traitement, même s'il parait relativement simple, s'avèrait coûteux lorsqu'il y a un trafic important et une table de routage de grande taille. Le fonctionnement beaucoup plus simple de MPLS, créé à cet effet, avait donc en partie pour but de décharger les routeurs. Les progrès qui ont été effectués sur les matériels de routage ont cependant rendu caduc cet aspect de MPLS.

Il n'en reste pas moins que MPLS permet une ingénierie du trafic, aspect qui reste utile pour optimiser les performances ou pour séparer des trafics coexistant dans un même réseau — afin de créer des VPN [94, 95] par exemple.

5.1.1 Fonctionnement de MPLS

Le protocole MPLS se base, comme son nom l'indique, sur la commutation par étiquettes pour la transmission des paquets. Une étiquette permet d'identifier une classe d'équivalence de transmission (*forwarding equivalence class*, FEC), et pour chaque FEC donnée existe un chemin commuté par étiquettes (*label switched path*, LSP), utilisé pour transmettre tous les paquets appartenant à la FEC. On parle aussi de tunnel MPLS pour un LSP. Bien que MPLS est généralement utilisé pour transporter des paquets IP, son comportement n'est absolument pas spécifique à IP et d'autres protocoles peuvent être transportés.

Une FEC représente donc un ensemble de paquets qui sont transmis de la même manière. Tous les paquets d'un même flot appartiennent généralement à la même FEC, mais un des intérêts de MPLS est qu'il aussi possible d'agréger dans une même FEC différents flots respectant des critères autres que l'appartenance à un flot, tels que le type de trafic, la destination du trafic ou encore des contraintes de qualité de service devant être vérifiées. Tous les paquets appartenant à une FEC sont traités de manière identique, indépendamment de leur contenu, de leur source ou de leur destination : seule l'étiquette associée à la FEC est utilisée pour la transmission du paquet.

Un LSP est formé par un ensemble de routeurs MPLS. Il n'y a pas de connaissance globale du LSP : chaque routeur détermine le routeur suivant en fonction de l'étiquette du paquet reçu. La façon dont un LSP est établi n'est pas défini par MPLS en tant que tel. En général, un protocole de signalisation tel que LDP [96], CR-LDP [97] ou RSVP-TE [98] est utilisé pour construire les chemins. Des critères de qualité de service entrent alors en jeu pour le choix des différents routeurs formant le chemin.

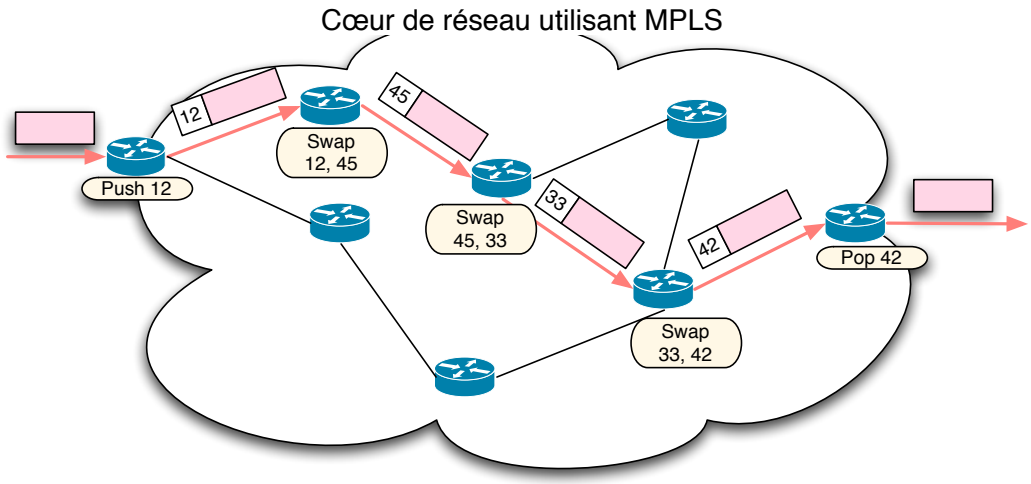

FIG. 5.1 – Exemple d'utilisation de MPLS : acheminement d'un paquet appartenant à une FEC

Un paquet devant être transmis dans un réseau MPLS entre dans un LSP par un routeur d'entrée (*label edge router*, LER). Ce routeur analyse le paquet pour déterminer la FEC à laquelle il appartient et trouver l'étiquette correspondante. Le routeur encapsule alors le paquet avec un en-tête MPLS contenant l'étiquette et envoie le paquet au routeur suivant. Le routeur suivant détermine l'action à effectuer en fonction de l'étiquette. Cette opération est extrêmement rapide car il s'agit simplement d'aller chercher l'information dans une table indexée par les étiquettes. Trois actions sont disponibles :

- encapsulation : le paquet MPLS est encapsulé avec un en-tête MPLS supplémentaire. On obtient ainsi une pile d'encapsulation MPLS, permettant de faire du routage hiérarchique ;
- décapsulation : l'en-tête MPLS est supprimé et le paquet sort du LSP. En fonction de la configuration du LSP, le paquet peut être transmis à un dernier routeur ou être traité en fonction de son contenu. Dans ce dernier cas, s'il s'agissait d'un paquet IP, l'opération de routage IP habituelle est effectuée. Dans le cas d'une pile d'encapsulation MPLS, une nouvelle étiquette est obtenue et celle-ci est donc utilisée pour traiter le paquet décapsulé ;
- changement d'étiquette : c'est l'action servant à transmettre le paquet MPLS au routeur suivant dans le LSP. Comme il n'est pas possible de garantir qu'une même valeur d'étiquette est disponible pour un LSP sur chaque routeur du LSP, la FEC est identifiée par une étiquette propre sur chaque routeur du LSP. Il est donc nécessaire de changer l'étiquette du paquet MPLS avec la valeur attendue par le routeur suivant. Après ce changement d'étiquette, le paquet MPLS est transmis au routeur suivant.

On peut donc noter que la table utilisée pour le traitement des étiquettes doit contenir l'action à effectuer, ainsi que, dans le cas d'un changement d'étiquette, l'adresse du routeur suivant et la nouvelle étiquette à utiliser.

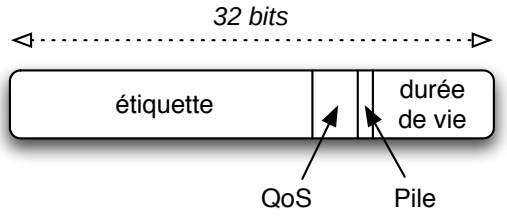

FIG. 5.2 – En-tête MPLS

L'en-tête MPLS est composé de 32 bits répartis en quatre champs : l'étiquette sur 20 bits, un champ *QoS* sur 3 bits indiquant la classe de services, un bit indiquant si un autre en-tête MPLS est empilé sous cet en-tête et un champ de durée de vie sur 8 bits afin de réduire les problèmes liés à l'existence d'une boucle.

5.1.2 Choix de MPLS pour l'architecture

La décision de ne pas installer toute l'architecture au niveau 3 afin de ne pas complexifier encore cette dernière a été expliqué dans la partie 4.2.6. L'utilisation d'une couche inférieure s'est donc avérée nécessaire. Il était possible de développer un protocole simple correspondant exactement aux besoins de l'architecture, mais le choix de MPLS a rapidement été fait.

Tout d'abord, MPLS est un protocole qui existe d'ores et déjà, qui est clairement défini et même standardisé. En outre, le fonctionnement basique de MPLS est extrêmement simple tout en répondant aux besoins de l'architecture proposée : avec un en-tête de taille réduite, MPLS fonctionne sous la couche IP, et plus généralement au niveau 2.5, permettant donc d'encapsuler de manière transparente les paquets. La création d'un nouveau protocole similaire serait une duplication de travail sans intérêt majeur, d'autant que le résultat serait relativement proche de MPLS.

La notion de FEC est aussi un des aspects de MPLS qui rend ce protocole intéressant. L'architecture proposée a pour but de permettre un traitement différent selon le trafic comme par exemple l'isolation d'un trafic particulier ou l'utilisation de chemins distincts pour des trafics correspondant à des services différents, même s'ils proviennent de la même source. Associer à chaque trafic une FEC rend cette différenciation des trafics aisée, et permet éventuellement de bénéficier de toute l'expérience en ingénierie du trafic qui existe pour MPLS.

Enfin, des implémentations de MPLS existent et le protocole est même déployé dans certains matériels. Cet aspect est important dans la mesure où une grande

partie du travail de cette thèse a consisté à implémenter l'architecture afin de la valider. Néanmoins, comme il le sera montré dans la partie 6.2.1, il s'est avéré que les implémentations n'étaient pas exemptes de défauts.

L'utilisation de MPLS ne pouvant engendrer que des bénéfices, certaines conséquences de cette utilisation peuvent poser problème. Le problème majeur réside dans le fait qu'il n'y a aucune information dans l'en-tête MPLS à propos des données qui sont encapsulées. En ne connaissant qu'un simple paquet, il est donc impossible de savoir si ce paquet est un paquet IP ou s'il s'agit d'un autre protocole qui a été encapsulé. Il faut donc gérer et garder cette information lors de l'établissement d'un LSP.

5.2 Présentation globale de l'architecture

Le principal rôle d'une architecture pour les réseaux de bordure spontanés est de permettre la connectivité entre les différents nœuds de ces réseaux. L'architecture est donc fortement liée au routage, mais ne se limite pas à cet aspect. Les choix expliqués dans les parties 4.2.3 et 4.2.4 d'avoir une approche orientée connexion et de garantir les performances créent un second aspect important consistant en une optimisation continue des ressources mises en jeu.

5.2.1 Définitions

Il convient dès à présent de distinguer différents termes :

- *route* : une route est une suite de liens permettant d'aller d'une source à une destination ;
- *chemin* : un chemin est une route particulière, qui a été choisie parce qu'elle correspond à certains critères en terme de qualité ;
- *tunnel* : un tunnel est l'instanciation d'un chemin à l'aide d'un tunnel MPLS. Il s'agit donc de ce qui permet d'utiliser un chemin ;

Le choix d'un mode connecté avec une connexion correspondant à chaque flot a pour conséquence le fait que l'acheminement de données d'un unique flot allant d'une source vers une destination a lieu sur un chemin. Deux flots distincts d'une même source vers une même destination peuvent utiliser soit le même chemin, soit deux chemins différents. S'il s'agit du même chemin, les connexions correspondant à chaque flot restent néanmoins différentes l'une de l'autre, et il en résulte que deux tunnels sont utilisés. Un chemin peut donc avoir plusieurs instances sous la forme de tunnels distincts.

5.2.2 Description de l'architecture

Pour que le fonctionnement d'un réseau de bordure spontané soit optimal, chaque nœud du réseau se doit de contribuer dans la mesure du possible. L'acheminement des données peut donc passer par tout nœud, et ainsi tout nœud est un routeur potentiel. Le réseau de bordure spontané ayant une topologie variable dans le temps de manière similaire à un réseau MANET, un algorithme de routage de type MANET est utilisé. Comme expliqué dans la partie 4.2.2, l'architecture utilise un algorithme de routage réactif similaire à DSR et AODV, décrit plus en détail dans la partie 5.3. Plusieurs routes sont découvertes et c'est la source qui choisit le chemin à utiliser.

À la différence des réseaux MANET, l'algorithme de routage est mis en œuvre pour chaque nouveau flot, même si le flot a pour destination un nœud déjà destination d'un autre flot. Une conséquence gratuite de cette approche est que l'architecture tire profit des chemins multiples entre deux nœuds car elle permet de les utiliser. Une fois un chemin choisi, la source instancie le chemin par la création d'un tunnel et c'est dans ce tunnel que les données du flot sont émises. Une optimisation envisageable consisterait à utiliser temporairement le tunnel d'un autre flot, jusqu'à ce que l'algorithme de routage retourne un chemin et que celui-ci soit instancié. Le délai lié à la découverte de routes et l'instanciation de chemin serait ainsi évité. Cette optimisation à néanmoins des effets de bord sur les mécanismes de maintenance et d'optimisation présentés ci-après et il reste donc préférable de ne pas la mettre en œuvre.

Une difficulté à ne pas négliger pour les chemins multiples reste la mise en place du processus décisionnel pour le choix de chemins distincts : l'architecture permet en effet de distinguer facilement les flots, mais le choix d'utiliser toujours des chemins distincts pour deux flots n'est pas nécessairement judicieux. Une approche inter-couches permettant d'évaluer les besoins d'un flot en terme de bande passante ou de latence donnerait des éléments de choix, mais malheureusement la pile des protocoles actuelles fait qu'une telle approche n'est pas envisageable. Il reste possible de deviner la nature du trafic en fonction, par exemple, des ports utilisés ou en l'analysant, mais ceci est clairement sous-optimal.

Les connexions sont de bout-en-bout entre deux nœuds et donc chaque routeur intermédiaire n'a pas de choix de routage à effectuer, et doit n'effectuer que l'acheminement des données. Il s'agit ainsi d'un routage par la source, mais l'utilisation de tunnels fait qu'il n'y pas la même surcharge qu'on peut trouver dans DSR avec l'ensemble du chemin transmis dans chaque paquet. En outre, les nœuds n'effectuant qu'un simple acheminement, les données transmises ne remontent pas jusqu'à la couche réseau dans la pile des nœuds intermédiaires comme l'illustre la figure 5.3 : une couche 2.5 est ainsi mise en œuvre.

La garantie du bon fonctionnement d'une connexion étant primordiale, la notion de chemin de secours est introduite. Un *chemin de secours* est un second chemin

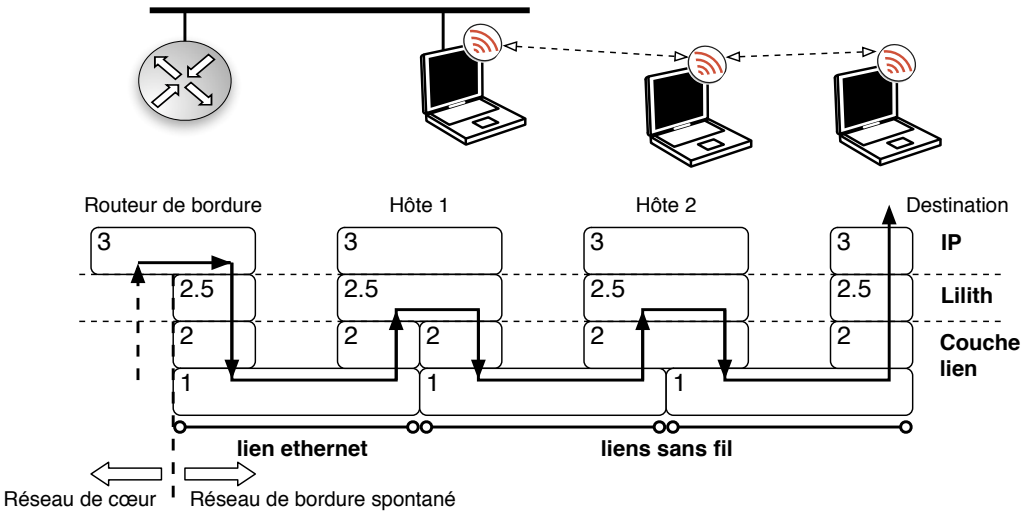

FIG. 5.3 – Pile des protocoles utilisés le long d'un chemin

instancié pour un flot donné, sur un chemin distinct du premier. Le chemin de secours permet de limiter les délais qu'impose l'algorithme de routage réactif lorsque le chemin principal devient non-fonctionnel à cause d'un changement de topologie ou parce qu'un lien entre deux nœuds intermédiaires est surchargé. Ce chemin de secours ne nécessite aucune surcharge pour être obtenu, en dehors de son instanciation : en effet, l'algorithme de routage permet de découvrir plusieurs routes à la fois, et donc outre le chemin utilisé, un chemin de secours peut être sélectionné par la source. L'utilisation des tunnels permet de passer du chemin principal au chemin de secours immédiatement et sans aucune perte puisque les nœuds intermédiaires ne font que continuer l'acheminement des données dans un tunnel donné et n'ont pas à être prévenus, et parce que le changement de chemin à utiliser au niveau de la source est une opération atomique. Une autre conséquence liée aux chemins de secours est que les tempêtes de paquets de diffusion [99] (*broadcast storms*) sont réduites lors d'un brusque changement de topologie, puisque le changement de topologie peut n'impacter que le chemin principal d'un flot, et non le chemin de secours.

Différentes informations sont collectées localement par chaque nœud intermédiaire pour chaque tunnel. Des exemples de telles informations peuvent être le nombre de paquets reçus et retransmis, ou la quantité des données reçues et retransmises pour un tunnel. Ces informations sont envoyées aux seuls nœuds voisins, ce qui permet d'avoir une vision locale du réseau et de l'état des tunnels. Ainsi, pour un tunnel donné qui le concerne, un nœud intermédiaire peut connaître la quantité de données qu'il a transmises au nœud suivant dans le tunnel et la quantité de données que ce dernier a reçues. Un écart significatif permet d'établir que le lien entre les deux nœuds est de mauvaise qualité, voire qu'il a été en partie rompu.

Cette information peut alors être remontée jusqu'à la source qui peut décider d'utiliser le chemin de secours.

Des approches inter-couches permettraient aussi de collecter des informations sur les liens utilisés par le tunnel pour entrer et sortir d'un nœud. Concrètement, le nombre de collisions ou certaines valeurs de paramètres de la couche MAC permettraient de savoir que le lien est particulièrement chargé. Ce type d'informations est pertinent pour la source qui pourrait décider d'utiliser un autre chemin. Malheureusement, à l'heure actuelle et avec le matériel actuel, de telles approches inter-couches ne sont pas réalisables car les cartes réseau ne permettent pas de transmettre ces informations aux couches supérieures.

Une autre application des informations collectées est la destruction automatique des chemins instanciés lorsque ceux-ci deviennent inutilisés. À la différence de simples routes, un chemin instancié nécessite des ressources sur chaque nœud intermédiaire, comme par exemple les informations nécessaires au fonctionnement du tunnel. Si la quantité de données émises ne change plus sur une longue période, alors cela signifie que le flot est terminé et que le chemin instancié peut être détruit. Cela permet d'éviter une signalisation supplémentaire pour la destruction des chemins. Un chemin de secours étant par définition instancié mais sans trafic, il convient alors d'envoyer régulièrement un paquet dans le tunnel correspondant afin qu'il ne soit pas détruit automatiquement. Le maintien du chemin de secours est ainsi réalisé de manière proactive, tandis que la destruction automatique des tunnels se rapproche d'un fonctionnement réactif.

Un dernier aspect non négligeable de l'architecture est l'optimisation continue en arrière-plan des connexions existantes. Le chemin principal — et il en va de même pour le chemin de secours — peut continuer à fonctionner tout en devenant moins intéressant à cause de l'évolution dans le temps du réseau. D'autres routes peuvent alors devenir de meilleurs choix comme chemins, mais le chemin instancié continuant de fonctionner, aucun élément ne peut déclencher directement la découverte de meilleures routes. Un mécanisme de recherche de nouvelles routes efficaces tourne donc en arrière-plan pour chaque flot. De manière périodique, une version modifiée de l'algorithme de routage est utilisée ; les routes sont ainsi réévaluées, la route correspondant au chemin instancié y comprise, et il est ainsi possible pour la source de déterminer si de meilleures options existent. Le cas échéant, elle peut décider d'utiliser les chemins trouvés comme chemin principal ou comme chemin de secours. Ce mécanisme permet de garantir un fonctionnement de bonne qualité sur une longue période pour une connexion.

5.2.3 Fonctionnement interne de l'architecture

En termes de fonctionnement interne, l'architecture proposée peut être séparée en différents modules illustrés dans la figure 5.4.

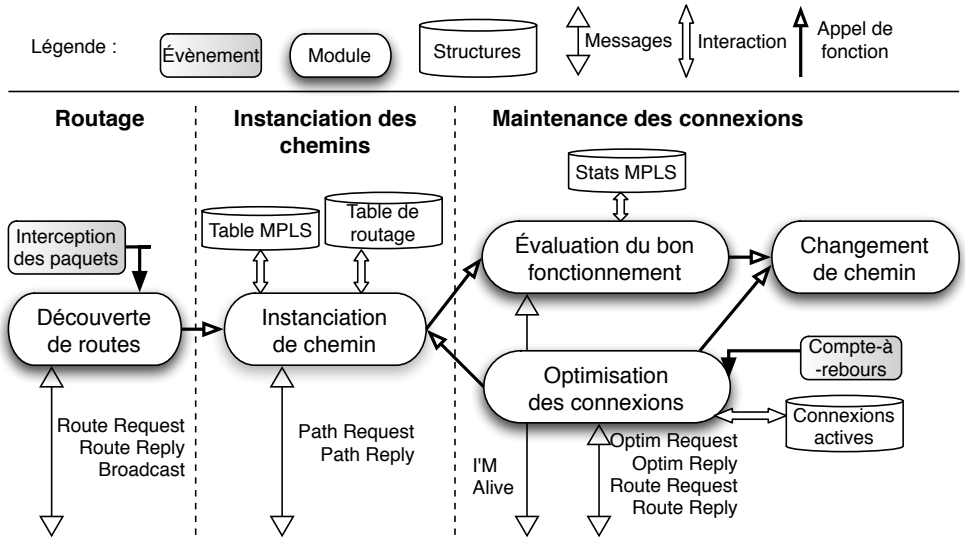

FIG. 5.4 – Modules de l'architecture

Il est intéressant de noter que l'utilisation de MPLS est transparente dans cette architecture : MPLS n'est pas utilisé parce que le protocole MPLS apporte un élément fondamental, mais parce que sa sémantique correspond parfaitement à la notion d'instanciation de chemin de l'architecture. En outre, la place de MPLS entre la couche MAC et la couche réseau permet d'insérer au niveau 2.5 tout l'acheminement dans le réseau de bordure spontané, et répond ainsi au besoin formulé dans la partie 4.2.6.

Une conséquence de l'utilisation de MPLS à prendre en compte est l'utilisation de l'en-tête MPLS. L'ajout de cet en-tête supplémentaire à chaque paquet est une surcharge qu'il faut mesurer pour savoir si son impact peut être négligé. L'en-tête MPLS pèse 32 bits, ce qui reste peu : même avec un protocole créé sur mesure, il aurait été difficile d'utiliser moins de données pour jouer le rôle demandé à MPLS.

Les sections suivantes décrivent plus en détail le fonctionnement des modules.

5.3 Routage

Le module de routage a pour objectif d'obtenir des routes pour tout nouveau flot. Deux routes seront sélectionnées comme chemin et chemin de secours, puis instanciées pour permettre l'acheminement des données du flot en question.

Un premier sous-module est chargé de l'interception des paquets ; celui-ci indique à un second sous-module de découverte de routes de démarrer le processus

de découverte d'une route vers la destination des paquets lorsque cela s'avère nécessaire. Un dernier sous-module gère les paquets de diffusion.

5.3.1 Interception des paquets

Tous les paquets émis sur une interface réseau passent par le sous-module d'interception des paquets. Le sous-module détermine alors la marche à suivre en fonction du paquet :

- si le paquet est un paquet d'unidiffusion (*unicast*) correspondant à une connexion existante — c'est-à-dire pour lequel un chemin existe —, alors le paquet est ignoré. Celui-ci continue de descendre dans la pile réseau du système d'exploitation jusqu'à la couche MPLS, qui se chargera de commencer son acheminement jusqu'à la destination ;
- si le paquet est un paquet d'unidiffusion ne correspondant à aucune connexion existante, alors le paquet est mis en tampon en attendant l'instanciation d'un chemin pour la connexion à laquelle il appartient. Le sous-module de découverte de routes est contacté afin qu'il lance une découverte de routes pour cette destination. Si aucune route n'est découverte, alors le paquet n'est pas émis. Dans le cas contraire, lorsque la connexion est établie, le module d'instanciation des chemins signalera cet évènement et les paquets mis en tampon pour la connexion seront réinsérés dans la pile réseau ;
- si le paquet est un paquet de diffusion, alors le paquet est transmis au sous-module de gestion des paquets de diffusion.

5.3.2 Découverte de routes

Le sous-module de découverte de routes a pour tâche d'obtenir un chemin pour une route d'une source à une destination. Il implémente un protocole de routage qui peut être réactif ou proactif. Bien que nous pensons que l'approche réactive est la plus adaptée, il nous a semblé intéressant de ne pas empêcher un protocole de routage proactif qui pourrait être bénéfique dans des cas particuliers, comme le cas où le nombre de nœuds est très limité par exemple. L'utilisation d'un protocole proactif nécessiterait néanmoins quelques ajustements.

Un protocole de routage réactif simple, similaire à DSR et AODV, est utilisé. Ce protocole se base sur deux messages : ROUTE REQUEST et ROUTE REPLY.

ROUTE REQUEST

Le message ROUTE REQUEST est le message émis par la source en diffusion par inondation pour trouver des routes vers la destination. Plusieurs champs le composent :

- un identifiant : il s'agit d'une valeur qui identifie de manière unique pour une source donnée la requête émise ;
- un champ de durée de vie : le champ de durée de vie permet de limiter la diffusion du message à un nombre de sauts. Chaque nœud recevant le message décrémente la valeur de ce champ et ne peut le retransmettre que si la nouvelle valeur est non-nulle. La valeur initiale correspond donc au nombre de sauts maximal que la requête effectuera ;
- un compteur de métrique : ce champ permet de calculer une métrique mesurant la valeur de la route parcourue par le message. La métrique est modifiée lors de chaque retransmission pour prendre en compte le lien sur lequel le message sera retransmis ;
- une liste des nœuds parcourus : à la réception d'un message ROUTE REQUEST, le nœud vérifie qu'il n'est pas déjà dans cette liste. En effet, si c'est le cas, le message est entré dans une boucle et il est donc inutile de le retransmettre. Si ce n'est pas le cas, le message sera retransmis. Avant de retransmettre celui-ci, le nœud s'ajoute à cette liste. Elle permet aussi d'enregistrer la route parcourue dans le message même, ce qui est une information nécessaire pour l'acheminement du message ROUTE REPLY.

Bien que le message utilise une inondation pour atteindre tous les nœuds du réseau, il n'est pas souhaitable d'utiliser un mécanisme d'optimisation de l'inondation comme décrit dans la partie 3.1.1 afin d'éviter de multiples retransmissions par un même nœud. En effet, plusieurs routes distinctes peuvent passer par un même nœud et il peut être intéressant pour la source de disposer de toutes ces routes. Néanmoins, un nœud peut garder un cache des identifiants de messages ROUTE REQUEST reçus. Dans ce cas, il ne retransmet les messages correspondant à un identifiant que s'ils sont reçus dans un intervalle de temps court après le premier message reçu avec cet identifiant ou si la métrique contenue dans le message est aussi bonne ou meilleure que celle enregistrée dans le cache.

ROUTE REPLY

Lorsque la cible d'un message ROUTE REQUEST reçoit un tel message, celle-ci envoie à la source du message une réponse ROUTE REQUEST. Ce message contient des champs similaires aux champs de la requête reçue :

- un identifiant : la valeur de l'identifiant contenu dans la requête ROUTE REQUEST est utilisée ;
- une métrique : la valeur du compteur de métrique contenu dans la requête ROUTE REQUEST est utilisée ;
- une liste de nœuds : la valeur de la liste des nœuds parcourus contenue dans la requête ROUTE REQUEST est utilisée. Elle représente aussi la route par laquelle la réponse doit être acheminée jusqu'à la source de la requête.

Chaque nœud sur la route correspondant à la liste de nœuds incluse dans la réponse transmet la réponse au nœud précédent dans cette même liste, jusqu'à la source de la requête.

Fonctionnement

Lors d'une découverte de routes pour une cible donnée, la source émet tout d'abord un message ROUTE REQUEST avec une durée de vie courte. Si aucune réponse n'est obtenue après expiration d'un délai, d'autres messages ROUTE REQUEST sont envoyés avec une durée de vie allant toujours croissante. Cette approche, dite de la recherche en anneaux croissants (*expanded ring search*) et basée sur une modification d'AODV [100], permet de limiter les requêtes à un niveau local et de ne pas encombrer l'ensemble du réseau lorsque la source et la cible sont proches. La valeur du délai d'attente avant retransmission de la requête avec une durée de vie plus importante est proportionnelle à la durée de vie de la requête. La taille des réseaux considérés étant limitée, comme décrit dans la partie 4.1.4, seul un nombre faible de retransmissions avec des durées de vie allant croissantes s'avère nécessaire. Si le nombre de retransmissions dépend évidemment du réseau, de trois à cinq palliers peuvent typiquement être utilisés afin de couvrir progressivement tout le réseau : ce nombre de palliers correspond à un compromis raisonnable entre le délai de la découverte de route et la charge que ce processus impose au réseau.

En cas d'absence définitive de réponse, la source supprimera l'entrée correspondant à la cible dans la liste des destinations en attente de chemin.

Lorsque la cible reçoit le message ROUTE REQUEST, elle transmet une réponse ROUTE REPLY à la source. Afin d'éviter de trop surcharger le réseau, il est possible d'effectuer un tri et de ne pas répondre à tous les messages ROUTE REQUEST correspondant à un unique identifiant. Ainsi, une réponse sera toujours envoyée pour les deux premières requêtes reçues — afin que la source dispose d'au moins un chemin principal et un chemin de secours —, mais cela ne sera pas automatiquement le cas pour les requêtes suivantes. Seules celles qui auront une métrique comparable ou meilleure à celles des requêtes déjà reçues, ou celles utilisant une route vraiment différente seront à l'origine d'une réponse.

Lorsque la source reçoit une première réponse ROUTE REPLY, le module d'instanciation des chemins est utilisé pour créer un tunnel correspondant à la route contenue dans la réponse. Si la source reçoit une seconde réponse, celle-ci permet de créer un chemin de secours. Toute autre réponse reçue ne sera utilisée pour remplacer le chemin principal ou le chemin de secours que si elle est considérée comme meilleure que les réponses précédentes. Le champ métrique est utilisé à cette fin en ce qui concerne le chemin principal ; pour le choix du chemin de secours, la diversité des routes est aussi prise en compte afin de diminuer la probabilité que le chemin principal et le chemin de secours disparaissent simultanément.

Comme indiqué dans la partie 4.2.4, il n'est pas possible de définir de manière unique une métrique permettant un fonctionnement optimal pour tous les réseaux de bordure spontanés. La métrique est donc un élément paramètrable dans l'architecture.

5.3.3 Gestion des paquets de diffusion

Ce sous-module gère l'acheminement des paquets de diffusion dans le réseau. Lorsqu'un paquet de diffusion est intercepté lors de son émission, il est encapsulé dans un message de type BROADCAST. Ce message est alors transmis par inondation à tous les nœuds du réseau.

Un message BROADCAST contient un identifiant du nœud émetteur ainsi qu'un identifiant local à ce dernier du paquet de diffusion. Ces informations permettent d'identifier de manière unique chaque paquet de diffusion et, en utilisant l'approche décrite dans la partie 3.1.1, d'éviter toute retransmission inutile.

Aucun mécanisme pour gérer la multidiffusion n'est actuellement spécifié et les paquets émis en multidiffusion sont donc gérés de la même manière que les paquets émis en diffusion.

5.4 Instanciation des chemins

Ce module assez simple a pour but de créer le tunnel correspondant à un chemin trouvé par le module de découverte de routes. Avec MPLS, un tunnel est toujours créé de la destination vers la source afin de garantir l'unicité des étiquettes utilisées. Le module est basé sur un protocole de signalisation composé de deux messages :

- PATH REQUEST : ce message est émis par une source souhaitant instancier un chemin vers une destination après avoir reçu une réponse ROUTE REPLY. Le message contient la liste des nœuds composant le chemin, provenant du message ROUTE REPLY ;
- PATH REPLY : il s'agit de la réponse au message PATH REQUEST, émise par la destination et acheminée jusqu'à la source. Cette réponse contient la liste des nœuds composant le chemin ainsi que les informations nécessaires à la création du tunnel, dont la valeur de l'étiquette et l'adresse à utiliser pour atteindre le nœud suivant dans le chemin.

5.4.1 Fonctionnement

Lorsque le module d'instanciation des chemins est contacté par le module de routage pour instancier un chemin, la source émet un message PATH REQUEST vers la destination. Ce message est acheminé par les différents nœuds composant le chemin en question.

Après avoir reçu une requête PATH REQUEST, la destination crée les ressources nécessaires localement pour le tunnel — dont notamment l'allocation d'une étiquette — et les transmet au nœud précédent dans le chemin grâce à un message

PATH REPLY. Ce nœud utilise alors ces informations pour lui aussi créer les ressources locales liées au tunnel et retransmettre la réponse PATH REPLY, avec les nouvelles informations liées au tunnel, au nœud qui le précède dans le chemin. Cette étape est ensuite répétée pour chaque nœud dans le chemin, jusqu'à la source.

Lorsque la source reçoit la réponse PATH REPLY, alors le chemin est instancié. Le module de routage est alors contacté afin qu'il libère les paquets qui avaient été mis en tampon en attente de l'instanciation d'un chemin pour leur flot.

Si aucun message PATH REPLY n'est reçu, ni pour le chemin principal, ni pour le chemin de secours, alors la source peut décider de promouvoir d'autres chemins dans ces rôles et de les instancier. En cas d'échecs répétés, la destination est considérée comme inaccessible. Les paquets qui avaient été mis en tampon pour cette destination sont alors simplement supprimés du tampon.

5.4.2 Fusion avec la découverte de routes

Une optimisation de l'architecture est ici possible. Rien ne rend impossible le regroupement de la découverte de routes et de l'instanciation des chemins en une seule étape. Concrètement, cela signifie qu'un message ROUTE REQUEST est aussi de manière implicite un message PATH REQUEST. Il suffit alors de fusionner les réponses ROUTE REPLY et PATH REPLY. La source de la requête recevra alors l'information sur l'existence de la route et l'information permettant d'utiliser le tunnel correspondant à la route.

Cette fusion permet d'éviter d'avoir un double délai d'attente lié à la découverte de routes et à l'instanciation de chemins avant de pouvoir envoyer le premier paquet d'un flot. L'impact n'est donc pas négligeable au niveau des performances.

Une conséquence négative est que chaque route découverte correspond alors à un chemin qui sera automatiquement instancié. Des ressources locales sur chaque nœud intermédiaire seront donc utilisées pour les tunnels. Le module de maintenance des connexions fera que ces tunnels seront automatiquement détruits car ils ne sont pas tous utilisés. En outre, comme la destination ne répond pas systématiquement à tous les messages ROUTE REQUEST, mais seulement aux deux premières requêtes ainsi qu'aux requêtes suivantes jugées intéressantes, ce problème reste limité. Cette conséquence négative n'a donc pas d'impact majeur, mais fait que cette fusion n'est pas des plus élégantes.

Boucle de création et destruction d'un tunnel

Un problème potentiel est à noter ici, dans le cas où 802.11 est une des technologies employées dans le réseau de bordure spontané. En 802.11, un lien n'est

pas nécessairement bidirectionnel ; en outre, une trame émise en diffusion au niveau MAC a une plus grande portée qu'une trame normale. Un cas particulier est donc possible : si la source ne peut joindre la cible qu'en diffusion et que la cible peut joindre la source, alors les messages ROUTE REQUEST et ROUTE REPLY seront émis et reçus avec succès. Néanmoins, tout paquet non envoyé en diffusion de la source vers la cible sera perdu.

S'il n'y a pas de fusion de la découverte de routes et de l'instanciation des chemins, alors le message PATH REQUEST n'atteindra pas la cible et tout fonctionnera normalement. Mais dans le cas contraire, le chemin sera instancié directement avec le message ROUTE REPLY et les paquets de données seront envoyés dans un tunnel ne fonctionnant pas. Le module de maintenance des connexions fera que le chemin de secours sera utilisé après un certain délai, et que ce tunnel sera détruit. Cependant, rien n'empêche l'optimisation des connexions décrite dans la partie 5.5 de trouver à nouveau le même chemin unidirectionel. Il faut alors que la source se souvienne que ce chemin ne fonctionnait pas afin d'éviter une boucle « création, utilisation avec pertes et destruction du tunnel » qui aurait un impact sur le trafic.

5.5 Maintenance des connexions

Le module de maintenance des connexions joue un rôle important dans l'architecture car c'est lui qui permet de détecter lorsqu'un lien disparait, et donc qu'il faut utiliser le chemin de secours. Son secon rôle est l'optimisation des connexions existantes, qui a pour objectif de déterminer si de meilleures routes existent pour les connexions en cours, et si c'est le cas, de les utiliser.

5.5.1 Évaluation du bon fonctionnement des chemins instanciés

Un tunnel étant composé de plusieurs liens, il y a plusieurs points de rupture possibles ; une évaluation locale du bon fonctionnement de chaque lien est donc mise en place. Chaque nœud par lequel passe au moins un tunnel envoie à tous ses voisins un message I'M ALIVE de manière périodique. Ce message contient des informations pour chaque tunnel passant par le nœud :

- les informations nécessaires pour que le nœud précédent dans le tunnel puisse détecter que le message le concerne ; il s'agit de l'étiquette utilisée pour le tunnel sur ce nœud ainsi que l'adresse de l'interface recevant le tunnel ;
- des statistiques sur le tunnel, telles que la quantité de trafic reçu ; MPLS permet d'obtenir aisément ces informations ;
- l'information selon laquel le tunnel a été détruit ou non ;
- des statistiques sur le lien utilisé par le tunnel : des approches inter-couches sont ici envisageables lorsque le matériel permettra de faire remonter des informations pertinentes sur la qualité du lien.

Lorsqu'un nœud reçoit un message I'M ALIVE, il l'analyse pour savoir si un ou plusieurs tunnels décrits dans ce message le concerne. C'est le cas si l'émetteur du message est le nœud suivant le récepteur dans un tunnel. Le récepteur analyse ensuite les informations sur le(s) tunnel(s) le concernant et les informations sur le lien tel que perçu par l'émetteur du message. Il peut par exemple comparer la quantité de trafic reçue de l'autre côté du lien pour un tunnel avec la quantité qu'il a émise. Le nœud peut ainsi détecter une anomalie indiquant une rupture[1] ou la surcharge du lien.

De manière indirecte, les messages I'M ALIVE jouent le rôle d'acquittement pour les données envoyées dans un tunnel grâce aux statistiques qui sont ainsi transmises ; il devient donc logique de tirer profit de ces messages dans ce sens. Si aucune information de type I'M ALIVE n'a été reçue pour un tunnel donné sur une certaine période, alors le nœud considère que le tunnel a été détruit. En pratique, cela arrive si les messages I'M ALIVE ne sont pas reçus — donc lorsque le lien est effectivement rompu — ou lorsque le tunnel n'existe plus dans le nœud en aval mais que l'information de sa destruction n'a pas été reçue.

Dès qu'un nœud détecte une anomalie, ou est informé qu'un tunnel a été détruit en aval, le nœud envoie immédiatement un message I'M ALIVE avec cette information. L'information est ainsi remontée sans délai jusqu'à la source du tunnel, qui pourra utiliser le chemin de secours afin de garantir le bon fonctionnement de la connexion.

Une trop faible fréquence des messages I'M ALIVE peut avoir pour conséquence l'utilisation sur une période non négligeable d'un tunnel qui ne peut plus fonctionner en raison de la disparition d'un lien entre deux nœuds, et donc la perte de données. À l'inverse, une fréquence trop importante peut avoir un impact important sur la charge du réseau. Un bon compromis entre ces deux extrêmes doit donc être mis en œuvre et une fréquence de l'ordre de deux secondes est typiquement utilisée.

Limitation de la charge réseau liée aux messages I'M ALIVE

La fréquence des messages I'M ALIVE et le fait que chaque nœud intermédiaire d'un tunnel transmette de tels messages peut faire que l'ensemble de ces messages constituent une utilisation non négligeable du réseau.

Une solution, qui peut s'avérer complexe à mettre en œuvre, pour limiter l'impact de ces messages est de les envoyer, quand c'est possible, « sur le dos » d'autres messages (*piggyback*). Une particularité des messages I'M ALIVE est en effet qu'ils

[1] Il est en effet possible de recevoir un message I'M ALIVE dans un sens et de ne pas pouvoir transmettre des données dans l'autre sens car les liens peuvent ne plus être bidirectionnels.

ne sont envoyés en diffusion que localement, aux nœuds voisins. Il est alors possible pour un nœud d'utiliser les messages BROADCAST et ROUTE REQUEST qu'il retransmet et d'y inclure le contenu du message I'M ALIVE lorsqu'il doit en envoyer un. Les nœuds voisins recevront le paquet contenant les deux messages et pourra les traiter comme s'ils étaient arrivés séparément.

5.5.2 Optimisation des connexions

L'optimisation d'une connexion est réalisée en deux étapes. La première consiste à réévaluer la qualité globale, de bout-en-bout, des chemins utilisés pour la connexion. Ensuite, un mécanisme de découverte de routes modifié est mis en œuvre afin de ne découvrir que les routes de meilleure qualité que celles correspondant aux chemins utilisés.

Cette optimisation a lieu en arrière-plan, et donc n'a aucun impact sur le trafic des connexions existantes. Elle est périodique, avec une période assez grande pour ne pas avoir d'influence négatives sur les performances globales du réseau.

Évaluation des chemins

L'évaluation des chemins est un mécanisme simple qui calcule la qualité des chemins instanciés en envoyant des messages qui seront acheminés sur les chemins en question.

Pour chaque chemin instancié, la source envoie un message OPTIM REQUEST, contenant un compteur pour la métrique du chemin et la liste des nœuds composant le chemin. Le message est acheminé jusqu'à la destination en passant par chacun des nœuds du chemin mais sans utiliser le tunnel. Chaque nœud intermédiaire met à jour la métrique de la même manière que lors d'une découverte de routes.

Lorsque la destination reçoit le message OPTIM REQUEST, elle n'a qu'à envoyer un message OPTIM REPLY contenant la nouvelle valeur de la métrique du chemin et la liste des nœuds composant le chemin. Cette dernière permet en effet d'acheminer le message à la source. À la réception de la réponse pour le chemin principal par la source, celle-ci lance une découverte de meilleures routes.

Découverte de meilleures routes

La découverte de meilleures routes utilise le mécanisme de découverte de routes présenté dans la partie 5.3.2, avec deux modifications :

– la requête ROUTE REQUEST contient une valeur de métrique indiquant la qualité minimale recherchée ;

– un nœud recevant un tel message ROUTE REQUEST incrémente le compteur de métrique avant de retransmettre le message, comme s'il s'agissait d'une découverte de routes normale, mais il ne retransmet le message que si la métrique mesurée est toujours meilleure que la métrique indiquant la qualité minimale recherchée. Cela permet d'éviter des retransmissions du message qui ne sont pas intéressantes pour la source.

La métrique indiquant la qualité minimale est déterminée grâce à l'évaluation des chemins qui a été réalisée auparavant. Cela permet ainsi de limiter la découverte de routes aux seules routes de qualité intéressante, et ainsi de réduire le trafic qu'implique cette partie de l'architecture. Si de meilleures routes sont découvertes, alors la source instancie les chemins correspondant et les utilise pour la connexion.

Dans le cas où il n'existe pas de chemin de secours, il est possible de réduire la qualité minimale recherchée afin de ne pas chercher à découvrir uniquement de meilleures routes, et donc d'obtenir une route pouvant devenir un chemin de secours.

Fusion des deux étapes

Une optimisation assez simple de cette partie de l'architecture consiste à fusionner l'évaluation des chemins et la découverte de meilleures routes. Dans ce cas, les messages OPTIM REQUEST sont utilisés comme des messages ROUTE REQUEST. Un nœud décide de retransmettre le message résultant si la métrique mesurée est meilleure que la métrique indiquant la qualité minimale recherchée ou si le message effectue le parcours correspondant à un chemin instancié de la connexion. Le message OPTIM REPLY est quant à lui émis normalement.

La seule difficulté qu'implique cette fusion est que la qualité courante des chemins n'est pas connue et donc une qualité passée est utilisée pour déterminer si une route est meilleure ou non. Cet asynchronisme ne pose pas de réel problème puisque le pire des cas est simplement qu'une meilleure route ne sera utilisée qu'avec un peu de retard.

5.6 Conclusion

L'architecture proposée ici est basée sur un niveau 2.5 par lequel les paquets sont acheminés dans le réseau de bordure spontané. Ce niveau 2.5 permet à la couche IP d'avoir l'illusion d'un réseau local unique. En raison de sa simplicité et de sa standardisation, et parce qu'il répond aux besoins de l'acheminement, MPLS sert de base pour créer cette nouvelle couche.

L'acheminement est rendu possible grâce à un protocole de routage réactif qui crée un chemin pour chaque nouvelle connexion. La connexion est ensuite maintenue tout au long de sa durée de vie grâce à des protections telles que les chemins de secours, la vérification continue du bon fonctionnement du chemin, ou encore grâce à un processus d'optimisation permettant de trouver en arrière-plan de meilleurs chemins à utiliser.

Bien que le couplage d'une approche orientée connexion avec un routage réactif peut sembler paradoxale, les caractéristiques obtenues permettent de répondre au cahier des charges défini dans le chapitre 4.

Chapitre 6

Du papier à la machine : implémentation

> If we can get a picture of Julia Roberts in a thong, we can certainly get a picture of this weirdo.
>
> J. Jonah Jameson, *Spider-Man*

Il existe plusieurs possibilités non exclusives entre elles pour valider une architecture telle que celle proposée dans le chapitre 5 :

- garder une approche purement théorique pour évaluer le travail ;
- utiliser des simulations pour évaluer le comportement ;
- réaliser un prototype afin d'expérimenter.

Le côté théorique a évidemment joué un rôle lors du développement de l'architecture pour anticiper les différentes difficultés à résoudre et choisir quelles voies explorer. Néanmoins, la validation formelle dans le cadre des réseaux sans fil ne permet pas de prévoir le comportement exact ou l'impact d'un protocole dans l'utilisation qui en est faite dans un réseau.

La plupart des recherches récentes dans le domaine des réseaux sans fil s'appuient sur des simulations, réalisées à l'aide d'outils comme ns-2 [101] ou OPNET Modeler [102]. Les simulations ont l'avantage de permettre de créer aisément différentes topologies et de pouvoir faire varier différents paramètres tels que le nombre de nœuds, la quantité de trafic simulée ou encore le modèle de mobilité des nœuds. Néanmoins, les résultats obtenus ne reflètent pas toujours la réalité [103, 104, 105, 106] : la configuration du simulateur peut avoir une influence sur les résultats obtenus ; la même simulation sur deux simulateurs peut aboutir à des conclusions différentes ; certains paramètres réels, comme l'impact des murs en terme d'atténuation du signal, sont rarement modélisés lors des simulations ; enfin,

la validité des implémentations de l'ensemble des protocoles mis en œuvre dans les simulateurs n'est généralement pas prouvée.

L'implémentation d'un prototype est rarement une voie explorée car le développement de celui-ci nécessite beaucoup de temps et les tests ensuite réalisés sont généralement de taille réduite en raison des contraintes matérielles. Ainsi, alors qu'une simulation est généralement effectuée avec une, voire plusieurs centaines de nœuds, l'expérimentation avec un prototype ne dépasse que très rarement la dizaine de nœuds. Il faut en effet posséder une importante quantité de matériel, le configurer et le déployer pour parvenir à réaliser des mesures conséquentes. Cependant, les résultats obtenus avec de telles expérimentations permettent de confronter directement à la réalité les théories proposées.

Nous avons choisi de développer très tôt Lilith, un prototype de l'architecture proposée dans le chapitre 5, afin de pouvoir nous appuyer sur des résultats réels pour améliorer l'architecture. Cette prise en compte de la réalité est un aspect central dans l'évolution de l'achitecture.

6.1 Outils et bibliothèques utilisés

Le travail de développement réalisé a été mené comme un projet de développement logiciel. Il a donc été naturel d'utiliser tous les outils appropriés, ainsi que de réutiliser du code disponible sous forme de bibliothèque ou logiciel.

Tous les outils standards de programmation ont été mis en œuvre. Cela inclut évidemment la gestion de la chaîne de compilation avec les `autotools` (`autoconf`, `automake` et `libtool`) et `make`, ainsi que le compilateur `gcc` et le débogueur `gdb` pour le code écrit en C.

L'utilisation d'un système de contrôle de version a permis de conserver l'historique du développement, ainsi que l'utilisation de branches pour des expérimentations sur le code. Après quelques temps avec CVS pour le contrôle de version, Bazaar a été choisi pour sa robustesse et sa simplicité.

Quelques autres outils ou interfaces ayant eu un impact considérable sur le développement sont présentés avec plus de détails dans cette section.

6.1.1 La bibliothèque `libpcap`

La bibliothèque `libpcap` [107] fournit une interface de programmation pour la capture de paquets ainsi que l'émission bas-niveau de paquets. Elle est utilisée dans de très nombreux outils tels que `tcpdump`, `wireshark`, `nmap` ou encore `snort`. L'intérêt majeur de cette bibliothèque, outre sa facilité d'utilisation, consiste en

sa portabilité sur de très nombreuses plate-formes. En effet, les opérations bas-niveau fonctionnent souvent d'une manière légèrement différente d'un système à un autre, et il est donc nécessaire d'implémenter ces opérations de manière native pour chaque système. La bibliothèque `libpcap` permet de s'affranchir de cette étape.

La capture des paquets s'effectue à l'aide d'un filtre qui détermine quels paquets doivent être capturés. Il est ainsi possible de limiter les paquets capturés aux paquets entrants ou sortants, en fonction d'un port source ou destination, d'une adresse source ou destination, d'un protocole comme TCP, IP ou même ARP, etc.

Il est aussi possible d'émettre des paquets ou des trames directement sur une interface donnée, en contournant toute la pile TCP/IP et même la résolution des adresses MAC. Dans ce derniers cas, il faut alors fournir la trame complète, y compris l'en-tête de la trame. Cela permet donc d'émettre des données sans interférence du noyau.

6.1.2 L'interface netlink

Dans la majorité des systèmes de type UNIX, l'interface avec le noyau à partir de l'espace utilisateur se fait à l'aide d'appels `ioctl`. Dans le noyau Linux, il a été décidé de remplacer ces appels `ioctl` par un protocole de communication particulier, netlink. Son objectif est d'être utilisé pour toute communication entre espace utilisateur et noyau. Cela inclut notamment une grande partie de l'interface avec la couche réseau du noyau, en particulier tout ce qui concerne le contrôle de la table de routage et des services IP [108]. Étant donné le rôle de netlink, il aurait été difficile de ne pas utiliser ce protocole de communication.

Cette interface netlink a l'avantage de rendre les choses plus propres, en définissant clairement les messages à transmettre, et en évitant d'avoir à faire des surcharges ambigues sur le type des variables. Cependant, netlink complexifie la programmation car il s'agit d'un protocole asynchrone et avec acquittements.

6.1.3 Interception des paquets en émission

Pour l'implémentation de l'architecture, il est nécessaire d'intercepter les paquets lors de leur émission avant qu'ils ne soient transmis par l'interface réseau. Le sous-module d'interception des paquets décrit dans la partie 5.3.1 a notamment besoin de cette fonctionnalité, ainsi que l'implémentation de MPLS en espace utilisateur qui sera présentée dans la partie 6.2.3. Plusieurs approches sont envisageables.

La première solution, et la plus intuitive, est d'utiliser la bibliothèque `libpcap`, décrite dans la partie 6.1.1. Cette bibliothèque permet de capturer des paquets selon

un filtre que l'on peut librement définir. Bien qu'assez facile à mettre en œuvre, et fonctionnant sur toutes les plate-formes, cette solution ne remplit pas totalement le rôle d'interception des paquets. En effet, `libpcap` ne permet que de capturer des paquets, et il est impossible de bloquer leur émission. En pratique, cela signifie que si un paquet est capturé pour être réémis plus tard, il sera émis deux fois : une fois au moment de sa capture et une fois lors de sa réémission. En fonction du contenu du paquet, cela peut avoir des effets de bord non souhaités.

Une seconde approche est d'utiliser l'infrastructure de pare-feu disponible dans le noyau pour détourner les paquets qui sont émis vers le programme. Ceci est rendu possible sous Linux grâce à `libipq` [109] et sous FreeBSD grâce aux sockets `divert` [110].

Le fonctionnement de `libipq` est relativement simple. Il suffit d'installer une règle `netfilter` indiquant quels paquets doivent être interceptés, et d'utiliser la bibliothèque `libipq` pour récupérer ces paquets. Il est alors possible d'examiner le contenu du paquet, et de décider si le noyau doit continuer l'émission de ce paquet, ou simplement ignorer le paquet. Par exemple, pour intercepter tous les paquets sortant sur l'interface `eth0`, la règle `netfilter` est la suivante :

```
iptables --append OUTPUT --out-interface eth0 --protocol all
         --jump QUEUE
```

L'annexe B présente un simple programme utilisant `libipq`, qui a été développé pour une expérimentation du chapitre 7.

Un inconvénient de `libipq` est cependant qu'un seul processus à la fois peut utiliser cette bibliothèque : il n'est pas possible de préciser vers quelle file d'attente en espace utilisateur les paquets sont redirigés. Cette restriction a été récemment levée par `libnetfilter_queue` [111], qui remplace `libipq`.

Sous FreeBSD, pour obtenir un résultat équivalent, il suffit de créer une socket `divert`, de lui assigner un numéro unique identifiant la socket — cela permet d'utiliser des sockets `divert` à partir de différents processus sans conflit — et d'installer une règle dans le pare-feu. De même que sous Linux, les paquets interceptés sont reçus par le processus et peuvent être traités par ce dernier. Ainsi, pour intercepter tous les paquets émis sur l'interface `bge0` *via* la socket `divert` identifiée par le numéro 13800, la commande à utiliser est :

```
ipfw add divert 13800 all from any to any xmit bge0
```

6.2 MPLS

Un des arguments en faveur du choix de MPLS a été le fait qu'il s'agit d'un protocole standardisé. Bien qu'il est implémenté dans de nombreux matériels de

routage, il n'en va pas de même pour les ordinateurs et autres appareils qui visent l'utilisateur final. Au début des travaux, seules une implémentation pour NetBSD et une autre pour Linux étaient disponibles.

L'implémentation MPLS pour NetBSD a été réalisée dans le cadre du projet AYAME [112], mais le développement a été arrêté avant même le début des travaux. L'implémentation MPLS pour le noyau Linux était quant à elle fonctionnelle et le développement était toujours actif.

6.2.1 MPLS Linux

L'implémentation de MPLS pour le noyau Linux [113] a été commencée en 2000 et est rapidement devenue fonctionnelle, même si le développement a évidemment continué avec quelques évolutions majeures.

La première version qui a été utilisée dans le cadre de nos travaux était une implémentation pour le noyau 2.4. La communication avec le noyau pour la gestion des tunnels MPLS avait lieu à l'aide de messages `ioctl` et les différentes statitisques disponibles pour chaque tunnel étaient exportées par le noyau dans un fichier se situant dans `/proc`. Bien que le protocole MPLS fonctionnait, il y avait quelques petits bogues, quelques problèmes de stabilité et un problème majeur : le noyau 2.4 était déjà remplacé par le noyau 2.6 dans de nombreuses distributions. Cependant, la différence entre le code du noyau 2.4 et celui du noyau 2.6 était telle qu'un simple portage était difficile. Il en a résulté que l'implémentation pour le noyau 2.6 a nécessité beaucoup de travail et est très différente.

Arrivée assez tardivement, l'implémentation pour le noyau 2.6 a aussi un changement de fonctionnement majeur : la communication avec le noyau se déroule par l'intermédiaire de netlink pour la gestion des tunnels MPLS et pour l'obtention des statistiques. Bien qu'il soit logique que netlink soit adopté pour les communications liées à MPLS entre espace utilisateur et noyau, ce changement a complexifié la programmation. Alors qu'un simple appel `ioctl()` était suffisant pour créer une étiquette MPLS, un traitement beaucoup plus important est devenu nécessaire.

Un problème majeur de cette implémentation MPLS est qu'elle n'est pour l'instant pas intégrée par défaut au noyau Linux : il faut appliquer un patch et utiliser une configuration particulière afin que le noyau gère le protocole. Malheureusement, ce patch n'est pas intégré dans les distributions et il faut donc gérer soi-même cette étape. Même si la plupart des changements ne sont pas intrusifs, quelques modifications du noyau touchent des parties du noyau qui varient dans le temps. Or le patch fourni n'existe que pour une version du noyau — trop souvent assez ancienne — et ne s'applique donc pas sans adaptation aux autres versions du noyau Linux. Il faut aussi gérer le déploiement de ce noyau sur les machines de test.

Une autre difficulté liée à l'utilisation de cette implémentation de MPLS est qu'elle n'existe que pour le noyau Linux et il est donc impossible d'utiliser le prototype qui a été implémenté sur une autre plate-forme. Si à première vue cela ne semble pas être un problème majeur, il s'avère que cela complique quelque peu les tests car cela ajoute une nouvelle barrière pour disposer du matériel adéquat.

Enfin, et il s'agit du problème le plus important, il subsiste quelques bogues majeurs dans cette implémentation. Il est ainsi arrivé régulièrement qu'une machine se bloque totalement et devienne inutilisable. Il est fort probable que ce blocage soit lié à une erreur dans la gestion des tunnels MPLS par le prototype, mais le blocage de la machine rend le débogage extrêmement compliqué et le noyau ne devrait pas être mis hors d'état de fonctionner par une telle erreur.

Un autre bogue lié à l'implémentation même rendait impossible l'utilisation des outils d'observation de réseau basés sur `libpcap`, tels que `tcpdump` et `wireshark`. Une analyse complète du problème, disponible en annexe A, a révélé le bogue, qui nécessite pour sa résolution une réécriture partielle de l'implémentation.

En raison des difficultés rencontrées avec cette implémentation de MPLS, et malgré ses qualités, il a semblé intéressant de disposer d'une autre implémentation qui aurait moins de limitations pour effectuer des tests. Deux critères importants ont donc été retenus : avoir un aspect multi-plate-forme pour ne pas être limité au noyau Linux et tourner en espace utilisateur afin de ne pas subir de blocages au niveau du noyau en cas de bogue.

6.2.2 scapy-mpls

Scapy [114] est un outil pour la manipulation des paquets écrit en python et fonctionnant sur de très nombreuses plate-formes. Il s'agit d'un logiciel très puissant car il permet de faire à peu près tout ce qui est envisageable : interception de trafic, analyse du réseau, interprétation de protocole, génération de paquets bruts ou par l'intermédiaire d'interface pour chaque protocole, etc. Un des intérêts majeur de scapy est sa flexibilité. En effet, alors qu'il n'est généralement pas possible d'utiliser un logiciel réseau à des fins non prévues, le but même de scapy est de laisser toujours une porte ouverte pour implémenter tout fonctionnement désiré. Ainsi, toutes les informations à propos d'un paquet sont disponibles, sans avoir été interprétée auparavant. L'utilisation du langage python permet aussi de développer un prototype complet en un temps réduit. Il en résulte que scapy est principalement utilisé comme une base pour créer un autre logiciel que comme un logiciel complet.

La compréhension du fonctionnement de scapy n'étant pas triviale, il a fallu un certain temps de prise en main pour appréhender correctement comment utiliser ce logiciel. Après cette première étape, il est vite apparu que scapy pouvait être utilisé afin de créer une implémentation de MPLS et, éventuellement, un second prototype pour l'architecture proposée. Quelques heures ont suffi pour créer scapy-mpls,

une implémentation de MPLS basée sur scapy. La caractéristique la plus remarquable de cette implémentation tient probablement dans la taille ce celle-ci : moins de 250 lignes de code python — commentaires et lignes vides comprises — sont nécessaires, même si bien entendu, tout cela repose sur scapy. L'annexe C contient l'ensemble du code nécessaire.

Le prototype scapy-mpls fonctionne, mais il n'a pas été développé au point de s'intégrer naturellement dans le système. En effet, l'émission de paquets par MPLS n'est pas transparente dans ce simple prototype, et nécessite d'écrire du code spécifique pour spécifier quels paquets envoyer dans un tunnel. En outre, les premiers tests ont immédiatement révélé que le fonctionnement est extrêmement lent et gourmand en utilisation du processeur. Trois raisons permettent d'expliquer ce problème de performance. Tout d'abord, le fait d'implémenter MPLS en espace utilisateur nécessite un changement de contexte du noyau à l'espace utilisateur pour le traitement de chaque paquet reçu, ce qui a un coût ; pour un unique paquet, ce coût peut cependant se révéler négligeable avec les ordinateurs d'aujourd'hui. L'utilisation du langage python a aussi sans aucun doute un impact : même si le script python n'est pas simplement interprété, mais est compilé avant d'être exécuté, il n'en reste pas moins qu'il s'agit d'un langage haut-niveau pour lequel toute manipulation se traduit en un nombre important d'instructions au niveau du processeur. L'apparente simplicité de python cache effectivement la complexité du traitement des données. Enfin, scapy est probablement un outil à double tranchant puisque sa flexibilité implique un impact sur les performances. La flexibilité implique de garder la possibilité d'effectuer un traitement spécifique lors de différentes étapes de la vie d'un paquet alors que le besoin de performances pousse vers un traitement simple et direct.

Un des critères d'évaluation de l'architecture est la performance qu'elle permet d'obtenir. En raison des problèmes rencontrés au niveau des performances, scapy-mpls n'a donc que très peu été utilisé. Cette implémentation de MPLS n'en reste pas moins intéressante par le fait qu'elle permet d'interagir en émission et en réception avec des paquets MPLS, et donc de déboguer une implémentation MPLS, très facilement.

6.2.3 `umpls`

Après l'expérimentation qu'a été scapy-mpls, le développement d'une implémentation performante de MPLS en espace utilisateur a semblé pertinente. C'est ainsi qu'a commencé le projet `umpls`, écrit en C. Il utilise la bibliothèque `libpcap` pour l'émission des paquets, ainsi que `libipq` et les sockets `divert` pour l'interception des paquets, ce qui le rend relativement portable puisque fonctionnel pour Linux et FreeBSD.

En terme de fonctionnalités, le projet a abouti puisque l'implémentation est complète et l'utilisation d'`umpls` est aisée : il suffit de lancer le démon, et il est ensuite

possible de le contrôler par son interface de contrôle. Un outil de ligne de commande a été écrit afin de ne pas avoir à écrire un programme pour réaliser chaque manipulation. Ainsi pour réaliser le tunnel MPLS de la figure 6.1, encapsulant le trafic à destination de C passant ou émis par A, il suffit d'effectuer les commandes des listings 6.1, 6.2 et 6.3.

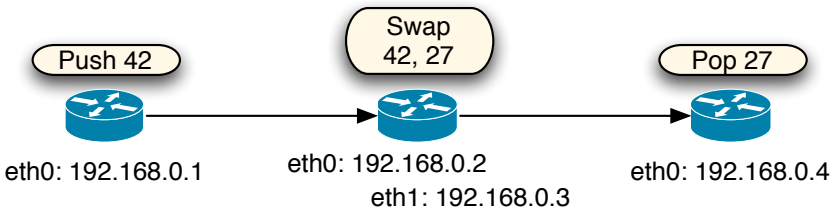

FIG. 6.1 – Exemple d'utilisation d'`umpls`

Listing 6.1 – Commandes à effectuer sur A

```
umpls iface add eth0
umpls out add 42 nexthop 192.168.0.2 iface eth0
# affiche la valeur de la cle correspondant a l'etiquette: $KEY1
umpls rule add 192.168.0.3 $KEY1
```

Listing 6.2 – Commandes à effectuer sur B

```
umpls iface add eth0
umpls iface add eth1
umpls out add 27 nexthop 192.168.0.3 iface eth1
# affiche la valeur de la cle correspondant a l'etiquette: $KEY2
umpls in 42 add swap $KEY2
```

Listing 6.3 – Commandes à effectuer sur C

```
umpls iface add eth0
umpls in add 27 pop inet
```

Les performances d'`umpls` seront étudiées dans le chapitre 7.

Organisation générale

L'architecture d'`umpls` est composée de plusieurs modules ayant chacun un rôle bien déterminé. D'une vision haut niveau du projet, ces modules sont regroupés pour former des composants.

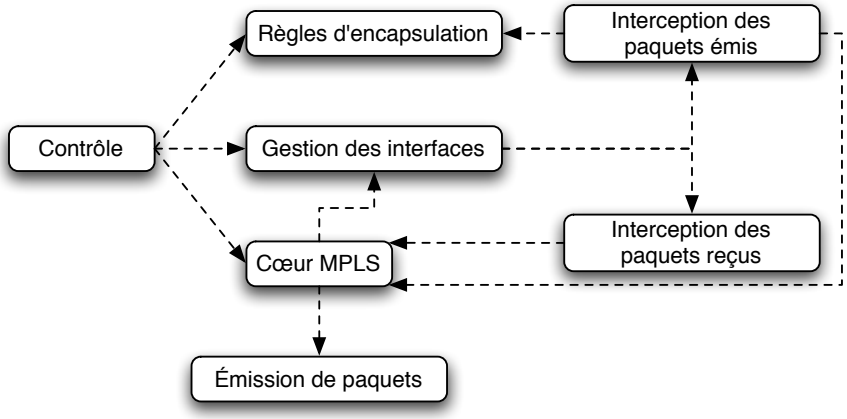

FIG. 6.2 – Diagramme de composants d'`umpls`

Contrôle. Le composant de contrôle est la partie qui permet l'interaction d'un programme extérieur avec `umpls`. Par l'intermédiaire de ce composant, il est possible de configurer quelles interfaces réseau sont utilisées par `umpls` (gestion des interfaces), de définir quels paquets émis doivent entrer dans un tunnel MPLS (règles d'encapsulation) et de créer, modifier et supprimer des tunnels MPLS (cœur MPLS). La communication interprocessus a lieu *via* une socket UNIX, à l'aide d'un protocole créé sur mesure.

Règles d'encapsulation. Il est nécessaire de déterminer quels paquets doivent être encapsulés dans MPLS. Pour cela, des règles sont définies par un processus extérieur à `umpls` et transmises *via* le composant de contrôle. En l'état actuel, un seul critère de règle a été implémenté : il s'agit de l'adresse IP destination du paquet. Cette limitation n'est en rien architecturale, mais simplement liée aux contraintes de temps. Très peu d'efforts seraient nécessaires pour pouvoir établir des règles en fonction d'autres critères comme l'adresse IP source, les ports utilisés, voire même le contenu du paquet. En effet, l'ensemble du paquet étant disponible, il peut donc être analysé de toutes les manières imaginables. Il faut néanmoins garder à l'esprit que cette analyse, si elle est coûteuse, a nécessairement un impact sur les performances réseau de la machine puisque tous les paquets émis doivent la subir. La règle spécifie évidemment la clé définissant le tunnel MPLS à utiliser pour l'encapsulation.

Gestion des interfaces. Un composant de gestion des interfaces est nécessaire afin de centraliser tout ce qui est relatif à celles-ci, tel que le contrôle de l'interception des paquets sur chaque interface ou l'espace d'étiquettage utilisé sur chacune d'entre elles. Il permet donc d'ajouter à la volée, sans avoir à redémarrer `umpls`, une

nouvelle interface à utiliser, ou d'en supprimer une. Il est ainsi possible d'utiliser `umpls` dans des conditions très souples, que peuvent requérir des expérimentations et même une utilisation réelle. Par exemple, dans une expérimentation, il peut arriver que l'on désactive une interface pour simuler une coupure de lien.

Cœur MPLS. Il s'agit du cœur d'`umpls` puisque c'est dans ce composant que toute la gestion de MPLS a lieu. Ce composant étant assez complexe de par sa structure et ses interactions avec les autres composants, il est détaillé ci-après.

Émission des paquets. Le composant d'émission des paquets utilise la bibliothèque `libpcap`. Malgré la simplicité apparente de la tâche, l'émission des paquets est en réalité relativement complexe lorsqu'il s'agit de paquets MPLS. En effet, il n'est pas possible de simplement injecter un paquet MPLS dans le noyau pour qu'il soit émis : l'information indiquant à quelle nœud ce paquet doit être transmis n'est pas disponible au noyau. Il est donc nécessaire d'ajouter l'en-tête MAC avec notamment les adresses MAC, qui elles ne sont pas nécessairement connues. Ce processus nécessite une gestion ARP, décrite ci-après.

Interception des paquets émis. Tous les paquets émis sont interceptés par ce composant. Les règles d'encapsulation sont alors consultées pour savoir si le paquet doit être encapsulé ou non. Si c'est le cas, l'émission du paquet par le noyau est annulée et le paquet est transmis au cœur MPLS qui le prend en charge ; dans le cas contraire, le paquet continue son parcours dans le noyau. L'implémentation est réalisé avec la bibliothèque `libipq` pour Linux et avec une socket `divert` pour FreeBSD. Il serait aussi possible d'utiliser la bibliothèque `libpcap`, mais il s'agirait alors d'une simple capture et non d'une réelle interception des paquets, avec les problèmes déjà indiqués dans la partie 6.1.3.

Interception des paquets reçus. Pour traiter la réception des paquets MPLS, il est nécessaire d'intercepter les paquets reçus. En pratique, ce sont les trames qui sont interceptées, afin de connaître le type du contenu de la trame. S'il s'agit d'un contenu MPLS, alors le paquet MPLS est transmis au cœur MPLS pour traitement. S'il s'agit d'une réponse ARP, alors celle-ci est aussi prise en compte comme décrit ci-après dans la gestion ARP. Dans l'implémentation, il ne s'agit pas d'une réelle interception, mais d'une capture à l'aide de la bibliothèque `libpcap` ; la raison est que l'interception de paquets MPLS n'est pas forcément possible par l'intermédiaire d'un pare-feu, notamment si le noyau n'a pas d'implémentation de MPLS. Comme il s'agit d'une simple capture, le paquet MPLS est aussi transmis au noyau, mais celui-ci l'ignorera s'il ne supporte pas le protocole. En pratique, il n'y a pas d'intérêt à utiliser `umpls` si le noyau supporte MPLS, et il ne s'agit donc pas d'un problème.

Détails du cœur MPLS

Le cœur MPLS d'`umpls` est le composant le plus important du projet, et comme le montre la figure 6.3, il est lié à presque tous les autres composants. Ce composant est découpé en quatre modules.

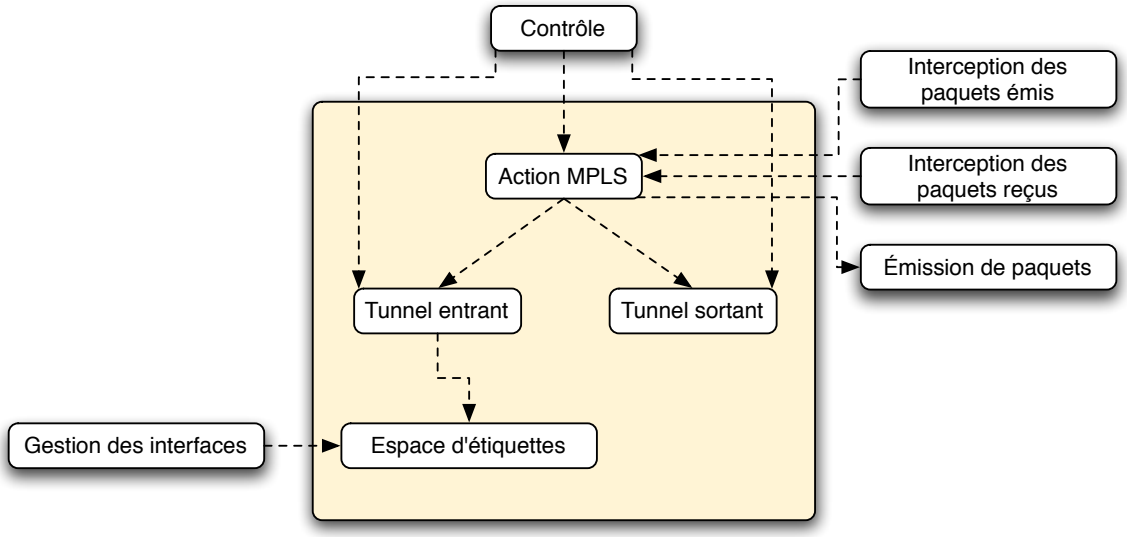

FIG. 6.3 – Détail du composant cœur MPLS

Espace d'étiquettes. En ce qui concerne les étiquettes entrantes, MPLS a deux modes de fonctionnement, correspondant chacun à la définition des espaces d'étiquettes (*label spaces*) utilisés :

- un espace d'étiquettes par plate-forme : dans ce cas, il n'est pas possible d'utiliser la même étiquette sur deux interfaces et toute étiquette entrante doit être unique sur la machine ;
- un espace d'étiquettes par interface : il est alors possible d'avoir deux étiquettes entrantes de la même valeur, si elles sont associées à deux interfaces différentes.

Généralement, l'espace d'étiquettes par plate-forme est suffisant, et c'est donc le comportement par défaut d'`umpls`. Ce module gère les espaces d'étiquettes, quelle que soit la configuration choisie.

Tunnel entrant. Ce module gère l'ensemble des étiquettes entrantes. Il s'agit donc principalement d'une structure de données qui doit être rapide à interroger et qui contient la liste des étiquettes entrantes ainsi que, pour chacune d'entre elles,

l'espace d'étiquettes auquel elle appartient et l'action à effectuer. Lors de la réception d'un paquet MPLS sur une interface, l'étiquette entrante du paquet est définie dans l'en-tête MPLS et l'espace d'étiquettes du paquet est défini par l'interface sur laquelle a été reçue le paquet. Les statistiques sur la quantité de données reçues pour chaque étiquette entrante sont aussi disponibles dans ce module.

Tunnel sortant. Ce module gère l'ensemble des étiquettes sortantes. Une étiquette sortante est réellement constituée d'une étiquette ainsi que de l'adresse du nœud auquel il faut envoyer les paquets. Une clé unique permet d'identifier les étiquettes sortantes. La valeur de la clé n'a aucun lien avec la valeur de l'étiquette, puisque plusieurs étiquettes sortantes peuvent être à destination de nœuds différents et avoir la même valeur d'étiquette. Les statistiques sur la quantité de données émises pour chaque étiquette sortante sont disponibles dans ce module.

Action MPLS. Les actions liées à MPLS — encapsulation, décapsulation et changement d'étiquette — sont mises en œuvre dans ce module. Lorsqu'un paquet MPLS est reçu *via* le module d'interception des paquets reçus, ce module interroge le module de tunnel entrant pour connaître l'action à effectuer. Dans le cas d'une encapsulation ou d'un changement d'étiquette, la clé correspondant à l'action est transmise au module de tunnel sortant pour l'obtention de l'étiquette à utiliser ainsi que l'adresse à laquelle il faut envoyer le paquet MPLS ; après le remplissage ou la modification de son en-tête, celui-ci est alors transmis au module d'émission des paquets. Dans le cas d'une décapsulation, le module transmet simplement le paquet encapsulé au module d'émission des paquets. En outre, les paquets dont le module d'interception des paquets émis détecte qu'ils doivent être encapsulés sont transmis à ce module qui les gère comme dans le cas d'encapsulation vu précédemment. Enfin, ce module s'occupe aussi de mettre à jour les statistiques de trafic pour chaque étiquette entrante et sortante.

Gestion ARP

Lorsqu'il faut émettre un paquet MPLS, `umpls` doit émettre une trame complète, en-tête MAC compris. Or, `umpls` connait l'adresse IP du nœud auquel il faut envoyer le paquet, mais pas son adresse MAC. Il faut donc interroger la table ARP pour trouver cette information. Si celle-ci n'est pas disponible dans la table ARP, il s'avère nécessaire d'envoyer une requête ARP *who-has* et d'attendre la réponse. Comme on ne peut pas demander au noyau de jouer ce rôle, il faut donc recréer en espace utilisateur le protocole ARP.

Il n'est pas envisageable de bloquer le programme en attendant la réponse ARP et l'utilisation d'une file d'attente s'avère donc nécessaire. Il existe pour chaque

adresse IP en attente de résolution ARP une file d'attente, contenant un ou plusieurs paquets MPLS. En cas de réponse, les paquets en file d'attente peuvent être encapsulés avec une trame MAC et émis. Dans le cas contraire, après un délai sans réponse, signifiant que le nœud possédant l'adresse IP demandée n'est pas joignable, la file d'attente est vidée sans émission des paquets.

Les réponses ARP sont reçues par le noyau et `umpls` ne peut pas en être directement informé. Il faut donc utiliser le composant d'interception des paquets reçus pour analyser toutes les réponses ARP reçues et savoir si elles correspondent à des requêtes qui ont été émises.

6.3 Lilith, une implémentation de l'architecture

Lilith est une implémentation de l'architecture proposée au chapitre 5. Il s'agit d'un démon écrit en C, et dont l'objectif est de fonctionner sur une machine disposant d'un noyau Linux avec le patch MPLS. Cette plate-forme a été arrêtée afin de simplifier le développement, mais il s'est avéré que le patch MPLS Linux a évolué pendant ce temps, et n'était pas totalement satisfaisant. Une certaine abstraction, non souhaitée au début du prototypage, a donc dû être ajoutée pour permettre l'utilisation de différentes implémentations de MPLS.

Une particularité du développement a été que celui-ci a eu un impact sur l'architecture : les contraintes de la réalité ont parfois mis en évidence certains aspects de l'architecture qui nécessitaient plus d'approfondissement. Cette approche fut délibérée, le prototypage ayant commencé relativement tôt.

6.3.1 Optimisations de l'architecture utilisées et aspects non implémentés

Lors de la présentation de l'architecture dans le chapitre 5, plusieurs optimisations de l'architecture ont été proposées afin de la rendre plus efficace. Lilith implémente certaines de ces optimisations. Le tableau 6.1 récapitule la liste des optimisations utilisée.

Il est à noter que la gestion des boucles de création et destruction de tunnels liées à la fusion de la découverte de routes et de l'instanciation des chemins n'a pas été implémentée. Il ne s'agit pas d'un problème d'implémentation particulièrement complexe, mais le cas est resté purement théorique et n'est pas apparu dans les expérimentations et cela n'a donc pas été une priorité pour le développement.

La distinction de différents flots ne se fait pas réellement sur les flots en tant que tels car il est impossible de distinguer ceux-ci sans information venant des couches supérieurs. La distinction a donc lieu sur les adresses et les ports utilisés ; bien que

TAB. 6.1 – Optimisations possibles de l'architecture et leur statut dans Lilith

Optimisation	Statut
Ré-utilisation d'un tunnel d'un autre flot pour un flot en attente de découverte de routes	Non implémentée
Fusion de la découverte de routes et de l'instanciation des chemins	Implémentée
Envoi « sur le dos » (*piggyback*) des messages I'M ALIVE	Non implémentée
Fusion de l'évaluation des chemins et de la découverte	Implémentée

cette approche soit simple, elle reste cependant suffisamment fiable pour obtenir de bons résultats.

Les messages I'M ALIVE ne contiennent que la liste des tunnels actifs et la liste des tunnels ayant disparu. Bien que les statistiques sur la quantité de données reçues soient disponibles et gérées par Lilith, elles ne sont pas transmises dans ces messages dans le prototype actuel. Cela n'empêche pas le fonctionnement basique de l'évaluation des chemins instanciés.

6.3.2 Métrique utilisée

La métrique joue un rôle important dans l'architecture proposée car c'est elle qui permet de déterminer si une route est meilleure qu'une autre. Elle a donc une influence sur le chemin principal utilisé pour la connexion, mais aussi sur le chemin de secours et sur tous le processus d'optimisation des connexions.

Il est délicat de définir une métrique efficace, mais le prototypage avec Lilith a nécessité d'implémenter une métrique au moins basique. Une approche simple a donc été choisie : la valeur d'une route dépend de son nombre de sauts, la valeur de chaque saut étant pondérée en fonction de la technologie entrant en jeu. Par exemple, un saut sur un réseau Ethernet vaut 1 tandis qu'un saut sur un réseau 802.11 vaut 2. La valeur la plus faible est la meilleure. Il s'agit donc d'une métrique similaire à celle utilisée par OSPF [44, 45], qui se base sur le coût des interfaces.

Il aurait été intéressant de prendre en compte des élements venant des couches MAC, notamment pour les réseaux 802.11, pour pouvoir adapter la métrique en fonction de l'activité sur le réseau, mais aucune technologie inter-couches n'est actuellement disponible pour réaliser cela.

6.3.3 Paramètres configurables

L'architecture proposée n'est pas une architecture fixe : différents paramètres entrent en jeu, et il est difficile de leur associer une valeur définitive car l'utilisation

des réseaux de bordures spontanés peut être variée. C'est pourquoi Lilith permet de changer aisément ces paramètres en ligne de commande. Le tableau 6.2 liste les différents paramètres configurables et les valeurs par défaut qui ont été choisies.

TAB. 6.2 – Paramètres de l'architecture configurables

Paramètre	Valeur par défaut
Nombre de chemins de secours	2
Délai avant de réémettre un message ROUTE REQUEST avec une durée de vie plus importante	0.5 seconde
Durée de vie du premier message ROUTE REQUEST	1 saut
Durée de vie du dernier message ROUTE REQUEST avant abandon	4 sauts
Durée de vie maximale des messages OPTIM REQUEST	4 sauts
Fréquence des messages I'M ALIVE	2 secondes
Temps minimal sans réception de messages I'M ALIVE avant suppression des tunnels non acquittés	7 secondes (soit trois messages)
Temps d'inactivité d'un tunnel avant sa suppression automatique	30 secondes
Fréquence du processus d'optimisation d'une connexion	15 secondes

6.3.4 Notion de bus

Dans Lilith, le réseau est considéré comme un unique bus sur lequel des données sont envoyées. Cette vision est en quelque sorte une transposition pour l'implémentation de l'idée présentée dans la partie 4.1.2 selon laquelle un réseau de bordure spontané est conceptuellement un simple réseau local. Il en découle que toutes les opérations de communication à travers le réseau se traduisent en accès au bus.

Deux types de paquets sont utilisés par Lilith :
- des paquets de signalisation : ces paquets sont utilisés pour tous les messages échangés entre les différentes instances de Lilith fonctionnant sur les nœuds d'un même réseau ;
- des paquets MPLS : bien que faisant partie de l'architecture au sens fonctionnel, ces paquets ne sont pas gérés directement par Lilith, mais par l'implémentation de MPLS qui est utilisée. Ils ne sont donc pas formellement considérés ici.

Le bus permet trois types d'accès en écriture :
- accès global : cet accès est utilisé pour tous les messages qui doivent atteindre tous les nœuds dans le réseau (BROADCAST, ROUTE REQUEST, OPTIM REQUEST) ;
- accès local : cet accès est utilisé pour les messages qui doivent atteindre tous les nœuds voisins (I'M ALIVE) ;

– accès vers une adresse donnée : cet accès est utilisé pour tous les messages devant atteindre une destination précise, le chemin pour l'atteindre étant connu (ROUTE REPLY, OPTIM REPLY).

L'introduction de la notion de bus permet d'exprimer clairement la sémantique utlisée par les différents messages, et la vision obtenue a pour conséquence une mise en commun maximale du code d'accès au réseau.

6.3.5 Organisation générale

Une partie non-négligeable de l'organisation en terme d'architecture du code de Lilith correspond au composant près à l'architecture proposée dans le chapitre 5 et détaillée dans la figure 5.4 et les parties 5.3, 5.4 et 5.5. Ces composants de Lilith constituent l'essentiel de l'organisation du programme, mais ils ne seront pas repris ici afin d'éviter une répétition.

Bus. La notion de bus établie dans la partie 6.3.4 se traduit sous la forme d'un composant avec une interface très simple permettant d'envoyer des messages sur le bus. C'est aussi ce composant qui gère la réception des messages et qui les fait parvenir aux autres composants.

Gestion des interfaces. Le composant de gestion des interfaces fonctionne de manière très similaire à celui existant pour `umpls` et décrit dans la partie 6.2.3. Il permet de contrôler quelles interfaces sont utilisées et de gérer les ressources propres à chaque interface.

Gestion de MPLS. Ce composant est une abstraction des primitives permettant de communiquer avec une implémentation MPLS. Il permet de créer et gérer des tunnels, ainsi que d'obtenir les informations sur les tunnels existant, dont notamment les statistiques sur la quantité de trafic émis ou reçu. Le composant a évolué dans le temps : alors qu'il était spécifique à MPLS Linux pour le noyau 2.4 à l'origine, le changement vers MPLS Linux pour le noyau 2.6 puis la volonté de pouvoir utiliser `umpls` ont rendu cette abstraction générique.

Interception des paquets émis. L'interception des paquets émis est l'élément déclencheur de la découverte de routes. La première implémentation avait été réalisée avec `libpcap`, mais le problème lié au fait qu'il s'agit d'une capture et non d'une interception, décrit dans la partie 6.1.3, a vite été rencontré : une conséquence était, par exemple, qu'entre deux nœuds voisins, le premier paquet ICMP *Echo Request* était reçu deux fois, une fois directement et une seconde fois en raison de la réémission du paquet dans le tunnel MPLS créé par la suite — les paquets suivants

étaient quant à eux émis directement dans le tunnel MPLS et donc reçus une seule fois. La bibliothèque `libipq` a ensuite été utilisée.
Une particularité de Lilith est que les messages de signalisation utilisés sont les seuls paquets qui ne doivent pas être encapsulés avec MPLS, et il ne faut donc pas les intercepter. Une configuration fine du pare-feu permet de rediriger tout le trafic sauf celui-ci vers `libipq`.

Tables de données Un ensemble de structures de données permet de garder en mémoire le lien entre les connexions correspondant à des flots et les tunnels entrant et sortant MPLS — un tunnel MPLS est représenté sous la forme d'un tunnel entrant et d'un tunnel sortant au niveau d'un nœud intermédiaire. Il existe une forte interdépendance entre toutes les structures, qui complexifie leur gestion. Par exemple, la structure représentant un tunnel sortant sur un nœud intermédiaire correspond à un tunnel entrant particulier et lorsqu'un des deux tunnels disparait, l'autre doit aussi disparaitre. C'est aussi dans ces structures qu'est définie la notion de connexion en terme de code. Il s'agit donc d'une couche de base de Lilith.

Messages Lilith. Différents modules ont été créés pour gérer de manière transparente les messages de signalisation utilisés par Lilith. Ces modules effectuent notamment la traduction des messages du mode de données brutes utilisé lors de la transmission des paquets vers des structures C faciles à manipuler dans le code, et l'opération inverse. Ce composant est un outil qui simplifie l'implémentation des autres composants travaillant avec les messages.

6.4 Conclusion

Lilith est le résultat du passage à l'implémentation d'un prototype pour l'architecture proposée dans le chapitre 5. Cela a nécessité l'utilisation de bibliothèques et d'outils bas-niveau qui sont généralement peu connus dans le monde de la recherche. Cela s'explique par le fait que les simulations ne nécessitent pas d'implémentation dans un cadre réel, et sont souvent préférées aux expérimentations.

L'implémentation est une étape qui s'avère souvent plus complexe que ce qu'il parait de prime abord car il n'est pas toujours possible de concrétiser certaines idées proposées de manière théorique. En outre, des limites ou des bogues du noyau ou d'autres composants mis en jeu peuvent influencer l'architecture générale, ou remettre en cause l'implémentation réalisée. Ainsi, différents problèmes de MPLS Linux ont poussé à la création d'une nouvelle implémentation de MPLS en espace utilisateur, `umpls`.

Implémenter est un travail réellement exigeant et nécessitant du temps pour parvenir à un résultat fonctionnel. Le développement de Lilith a ainsi requis plusieurs mois de travail avant de parvenir à un programme à la fois fonctionnel et souple, permettant de réaliser aisément tout type d'expérimentations. L'outil SLOCCount [115] de David A. Wheeler a été utilisé pour analyser le code de `umpls` et de Lilith. L'analyse indique qu'il existe 5606 lignes de code effectives pour 6778 lignes réelles pour `umpls` et 7799 lignes de code effectives pour 10815 lignes réelles pour Lilith.

Chapitre 7

Expérimentations avec Lilith

> Sallah, I said *no* camels. That's *five* camels. Can't you count ?
>
> Indiana Jones, *Indiana Jones and the Last Crusade*

Le développement de Lilith a permis de faire évoluer assez tôt l'architecture proposée afin de l'adapter avec une démarche de développement itératif pour répondre à différents problèmes. Mais le prototype a aussi servi à valider certaines idées et à montrer qu'elles étaient réalisables. Différents tests et expérimentations ont été utilisés à cette fin pendant toute la durée des travaux.

Dans un premier temps, les tests avaient pour objectif de vérifier le bon fonctionnement de Lilith, et les performances n'étaient donc pas un aspect essentiel. User-Mode Linux, présenté dans la partie 7.1, s'est révélé comme un outil extrêmement puissant durant cette phase, puisqu'un matériel limité à un ordinateur a suffi pour tous ces tests.

Par la suite, pour des mesures plus avancées, un CD-ROM basé sur Linux LiveCD Router [116] a été réalisé afin de n'avoir qu'à démarrer des ordinateurs avec ce CD-ROM pour disposer de Lilith. Plusieurs expérimentations réelles ont ainsi été réalisées, notamment pour mesurer des performances ou vérifier le comportement de Lilith dans différentes conditions. Néanmoins, certaines limites pour les expérimentations sont rapidement apparues : le matériel n'est disponible qu'en quantité limitée et ne permet pas de faire des expériences à grande échelle, et des observations impliquant une mobilité avec plus d'un ordinateur connecté en sans fil créent des difficultés logistiques non négligeables.

Certaines expérimentations à plus grande échelle ont été réalisées sur Emulab [117], mais des problèmes de stabilité sur cette plate-forme — notamment des blocages intempestifs des nœuds, probablement liés aux versions utilisées du système d'exploitation — n'ont pas permis de parvenir à des résultats concluants.

7.1 User-Mode Linux

Une grande difficulté liée au développement d'un prototype est qu'il faut pouvoir tester régulièrement le prototype. Il est généralement rare de disposer en permanence du matériel nécessaire, et il s'avère nécessaire de contourner cette difficulté. User-Mode Linux [118] permet de résoudre ce problème proprement en fournissant une machine virtuelle aisément exécutable sur tout ordinateur. Ce dernier joue alors le rôle d'hôte. C'est donc une solution adaptée pour expérimenter avec un logiciel pouvant avoir un impact sur la stabilité du système sans courir de réel risque.

Une caractéristique de User-Mode Linux est qu'il s'agit en réalité d'une architecture particulière pour le noyau Linux, au même titre que 386, PowerPC ou x86-64. Le noyau compilé est en réalité un programme qu'il suffit d'exécuter. Il est donc possible de modifier aisément le noyau Linux et d'utiliser cette version modifiée dans la machine virtuelle. Ceci est particulièrement intéressant car MPLS Linux, décrit dans la partie 6.2.1, n'existe que sous la forme d'un patch pour le noyau Linux. En outre, cela signifie que tout le trafic réseau de la machine virtuelle passe par la vraie pile réseau du noyau, sans modification, garantissant un comportement fidèle.

Il est évidemment possible d'éxecuter plusieurs machines virtuelles User-Mode Linux sur un unique ordinateur, et de les relier en réseau avec une topologie que l'on détermine ; un processus extérieur permet cette mise en réseau en jouant le rôle de commutateur. Il est en outre possible de placer les machines virtuelles sur le réseau réel auquel l'ordinateur hôte est connecté — celui-ci joue alors le rôle de pont. De très nombreuses configurations réseaux peuvent ainsi être réalisées, et la limite du nombre de machines virtuelles que le matériel peut supporter peut être contournée en utilisant plusieurs ordinateurs.

Néanmoins, il ne faut pas oublier le fait que les essais du prototype ont lieu sur des machines virtuelles, sans virtualisation du matériel. Cela implique un impact non négligeable sur les performances obtenues, et il n'est donc pas possible d'utiliser des mesures effectuées ainsi pour en tirer des conclusions sur les performances dans l'absolu. Une comparaison de deux séries de mesures effectuées dans ce contexte peuvent néanmoins être comparées.

Il est à noter qu'avec des ordinateurs relativement récents, il est désormais possible de tirer profit de la virtualisation avec des outils comme VMware [119], Xen [120] ou encore KVM [121]. Ces outils sont plus performants, mais n'étaient pas disponibles au moment des travaux.

7.2 Performances de MPLS

Avant de mesurer expérimentalement les performances de Lilith, il convient de connaître l'impact de l'implémentation MPLS utilisée. En effet, le coût en perfor-

mance de l'architecture proposée est à partager entre MPLS et Lilith en tant que tel. Les performances des différentes implémentations de MPLS ont donc été mesurées lors de différents tests.

La figure 7.1 illustre le réseau utilisé pour ces tests. Il s'agit d'un réseau filaire composé de quatre nœuds connectés par un répéteur — le canal est alors partagé par tous les nœuds — ou un commutateur — des transmissions simultanées peuvent avoir lieu sur différents ports — Ethernet 100Mb/s. Une expérimentation sur réseau filaire a été préférée à une expérimentation sur réseau sans fil car l'objectif est ici de mesurer les performances brutes des implémentation MPLS et il est donc important de maximiser la bande passante et de limiter les aléas, qui auraient été importants avec un réseau sans fil, causés par exemples par le problème des terminaux cachés. Le répéteur Ethernet fonctionne de manière similaire à un medium partagé sans fil.

Les nœuds A et D communiquent entre eux, pendant que B et C jouent le rôle de nœuds intermédiaires. Dans cette expérience, l'interconnexion se situe au niveau 2, des règles de filtrage qui ne permettent au nœuds de communiquer qu'avec leurs voisins ont été mises en place, afin d'émuler le comportement d'un réseau à sauts multiples.

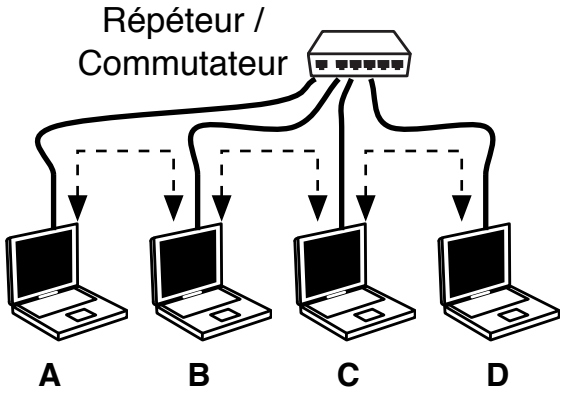

FIG. 7.1 – Réseau utilisé pour les mesures de performances

L'objectif est d'évaluer la surcharge due à MPLS par rapport à un acheminement IP purement statique. Les différentes mesures ont été réalisés dans les condition suivantes :

- acheminement IP *via* une route statique traversant les quatre nœuds : c'est le cas de référence car il mesure les performances lorsque l'interconnexion a lieu au niveau 3 ;
- acheminement IP comme dans le cas précédent, mais tous les paquets passent en outre par un programme en espace utilisateur : ce cas permet de comparer

l'implémentation en espace utilisateur de MPLS avec l'acheminement IP. Le code utilisé pour le programme est disponible dans l'annexe B ; il s'agit d'un exemple simple montrant comment utiliser `libipq` ;
– acheminement MPLS, pour les différents implémentations : des chemins sont alors établis manuellement afin de permettre toutes les communications nécessaires.

7.2.1 Mesures de délai

À l'aide de `ping`, les temps aller-retour pour différentes tailles de paquets ont été mesurés.

TAB. 7.1 – Comparaison du temps aller-retour pour les implémentations MPLS

Cas	IP	IP utilisateur	MPLS Linux	scapy-mpls	`umpls`
Ping moyen (ms) Paquet de 64 octets	0.3	0.3	0.3	93	0.4
Ping moyen (ms) Paquet de 1000 octets	0.8	0.8	0.8	95	0.8

La première conclusion qui s'impose à la vue des résultats présentés dans le tableau 7.1 est que l'implémentation de MPLS scapy-mpls n'est pas utilisable. Un simple `ping` nécessite presque un dixième de seconde : il s'agit d'une durée très longue pour un simple test. Ce problème de performance flagrant s'explique par le fait que scapy-mpls est de très loin non optimisé, et ne peut pas l'être, comme expliqué dans la partie 6.2.2. Cette simple expérimentation élimine donc déjà l'utilisation de scapy-mpls.

En ce qui concerne les deux autres implémentations de MPLS, on peut constater que les performances sont quasiment identiques à celles obtenues dans le cas du simple acheminement IP. `umpls` s'avère légèrement plus lent, et cela peut s'expliquer par le traitement plus important qui a lieu en espace utilisateur par rapport à de l'acheminement IP, qui s'ajoute au temps nécessaire pour le changement de contexte lors du passage du mode noyau au mode espace utilisateur, avec les transferts de données que ce changement de contexte implique.

Ces performances ne constituent pas une surprise car il n'y a pas de réelle raison d'observer un ralentissement à cause de MPLS en ce qui concerne les délais : de manière théorique, le traitement d'un paquet MPLS est plus simple que le traitement d'un paquet IP et l'en-tête MPLS est suffisamment petit pour n'avoir qu'un impact négligeable sur le délai de transmission du paquet.

7.2.2 Mesures de débit

L'utilitaire `netperf` [122] a été utilisé pour mesurer le débit entre les nœuds A et D. Le tableau 7.2 présentent les résultats.

TAB. 7.2 – Comparaison du débit pour les implémentations MPLS

Cas	IP	IP utilisateur	MPLS Linux	`umpls`
Débit (Mb/s)	22.6	22.5	22.5	22.4
Connexion par un répéteur	*(100%)*	*(100%)*	*(99.6%)*	*(99.1%)*
Débit (Mb/s)	91.4	91.4	91.1	91.0
Connexion par un commutateur	*(100%)*	*(100%)*	*(99.7%)*	*(99.6%)*

On constate que le débit est à peine moindre avec MPLS qu'avec l'acheminement IP puisqu'il n'y a jamais plus de 1% d'écart dans le débit observé. Le faible écart peut tout de même être expliqué : l'en-tête MPLS diminue la charge utile des données transportées, et donc le débit utile. En outre, dans le cas d'`umpls`, les changements de contexte entre le mode noyau et le mode espace utilisateur ont encore certainement un léger impact.

Une information qui n'apparaît pas dans ces mesures est qu'avec `umpls`, le processeur, un Pentium IV 1.6 GHz, était utilisé à plus de 90%, en raison du nombre excessivement important de changements de contexte par seconde.

7.3 Performances de Lilith

Les mesures précédentes ont permis de vérifier que MPLS peut être utilisé sans impact majeur sur les performances par rapport à un acheminement IP statique. L'architecture pour les réseaux de bordure spontanés ne souffrira donc pas de ce choix. Il faut désormais s'intéresser à l'impact de l'architecture elle-même sur les performances, et vérifier que certaines de ses caractéristiques apportent un réel intérêt.

7.3.1 Mesures de délai

La même expérimentation que celle présentée dans la partie 7.2.1 est réalisée, mais avec chaque nœud utilisant Lilith et MPLS Linux. Les mesures sont présentées dans le tableau 7.3.

On constate immédiatement que le temps aller-retour du premier `ping` est particulièrement long avec Lilith. En effet, dans ce cas, un certain nombre d'évènements est déclenché. En voici la liste pour le paquet ICMP *Echo Request* émis par

TAB. 7.3 – Mesures de délai avec Lilith

Cas	IP	Lilith
Premier ping (ms) Cache ARP vide, Paquet de 64 octets	0.9	6.0
Ping moyen (ms) Paquet de 64 octets	0.3	0.5
Ping moyen (ms) Paquet de 1000 octets	0.8	1.0

A à destination de D — l'envoi de la réponse ICMP *Echo Reply* par D déclenche une suite d'évènements similaire :

- A intercepte le paquet ICMP *Echo Request* et regarde si un tunnel existe pour ce flot ; comme ce n'est pas le cas, le paquet est mis en tampon temporairement jusqu'à ce qu'un chemin à destination de D soit instancié ;
- A lance le mécanisme de découverte de routes avec pour cible D ; une requête combinant les messages ROUTE REQUEST et PATH REQUEST pour D est donc envoyée par inondation ;
- B reçoit la requête et la retransmet ;
- C reçoit la requête et la retransmet ;
- D reçoit la requête et lance l'établissement d'un tunnel en envoyant un message combinant ROUTE REPLY et PATH REPLY à A ;
- la réponse passe par C puis par B, qui allouent les ressources locales nécessaires au fonctionnement du tunnel ;
- A reçoit la réponse, finalise la création du tunnel, ce qui instancie un chemin vers D ; A envoie alors dans le tunnel le paquet ICMP *Echo Request* qui avait été mis en tampon.

Le temps aller-retour du premier `ping` en acheminement IP statique est lui aussi plus long que le temps aller-retour moyen. Cela s'explique par le temps requis pour les requêtes ARP sur chaque lien que doit traverser le paquet ; on peut noter que le paquet ICMP *Echo Reply* ne nécessite pas de telles requêtes ARP car le cache ARP est d'ores et déjà rempli. L'impact sur le délai reste néanmoins moins important que dans le cas où Lilith est utilisé. Lilith paie ici deux fois le prix de l'instanciation réactive d'un chemin : une fois pour l'aller et une fois pour le retour. Il est légitime de se demander s'il ne conviendrait pas de créer automatiquement un tunnel retour, et ainsi de créer des connexions bidirectionnelles. Néanmoins, la valeur obtenue de 6 ms reste raisonnable.

Par la suite, les temps moyens observés montrent que la différence entre l'acheminement IP et l'acheminement en utilisant Lilith est de 0.2 ms, quelle que soit la taille des données émises. Il n'y a plus de délai induit par l'instanciation des chemins et les paquets sont émis directement dans les tunnels. La différence de 0.2 ms

s'explique par le travail effectué en espace utilisateur pour chaque paquet : le module d'interception des paquets émis entre en jeu pour chaque nœud car `netfilter` ne permet pas d'autoriser le trafic MPLS avec une simple règle, et c'est donc à ce module d'effectuer cette tâche. Un léger délai similaire avait aussi été observé pour `umpls` dans la partie 7.2.1. 0.2 ms est cependant un délai négligeable en terme de performances.

7.3.2 Mesures de débit

L'impact de Lilith sur les performances en terme de débit a ensuite été mesuré avec la même expérimentation que celle présentée dans la partie 7.2.2, chaque nœud utilisant désormais Lilith complémenté par MPLS Linux.

TAB. 7.4 – Mesures de débit avec Lilith

Cas	IP	Lilith
Débit (Mb/s)	22.6	22.3
Connexion par un répéteur	*(100%)*	*(98.7%)*
Débit (Mb/s)	91.4	91.0
Connexion par un commutateur	*(100%)*	*(99.6%)*

Le tableau 7.4 détaille les résultats obtenus. On peut observer que le débit est à peine diminué par rapport à l'acheminement IP statique : cette diminution est de 1.3% lorsqu'un répéteur est utilisé et de 0.4% lorsqu'un commutateur est utilisé. Comme vu dans la partie 7.2.2, l'utilisation de MPLS, en raison de la surcharge causée par les en-têtes, participate à la diminution de performances. Un autre facteur entrant en jeu avec Lilith est l'émission des messages I'M ALIVE et l'optimisation des routes en arrière-plan, qui peuvent avoir un effet mineur sur l'utilisation de la bande passante. Les performances observées n'en restent pas moins comparables à celles obtenues avec un acheminement IP statique et le débit n'est qu'à peine affecté par Lilith.

7.3.3 Impact des chemins multiples

Une caractéristique intéressante de l'architecture proposée est qu'elle permet l'utilisation de chemins multiples pour différents flots d'une même source à une même destination. Cette caractéristique peut permettre d'améliorer les performances observées dans différentes conditions. La figure 7.2 illustre un cas pour lequel l'utilisation de chemins multiples peut s'avérer bénéfique. En effet, il existe deux chemins entre A et D : le chemin 1 passant par le nœud B et le chemin 2 passant par le nœud C. `netperf` est utilisé entre A et D pour générer un important trafic

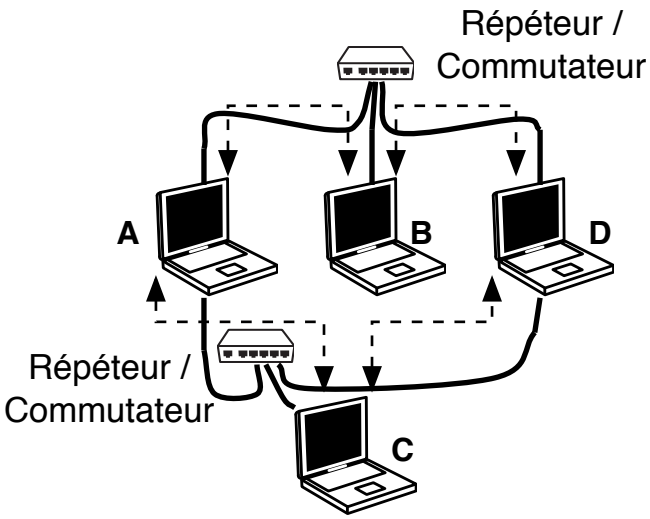

FIG. 7.2 – Réseau utilisé pour l'expérimentation des chemins multiples

TAB. 7.5 – Performances sur des routes multiples

Chemin	Répéteur / Commutateur	Temps aller-retour moyen (ms)	Écart-type (ms)
1	Switch	5.3	0.08
2	—	0.5	0.07
1	Commutateur	12.0	3.55
2	—	0.3	0.07

TCP sur le chemin 1. On mesure ensuite le temps aller-retour sur chacun des deux chemins en utilisant `ping`.

Le tableau 7.5 présente les résultats des mesures réalisées. On observe, avec peu de surprise, l'impact important du trafic TCP généré par `netperf` sur le temps aller-retour pour le chemin 1. Le temps aller-retour sur le chemin 2 est quant à lui épargné de tout impact : il est dix fois moins important que celui pour le chemin 2.

Bien que l'exemple donné par cette expérimentation est extrême — dans des conditions réelles, il est probable que d'autres trafics entrent en jeu et rendent les résultats moins flagrants —, il montre néanmoins que la possibilité d'utiliser de multiples chemins entre deux nœuds peut s'avérer particulièrement intéressante. Un acheminement IP standard ne permet pas d'obtenir cette fonctionnalité car il n'existe qu'une route par destination et il n'est donc pas possible de choisir les routes utilisées pour chaque flot.

7.3.4 Optimisation et chemin de secours

Un élément primordial de l'architecture qu'implémente Lilith est la maintenance des connexions utilisées, et la forte volonté de garantir de bonnes performances pour les connexions existantes. Deux mécanismes mis en œuvre dans cette optique sont les chemins de secours et le processus d'optimisation cherchant de meilleures routes en arrière-plan.

Une expérimentation a été réalisée pour mesurer l'impact de ce comportement. Le scenario présenté dans la figure 7.3 a pour objectif d'illustrer le comportement dynamique de Lilith dans une configuration avec plusieurs routes possibles entre une même source et une même destination, lorsque des ruptures et des apparitions de lien interviennent.

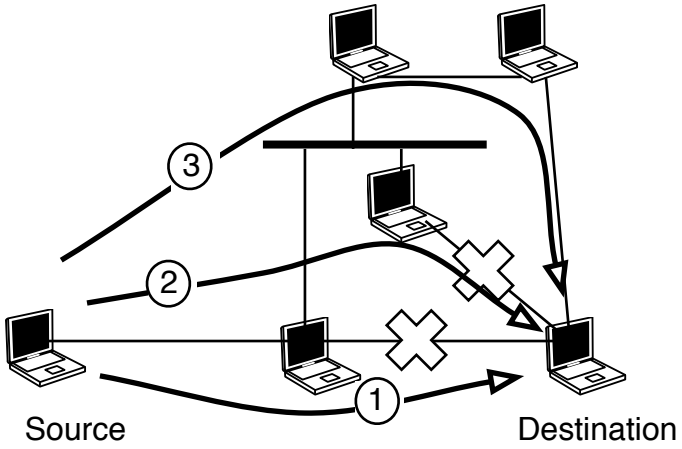

FIG. 7.3 – Réseau utilisé pour l'expérimentation des chemins de secours

Dans cette expérimentation, trois chemins existent entre la source et la destination : le chemin 1 est optimal, le chemin 2 a un saut de plus que le chemin 1 et le chemin 3 est le plus long. Le nœud source émet un trafic continu vers la destination. Après un premier temps d'attente, le chemin 1 est interrompu ; après un autre temps d'attente, le chemin 2 est lui aussi interrompu ; enfin, le chemin 1 est rétabli. La figure 7.4 présente les temps aller-retour (en ms) mesurés.

On peut tout d'abord observer un premier phénomène : les temps aller-retour des premiers paquets n'apparaissent pas dans les résultats. En réalité, leur valeur est tellement importante qu'il faudrait changer d'échelle pour les observer. Cela s'explique simplement par le fait que pour les premiers paquets, le nœud source doit trouver un chemin vers la destination et l'instancier. Pendant le délai causé par ces étapes, les paquets sont mis en tampon. Lorsqu'un chemin vers la destination

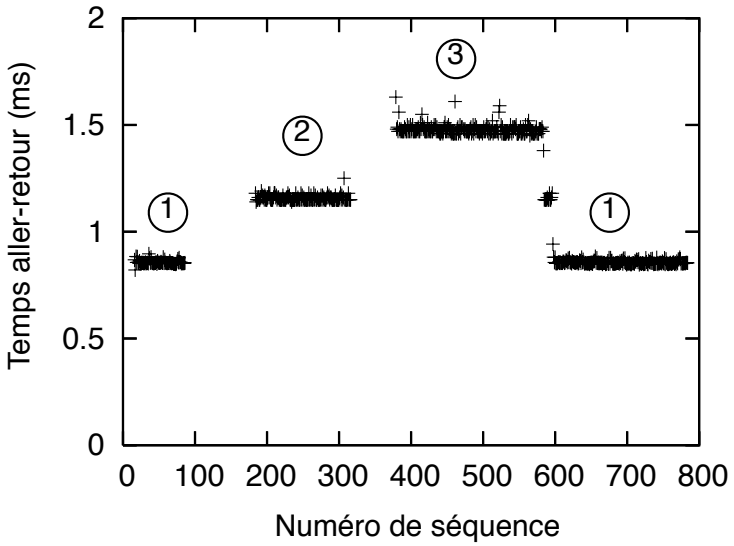

FIG. 7.4 – Temps aller-retour mesuré

est instancié, ces paquets sont alors émis dans le tunnel correspondant. Ensuite, le nœud destination reçoit les premiers paquets et doit effectuer le même travail pour pouvoir envoyer la réponse. C'est donc le délai induit par tout le processus de découverte de chemin qui crée ce phénomène.

La seconde observation est que les délais se placent principalement sur quatre palliers, correspondant à trois valeurs moyennes d'aller-retour. Le premier pallier correspond à l'utilisation du chemin 1. Lorsque celui-ci est interrompu, on passe à un second pallier avec une valeur plus élevée, correspondant au chemin 2 ; il en va de même lors de l'interruption du chemin 2, qui a pour conséquence le troisième pallier avec une valeur encore plus élevée. Enfin, le rétablissement du chemin 1 permet de retourner à un faible temps aller-retour. Le comportement que traduisent ces valeurs est le suivant :

– lors du démarrage de l'expérimentation, le nœud source cherche des routes vers la destination et en trouve trois. Le chemin 1 est le plus court chemin et est donc utilisé comme chemin principal et le chemin 2 est gardé comme chemin de secours ;
– à la réception des premiers paquets, le nœud destination effectue le même travail avec les mêmes résultats car il doit renvoyer une réponse ;
– après l'interruption du chemin 1, des messages I'M ALIVE ne sont plus reçus et permettent de découvrir la rupture de lien qui a eu lieu. Le chemin de secours est donc utilisé et une nouvelle recherche de routes en arrière-plan

permet de trouver le chemin 3 comme chemin de secours. Cette étape a lieu à la fois pour la source et la destination ;
– après l'interruption du chemin 2, le chemin de secours, à savoir le chemin 3, devient le chemin principal. Une recherche de routes en arrière-plan est aussi lancée mais ne donne aucun résultat. Cette étape a lieu à la fois pour la source et la destination ;
– lorsque le chemin 1 est rétabli, la prochaine recherche en arrière-plan pour optimiser la connexion le découvre. Dès sa découverte, le chemin est jugé meilleur et donc instancié pour devenir chemin principal. Le chemin 3 redevient alors un chemin de secours. Cette étape a lieu à la fois pour la source et la destination.

On remarque que des pertes de paquets ont lieu à chaque interruption de chemin. L'interruption étant brutale, il n'est pas possible de la prévoir et de détecter par avance qu'un changement de chemin sera nécessaire. En outre, il existe un délai avant de pouvoir détecter que le chemin a été interrompu. Un réel gain serait observé à ce niveau avec une approche inter-couches permettant, si possible, de prévoir une rupture de lien, ou du moins de la détecter rapidement.

Enfin, il est intéressant de noter le temps aller-retour qu'un ensemble de point marque lorsque le chemin 1 est rétabli. Cette valeur correspond en réalité à l'utilisation du chemin 1 pour l'aller et du chemin 3 pour le retour — ou vice-versa : les optimisations de la connexion aller et de la connexion retour ne sont pas simultanées. Le nombre de sauts total pour l'aller-retour est alors le même que si le chemin 2 était utilisé pour l'aller et le retour, ce qui explique que la valeur soit la même que dans ce cas.

7.3.5 Expérimentations qualitatives

Après avoir fait plusieurs expériences visant à mesurer quantitativement les performances de Lilith, il est apparu intéressant de se placer au niveau d'un utilisateur normal pour tester de manière qualitative cette architecture. À l'aide d'ordinateurs portables disposés dans le laboratoire, quelques expériences ont été effectuées. La figure 7.5 illustre une des dispositions utilisées.

Différents usages ont été testés avec des logiciels non modifiés :
– téléchargement de fichiers ;
– connexion à un terminal en ligne de commande (`ssh`) ;
– appel en voix sur IP (VoIP) ;
– lecture en continu (*streaming*) de vidéo.

Quelques éléments de mobilité ont été aussi introduits avec des déplacements d'ordinateur. Les résultats ont été généralement probants puisque tout a fonctionné, dans la limite où un chemin était toujours disponible entre source et destination. Néanmoins, il y a parfois eu des problèmes de « coupures », avec les connexions

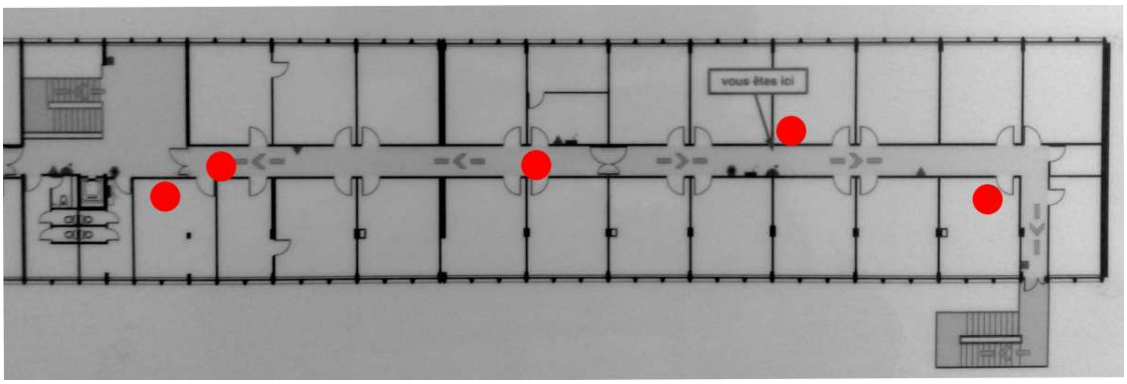

FIG. 7.5 – Placement des ordinateurs pour des expérimentations qualitatives

qui ne fonctionnaient plus. Après plusieurs tentatives pour comprendre ces phénomènes de coupures, il est apparu qu'ils apparaissaient de manière identique avec un acheminement IP statique utilisant des tables de routage manuellement modifiées. Il ne s'agissait donc pas de problèmes causés par Lilith, mais de problèmes liés à la couche physique, qui seront discutés dans la partie 8.3.

7.4 Confrontation avec la réalité

De simples tests avec un prototype permettent de rencontrer différents problèmes très concrets qui, pour la plupart, n'existent pas ou ne peuvent pas être découverts lorsqu'on réalise des simulations. L'écart entre simulation et expérimentation est alors bien réel, même au-delà des différences qui peuvent exister lorsqu'on cherche à mesurer des critères d'évaluation : par exemple, le fonctionnement d'un noyau peut avoir un impact non-négligeable sur le prototype, voire sur l'architecture proposée.

Le prototypage avec Lilith a permis de découvrir des problèmes de ce type. Si certains d'entre eux ont pu être résolus avec des solutions parfois simples et parfois plus complexes, d'autres problèmes n'ont abouti qu'à une solution incomplète bien que généralement fonctionnelle.

7.4.1 Interfaces avec une même adresse IP

Il peut être intéressant d'utiliser la même adresse IP sur toutes les interfaces des nœuds (*multihoming*). L'adresse devient alors un identifiant unique pour chaque nœud, et cela permet de gérer dans une certaine mesure la mobilité ainsi que de gérer de manière transparente l'apparition et la disparition d'interfaces réseau sur chaque nœud.

Les tout premiers tests de Lilith ont été réalisés avec une telle configuration, et ne fonctionnaient pas sous Linux. Le problème n'existait pas sous FreeBSD. Une recherche fastidieuse, incluant la lecture d'une partie du code du noyau, a permis de déterminer qu'une option de configuration était la source de ce problème. Le paramètre `/proc/sys/net/ipv4/conf/all/rp_filter` contrôle si la validation suivante est réalisée pour chaque paquet :

- si la route utilisée pour atteindre la source du paquet passe par l'interface qui a reçu le paquet, alors le paquet est accepté ;
- sinon il est ignoré.

Cette vérification par chemin inverse (*reverse path*) est généralement activée par défaut pour des raisons de sécurité afin de limiter les problèmes d'usurpation d'adresse IP. Or, avec deux interfaces sur le même sous-réseau, une seule des deux interfaces permet de passer cette validation avec succès et la seconde interface s'avère alors inutilisable. Il est donc nécessaire de désactiver cette validation.

7.4.2 Décapsulation de paquets sur la bonne interface

Certains programmes, voire le noyau, peuvent utiliser la couche réseau de manière assez bas niveau, et notamment utiliser l'interface sur laquelle a été reçu un paquet comme une information pour le traitement du paquet.

Pour les paquets envoyés en unidiffusion (*unicast*), il est donc important qu'un paquet encapsulé dans MPLS à destination d'une adresse IP soit reçu sur l'interface ayant cette adresse IP. Les implémentations de MPLS gèrent cet aspect nativement sans grande difficulté.

Ce problème est aussi important pour les paquets de diffusion. À l'origine, Lilith envoyait tous les paquets de diffusion encapsulés dans un message BROADCAST sur l'interface locale (*loopback*), ce qui était suffisant d'après les tests effectués. Il s'est avéré que cette méthode a posé problème dans le cas de DHCP, comme le montre la partie 7.4.4, mais aussi pour les paquets ICMP. En effet, le noyau Linux vérifie que le paquet ICMP a été reçu sur une interface qui correspond à l'adresse IP destination du paquet — ou dont l'adresse de diffusion est celle de l'adresse IP destination du paquet dans le cas d'un paquet de diffusion. L'utilisation de l'interface locale rendant ceci impossible, et l'adresse IP destination du paquet ne permettant pas de spécifier quelle interface destination utiliser lors de l'envoi d'un paquet vers soi-même, il a fallu modifier Lilith afin d'utiliser des fonctions bas niveau permettant d'envoyer une trame complète à destination de la bonne interface.

7.4.3 Envoi des paquets de diffusion à soi-même

Un comportement souvent oublié lors de l'envoi d'un paquet de diffusion sur une interface est que ce paquet est aussi reçu sur cette interface si l'adresse destination

du paquet est la même que l'adresse de diffusion de l'interface. Or, le cycle de vie d'un paquet de diffusion dans Lilith ne permettait originalement pas cela. En effet, un paquet de diffusion est intercepté puis encapsulé dans un message BROADCAST ; ce message est ensuite émis sur le bus en accès global. Or, la méthode d'accès global du bus fait en sorte que tout paquet émis n'est pas reçu par l'interface émettrice afin de ne pas recevoir des messages ROUTE REQUEST provenant de soi-même.

Certaines applications dépendant potentiellement de ce comportement, il est important de le reproduire afin de rester compatible au maximum avec le fonctionnement actuel de la pile réseau. Il était donc possible d'envoyer le paquet dans l'interface locale lors de l'encapsulation dans le message BROADCAST[1], ou de modifier le comportement du bus en accès global. La première solution a été mise en œuvre car elle est légèrement plus simple.

7.4.4 Gestion de DHCP

L'autonomie est une caractéristique importante d'un réseau de bordure spontané, et il est utile d'avoir une configuration minimale pour chaque nœud. Avoir un serveur DHCP sur le réseau permet en grande partie de faciliter ces aspects. Néanmoins, le fonctionnement même du protocole DHCP [84] crée des difficultés. Voici de manière simplifiée les différentes étapes de l'obtention par un nœud d'une adresse avec DHCP :

- le nœud envoie une requête dans un paquet de diffusion dont l'adresse source est 0.0.0.0. La requête peut contenir un champ dit « magique » permettant notamment de savoir sur quelle interface le paquet a été émis ;
- le serveur reçoit la requête et la traite. Une réponse avec le même champ « magique » est alors émise à destination du nœud ayant envoyé la requête : l'adresse MAC destination de la trame est celle du nœud — ce qui permet la réception de la réponse par le nœud — et l'adresse IP destination est celle allouée ;
- le nœud reçoit la réponse, vérifie que le champ « magique » est valide pour l'interface sur laquelle la réponse a été reçue, et si c'est le cas utilise l'adresse IP obtenue.

Un premier problème immédiat est que ceci ne peut pas fonctionner en multi-sauts en raison de la seconde étape : la réponse ne parviendra pas jusqu'au client si plusieurs sauts sont nécessaires car aucun nœud ne la retransmettra en raison de l'adresse destination MAC. Il existe cependant une option permettant au client de demander à ce que la réponse soit émise en diffusion, ce qui a pour effet de changer l'adresse MAC destination mais aussi l'adresse IP destination. Celle-ci est alors 255.255.255.255, et correspond donc à un paquet de diffusion : la réponse sera alors encapsulé dans un message BROADCAST et le client la recevra.

[1] Dans ce cas, le paquet de diffusion envoyé dans l'interface est reçu par celle-ci, et non émis par celle-ci. S'il avait été émis, une boucle d'envoi-réception du paquet aurait été créé.

Un second problème potentiel est qu'en raison du champ « magique », il faut que la réponse soit reçue sur la bonne interface par le client. Ceci est possible car Lilith envoie tout message BROADCAST dont la destination est 255.255.255.255 sur toutes les interfaces réseau.

7.4.5 Taille maximale des paquets

Un aspect qui peut facilement être oublié lorsqu'on encapsule les données est la taille maximale des paquets lors de la transmission (*Maximum Tranmission Unit*, MTU). La simple encapsulation crée de fait un changement de MTU afin que l'entête ajouté n'empêche pas la transmission des paquets.

En terme d'implémentation, deux choix sont possibles : gérer soi-même la fragmentation des paquets encapsulés ou modifier la MTU des interfaces. Fragmenter soi-même les paquets nécessite un travail non négligeable, mais a surtout le problème d'interférer avec le bon fonctionnement de la segmentation TCP. Il est donc plus intéressant de modifier la MTU des interfaces.

Un second problème lié à la MTU est qu'avec MPLS, il n'est pas possible de disposer des paquets ICMP informant qu'une fragmentation est nécessaire sur la route comme c'est le cas pour IP. Il faut donc connaître à l'avance la MTU minimale qui sera rencontrée sur un chemin utilisé, ce qui implique une signalisation supplémentaire qui n'a pas été prévue dans l'architecure proposée. Ce problème reste néanmoins rare et prendre en compte l'encapsulation suffit généralement.

Deux encapsulations existent dans Lilith : l'encapsulation par MPLS et celle par le message BROADCAST. Seule celle ayant l'en-tête le plus long doit être prise en compte. L'implémentation actuelle fait qu'il s'agit de l'en-tête des messages BROADCAST.

7.5 Conclusion

Le choix de développer le prototype Lilith a permis d'observer dans un environnement réel l'architecture proposée dans le chapitre 5. Les difficultés matérielles qu'impliquent des expérimentations sont néanmoins la cause de restrictions sur celles-ci, notamment en ce qui concerne le nombre de nœuds présents sur le réseau.

Les expérimentations effectuées permettent néanmoins de montrer clairement certains des avantages de cette architecture ainsi que de mesurer le faible impact que cette dernière induit sur le réseau. Les performances des nœuds ne sont pas affectées par l'utilisation de Lilith ; en outre, les nœuds parviennent à tirer profit en situation réelle des routes multiples, des routes de secours ou encore de l'optimisation effectuée en arrière-plan.

Enfin, les tests effectués au fur et à mesure du développement pour vérifier le bon fonctionnement de Lilith et des concepts de l'architecture ont permis de confronter la théorie à la réalité et de découvrir certains problèmes qui peuvent être complexes à résoudre en pratique. Le problème de la MTU est par exemple un problème qui existe dès qu'il y a encapsulation du paquet émis, et pourtant extrêmement peu de propositions utilisant un en-tête supplémentaire abordent ce sujet.

Chapitre 8

Perspectives

> I'll be back.
>
> The Terminator, *The Terminator*

Proposer une architecture pour les réseaux de bordure spontanés implique de s'intéresser à un domaine assez large, et les travaux réalisés n'ont pas permis d'étudier tous les aspects en profondeur. Cependant, l'expérience qu'ont offert le développement de l'architecture ainsi que le prototypage avec Lilith offre la possibilité de prendre du recul.

De nombreux travaux restent possibles au sein de l'architecture même qui a été proposée, mais des pistes dans d'autres cadres sont aussi apparues au fur et à mesure des différentes avancées. Ces pistes n'ont pas pu être suivies, mais elles peuvent faire l'objet de futurs travaux.

8.1 Limitations et améliorations envisageables de l'architecture

Entre la première vision de l'architecture et celle qui a été implémentée par Lilith, de nombreux changements ont eu lieu. Différents problèmes ont en effet été rencontrés au cours du temps et ont été résolus et des difficultés liées purement à l'implémentation ont pu être dépassées. Cependant, il reste différentes améliorations et optimisations possibles, qui sont liées à certaines limitations dans l'architecture actuelle.

8.1.1 Dépendance envers IP

MPLS ne différencie pas les protocoles et permet donc d'encapsuler tout protocole de niveau 3 ; néanmoins, le fonctionnement concret fait qu'en général, il est supposé que le protocole est IP dans sa version 4 ou 6. En outre, l'implémentation actuelle de Lilith présuppose à de nombreux endroits l'utilisation d'IP, voire parfois même plus spécifiquement d'IPv4. L'utilisation d'un autre protocole nécessiterait donc un travail non négligeable.

Cette dépendance envers IP n'est pas particulièrement gênante étant donnée la prédominance d'IP aujourd'hui, mais il peut s'avérer intéressant de s'en libérer dans l'éventualité d'un changement radical de la pile réseau. Un tel changement n'aurait *a priori* pas lieu sur tous les réseaux utilisés, mais peut être considéré dans la recherche. On peut raisonnablement penser que dans le cas d'un tel changement, l'architecture proposée ici ne serait plus adaptée — l'architecture est basée entre autres sur des contraintes de compatibilité — et nécessiterait de nombreux ajustements, donc la limitation due à la dépendance envers IP reste un élément mineur.

8.1.2 Gestion de la diffusion

La méthode de diffusion utilisée dans l'architecture proposée est l'inondation. La diffusion par inondation a un coût important en terme d'utilisation réseau, mais elle permet de tirer profit des redondances pour combler les pertes qui sont liées à la mauvaise qualité des liens dans un réseau chargé. D'autres approches peuvent néanmoins s'avérer être intéressantes car l'utilisation de protocoles d'autoconfiguration et de découverte de service augmente sensiblement le nombre de paquets de diffusion.

Parmi les autres approches possibles, l'utilisation de relais multi-points [56] comme le fait OLSR est certainement une des premières à explorer. Ces relais multi-points permettent d'atteindre l'ensemble des nœuds du réseau tout en limitant le nombre d'émissions du paquet émis en diffusion. On peut raisonnablement envisager de coupler les messages nécessaires à la découverte des relais multi-points avec les messages I'M ALIVE. la seule différence impliquée par ce changement réside dans le fait qu'un nœud par lequel aucun tunnel ne passe n'émettait pas de tels messages auparavant et devrait alors en émettre.

Enfin, en l'état actuel, il n'y a aucune gestion spécifique de la multidiffusion dans l'architecture proposée, et donc celle-ci est gérée comme une diffusion normale. Étant donné qu'un réseau de bordure spontané est considéré comme un réseau local et qu'il est de taille limitée, cette absence de gestion de la multidiffusion ne pose pas un problème majeur. Il peut cependant être intéressant d'approfondir le sujet.

8.1.3 Choix d'une métrique

Comme indiqué dans la partie 6.3.2, la métrique actuellement mise en œuvre dans Lilith utilise une approche très simple. Les limites imposées par le matériel actuel fait qu'il n'est pas possible de prendre en compte des informations provenant de couches bas-niveau dans le calcul de la métrique. Cela reste théoriquement possible, et il faut donc déterminer quels critères apportent un intérêt significatif. De nombreux travaux existent déjà dans le domaine [123, 124, 125, 126], et la non-émergeance d'une solution principale peut laisser penser qu'il n'y a aucune métrique optimale. Ceci reste une possibilité à envisager car des réseaux de différentes natures existent, même au sein des réseaux de bordure spontanés.

Il faut aussi remarquer que la métrique d'un chemin est calculée en fonction des informations fournies par les différents nœuds sur le chemin, et généralement les calculs de métrique sur un réseau multi-sauts impliquent une confiance dans tous les nœuds du réseau. Cette confiance doit être évaluée, et peut-être remise en question. La partie 8.1.4 discute de ce type de problèmes.

8.1.4 Sécurité

Le problème de la sécurité dans le domaine des réseaux est un problème essentiel car les données transmises sont potentiellement sensibles et il est souvent aisé de les intercepter ou de les manipuler, notamment dans un réseau possédant une composante sans fil, ou pour lequel tout nœud peut être routeur. Deux aspects sont ici importants : la sécurité du routage et la sécurité des données transmises.

Comme la majorité des travaux réseaux dans ce domaine, l'architecture proposée actuellement présuppose que tous les nœuds coopèrent et sont « de bonne foi ». Cette décision est, en quelque sorte, culturelle car la sécurité est souvent considérée comme auxiliaire dans un premier temps, et ajoutée par la suite. Il est en partie possible d'intégrer certains aspects de sécurité dans l'architecture proposée dans le cadre de ces travaux.

Il est possible de considérer que la sécurité dans un réseau de bordure spontané utilisant Lilith est plus ou moins équivalente à celle existant dans un réseau normale ou sur Internet. C'est effectivement en partie vrai puisque les mêmes applications sont utilisées de la même façon. Pourtant, ajouter un nœud au réseau est beaucoup moins complexe dans un réseau de bordure spontané car le réseau a pour but de gérer facilement l'apparition des nouveaux nœuds. Une première piste serait donc le contrôle d'accès au réseau. La contrainte de non-centralisation peut néanmoins rendre cet aspect assez difficile, puisque la décision doit être distribuée et prise en prenant en compte la sécurité.

En ce qui concerne le routage proprement dit, les différents problèmes pouvant être rencontrés sont :

- l'absence de coopération d'un nœud : ce cas n'est pas particulièrement grave, mais il peut s'apparenter au cas suivant ;
- un nœud qui cherche à tromper d'autres nœuds pour maximiser ses capacités sur le réseau : il est tout d'abord important de savoir comment caractériser un tel comportement car rien ne peut être fait si aucun autre nœud ne s'en rend compte. Pour minimiser le problème, il est possible de s'inspirer de la théorie du jeu, et notamment du dilemme répété du prisonnier [127]. La stratégie la plus efficace sur le long terme, même si elle n'est pas toujours optimale, est « œil pour œil », qui consiste à reproduire le comportement de l'autre : si l'adversaire coopère, alors il faut coopérer avec lui et s'il ne coopère pas, il ne faut pas coopérer avec lui. Transposer cette stratégie dans les réseaux est une solution qui peut mériter une étude approfondie ;
- un nœud qui n'est pas intéressé par utiliser le réseau, mais qui cherche simplement à empêcher son utilisation : il convient probablement ici de disposer d'un mécanisme de réputation permettant d'éviter de tels nœuds. Cependant, si le réseau est en partie sans fil, il est aisé d'utiliser le medium en permanence et de bloquer la partie sans fil du réseau. Il n'y a pas de réelle solution dans ce cas.

La sécurité des données transmises pourraient être garantie en intégrant une signature garantissant l'intégrité des données et l'authentification de l'émetteur, et si besoin est, un chiffrement des données. L'encapsulation par MPLS crée une couche qui permettrait de réaliser ces opérations de manière transparente. Il convient néanmoins de définir comment mettre en place une infrastructure de clés qui serait alors nécessaire. La difficulté majeure réside dans le fait que cette infrastructure doit pouvoir fonctionner de manière décentralisée. Il s'agit d'un sujet d'étude extrêmement complexe car la moindre erreur peut supprimer toute sécurité.

8.2 Autoconfiguration sur un réseau multi-sauts

L'autoconfiguration permet à différents nœuds se regroupant en réseau de pouvoir communiquer entre eux sans nécessiter une infrastructure particulière. Une des premières étapes de l'autoconfiguration est l'obtention d'une adresse IP. Une méthode à la fois simple et puissante pour cela est définie dans la RFC 3927 [70]. Il s'agit de la méthode utilisée par Zeroconf, dont le fonctionnement est le suivant :

- le nœud choisit une adresse dans le réseau 169.254/16 de manière pseudo-aléatoire, ou si c'est possible, choisit l'adresse qu'il avait utilisée lors de la précédente configuration ;
- le nœud envoie un message sur le réseau pour demander l'autorisation d'utiliser cette adresse. La RFC ne définit le fonctionnement de cette étape que dans le cas d'un réseau local 802 : une requête ARP pour l'adresse choisie est envoyée ; l'absence de réponse signifie que le nœud peut utiliser l'adresse ; en cas de réponse, le nœud recommence le processus avec une autre adresse ;

– le nœud envoie enfin un message pour annoncer qu'il utilise l'adresse choisie. Un paquet ARP est utilisé à cette fin.

Un mécanisme de détection des conflits est aussi disponible.

La spécification de cette méthode dépend fortement de ARP, et cela la rend donc inutilisable dans le cadre d'un réseau multi-sauts, sauf si les requêtes ARP sont retransmises. Il est néanmoins envisageable de la modifier pour qu'elle utilise des paquets IP, ce qui la rend utilisable dans un tel environnement :

– le message envoyé dans la seconde étape est alors un paquet IP envoyé en diffusion locale — avec 255.255.255.255 comme adresse IP destination — avec pour adresse IP source l'adresse choisie dans la première étape ;
– si un nœud reçoit le message et utilise déjà l'adresse choisie, alors il envoie une réponse en diffusion pour indiquer le conflit. Le premier nœud n'ayant pas encore d'adresse IP configurée sur son interface et comme il n'est pas possible de se baser sur une adresse MAC pour l'atteindre, il est nécessaire d'envoyer la réponse en diffusion afin qu'elle lui parvienne ;
– le processus gérant l'autoconfiguration capture les paquets sur l'interface, ce qui lui permet de recevoir la réponse l'informant du conflit le cas échéant.

L'utilisation de la diffusion au niveau IP permet de ne pas être limité à un unique saut, puisque les paquets IP sont acheminés à travers les différents sauts.

Une implémentation très basique de cette modification de la RFC a été réalisée et est fonctionnelle. Des tests plus poussés sont cependant nécessaires pour vérifier le bon fonctionnement dans tous les cas, et pour observer si l'utilisation d'une telle approche fait surgir d'autres problèmes.

Au delà de l'obtention automatique d'une adresse IP, il serait particulièrement intéressant d'effectuer des mesures réelles de trafic pour observer si l'utilisation de protocoles tels que MDNS et DNS-SD ont un quelconque impact sur le réseau, notamment en terme de quantité de trafic induit.

8.3 Problèmes liés à la couche physique 802.11

Lors des expérimentations réalisées pour la partie 7.3.5, il est apparu qu'avec un simple acheminement IP configuré de manière statique, les communications entre deux nœuds étaient d'une qualité extrêmement variable en fonction de facteurs totalement extérieurs au réseau. Par exemple, il arrivait parfois que le simple passage d'une ou deux personnes dans le couloir empêchait toute communication. De manière plus inquiétante, le four micro-ondes de la cafétéria pouvoir avoir lui aussi un impact sur le délai de transmission lorsqu'il était en marche.

La figure 8.1 illustre ce phénomène. Elle correspond aux mesures du temps aller-retour tel que calculé pendant 15 minutes par `ping` pour des paquets de 1000 bits

entre deux nœuds connectés directement entre eux, donc sans sauts multiples. Pendant les dix premières minutes, malgré une variabilité non négligeable, on observe une certaine stabilité autour de 3.5 ms. Les mesures des cinq dernières minutes sont quant à elles beaucoup plus instables. La seule différence entrant en jeu est que des personnes sont allées dans le couloir, et donc indirectement entre les deux nœuds, et se sont arrêtées pour discuter.

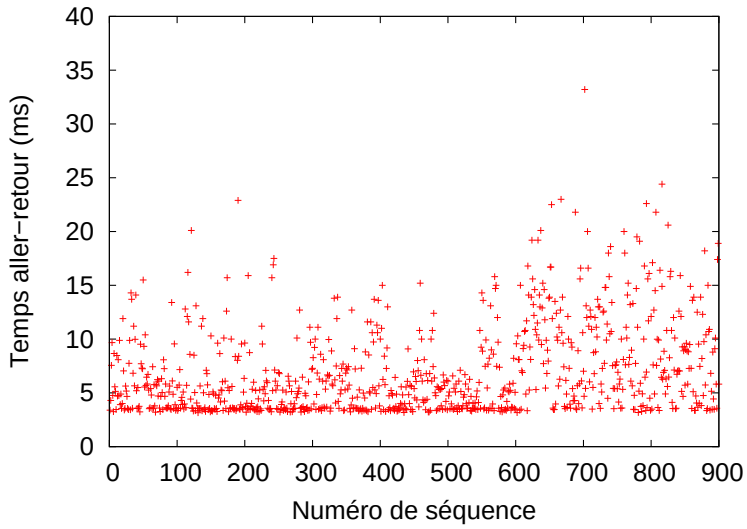

FIG. 8.1 – Illustration de la sensibilité du temps aller-retour à des éléments extérieurs

Un second constat qui a été réalisé pendant ces expérimentations est l'influence de l'implémentation de 802.11 pour les différentes cartes disponibles, d'un point de vue global (matériel et logiciel). Des mesures de temps aller-retour avec `ping` ont été effectuées entre deux nœuds distants de deux sauts, fonctionnant sous Linux. Les paquets ont une taille de 1250 bits et la MTU des interfaces réseau a été réduite à 400 bits, causant ainsi une fragmentation[1]. La figure 8.2 montre les résultats pour une carte Intersil utilisant le pilote prism54, et la figure 8.3 montre les résultats pour une carte Intel utilisant le driver ipw2100. On remarque immédiatement que les valeurs obtenues avec la carte Intersil sont beaucoup plus stables que celles obtenues avec la carte Intel : dans ce dernier cas, toutes les valeurs sont éparpillées, avec un écart-type de 7.4 ms, alors que l'écart-type pour la carte Intersil est de 3.7 ms. En outre, la carte Intersil obtient des délais en moyenne deux fois moins

[1]L'expérience visait au départ à comparer la fragmentation réalisée au niveau du noyau en raison de la MTU avec la fragmentation réalisée au niveau 802.11 par la carte. Les résultats étaient similaires, sauf pour la carte Intel ipw2100 qui obtenait un résultat légèrement meilleur avec la fragmentation au niveau 802.11.

importants que la carte Intel : la moyenne est de 9.2 ms dans le premier cas contre 21.2 ms dans le second. Il y a donc une différence de comportement flagrant entre les deux cartes.

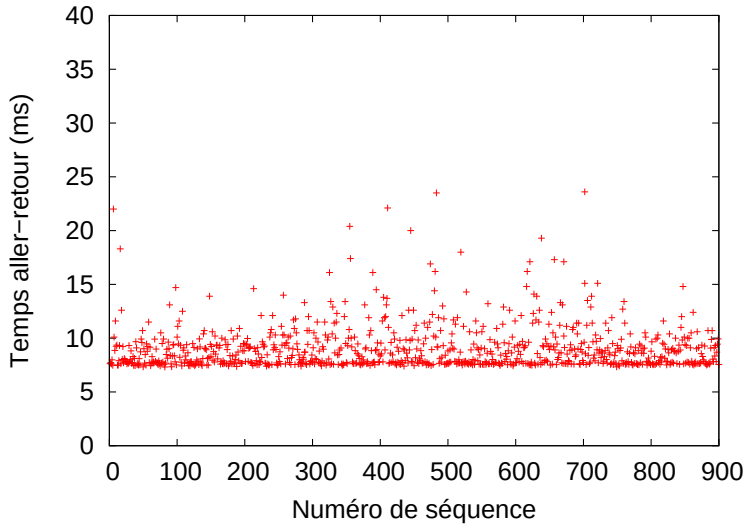

FIG. 8.2 – Temps aller-retour mesuré avec une carte Intersil

Ces quelques expériences permettent de constater que la couche physique de 802.11, mais aussi peut-être l'implémentation de 802.11 dans les cartes réseau ne sont pas exemptes de problèmes. Cela laisse donc penser qu'étudier avec plus de détails et de temps ces aspects, notamment à travers toute une série d'expérimentations pratiques, est un sujet potentiellement riche.

8.4 Les réseaux sans fil étendus

L'étude des réseaux de bordure spontanés a permis de s'intéresser de près aux différentes solutions possibles dans un réseau qui reste, au final, un réseau local. Mais le travail effectué n'en reste pas moins utile dans un tout autre cadre, beaucoup plus élargi tels que les réseaux sans fil étendus. Ces réseaux sont de très grande taille, tant en nombre de nœuds que dans la répartition géographique de ces derniers. Le cadre des réseaux sans fil étendus n'est pour l'instant pas clairement défini, et plusieurs questions doivent trouver une réponse avant de pouvoir commencer à chercher des solutions aux différents problèmes.

Par exemple, il semble beaucoup plus élégant et attrayant de considérer ces réseaux comme des réseaux totalement distribués. Cependant, la réalité nous montre

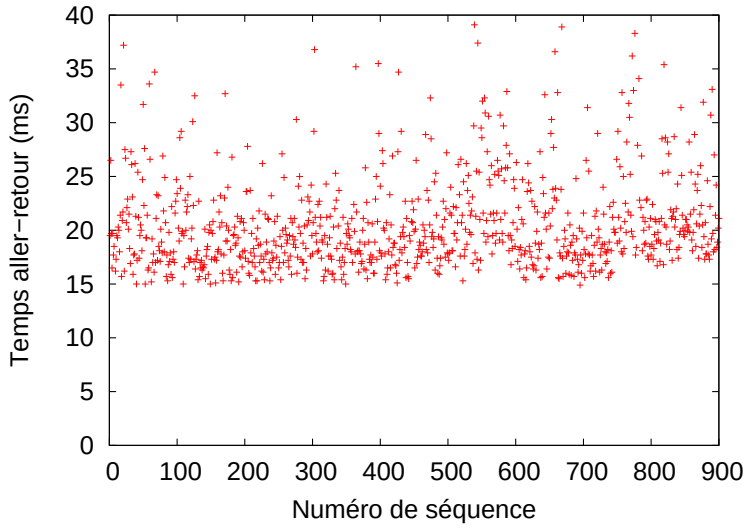

FIG. 8.3 – Temps aller-retour mesuré avec une carte Intel

qu'un changement majeur est en cours depuis quelques années : en France, de plus en plus de ménages disposent d'un modem-routeur avec des capacités sans fil fourni par le fournisseur d'accès. Il est difficile d'ignorer cette tendance, et il devient peut-être raisonnable de prendre comme acquis le fait qu'une certaine infrastructure sans fil existe grâce à ce matériel déployé un peu partout. Considérer cette infrastructure semble aller contre le choix d'une solution totalement distribuée, mais permet d'un autre côté d'ancrer la recherche dans une réalité toujours plus présente.

Il existe un lien entre réseau sans fil étendu et réseau de bordure spontané : d'un point de vue local, un réseau sans fil étendu peut être considéré comme un réseau de bordure spontané. Localement, les caractéristiques du réseau font en effet qu'il est possible d'appliquer l'expérience qui a été acquise sur les réseaux de bordure spontanés aux nœuds souhaitant communiquer entre eux. Une distinction entre communication locale et communication globale peut ainsi être introduite.

En cas d'approche distribuée des réseaux sans fil étendus, étant donnée la taille des réseaux, il semble nécessaire d'utiliser pour toutes les communications globales un routage géographique, ainsi que d'inclure dans les adresses des nœuds une composante géographique. L'adresse n'est alors plus un identifiant fixe du nœud, et il devient nécessaire de séparer ce rôle dans un identifiant distinct de l'adresse. Un service de localisation distribué s'avère nécessaire pour faire le lien entre identifiant et adresse géographique.

Comme le montre les quelques idées exposées dans cette partie, imaginer des pistes à suivre pour les réseaux sans fil étendus ne constitue pas une réelle difficulté. Mais développer les pistes pour résoudre chaque problème soulève à chaque fois de nouveaux problèmes et constitue un défi majeur.

Chapitre 9

Conclusions

> You have been a fabulous audience ! Tell you what, you're the best audience in the whole world. Take care of yourselves ! Good night, fellas ! Good night, Agrabah ! Adios, amigos !
>
> Genie, *Aladdin*

L'objectif initial de cette thèse était d'étudier la connectivité dans les réseaux sans fil. Il est rapidement apparu qu'il était possible d'élargir le cadre à des réseaux plus généraux, les réseaux de bordure spontanés. Ceci est justifié par l'omniprésence de tels réseaux autour de nous, que ce soit dans un bâtiment, à un étage, au travail ou chez soi. Ces réseaux conceptuellement locaux sont quotidiennement utilisés de manière sous-optimale et nécessitent trop de configuration de la part des utilisateurs, alors qu'ils devraient être transparents et faciliter l'utilisation des différents services existants.

Des solutions de connectivité existent pour des réseaux relativement similaires comme les réseaux sans fil ad hoc. De nombreux protocoles de routage ont ainsi été développés dans le cadre des réseau ad hoc, particulièrement dans le groupe de travail MANET. Quelques années plus tard, des approches différentes sont apparues avec des ambitions plus importantes que le simple routage, et intégrant d'autres aspects qui étaient auparavant développés séparément. Ces approches se basent sur un niveau 2.5 pour proposer de nouvelles architectures.

Afin de déterminer les besoins précis auxquels devraient répondre une solution, nous avons dans un premier temps défini plus précisément la nature des réseaux de bordure spontanés. Cela nous a permis de proposer par la suite une nouvelle architecture d'interconnexion au niveau 2.5 pour ce type de réseaux. Le choix d'un couplage d'un fonctionnement orienté connexion et d'un routage réactif apparait

comme central dans cette architecture. Cela semble néanmoins comme un choix paradoxal car les deux notions semblent se contredire. Centrer l'architecture sur les connexions et non sur les paquets est une décision qui peut aller jusqu'à choquer dans le monde de la recherche car cela peut évoquer un retour en arrière. Nous avons cependant montré que cette solution originale permet de garantir un bon fonctionnement du réseau et que les bonnes performances sont préservées, voire améliorées dans certains cas. La notion de chemin de secours, l'utilisation de chemins différents pour chaque flot ou encore l'optimisation en arrière-plan de manière continue constituent un ensemble de moyens mis en œuvre essentiels pour la préservation de la qualité des connexions.

Une part considérable du travail effectué a consisté en l'implémentation d'un prototype dans le but d'affronter les problèmes qui ne sont rencontrés que dans la réalité. Se plonger dans le concret de cette manière change nécessairement le point de vue par rapport au travail effectué, et ce nouvel angle apporte une touche particulière à l'architecture : prendre la mesure des difficultés qu'on peut rencontrer en implémentant le prototype, mais aussi pendant les tests de celui-ci, a un impact sur les idées qu'on cherche à concrétiser. Pourtant, souvent, le prototypage n'est considéré que comme une étape optionnelle dans la recherche.

Si l'on compare l'architecture proposée qu'implémente Lilith avec LUNAR et Ananas, deux autres propositions utilisant un niveau 2.5, on constate qu'il existe des similitudes dans la philosophie. Il existe en effet une volonté de virtualiser, pour les couches hautes, un réseau complexe avec de fortes contraintes en un réseau local, ainsi que la volonté de se confronter à la réalité avec un prototype. Alors que LUNAR a été créé pour la facilité d'utilisation par un utilisateur normal, et que la spécificité d'Ananas réside dans l'insertion d'une interface virtuelle pour « tromper » le système d'exploitation, notre proposition a comme particularité d'insister sur l'importance des connexions au dépens des paquets. Ces approches différentes montrent certainement que le niveau 2.5 peut apporter de nouvelles solutions et méritent d'être plus étudiées.

La travail réalisé au cours de cette thèse a permis de découvrir comment élaborer une architecture d'interconnexion, avec tous les problèmes qui se posent à chaque nouvelle étape passée. Le résultat est un ensemble cohérent, avec une philosophie particulière, qui se place dans le cadre des réseaux de bordure spontanés. Néanmoins, entrer dans les détails et parvenir à répondre aux questions a aussi eu pour effet d'ouvrir de nouvelles pistes à explorer comme expliqué dans le chapitre 8. Alors que certaines de ces pistes peuvent apparaître, probablement à tort, comme des sujets relativement simples — autoconfiguration sur un réseau multi-sauts, expérimentations pour observer les problèmes des implémentations 802.11 —, d'autres entrent immédiatemment dans la catégorie des sujets particulièrement ambitieux — sécurité dans les réseaux spontanés ou encore architecture pour les réseaux sans fil étendus. L'expérience acquise pendant la thèse aide

à prendre le mesure de l'étendue du travail qu'il reste à réaliser dans le monde toujours aussi vaste des réseaux.

Enfin, avoir travaillé sur un niveau 2.5 amène naturellement à se poser des questions sur le fonctionnement actuel des réseaux. La pile réseau actuelle fonctionne parfaitement dans le cadre d'utilisation qui existe aujourd'hui, mais n'est-elle pas non plus un frein à l'innovation ? Garder la compatibilité avec l'ensemble des applications utilisées aujourd'hui est en effet souvent devenu un des critères essentiels pour toute proposition se voulant sérieuse. Il est certes vrai qu'une migration vers une toute autre architecture se révèlerait particulièrement délicate, mais l'apparition rapide de nouveaux modes de vie comme la téléphonie mobile, ou plus récemment l'installation de matériel de connexion à Internet chez une grande partie de la population, pourrait offrir une possibilité de changement relativement centralisé. Sortir du modèle ouvre des perspectives nouvelles encore plus importantes, et l'insertion d'un niveau 2.5 au milieu d'un ensemble de couches qu'on suppose toujours indépendantes pose la question de la remise à plat de ces couches comme base de réflexion pour un nouveau modèle.

Troisième partie

Annexes

Annexe A

Bogue d'implémentation de MPLS pour Linux

> C'est quand même étrange, ces lettres que laisse le tueur. O, D, I et maintenant L. ODIL, qu'est que ca peut bien vouloir dire ? Lido ? Ça serait une danseuse ?
>
> Serge Karamazov, *La Cité de la Peur*

Un logiciel comme `tcpdump` ou `wireshark` permet d'observer le trafic transmis sur un réseau. Pour son bon fonctionnement, il doit pouvoir connaître les différentes interfaces disponibles sur l'ordinateur. L'obtention d'une liste des interfaces se fait en interrogeant le noyau *via* l'appel système `getifaddrs()` qui est implémenté par la bibliothèque libc.

Sur un système Linux, la bibliothèque libc est la glibc. Or depuis la version 2.3.3, la glibc implémente l'appel `getifaddrs()` en utilisant l'interface netlink. En effet, les développeurs du noyau Linux ont clairement fait savoir que les autres méthodes risquaient de disparaitre. Une socket netlink, de la famille `NETLINK_ROUTE` est donc ouverte et deux requêtes `RTM_GETLINK` et `RTM_GETADDR` sont envoyées pour le protocole `AF_UNSPEC`, ce qui signifie pour tous les protocoles utilisant netlink.

L'implémentation de MPLS utilise la famille `NETLINK_ROUTE` pour la communication entre espace utilisateur et noyau. En outre, elle surcharge le sens de toutes les valeurs `RTM_*`, et donc notamment `RTM_GETLINK` et `RTM_GETADDR`, pour ses propres besoins. Ainsi, `RTM_GETADDR` est surchargé par `MPLS_RTM_GETILM`.

Lorsque le noyau reçoit la requête `RTM_GETADDR` pour tous les protocoles, il utilise le tableau `rtnetlink_links` pour connaître pour chacun des protocoles le tableau contenant les fonctions à appeler pour gèrer ce type de requête. Ainsi, pour

MPLS, ce dernier tableau est `mpls_rtnetlink_table`. Et la fonction qui doit gérer la requête `RTM_GETADDR` pour MPLS gère, à cause de la surcharge, la requête `MPLS_RTM_GETILM`. Elle renvoie donc une valeur qui n'est pas une adresse à proprement parler, mais une information sur un tunnel MPLS.

Lorsque la glibc reçoit cette information, elle tente de la traiter comme s'il s'agissait d'une adresse. Comme c'est impossible et que la glibc pense qu'une telle erreur signifie qu'il y a incompatibilité entre le noyau et la bibliothèque, la glibc arrête le processus en appelant `abort()`. Le fonctionnement extrême de la glibc est normal car il est effectivement anormal qu'elle reçoive une information sur une adresse dont elle ne connait pas le format.

Ce bogue est lié à deux problèmes, et la correction d'un seul d'entre eux corrigerait le problème :

– l'implémentation de MPLS surcharge le sens des messages `RTM_*` ;
– l'implémentation de MPLS utilise la famille `NETLINK_ROUTE`.

Le premier problème n'est pas résolvable en tant que tel car le nombre de valeurs `RTM_*` est limité, et l'espace restant n'est pas suffisant pour le nombre de valeurs dont a besoin MPLS.

Le second problème peut être résolu par la création d'une nouvelle famille netlink, comme `NETLINK_MPLS`. Cela permettrait de ne pas avoir à surcharger le sens des messages `RTM_*`, et cela a l'avantage d'être une voie à suivre logique car la communication entre espace utilisateur et noyau au sujet de MPLS n'est pas sémantiquement une communication sur les routes. Ce changement nécessite une réecriture partielle du code de base de MPLS. Même s'il ne s'agit pas d'une réecriture complexe, elle demande un effort non négligeable.

Annexe B

Utilisation de `libipq` pour intercepter tous les paquets IP

> We interrupt this program to annoy you and make things generally irritating.
>
> BBC Announcer, *Monty Python's Flying Circus*

```
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <iptables.h>
#include <pthread.h>
#include <libipq/libipq.h>

pthread_t thread;
struct ipq_handle *handle;

/* the 'void *arg' argument and the 'void *' return value
 * are needed to create a thread */
static void *
iptables_thread (void *arg)
{
        sigset_t set;
        unsigned char buf[BUFSIZ];
        ipq_packet_msg_t *m;

        /* We set a signal mask for this thread because it should
         * not see signals: all signals will be handled by the main
         * thread. It * works around a small bug when killing the
         * program and canceling * this thread (which was already
```

```
24          * terminated because it caught the signal).*/
        sigfillset(&set);
26      pthread_sigmask(SIG_SETMASK, &set, NULL);

28      while (1) {
                /* this is to make a cancellation point available
30               * somewhere in the function so the thread is
                 * cancellable (needed for iptables_free()) */
32              pthread_testcancel();

34              /* get a new packet */
                if (ipq_read(handle, buf, sizeof(buf), 0) < 0)
36                      continue;

38              m = ipq_get_packet(buf);
                ipq_set_verdict(handle, m->packet_id,
40                              NF_ACCEPT, 0, NULL);
        }
42
        return NULL;
44  }

46  static void
    iptables_free (void)
48  {
        system("iptables -F");
50
        if (handle != NULL)
52              ipq_destroy_handle(handle);

54      pthread_cancel(thread);
    }
56
    static void
58  iptables_wait (void)
    {
60      pthread_join (thread, NULL);
    }
62
    static int
64  iptables_init (void)
    {
66      handle = ipq_create_handle(0, PF_INET);
        if (!handle)
68              return -1;

70      if (ipq_set_mode(handle, IPQ_COPY_PACKET, 0) < 0) {
                iptables_free();
72              return -1;
```

```
        }
74
        system("iptables -F");
76      system("iptables -A OUTPUT -p all -j QUEUE");

78      if (pthread_create(&thread, NULL,
                           &iptables_thread, NULL) != 0) {
80              iptables_free();
                return -1;
82      }

84      return 0;
}
86
static void
88 sig_handler (int sig)
{
90      iptables_free();
        exit(0);
92 }

94 int
main (int    argc,
96       char **argv)
{
98      signal (SIGINT, &sig_handler);
        signal (SIGTERM, &sig_handler);
100
        if (iptables_init())
102             return 1;

104     iptables_wait();

106     return 0;
}
```

Annexe C

Implémentation de MPLS avec scapy

> I just don't believe it. I won't believe it. I can't believe it. I shan't believe it.
>
> Roger Rabbit, *Who Framed Roger Rabbit ?*

```
from scapy import *

#
# scapy integration
#

MPLS_TYPE = 0x8847

class MiniMPLS (Packet):
  name = "MiniMPLS"
  fields_desc = [ BitField("label", 0, 20),
                  BitField("exp", 0, 3),
                  BitField("stack", 1, 1),
                  ByteField("ttl", 64) ]

  def send(self):
    action = MplsIlms.getaction (self.label)
    type = action.gettype ()
    if type == "pop":
      self.pop ()
    elif type == "push":
      self.push (action.getkey ())
    elif type == "swap":
```

```
24        self.swap (action.getkey ())
      else:
26        log_runtime.error("Unexpected type for action: %s (label %d)"
                            % (type, self.label))
28
    def pop(self):
30    if self.stack == 1:
        if IP in self:
32        send (self[IP], verbose = 0)
        else:
34        log_runtime.error("Does not know the nature of the payload "
                            "(label %d)" % (self.label))
36    elif not MiniMPLS in self.payload:
          log_runtime.error("Does not contain a stacked MPLS header "
38                          "(label %d)" % (self.label))
      else:
40      innermpls = self.getlayer(MiniMPLS, nb = 2)
        innermpls.send()
42
    def push(self, key):
44    nhlfe = MplsNhlfes.get (key)
      if nhlfe == None:
46      log_runtime.error("No NHLFE associated to key %d (label %d)"
                          % (key, self.label))
48      return

50    mpls = MPLS (label = nhlfe.getlabel ())/self
      iface = nhlfe.getiface ()
52    sendp(Ether(src=get_if_hwaddr(iface),
                  dst=getmacbyip(nhlfe.getnexthop))/mpls,
54          iface=iface)

56  def swap(self, key):
      self.ttl -= 1
58    if self.ttl == 0:
        log_runtime.error("Dropping packet (TTL = 0) (label %d)"
60                        % (self.label))
        return
62
      nhlfe = MplsNhlfes.get (key)
64    if nhlfe == None:
        log_runtime.error("No NHLFE associated to key %d (label %d)"
66                        % (key, self.label))
        return
68
      self.label = nhlfe.getlabel ()
70    iface = nhlfe.getiface ()
      sendp(Ether(src=get_if_hwaddr(iface),
72                dst=getmacbyip(nhlfe.getnexthop ()))/self,
```

```
          iface=iface, verbose = 0)

  def mysummary(self):
    return self.sprintf("label %MiniMPLS.label% "
                        "(stack=%MiniMPLS.stack%)")

# tell scapy that Ether can contain MPLS and that it should use
# MPLS_TYPE in this case
# limit the payload of MPLS to IP (at least for now)
layer_bonds = [ ( Ether,    MiniMPLS, { "type" : MPLS_TYPE } ),
                ( MiniMPLS, MiniMPLS, { "stack" : 0 }        ),
                ( MiniMPLS, IP,       { "stack" : 1 }        ) ]

for l in layer_bonds:
    bind_layers (*l)
del (l)

#
# Implementation of MPLS
#

class MplsIlmLabel:
  label = 16

  def get_new_label():
    MplsIlmLabel.label += 1
    return MplsIlmLabel.label

  get_new_label = staticmethod (get_new_label)

class MplsNhlfeKey:
  key = 0

  def get_new_key():
    MplsNhlfeKey.key += 1
    return MplsNhlfeKey.key

  get_new_key = staticmethod (get_new_key)

class MplsNhlfes:
  nhlfes = {}

  def add (label, next_hop, iface):
    nhlfe = MplsNhlfe (label, next_hop, iface)
    key = MplsNhlfeKey.get_new_key()

    MplsNhlfes.nhlfes[key] = nhlfe
    return key
```

```
122   def get (key):
        if key in MplsNhlfes.nhlfes:
124       return MplsNhlfes.nhlfes[key]
        else:
126       return None

128   def remove (key):
        if key in MplsNhlfes.nhlfes:
130       del MplsNhlfes.nhlfes[key]

132   def exists (key):
        return key in MplsNhlfes.nhlfes
134
      add = staticmethod (add)
136   get = staticmethod (get)
      remove = staticmethod (remove)
138   exists = staticmethod (exists)

140 class MplsNhlfe:
      def __init__ (self, label, next_hop, iface):
142     self.label = label
        self.next_hop = next_hop
144     self.iface = iface

146   def getlabel(self):
        return self.label
148
      def getnexthop(self):
150     return self.next_hop

152   def getiface(self):
        return self.iface
154
    class MplsIlms:
156   ilmspaces = {}

158   def add (label, action = None, labelspace = 0):
        if action == None:
160       action = MplsIlmAction()
        if not labelspace in MplsIlms.ilmspaces:
162       MplsIlms.ilmspaces[labelspace] = {}
        if label in MplsIlms.ilmspaces[labelspace]:
164       return
        MplsIlms.ilmspaces[labelspace][label] = action
166
      def remove (label, labelspace = 0):
168     if not labelspace in MplsIlms.ilmspaces:
          return
170     if not label in MplsIlms.ilmspaces[labelspace]:
```

```
      return
    del MplsIlms.ilmspaces[labelspace][label]

  def modifyaction (label, action, labelspace = 0):
    if not labelspace in MplsIlms.ilmspaces:
      return
    if not label in MplsIlms.ilmspaces[labelspace]:
      return
    MplsIlms.ilmspaces[labelspace][label] = action

  def getaction (label, labelspace = 0):
    if not labelspace in MplsIlms.ilmspaces:
      return None
    if not label in MplsIlms.ilmspaces[labelspace]:
      return None
    return MplsIlms.ilmspaces[labelspace][label]

  add = staticmethod (add)
  remove = staticmethod (remove)
  modifyaction = staticmethod (modifyaction)
  getaction = staticmethod (getaction)

class MplsIlmAction:
  key = None

  def __init__ (self, type = "pop", key = None):
    self.type = type
    if type == "swap" or type == "push":
      if key == None:
        return None
      if not MplsNhlfes.exists (key):
        return None
      self.key = key

  def gettype (self):
    return self.type

  def getkey (self):
    if self.type == "swap" or self.type == "push":
      return self.key
    else:
      return None


def listen_for_mpls(iface, verbose = 0):
  pks = conf.L2socket(iface=iface, nofilter=1)

  inmask = [pks]
  while 1:
```

```
220     try:
          inp, out, err = select(inmask, [], [], None)
222       if len(inp) == 0:
            break
224       if pks in inp:
            packet = pks.recv(MTU)
226       if packet is None:
            continue
228
          if MiniMPLS in packet:
230         mpls = packet[MiniMPLS]
            if verbose:
232           print mpls.mysummary()
            mpls.send()
234
        except KeyboardInterrupt:
236       break
```

Annexe D

Bibliographie

[1] N. Abramson. The aloha system - another alternative for computer communications. In *Proceedings of the AFIPS Conference*, volume 37, pages 295–298, 1970.

[2] N. Abramson. Development of the ALOHANET. *IEEE Transactions on Information Theory*, IT-31(2), March 1985.

[3] R. M. Metcalfe. PACKET COMMUNICATION. Technical report, Massachusetts Institute of Technology, Cambridge, MA, USA, 1973.

[4] R. M. Metcalfe and D. R. Boggs. Ethernet : distributed packet switching for local computer networks. *Commun. ACM*, 19(7) :395–404, 1976.

[5] L. G. Roberts. Multiple computer networks and intercomputer communication. In *SOSP '67 : Proceedings of the first ACM symposium on Operating System Principles*, pages 3.1–3.6, New York, NY, USA, 1967. ACM Press.

[6] L. Roberts. The Arpanet and computer networks. In *Proceedings of the ACM Conference on The history of personal workstations*, pages 51–58, New York, NY, USA, 1986. ACM Press.

[7] P. Baran. On distributed communications networks. Technical report, RAND Corp., September 1962.

[8] L. Kleinrock. Information flow in large communication nets. Technical report, Massachusetts Institute of Technology, July 1961.

[9] T. Tugend. Ucla to be first station in nationwide computer network. Technical report, UCLA Press Release, July 1969.

[10] RFC Editor and et al. 30 Years of RFCs. RFC 2555, 1999.

[11] V. Cerf and R. Kahn. A Protocol for Packet Network Intercommunication. *Communications, IEEE Transactions on [legacy, pre - 1988]*, 22(5) :637–648, 1974.

[12] T. Berners-Lee and R. Cailliau. WorldWideWeb : Proposal for a HyperText Project. Proposal, CERN, 1990.

[13] Wi-Fi Alliance.
http://www.wi-fi.org/.

[14] Autorité de Régulation des Communications Électroniques et des Postes. Suivi des indicateurs mobiles.
http://www.art-telecom.fr/index.php?id=35.

[15] Autorité de Régulation des Communications Électroniques et des Postes. L'Observatoire de l'Internet haut débit.
http://www.art-telecom.fr/index.php?id=9394.

[16] M. Weiser. The computer for the twenty-first century. *Scientific American*, 9 :94–100, Sep 1991.

[17] L. Krishnamurthy, S. Conner, M. Yarvis, J. Chhabra, C. Ellison, C. Brabenac, and E. Tsui. Meeting the Demands of the Digital Home with High-Speed Multi-Hop Wireless Networks. *Intel Technology Journal*, 6(4), 2002.

[18] IEEE 802 LAN/MAN Standards Committee.
http://www.ieee802.org/.

[19] Bluetooth SIG (Special Interest Group).
http://www.bluetooth.org/.

[20] 802.11 : Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications.
http://standards.ieee.org/getieee802/download/802.11-1999.pdf, 1999.

[21] 802.11a : High-speed Physical Layer in the 5 GHz band.
http://standards.ieee.org/getieee802/download/802.11a-1999.pdf, 1999.

[22] 802.11b : Higher speed Physical Layer (PHY) extension in the 2.4 GHz band.
http://standards.ieee.org/getieee802/download/802.11b-1999.pdf, 1999.

[23] 802.11g : Further Higher-Speed Physical Layer Extension in the 2.4 GHz Band.
http://standards.ieee.org/getieee802/download/802.11g-2003.pdf, 2003.

[24] 802.11n : Enhancements for Higher Throughput, 2008 (expected).

[25] 802.11i : Medium Access Control (MAC) Security Enhancements.
http://standards.ieee.org/getieee802/download/802.11i-2004.pdf, 2004.

[26] 802.11d : Specification for Operation in Additional Regulartory Domains.
http://standards.ieee.org/getieee802/download/802.11d-2001.pdf, 2001.

[27] 802.11h : Spectrum and Transmit Power Management Extensions in the 5GHz band in Europe.
http://standards.ieee.org/getieee802/download/802.11h-2003.pdf, 2003.

[28] 802.11j : 4.9 GHz–5 GHz Operation in Japan.
http://standards.ieee.org/getieee802/download/802.11j-2004.pdf, 2004.

[29] 802.11e : Medium Access Control (MAC) Quality of Service Enhancements.
http://standards.ieee.org/getieee802/download/802.11e-2005.pdf, 2005.

[30] A. Colvin. CSMA with collision avoidance. In *Computer Communications Techniques*, volume 6, pages 225–235. Butterworth & Co. (Publishers) Ltd., Oct 1983.

[31] D. Fumolari. Link performance of an embedded Bluetooth personal area network. In *Proceedings of IEEE ICC'01*, volume 8, pages 2573–2577, june 2001.

[32] M. Heusse, F. Rousseau, G. Berger-Sabbatel, and A. Duda. Performance anomaly of 802.11b. In *Proceedings of IEEE INFOCOM 2003*, San Francisco, USA, March-April 2003.

[33] F. A. Tobagi and L. Kleinrock. Packet switching in radio channels : part II the hidden terminal problem in carrier sense multiple-access and the busy-tone solution. *IEEE Transactions on Communication*, 23(12) :1417–1433, December 1975.

[34] C. L. Fullmer and J. J. Garcia-Luna-Aceves. Solutions to Hidden Terminal Problems in Wireless Networks. In *SIGCOMM*, pages 39–49, 1997.

[35] V. Bharghavan, A. J. Demers, S. Shenker, and L. Zhang. MACAW : A media access protocol for wireless LAN's. In *SIGCOMM*, pages 212–225, 1994.

[36] H. Lundgren, E. Nordströ, and C. Tschudin. Coping with communication gray zones in IEEE 802.11b based ad hoc networks. In *WOWMOM '02 : Proceedings of the 5th ACM international workshop on Wireless mobile multimedia*, pages 49–55, New York, NY, USA, 2002. ACM Press.

[37] Bluetooth.
http://www.bluetooth.com/.

[38] Bluetooth Network Encapsulation Protocol.
http://www.bluetooth.com/NR/rdonlyres/89B2BA02-D3FB-4717-97A1-A5B10D90F795/912/BNEPSpecification1.pdf, 2003.

[39] Personal Area Networking Profile (PAN).
http://www.bluetooth.com/NR/rdonlyres/279DC460-295E-42ED-8952-61B723620884/984/PAN_SPEC_V10.pdf, 2003.

[40] C. L. Hedrick. Routing Information Protocol. RFC 1058, 1988.

[41] G. Malkin. RIP Version 2. RFC 2453, 1998.

[42] J. M. McQuillan, I. Richer, and E. C. Rosen. The new routing algorithm for the ARPANET. *Innovations in Internetworking*, pages 119–127, 1988.

[43] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1 :269–271, 1959.

[44] J. Moy. OSPF specification. RFC 1131, 1989.

[45] J. Moy. OSPF Version 2. RFC 2328, 1998.

[46] R. W. Callon. Use of OSI IS-IS for routing in TCP/IP and dual environments. RFC 1195, 1990.

[47] IETF MANET Working Group. http://www.ietf.org/html.charters/manet-charter.html, 2000.

[48] S. Corson and J. Macker. Mobile Ad hoc Networking (MANET) : Routing Protocol Performance Issues and Evaluation Considerations. RFC 2501, 1999.

[49] D. B. Johnson and D. A. Maltz. Dynamic Source Routing in Ad Hoc Wireless Networks. In Imielinski and Korth, editors, *Mobile Computing*, volume 353. Kluwer Academic Publishers, 1996.

[50] D. Johnson, Y. Hu, and D. Maltz. The Dynamic Source Routing Protocol (DSR) for Mobile Ad Hoc Networks for IPv4. RFC 4728, 2007.

[51] Y.-C. Hu and D. B. Johnson. Implicit source routes for on-demand ad hoc network routing. In *MobiHoc '01 : Proceedings of the 2nd ACM international symposium on Mobile ad hoc networking & computing*, pages 1–10, New York, NY, USA, 2001. ACM Press.

[52] C. E. Perkins and E. M. Royer. Ad-hoc On-Demand Distance Vector Routing. In *WMCSA '99 : Proceedings of the Second IEEE Workshop on Mobile Computer Systems and Applications*, page 90, Washington, DC, USA, 1999. IEEE Computer Society.

[53] C. Perkins, E. Belding-Royer, and S. Das. Ad hoc On-Demand Distance Vector (AODV) Routing. RFC 3561, 2003.

[54] S. Ranjan Das, C. E. Perkins, and E. M. Royer. Performance Comparison of Two On-demand Routing Protocols for Ad Hoc Networks. In *INFOCOM (1)*, pages 3–12, 2000.

[55] P. Jacquet, P. Mühlethaler, T. Clausen, A. Laouiti, A. Qayyum, and L. Viennot. Optimized link state routing protocol for ad hoc networks. In *Proceedings of the 5th IEEE Multi Topic Conference (INMIC 2001)*, 2001.

[56] A. Qayyum, L. Viennot, and A. Laouiti. Multipoint Relaying : An Efficient Technique for flooding in Mobile Wireless Networks. Technical Report Research Report RR-3898, INRIA, February 2000.

[57] Optimized Link State Routing Protocol (OLSR). RFC 3626, 2003.

[58] T. H. Clausen, P. Jacquet, and L. Viennot. Investigating the Impact of Partial Topology in Proactive MANET Routing Protocols. In *WPMC '02 : 5th International Symposium on Wireless Personal Multimedia Communications*, volume 3, pages 1374–1378, 2002.

[59] Charles Perkins. Ad Hoc networks, IETF, and convergence in solution space. In *MobiArch '06 : Proceedings of first ACM/IEEE international workshop on Mobility in the evolving internet architecture*, pages 1–2, New York, NY, USA, 2006. ACM Press.

[60] I. Chakeres and C. Perkins. Dynamic MANET On-demand Routing Protocol. IETF Internet Draft, 2007.

[61] C. Adjih, P. Jacquet, and L. Viennot. Computing connected dominated sets with multipoint relays. *Ad Hoc and Sensor Wireless Networks*, 1(1-2), january 2005.

[62] R. Perlman. An Algorithm for Distributed Computation of a Spanning Tree in an Extended LAN. *ACM SIGCOMM Computer Communication Review*, 15 :44–53, 1985.

[63] R. Perlman. Rbridges : Transparent Routing. In *Proc. Infocom 2004*, Hong Kong, March 2004.

[64] 802.1w - Rapid Reconfiguration of Spanning Tree. http://www.ieee802.org/1/pages/802.1w.html, 2004.

[65] S. Abuguba and I. Moldován. Verification of RSTP convergence and scalability by measurements and simulations .

[66] P. Francois, C. Filsfils, J. Evans, and O. Bonaventure. Achieving sub-second IGP convergence in large IP networks. *SIGCOMM Comput. Commun. Rev.*, 35(3) :35–44, 2005.

[67] C. Tschudin, R. Gold, O. Rensfelt, and O. Wibling. LUNAR : a Lightweight Underlay Network Ad-hoc Routing Protocol and Implementation. In *Proc. Next Generation Teletraffic and Wired/Wireless Advanced Networking (NEW2AN'04)*, 2004.

[68] C. Tschudin and R. Gold. SelNet : A Translating Underlay Network. Technical report, Uppsala University, Department of Information Technology, 2001.

[69] R. Gold, P. Gunningberg, and C. Tschudin. A virtualized link layer with support for indirection. In *FDNA '04 : Proceedings of the ACM SIGCOMM workshop on Future directions in network architecture*, pages 28–34, New York, NY, USA, 2004. ACM Press.

[70] S. Cheshire, B. Aboba, and E. Guttman. Dynamic Configuration of IPv4 Link-Local Addresses. RFC 3927, 2005.

[71] C. Tschudin and C. Jelger. Underlay Fusion of DNS, ARP/ND, and Path Resolution in MANETs. In *Proc. 5th Scandinavian Workshop on Wireless Ad-hoc Networks (ADHOC'05)*, 2005.

[72] G. Chelius and E. Fleury. Ananas : An Ad hoc Network Architectural Scheme. In *Proc. MWCN*. IEEE, September 2002.

[73] 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.

[74] C. F. Tschudin, H. Lundgren, and E. Nordström. Embedding MANETs in the Real World. In *Personal Wireless Communications, IFIP-TC6 8th International Conference*, pages 578–589, September 2003.

[75] M. Handley. Why the Internet only just works. *BT Technology Journal*, 24(3) :119–129, 2006.

[76] J. Postel. Internet Protocol. RFC 791, 1981.

[77] J. Postel. Transmission Control Protocol. RFC 793, 1981.

[78] J. Nagle. Congestion control in IP/TCP internetworks. *SIGCOMM Comput. Commun. Rev.*, 14(4) :11–17, 1984.

[79] V. Jacobson, R. Braden, and D. Borman. TCP Extensions for High Performance. RFC 1323, 1992.

[80] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow. TCP Selective Acknowledgement Options. RFC 2018, 1996.

[81] M. Allman, V. Paxson, and W. Stevens. TCP Congestion Control. RFC 2581, 1999.

[82] K. Ramakrishnan, S. Floyd, and D. Black. The Addition of Explicit Congestion Notification (ECN) to IP. RFC 3168, 2001.

[83] S. Floyd, T. Henderson, and A. Gurtov. The NewReno Modification to TCP's Fast Recovery Algorithm. RFC 3782, 2004.

[84] R. Droms. Dynamic Host Configuration Protocol. RFC 2131, 1997.

[85] S. Cheshire and M. Krochmal. DNS-Based Service Discovery. IETF Internet Draft, 2006.

[86] D. C. Plummer. An Ethernet Address Resolution Protocol or Converting Network Protocol Addresses to 48.bit Ethernet Address for Transmission on Ethernet Hardware. RFC 826, 1982.

[87] P. Gupta and P. Kumar. Capacity of wireless networks, 1999.

[88] S. M. Das, H. Pucha, and Y. C. Hu. Performance comparison of scalable location services for geographic ad hoc routing. In *INFOCOM*, pages 1228–1239. IEEE, 2005.

[89] E. Rosen, A. Viswanathan, and R. Callon. Multiprotocol Label Switching Architecture. RFC 3031, 2001.

[90] E. Rosen, D. Tappan, G. Fedorkow, Y. Rekhter, D. Farinacci, T. Li, and A. Conta. MPLS Label Stack Encoding. RFC 3032, 2001.

[91] A. Conta, P. Doolan, and A. Malis. Use of Label Switching on Frame Relay Networks Specification. RFC 3034, 2001.

[92] Y. Ohba, Y. Katsube, E. Rosen, and P. Doolan. MPLS Loop Prevention Mechanism. RFC 3063, 2001.

[93] F. Le Faucheur, L. Wu, B. Davie, S. Davari, P. Vaananen, R. Krishnan, P. Cheval, and J. Heinanen. Multi-Protocol Label Switching (MPLS) Support of Differentiated Services. RFC 3270, 2002.

[94] E. Rosen and Y. Rekhter. BGP/MPLS VPNs. RFC 2547, 1999.

[95] K. Muthukrishnan and A. Malis. A Core MPLS IP VPN Architecture. RFC 2917, 2000.

[96] L. Andersson, P. Doolan, N. Feldman, A. Fredette, and B. Thomas. LDP Specification. RFC 3036, 2001.

[97] L. Andersson, R. Callon, R. Dantu, L. Wu, P. Doolan, T. Worster, N. Feldman, A. Fredette, M. Girish, E. Gray, J. Heinanen, T. Kilty, and A. Malis. Constraint-Based LSP Setup using LDP. RFC 3212, 2002.

[98] D. Awduche, L. Berger, D. Gan, T. Li, V. Srinivasan, and G. Swallow. RSVP-TE : Extensions to RSVP for LSP Tunnels. RFC 3209, 2001.

[99] S.-Y. Ni, Y.-C. Tseng, Y.-S. Chen, and J.-P. Sheu. The broadcast storm problem in a mobile ad hoc network. In *MobiCom '99 : Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking*, pages 151–162, New York, NY, USA, 1999. ACM Press.

[100] S.-J. Lee, E. M. Belding-Royer, and C. E. Perkins. Scalability study of the ad hoc on-demand distance vector routing protocol. *Int. J. Netw. Manag.*, 13(2) :97–114, 2003.

[101] The Network Simulator - ns-2. http://www.isi.edu/nsnam/ns/.

[102] OPNET Modeler. http://www.opnet.com/solutions/network_rd/modeler.html.

[103] J. Heidemann, K. Mills, and S. Kumar. Expanding Confidence in Network Simulation. *IEEE Network Magazine*, 15(5) :58–63, Sept./Oct. 2001.

[104] M. Takai, J. Martin, and R. Bagrodia. Effects of Wireless Physical Layer Modeling in Mobile Ad Hoc Networks, 2001.

[105] J. Heidemann, N. Bulusu, J. Elson, C. Intanagonwiwat, K. Lan, Y. Xu, W. Ye, D. Estrin, and R. Govindan. Effects of detail in wireless network simulation, 2001.

[106] D. Cavin, Y. Sasson, and A. Schiper. On the accuracy of MANET simulators. In *POMC '02 : Proceedings of the second ACM international workshop on Principles of mobile computing*, pages 38–43, New York, NY, USA, 2002. ACM Press.

[107] Équipe de `tcpdump`. libpcap.
http://www.tcpdump.org/.

[108] J. Salim, H. Khosravi, A. Kleen, and A. Kuznetsov. Linux Netlink as an IP Services Protocol. RFC 3549, 2003.

[109] Équipe de `netfilter`. libipq.
http://www.netfilter.org/projects/iptables/.

[110] Équipe de FreeBSD. divert.
http://www.freebsd.org/cgi/man.cgi?query=divert.

[111] Équipe de `netfilter`. libnetfilter_queue.
http://www.netfilter.org/projects/libnetfilter_queue/.

[112] Y. Uo, Z. Uda, and N. Ogashiwa. Projet AYAME.
http://www.ayame.org/.

[113] J. R. Leu. MPLS for Linux.
http://mpls-linux.sourceforge.net/.

[114] P. Biondi. Scapy.
http://www.secdev.org/projects/scapy/.

[115] D. A. Wheeler. SLOCCount.
http://www.dwheeler.com/sloccount/.

[116] WIFI.com.ar. Linux LiveCD Router.
http://www.wifi.com.ar/english/cdrouter/.

[117] 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 *Proc. of the Fifth Symposium on Operating Systems Design and Implementation*, pages 255–270, Boston, MA, December 2002. USENIX Association.

[118] J. Dike. User-Mode Linux.
http://user-mode-linux.sourceforge.net/.

[119] VMware, Inc. VMware.
http://www.vmware.com/.

[120] University of Cambridge Computer Laboratory. Xen.
http://www.cl.cam.ac.uk/research/srg/netos/xen/.

[121] A. Kivity and Y. Kamay. KVM.
http://kvm.qumranet.com/.

[122] R. Jones. Netperf.
http://www.netperf.org/.

[123] D. S. J. De Couto, D. Aguayo, J. Bicket, and R. Morris. A high-throughput path metric for multi-hop wireless routing. In *MobiCom '03 : Proceedings of the 9th annual international conference on Mobile computing and networking*, pages 134–146, New York, NY, USA, 2003. ACM Press.

[124] G. Karbaschi and A. Fladenmuller. A Link Quality and Congestion-aware Cross layer Metric for Multi-Hop Wireless Routing. In *2nd IEEE International Conference on Mobile Ad-hoc and Sensor Systems(MASS05)*, November 2005.

[125] A. Adya, P. Bahl, J. Padhye, A. Wolman, and L. Zhou. A Multi-Radio Unification Protocol for IEEE 802.11 Wireless Networks. In *BROADNETS '04 : Proceedings of the First International Conference on Broadband Networks (BROADNETS'04)*, pages 344–354, Washington, DC, USA, 2004. IEEE Computer Society.

[126] B. Awerbuch, D. Holmer, and H. Rubens. High throughput route selection in multi-rate ad hoc wireless networks, 2003.

[127] R. M. Axelrod. *The Evolution of Cooperation*. Basic Books, 1984.

Les réseaux sans fil spontanés pour l'Internet Ambiant

Résumé

Cette thèse propose une nouvelle architecture d'interconnexion pour les réseaux de bordure spontanés. Un réseau de bordure spontané est composé de différents réseaux utilisant des technologies variées mais ne formant, pour l'utilisateur, qu'un unique réseau local et autonome. La démocratisation des technologies sans fil, et en particulier de 802.11, a favorisé l'émergence de ce type de réseaux. Tranchant avec la philosophie d'IP dans laquelle tout paquet est indépendant, nous affirmons l'importance de la notion de connexion et le besoin de garantir le bon fonctionnement des connexions établies. Nous proposons à cette fin un niveau 2.5 entre les couches MAC et IP, qui se traduit par l'utilisation de chemins MPLS pour le transport des données. Couplée avec un routage réactif, cette architecture orientée connexion au niveau 2.5 offre de nombreux avantages. Une implémentation de notre proposition a été réalisée sous la forme d'un prototype, Lilith.

Mots clés : architecture d'interconnexion, réseaux spontanés, réseaux sans fil, routage ad hoc, couche 2.5, prototype, Internet ambiant, MPLS.

Spontaneous Wireless Networks for the Pervasive Internet

Abstract

This thesis proposes a new interconnection architecture for spontaneous edge networks. A spontaneous edge network is made of several networks based on various technologies, which appear like only one local and autonomous network to the user. The democratization of wireless technologies, and especially of 802.11, has enabled the rise of such networks. While the IP philosophy states that all packets are independent, we stress the importance of the concept of connection and the need to maintain established connections. To this end, we propose a 2.5 layer, between the MAC and IP layers, using MPLS paths to convey data. Coupled with a reactive routing, this connection-oriented architecture at the 2.5 layer presents several benefits. Our proposal has been implemented in a prototype called Lilith.

Keywords: interconnection architecture, ad hoc routing, spontaneous networks, wireless networks, 2.5 layer, prototype, pervasive Internet, MPLS.

Discipline : Informatique : Systèmes et Logiciels
Laboratoire : LIG – Équipe Drakkar
BP 72, 38402 Saint Martin d'Hères CEDEX, France