Algorithmes pour le routage dans les réseaux multicouches

Noureddine Mouhoub

THÈSE PRÉSENTÉE

POUR OBTENIR LE GRADE DE

DOCTEUR DE

L'UNIVERSITÉ DE BORDEAUX

ÉCOLE DOCTORALE DE MATHÉMATIQUES ET D'INFORMATIQUE

SPÉCIALITÉ INFORMATIQUE

Par **Noureddine MOUHOUB**

Algorithmes pour le routage dans les réseaux multicouches

Sous la direction de : **Damien MAGONI**
Avec la co-direction de : **Mohamed Lamine LAMALI**

Soutenue le 8 décembre 2023

Membres du jury :

M. Toufik Ahmed	Professeur des universités	Bordeaux INP	Président du jury
Mme Johanne Cohen	Directrice de recherche	CNRS	Rapporteure
M. Marceau Coupechoux	Professeur des universités	Institut Mines-Télécom	Rapporteur
Mme Katia Jaffrès-Runser	Professeure des universités	Toulouse INP	Examinatrice
M. Damien Magoni	Professeur des universités	Université de Bordeaux	Directeur
M. Mohamed Lamine Lamali	Maître de conférences	Université de Bordeaux	Co-directeur

Table des matières

Table des figures

Liste des tableaux

Liste des algorithmes

Remerciements

Cette thèse fut une très belle aventure, parfois difficile, mais très enrichissante. Ce voyage intellectuel a été très instructif et je suis fier des contributions significatives de mes recherches. Ces quatre dernières années, de nombreuses personnes m'ont aidé à terminer ce travail, et j'aimerais ici pouvoir les en remercier.

Avant tout, je suis extrêmement reconnaissant envers mes directeurs de thèse, Pr. Damien Magoni et Dr. Mohamed Lamine Lamali, pour leurs précieux conseils, leur soutien et leur encadrement tout au long de ce voyage.

Je tiens à exprimer ma sincère gratitude à tous les membres du jury. Je tiens à remercier Pr. Toufik Ahmed, Dr. Johanne Cohen, Pr. Marceau Coupechoux et Pr. Katia Jaffrès-Runser, pour leurs commentaires pertinents et leurs critiques constructives qui ont largement contribué au raffinement de mon travail.

Je tiens à exprimer ma gratitude à mes collègues de l'Université de Bordeaux et du Laboratoire Bordelais de Recherche en Informatique (LaBRI), pour les beaux moments que nous avons passés ensemble et pour avoir participé à mon voyage.

Enfin et surtout, cet accomplissement ne sera pas possible sans mentionner ma très profonde gratitude et mon plus profond amour à ma femme, à mon petit fils, à mes parents et à mes sœurs pour tout le soutien et la gentillesse qu'ils m'ont apporté tout au long de ma vie et particulièrement ces quatre dernières années.

Introduction générale

Je continue à poser ces questions de "comment" et "pourquoi". Parfois, je trouve une réponse.

– Stephen Hawking

Le routage dans les réseaux hétérogènes et multi-couches

L'histoire de la transmission de l'information est une succession de progrès technologiques qui a profondément changé notre manière de communiquer. Autrefois, les messages prenaient des chemins tortueux et incertains, que ce soit à dos de cheval ou par l'intermédiaire de pigeons voyageurs, pour relier des endroits éloignés. Cependant, l'avènement des télécommunications au XIXe siècle a considérablement raccourci les distances virtuelles grâce aux câbles sous-marins. Puis, l'ère numérique a ouvert une ère d'instantanéité et de mondialisation de l'information. Internet, cette toile cosmique qui transcende les frontières géographiques, a révolutionné la façon dont nous partageons et accédons aux connaissances. Les avancées technologiques ont surmonté les contraintes liées à la distance dans la communication et le partage de connaissances. Les réseaux de télécommunication ont progressé de la simple voix et du courrier électronique vers des technologies sophistiquées telles que la diffusion de vidéos en Haute Définition, le transfert de fichiers volumineux, la téléprésence et les jeux en ligne, entre autres. De plus, il est désormais possible d'assister en direct et en Haute Définition à des événements se déroulant à l'autre bout de la planète. Cette transformation remarquable de la transmission de l'information, de la lenteur du courrier à la vitesse de la lumière, soulève ainsi la question essentielle de comment garantir une telle transmission au sein des réseaux de télécommunication.

L'univers complexe des réseaux de télécommunication repose sur un élément fondamental, le *routage*. Il constitue l'épine dorsale de l'infrastructure numérique mondiale. À l'ère de l'information et de la connectivité croissante, il est essentiel de comprendre ce concept fondamental pour appréhender le fonctionnement des réseaux informatiques, téléphoniques et de données qui sous-tendent notre vie quotidienne. Le routage est le processus invisible mais crucial qui permet à nos données, qu'il s'agisse de messages, de fichiers, de vidéos ou de toute autre type d'information, de voyager de manière fluide et efficace à travers un labyrinthe complexe de dispositifs interconnectés. Il englobe diverses méthodes, parmi lesquelles se distinguent le routage *unicast* et le routage *multicast*. Le premier vise à diriger les données vers une destination unique, tel que le transfert

d'un fichier entre deux dispositifs, en empruntant un chemin de bout en bout. Le second, quant à lui, vise à diffuser les mêmes données à un groupe de destinataires, comme dans le cas du streaming ou de la vidéoconférence, en utilisant un *arbre de diffusion*. Ce panorama du routage ne serait pas complet sans mentionner les protocoles qui l'animent. De nombreux protocoles, tels que RIP, OSPF, BGP, DVMRP, PIM, pour n'en citer que quelques-uns, jouent un rôle crucial dans la mise en place d'une connectivité optimale entre chaque nœud pour l'ensemble des nœuds d'un réseau. Ils contribuent également à la création d'une *table de routage*, qui répertorie toutes les routes possibles, ou chemins, dans le réseau. Ces protocoles s'appuient sur des algorithmes classiques de calcul de chemins, parmi lesquels Dijkstra, Bellman-Ford, Floyd-Warshall, et d'autres, pour accomplir leur tâche. En parcourant le monde du routage, on découvre deux techniques fondamentalement différentes : le *routage par sauts* et le *routage par segments*. Dans le routage par sauts, les données sont transmises d'un nœud à un nœud voisin, chaque saut rapprochant un peu plus les données de leur destination finale. En revanche, dans le routage par segments, le chemin est décomposé en plusieurs sous-chemins, appelés segments, et les données sont transmises segment par segment.

De nos jours, la plupart des réseaux de communication sont *hétérogènes et multicouches i.e.,* comportent plusieurs protocoles [1] et plusieurs couches, parfois sur des technologies différentes. De plus, le nombre de protocoles croît rapidement du fait de l'apparition régulière de nouvelles technologies. Outre le cas classique d'IPv4/IPv6, de nouvelles technologies telles que l'*Internet des objets* (IoT), introduisent de nouveaux protocoles de routage tels que AODV dans la pile Zigbee ou RPL dans les réseaux 6LowPAN. De plus, certaines technologies basées sur le *Software Defined Networking* (SDN), telles que P4, permettent la création de tout nouveau protocole souhaité. L'interopérabilité des protocoles est cruciale pour assurer la connectivité (et donc la communication) entre les différents nœuds d'un réseau hétérogène et multicouche. Il existe principalement deux méthodes pour assurer la connectivité dans ce contexte : *i)* la *conversion* de protocole et *ii)* l'*encapsulation* de protocole. La première méthode convertit le format d'en-tête d'un protocole donné en un autre format d'en-tête appartenant à un protocole différent (*ex.,* proxy, dual-stack, NAT-PT, NAT-64, etc.). La deuxième méthode encapsule les unités d'information (*i.e.,* paquets composés d'en-têtes et de données) d'un protocole en plaçant chaque unité d'information dans le champ de données d'un autre protocole (*ex.,* IP-in-IP, 6over4, etc.), et en effectuant l'opération inverse (appelée *désencapsulation*) plus tard, *i.e.,* extraire l'unité d'informations intérieure du champ de données de l'unité d'informations extérieure. On appelle *fonctions d'adaptation*, ces fonctions de transformation (conversion, encapsulation et désencapsulation). Ces opérations induisent la création de *tunnels* où un tunnel est une partie d'un chemin qui commence par une encapsulation (point d'entrée) et qui se termine par la désencapsulation correspondante (point de sortie). Les tunnels sont parfois imbriqués, *i.e.,* plusieurs unités d'informations sont ensuite encapsulées dans d'autres dans un ordre défini. La suite des en-têtes des unités d'informations formant une *pile de protocole*. Les tunnels sont largement utilisés dans les réseaux actuels.

1. Afin d'éviter toute confusion entre *protocole de communication* et *protocole de routage* dans la suite de ce document, nous désignerons le premier par le seul mot *protocole*, tandis que le second est explicitement indiqué comme *protocole de routage*.

Outre l'interopérabilité des protocoles, ils sont également utilisés à des fins de sécurité dans les *Réseaux Privés Virtuels* (VPN) (*ex.,* IPsec, WPA, *Oignon Routing*, TLS, Noise, etc.).

Dans ce contexte, le routage avec la prise en compte des tunnels *i.e.,* le calcul de chemins sous l'hétérogénéité protocolaire et la présence des fonctions d'adaptation n'est pas trivial. En effet, un chemin traversant différents protocoles de communication doit permettre une continuité protocolaire. De plus, si ce chemin passant par une encapsulation doit passer par la désencapsulation correspondante plus tard, il doit ainsi comporter autant d'encapsulations que de désencapsulations dans un ordre défini. Ces conditions définissent des contraintes de faisabilité (*i.e.,* compatibilité) et un tel chemin est dit *faisable*. Le problème algorithmique sous-jacent est celui du calcul des chemins faisables, et plus particulièrement de décider où convertir ou encapsuler. Malheureusement, les protocoles de routage actuels ne sont pas capables de construire automatiquement des chemins avec la prise en compte de tunnels. Bien que plusieurs tentatives d'automatisation aient eu lieu (*ex.,* Teredo, 6over4, 6to4, TSP, ISATAP, etc.), dans les réseaux actuels, il n'existe aucun mécanisme de calcul de chemins avec tunnels totalement automatisé. Ceci est dû à la difficulté de concevoir des algorithmes de calcul de chemins prenant en compte les fonctions d'adaptation. En fait, la plupart de ces méthodes nécessitent l'accès à un *Domain Name System* (DNS) ou à un autre serveur dédié. Dans de tels serveurs, les routes sont pré-calculées. De plus, ils ne sont pas nécessairement optimaux en termes de coût de trajet. Surtout, aucun d'entre eux n'est en mesure de déterminer automatiquement les points d'entrée et sortie des différents tunnels. Cette tâche est effectuée soit par pré-calcul à l'avance, soit par configuration manuelle établie par des scripts.

Il faut donc une solution qui permette de calculer les plus courts chemins avec la prise en compte des tunnels (y compris les points d'entrée et de sortie) directement par le protocole de routage. Une telle solution permettrait à la fois une automatisation complète, mais aussi un recalcul rapide du chemin afin de s'adapter aux événements de changements de topologie.

Problèmes traités et contributions

Cette thèse a été effectuée dans le cadre du projet ANR JCJC HÉtérogénéité et algorithmes de RoutAge dans les réseaux (HÉRA), dont le but est de mieux comprendre les fondements théoriques structurels et algorithmiques des réseaux multicouches. Plus précisement, étudier les problèmes de communication dans de tels réseaux, en particulier le calcul de chemins centralisé et distribué, la communication multipoint via le calcul d'arbres de diffusion, ainsi que l'adaptation de ces algorithmes en protocoles de routage.

Dans un premier temps, nous étudions le calcul de chemins faisables par *la fermeture transitive* : deux chemins sont concaténés pour n'en former qu'un seul. L'algorithme de fermeture transitive le plus connu est celui de Floyd-Warshall. Cette approche ne peut pas être directement appliquée dans notre cas car deux chemins ne sont pas forcément compatibles (*ex.,* un premier chemin se terminant avant la fin d'un tunnel alors que le

deuxième commence par deux tunnels imbriqués – ceci sur le même nœud). La première partie de notre travail est donc de déterminer dans quels cas il est possible de concaténer deux chemins. Pour cela, nous proposons deux méthodes de décomposition de chemins faisables en sous-chemins. En se basant sur ces résultats, nous proposons deux algorithmes qui calculent le plus court chemin faisable entre toutes les paires de nœuds dans un réseau hétérogène et multicouche. Pour implémenter ces algorithmes, la seule nécessité est que chaque nœud ait accès à la table de routage des autres. Les algorithmes peuvent donc être totalement centralisés ou bien distribués avec mémoire partagée, et ils permettent un routage par sauts et par segment à la fois. Nous discutons quelques éléments de mise en œuvre afin de transformer les algorithmes proposés en un protocole de routage, en particulier le format d'en-têtes, le chiffrements des piles de protocoles, le routage par segment, le déploiment sous SDN, etc.

Ensuite, la définition d'un nouveau protocole de routage nécessite une conception et une validation, qui peuvent être facilement modélisées à l'aide d'une structure algébrique appelée *algèbre de routage*. Cette dernière est définie par un ensemble de routes et deux opérations binaires pour composer et choisir les routes. En général, les protocoles de routage standards reposent sur deux opérations, "min" pour la comparaison des poids des chemins et "+" pour calculer le poids d'un chemin à partir des poids des liens qui le composent. Les structures algébriques de base utilisées dans de tels contextes sont les semi-anneaux, les algèbres de fonctions et les algèbres de Sobrinho. Cependant, à ce jour, toutes les algèbres de routage existantes ont été appliquées dans le cas des réseaux monocouches. Autrement dit, elles permettent de modéliser des problèmes de routage sans la prise en compte de tunnels (*ex.,* chemins plus courts, chemins plus larges, chemins plus fiables, etc). Pour cela, nous proposons une modification de ces algèbres afin de modéliser le problème de routage avec tunnels (*i.e.,* chemins faisables) dans un réseau hétérogène et multicouche. En se basant sur cela, nous étudions des propriétés de convergence sur les algèbres proposées.

Enfin, la simulation dans le domaine de la communication et des réseaux informatiques est un outil puissant pour évaluer et analyser de nouveaux protocoles de routage. L'avantage supplémentaire des simulations informatiques est qu'elles ne sont pas limitées par des facteurs matériels. Ainsi des réseaux à plusieurs milliers de noeuds peuvent être simulés. L'évaluation de protocoles nécessite cependant que la topologie du réseau soit configurée de manière appropriée. Dans ce contexte, la collecte de cartes réelles de topologies réseaux et/ou la génération de cartes synthétiques de topologies réseaux est le premier pas à réaliser avant de pouvoir faire des simulations. Or, collecter des cartes de réseaux hétérogènes et multicouches est une tâche complexe et coûteuse. En effet, les nœuds hétérogènes ou multiprotocolaires ne sont pas connus à l'avance dans les outils de collecte de cartes. Ainsi, les zones situées derrière les NATs ou à l'intérieur des opérateurs de réseau ne peuvent pas être facilement explorées. De plus, les modèles de génération des cartes synthétiques de topologies réseaux permettent uniquement de générer des topologies monocouches. C'est pourquoi nous proposons un générateur de topologies de réseaux hétérogènes et multicouches qui possède trois méthodes de génération de réseaux hétérogènes et multicouches (mono-aléatoire, multi-aléatoires et multi-réels).

Organisation de la thèse

Cette thèse se compose en quatre chapitres :

— Le chapitre 1 aborde le principe d'encapsulation et les tunnels dans les réseaux hétérogènes et multicouches. Nous définissons le concept d'encapsulation avec quelques exemples dans les réseaux de communication. Nous expliquons brièvement le problème de faisabilité de chemins avec tunnels. Nous définissons également le modèle formel des réseaux multicouches. Nous expliquons quelques problèmes importants rencontrés dans de tels réseaux.

— Le chapitre 2 traite le problème du routage dans les réseaux hétérogènes et multicouches. Nous étudions le problème de calcul de chemins faisables par fermeture transitive. Nous proposons deux méthodes de décomposition des chemins faisables en sous-chemins. Nous proposons également deux algorithmes de calcul de chemins faisables par fermeture transitive. Nous discutons quelques éléments de mise en œuvre afin de transformer les algorithmes proposés en un protocole de routage.

— Le chapitre 3 étudie l'application des algèbres de routage dans le cas des réseaux hétérogènes et multicouches. Nous étudions la modification des algèbres de routage (semi-anneaux, algèbre de fonctions et algèbre de Sobrinho). Nous proposons des algèbres de routage pour le calcul de chemins avec tunnels. Nous étudions des propriétés de convergence sur les algèbres proposés.

— Le chapitre 4 traite le problème de génération de topologies de réseaux multicouches. Nous définissons différentes méthodes (mono-aléatoire, multi-aléatoires et multi-réels) de génération de topologies de réseaux multicouches. Nous proposons un générateur de topologies de réseaux multicouches.

Enfin, nous finissons par une conclusion générale dans laquelle nous résumons nos contributions et explorons quelques perspectives de recherche. Cette thèse a donné lieu à une publication dans une conférence internationale, une publication dans un atelier international, une publication dans une conférence nationale, un article en révision dans un journal international, un article en soumission dans un journal international, un article en soumission dans une conférence internationale, une bibliothèque d'algorithmes et un logiciel libre et open source publié dans un dépôt Gitlab. Tous les articles sont listés à la page 119.

Chapitre 1

Réseaux hétérogènes et multicouches

Lorsque vous aimez un problème, ses contours, ses obstacles et ses résistances font partie de son caractère.

– Steven Strogatz

Sommaire

1.1 Introduction

L'évolution rapide des technologies de communication a entraîné la création de nombreux protocoles adaptés à des besoins spécifiques. Bien qu'IPv4 et IPv6 restent les protocoles de base d'Internet, l'Internet des Objets (IoT) a donné naissance à des protocoles comme MQTT, CoAP, et Zigbee (avec AODV pour le routage). De même, dans les réseaux à faible puissance et à faible bande passante, tels que les réseaux de capteurs et 6LowPAN, RPL est essentiel pour une communication efficace. Les paradigmes émergents comme le Software Defined Networking (SDN) et les technologies telles que P4 permettent désormais la personnalisation des protocoles pour répondre aux besoins changeants des réseaux modernes.

Les réseaux hétérogènes et multicouches jouent un rôle fondamental dans l'architecture de communication moderne. Dans un monde caractérisé par une connectivité omniprésente, où une multitude de dispositifs et de technologies coexistent, ces réseaux représentent une solution robuste pour répondre à la diversité croissante des besoins en communication. L'utilisation d'encapsulation et l'établissement de tunnels se révèle essentielle dans la gestion de ces réseaux complexes, assurant ainsi une communication fluide entre des environnements technologiquement variés. Ces méthodes sophistiquées proposent des solutions efficaces pour harmoniser la coexistence de protocoles et d'infrastructures disparates au sein d'un même réseau, créant ainsi un écosystème interopérable et cohérent.

Le routage dans un environnement avec diverses encapsulations et tunnels est une opération complexe qui requiert une gestion dédiée. Les protocoles de routage classiques peuvent rencontrer des difficultés à gérer la variété des protocoles de communication. Concrètement, pour qu'un chemin puisse traverser différents protocoles de communication, il doit être compatible avec chacun d'entre eux et garantir une continuité protocolaire. C'est pourquoi, dans de tels contextes, on met souvent en place des mécanismes et protocoles spécifiques pour gérer ces cas particuliers tels que Teredo, 6over4, 6to4, TSP, ISATAP, etc. En revanche, tous les mécanismes d'établissement de tunnels actuels ne sont peut-être pas en mesure de déterminer automatiquement les points d'entrée et de sortie des différents tunnels, ce qui nécessite souvent une intervention manuelle ou une configuration préalable.

Dans ce chapitre, nous expliquons comment assurer la connectivé dans les réseaux hétérogènes et multicouches via le concept d'encapsulation. La section 1.2 définit le principe d'encapsulation et explique brievement son impact sur la présence de tunnels et la faisablité de chemins. Les techniques de calcul de chemins avec la prise en compte de tunnels sont présentées dans la même section. Le modèle formel des réseaux multicouches utilisé dans ce document est présenté dans la section 1.3.1. La section 1.4 explique d'une manière informelle les problèmes rencontrés dans des réseaux mutlicouches. Enfin, la section 1.5 conclut le chapitre.

1.2 Encapsulation et tunnels dans les réseaux

Dans cette section, nous définissons le principe d'encapsulation via des exemples classiques dans le monde réseau. Ensuite, nous expliquons d'une manière informelle la faisabilité de chemins avec la prise en compte des encapsulations. Enfin, nous citons quelques techniques utilisées pour établir les tunnels dans les réseaux de télécommunications.

1.2.1 Encapsulation dans les réseaux

1.2.1.1 Principe d'encapsulation

En général, l'encapsulation d'un objet X dans un autre objet Y permet de mettre l'objet X en entier à l'intérieur de Y sans aucun changement. L'opération inverse, *i.e.,* désencapsulation permet d'extraire l'objet X du Y. Dans un protocole de communication, l'objet de communication est représenté par la structure d'informations appelée *Unité de Données de Protocole* (Protocol Data Unit, PDU). La Figure 1.1 représente la structure générale de l'unité de données de protocole. Cette structure est composée de :

- Un en-tête : Il contient généralement les identifiants ou les adresses nécessaires pour l'acheminement du PDU.
- Un champ de données : Il contient les informations à transmettre.
- Une extension : Elle ne contient aucune informaion utile, et elle est utilisée uniquement si la taille des unités de données est limitée par le protocole.

FIGURE 1.1 – Structure de l'unité de données de protocole (PDU).

L'encapsulation d'un PDU du protocole X dans un autre PDU du protocole Y, consiste à mettre le PDU du protocole X en entier (en-tête et données) dans le champs de données du PDU du protocole Y. La figure 1.2 illustre une telle opération. L'opération inverse ou la désencapsulation du PDU du protocole X du PDU du protocole Y permet d'extraire le PDU encapsulé, *i.e.,* PDU du protocole X (ou simplement les données du PDU du protocole Y) et supprimer le reste du PDU du protocole Y (en-tête et extension).

Il faut noter que dans certains cas la taille du PDU est limitée par le protocole qui le définit. Ceci peut poser des problèmes de compatibilité si le PDU intérieur (à encapsuler) est de taille plus grande que le PDU extérieur. Dans cette situation, des solutions techniques peuvent être mise en place (*ex.,* fixer la taille des PDU, fragmenter les PDU, etc.).

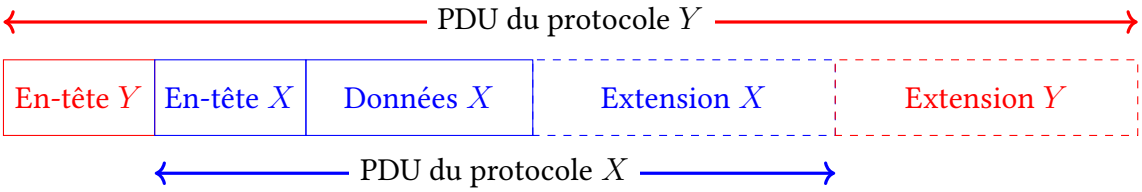


FIGURE 1.2 – Encapsulation d'un PDU du protocole X dans un PDU du protocole Y.

1.2.1.2 Exemples d'encapsulation dans les réseaux

Dans le modèle en couche OSI [50], un PDU est représenté par la trame pour Ethernet, le paquet pour IP, le segment pour TCP et UDP, et les données pour les couches applicatives. Dans la suite, nous citons quelques exemples abstraits d'encapsulations dans les réseaux actuels.

Encapsulation IP sur Ethernet. Le protocole Ethernet est un protocole de la couche physique du modèle OSI, sachant que le protocole IP est un protocole de la couche réseau. L'encapsulation IP sur Ethernet définie dans les RFC [41, 15] permet de transporter des paquets IP dans des trames Ethernet. La figure 1.3 représente l'encapsulation d'un paquet IP dans une trame Ethernet.

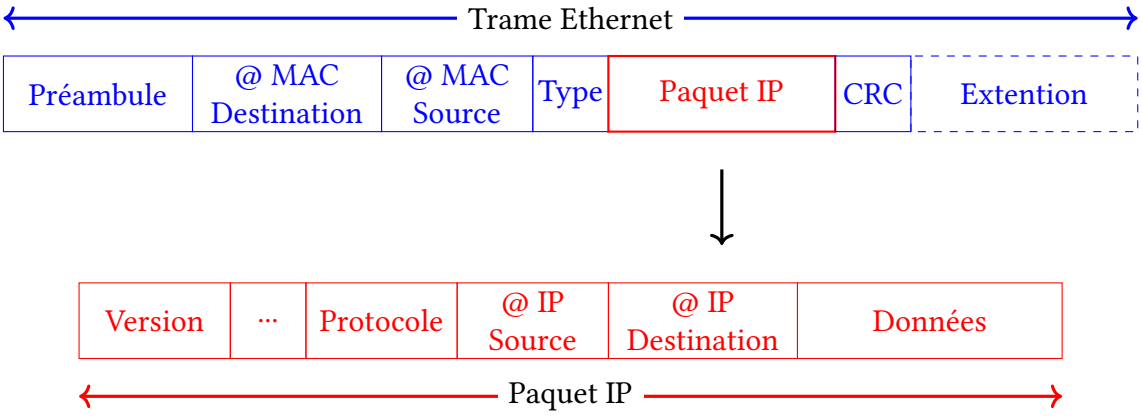


FIGURE 1.3 – Encapsulation d'un paquet IP dans une trame Ethernet.

Encapsulation IP sur IP. Dans l'exemple précédent nous avons vu l'encapsulation entre deux formats différentes de PDU. Mais ce n'est pas toujours le cas, *i.e.*, il est possible d'encapsuler deux PDU de même type. Un exemple correspondant est l'encapsulation d'un paquet IP dans un autre paquet IP. Les RFC [99, 92] définissent une telle encapsulation. La figure 1.4 représente l'encapsulation d'un paquet IP dans un autre paquet IP.

Il existe d'autres exemples d'encapsulation, nous citons également l'encapsulation Ethernet sur MPLS [72], l'encapsulation TDM sur IP [96], l'encapsulation Frame Relay sur MPLS [26], l'encapsulation ATM sur MPLS [25], etc.

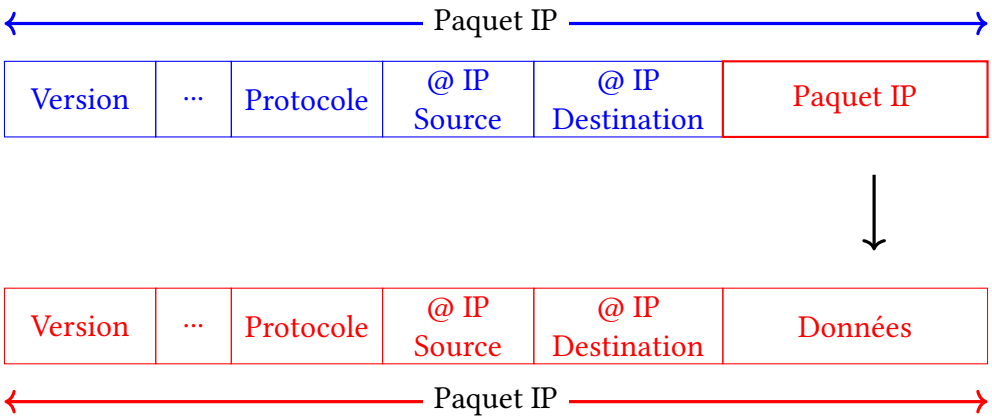


FIGURE 1.4 – Encapsulation d'un paquet IP dans un autre paquet IP.

1.2.2 Faisabilité de chemins avec prise en compte des tunnels

En général, un chemin d'une source à une destination est un ensemble de nœuds (les équipements) reliés entre eux avec des liens (les liens de communication). Dans un réseau classique, tous les nœuds utilisent le même protocole de communication. Or, dans les réseaux actuels, plusieurs protocoles de communication peuvent être omniprésents dans un même réseau. Dans cette situation, certains nœuds sont dit *monoprotocolaires* et d'autres sont *multiprotocolaires* (ou hétérogènes). Un nœud monoprotocolaire est uniquement capable de retransmettre les PDU tels qu'ils sont reçu, en revanche, un nœud multiprotocolaire est capable d'encapsuler (ou désencapsuler) certains protocoles dans d'autres.

Dans ce contexte, un tunnel est une proportion de chemin qui commence par une encapsulation et qui se termine par la désencapsulation correspondante. L'hétérogénéité protocolaire, et plus précisement, la présence de tunnels rend le calcul de chemins plus complexe. Par exemple, un chemin qui passe par une encapsulation doit passer par la désencapsulation correspondante plus tard, ainsi il doit comporter autant d'encapsulations que de désencapsulations. Ces conditions définissent des contraintes de faisabilité, en particulier, un chemin *faisable* est un chemin qui permet une continuité protocolaire. La figure 1.5 représente un réseau avec deux protocoles de communications X et Y. Les nœuds monoprotocolaires qui communiquent le protocole X (resp. Y) sont colorés en bleu (resp. rouge), et ils sont uniquement capables de retransmettre le procotole communiqué. Les nœuds colorés en noir sont multiprotocolaires tels que le nœud A est capable d'encapsuler des PDU X dans des PDU Y et le R est capable de faire l'opération inverse du nœuds A. La suite des nœuds L, A, B, R, I forment un chemin faisable (en vert) de L à I. Ce dernier contient un tunnel entre le nœud A et R avec une pile contenant le PDU X encapsulé dans le PDU Y. En revanche, le chemin du bas (en noir) est non faisable, car le nœud C en bleu ne peut pas désencapsuler le PDU X du PDU Y et il est uniquement capable de traiter les PDU X.

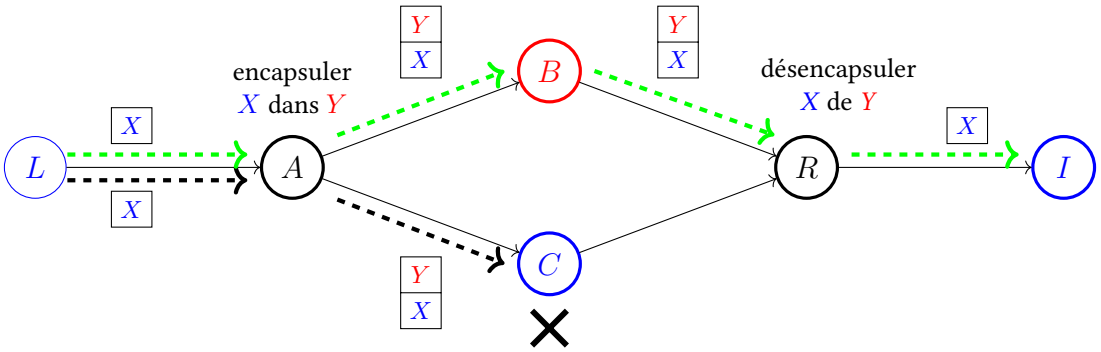


FIGURE 1.5 – Faisabilité de chemins avec la prise en compte des tunnels. L'encapsulation des PDU est représentée par une pile à côté de chaque lien sortant.

1.2.3 Techniques d'établissement de tunnels dans les réseaux

Dans la section précédente, nous avons expliqué la faisabilité de chemins avec la prise en compte des tunnels. Il existe différentes techniques pour établir les tunnels dans les réseaux de télécommunication. Dans la suite, nous citons la plupart de ces techniques avec les références correspondantes de chacune.

Tunnels chiffrés. IPsec [97] ; Wireguard [23] ; SSH [67] ; IEEE 802.11i [45] ; TLS [95].

Tunnels point-à-point. PPTP [116] ; L2TP [106] ; IP-in-IP [99, 92] ; GRE [29, 66] ; 6over4 [10] ; 6to4 [11] ; 6in4 [32] ; TSP [8] ; ISATAP [104] ; Teredo [44] ; L2 pseudowires [91] ; L3 pseudowires [103].

Tunnels multipoints. GMPLS [71] ; MPLS-TE [73] ; VPLS [65] ; IPLS [98] ; 802.1Q [46] ; IEEE 802.1ad [48] ; IEEE 802.1ah [47].

Toutes ces techniques permettent d'établir des tunnels soit par une configuration manuelle soit par un précalcul. D'un côté, les routes sont pré-calculées à l'avance et elles ne sont pas nécessairement optimales en termes de coût de trajet. D'autre côté, le choix des extrémités du tunnel *i.e.,* le calcul des points d'entrée et de sortie n'est pas automatique et il n'est pas géré directement par le protocole de routage.

1.3 Modèle formel d'un réseau multicouche

Dans cette section, nous reprenons le même modèle et les même définitions (pile de protocoles, chemin faisable et table de routage) que dans [59]. Nous généralisons également la définition du chemin faisable par le chemin valide.

1.3.1 Définition du réseau

Un réseau multicouche est un quadruplet $(\mathcal{G}, \mathcal{A}, \mathcal{F}, \omega)$, tels que :

- $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ est un graphe orienté représentant la topologie du réseau. Le nombre de nœuds est noté n et le nombre de liens orientés est noté m,
- $\mathcal{A} = \{x, y, ...\}$ est l'ensemble fini des protocoles disponibles dans le réseau (*ex.*, IPv4 et IPv6), leur nombre est noté λ. L'ensemble des protocoles qu'un nœud U peut recevoir (resp. émettre) est noté par $In(U)$ (resp. $Out(U)$),
- $\mathcal{F}$ est l'ensemble des fonctions d'adaptation disponibles dans le réseau. L'ensemble des fonctions d'adaptation que le nœud U peut effectuer est noté par $\mathcal{F}(U)$. Une fonction d'adaptation f peut être une :
  - *Conversion* : l'en-tête d'un paquet d'un protocole x est converti en en-tête d'un autre protocole y sans aucun changement dans les protocoles sous-jacents (*i.e.*, déjà encapsulés). Cette fonction est notée $(x \rightarrow y)$ (*ex.*, conversion IPv4/IPv6 via NAT-PT [105]). Si les paquets reçus et émis sont du même protocole, *i.e.*, une retransmission classique, il est noté $(x \rightarrow x)$. Ainsi, nous considérons une retransmission classique comme un type particulier de conversion ;
  - *Encapsulation* : l'ensemble du paquet de protocole x est encapsulé dans le champ de données d'un paquet de protocole y. Cette fonction est notée $(x \rightarrow xy)$ (*ex.*, encapsulation d'IPv4 dans IPv6). Notez qu'un paquet peut être encapsulé dans un autre du même protocole, comme pour IP-in-IP (*i.e.*, RFC 1853 [99] et RFC 2003 [92]). Dans ce cas, la fonction est simplement notée $(x \rightarrow xx)$;
  - *Désencapsulation* : un paquet de protocole x est extrait du champ de données d'un paquet de protocole y, c'est l'opération inverse de $(x \rightarrow xy)$, et donc il est noté $(xy \rightarrow x)$. À noter que cette opération ne peut être effectuée que si le paquet reçu de protocole y contient effectivement un paquet de protocole x dans son champ de données.
- $\omega : \mathcal{V} \times \mathcal{F} \times \mathcal{V} \rightarrow W$ est une fonction de poids où $\omega(U, f, V)$ est le coût d'utilisation de la fonction d'adaptation f par U et de l'envoi du paquet résultant sur le lien (U, V). Sa somme sur un chemin est le coût que nous voulons minimiser, ou maximiser, etc. Cette fonction est très générique et elle peut modéliser n'importe quelle métrique additive. Par exemple, si les poids sont des nombres réels ($W = \mathbb{R}$) alors la fonction de poids peut modéliser le nombre de sauts (en posant $\omega(U, f, V) = 1$ pour tous les triplets possibles), ou le nombre d'encapsulations (en posant $\omega(U, f, V) = 1$ lorsque f est une encapsulation et $\omega(U, f, V) = 0$ sinon), etc.

Un exemple d'un tel réseau, où les protocoles IPv4 et IPv6 coexistent, est illustré dans la figure 1.6.

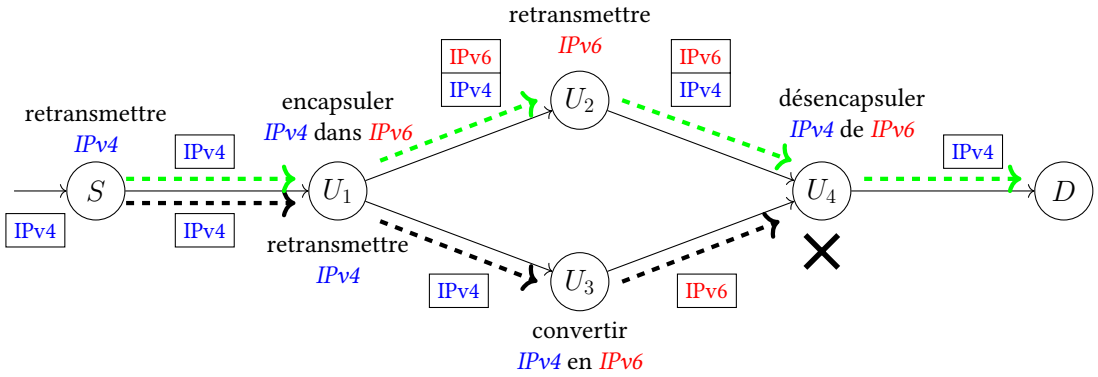


FIGURE 1.6 – Exemple de réseau englobant les conversions et encapsulations des protocoles IPv4 et IPv6. Le chemin du haut en vert est faisable (et valide), et contient un tunnel de U_1 à U_4. Le chemin du bas en noir n'est pas faisable. La pile de protocoles spécifique à chaque chemin est affichée à côté de chaque lien sortant.

1.3.2 Pile de protocoles

L'encapsulation entre les protocoles est représentée par une pile de protocoles H. Par exemple, le protocole x encapsulé dans le protocole z qui est lui même encapsulé dans un autre y est représenté par la pile xzy (le sommet de la pile est celui qui est le plus à droite). On notera par $Top(H)$ le sommet de la pile H et par $Hei(H)$ la hauteur de la pile H, *i.e.*, le nombre de protocoles encapsulés. Par abus de notation, on pourra considérer une pile de hauteur 1 ou sans protocole encapsulé comme étant un protocole.

Une séquence de fonctions d'adaptation appliquée à une pile de protocoles de départ H_{in} induit une pile de protocoles d'arrivée H_{out}. Par exemple, la séquence $(x \rightarrow xy)(y \rightarrow y)$ appliquée à la pile x induit la pile xy. On notera $f(H)$ l'application de la fonction d'adaptation f à la pile H. Parfois, l'application n'est pas possible. Par exemple, si $H = xx$ et $f = (y \rightarrow x)$, on ne pourra pas convertir le sommet de pile y en x vu que ce protocole est différent de y, *i.e.*, $Top(H) = x$. Dans cette situation, nous notons $f(H) = \phi$ (pile interdite), ainsi on ne pourra pas aller plus loin, *i.e.*, $f(\phi) = \phi$ quelle que soit la fonction d'adaptation f. On notera $\overline{f}$ la fonction inverse de f. Par exemple, si $f = (x \rightarrow y)$ alors $\overline{f} = (y \rightarrow x)$. La fonction inverse d'une encapsulation est la désencapsulation correspondante. La pile de protocole H_{out} induite par une séquence de fonctions $f_0 f_1 \ldots f_k$ appliquée à la pile H_{in} est récursivement définie comme suit :

$$H_{i+1} = f_i(H_i) \quad \text{pour tout} \quad 0 \leq i \leq k \quad \text{où} \quad H_0 = H_{in} \text{ et } H_{k+1} = H_{out}$$

Dans la figure 1.6, l'encapsulation de l'IPv4 dans l'IPv6 sur le nœud U_1 suivie par la retransmission de l'IPv6 sur le nœud U_2 et appliquée la pile de départ IPv6, induit la pile d'arrivée IPv4.IPv6. L'évolution de la hauteur de la pile de protocoles de cet exemple est représentée dans la Figure 1.7.

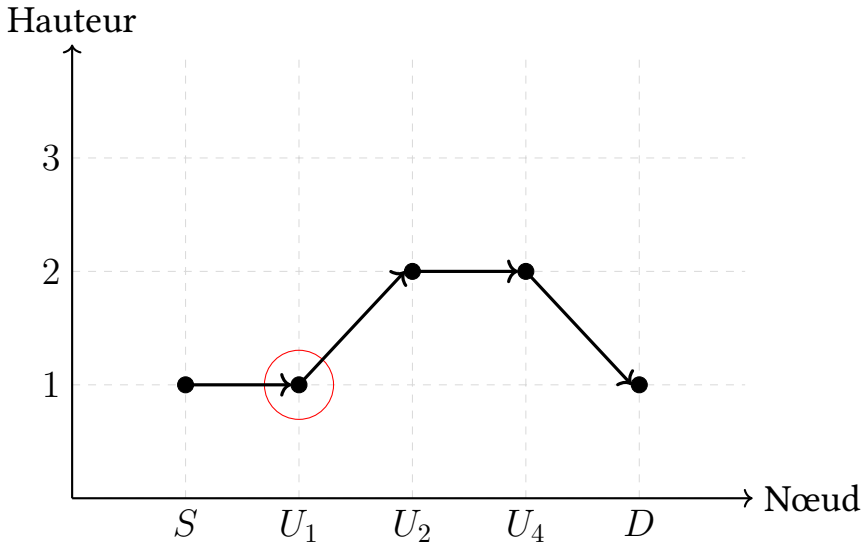


FIGURE 1.7 – Evolution de la hauteur de la pile de protocoles du chemin faisable (et valide) représentée dans la figure 1.6. Le pas montant représente l'encapsulation, le pas descendant représente la désencapsulation et les pas horizontaux sont des conversions. Le point entouré en rouge est une vallée.

1.3.3 Chemin faisable et chemin valide

Dans un contexte multicouche, un chemin faisable doit permettre à ses deux extrémités (et plus généralement à tous les nœuds qui le composent) de communiquer, *i.e.*, qu'il permet une continuité protocolaire. Il est composé d'une suite de nœuds, de fonctions d'adaptation et de piles. Plus formellement, un chemin $\mathcal{P}$ d'un nœud source S à un nœud destination D est une séquence $H_{in}Sf_0U_1f_1 \dots U_kf_kH_{out}D$, où chaque $U_i, i = 1, \dots, k$, est un nœud, et chaque f_i est une fonction d'adaptation disponible sur le nœud U_i. On dit que ce chemin est *faisable* si et seulement si :

1. La suite de nœuds $SU_1 \dots U_kD$ est un chemin orienté (au sens classique) dans le graphe $\mathcal{G} = (\mathcal{V}, \mathcal{E})$,
2. La suite de fonctions $f_0f_1 \dots f_k$ appliquée à la pile H_{in} induit la pile $H_{out} \neq \phi$,
3. Les piles H_{in} et H_{out} sont deux protocoles, *i.e.*, $Hei(H_{in}) = 1$ et $Hei(H_{out}) = 1$.

Il faut noter que dans [59], les auteurs rajoutent à cette définition l'hypothèse suivante :

— la source S applique une fonction f_0 de type retransmission classique $f_0 = (x \rightarrow x)$.

Cette hypothèse est un cas particulier de notre définition et elle n'impacte pas les résultats.

L'une des propriétés importantes des chemins faisables est la hauteur des piles de protocoles induites par chacune de ses fonctions d'adaptation. Un chemin faisable dont la hauteur de toutes ses piles de protocoles (sans les piles H_{in} et H_{out}) est toujours supérieure à 1 est dit sans-vallée. Plus formellement, soit $H_1, H_2, \dots H_k$ les piles de protocoles

induites dans un chemin faisable $\mathcal{P}$ par les fonctions d'adaptation $f_0, f_1, \ldots, f_k$. On dit que $\mathcal{P}$ est *sans-vallée* si est seulement si :

$$Hei(H_i) > 1 \quad \text{pour tout} \quad 1 \leq i \leq k$$

La définition précédente de chemin faisable est restreinte, et elle permet uniquemenet de spécifier les chemins dont la source émet un paquet sans encapsulation et la destination reçoit aussi un paquet sans encapsulation. Afin de généraliser cette dernière, *i.e.,* définir tous les types de chemins dans un modèle multicouche (en particulier les tunnels), nous définissons un type général du chemin faisable dit chemin valide. Un chemin $\mathcal{P} = H_{in}Sf_0U_1f_1 \ldots U_kf_kH_{out}D$ de S à D est *valide* si et seulement si :

- Mêmes conditons (1) et (2) qu'un chemin faisable,
- Les piles H_{in} et H_{out} peuvent contenir des protocoles encapsulés, *i.e.*, $Hei(H_{in}) \geq 1$ et $Hei(H_{out}) \geq 1$.

Dans la figure 1.6, Le chemin du haut (en vert) représente un chemin valide et faisable de S à D avec la pile H_{in}=IPv4 et la pile H_{out}=IPv4. Il contient un tunnel de U_1 à U_4 avec une pile de protocoles IPv4.IPv6. Le chemin du bas (en noir) est non faisable entre S et D, puisque le nœud U_4 reçoit le protocole IPv6 et il ne peut pas désencapsuler le protocole IPv4 du IPv6. Formellement, la pile en sortie de U_4 sur ce chemin est la pile interdite ϕ. La figure 1.7 représente la hauteur des piles de protocoles induites par les fonctions d'adaptation du chemin faisable entre S et D. Ce chemin contient une vallée au niveau du nœud U_1. Cependant, les deux sous-chemins de S à U_1 et de U_1 à D représentent des chemins faisables sans-vallées.

La longueur d'un chemin $\mathcal{P}$ (ou le nombre de sauts) est le nombre de liens qui le composent notée par $|\mathcal{P}|$. Le coût d'un chemin faisable (ou plus généralement valide) [1] $\mathcal{P} = H_{in}Sf_0U_1f_1 \ldots U_kf_kH_{out}D$ d'un nœud S à un nœud D est la somme des poids de ses liens et ses fonctions d'adaptation. Il est noté par,

$$\omega(\mathcal{P}) = \sum_{i=0}^{k} \omega(U_i, f_i, U_{i+1}) \quad \text{où} \quad U_0 = S \text{ et } U_{k+1} = D$$

Un chemin optimal est un chemin faisable (valide) qui optimise le poids $\omega(\mathcal{P})$.

1.3.4 Table de routage

Une table de routage $\mathcal{T}_U$ d'un nœud U est un ensemble de lignes, où chaque ligne représente un chemin valide (faisable) de la forme $(D, H_{in}, H_{out}, V, f, \omega, DST)$ tels que :

- D est la destination à atteindre,
- H_{in} est la pile qu'un paquet doit contenir pour parcourir la route de U à D,

1. Le coût d'un chemin invalide est non défini.

— H_{out} est la pile de protocoles reçue par D sur cette route,
— V est le prochain voisin sur cette route (next hop),
— f est la fonction que U applique à un paquet reçu avant de le transmettre à V,
— ω est le coût de cette route,
— DST est la pile de destinations pour le routage par segment.

Le clé de la table de routage est représentée par le triple (D, H_{in}, H_{out}) et elle permet la gestion de cette table (l'ajout d'une ou plusieurs routes, la mise à jour, etc.). Dans la suite, nous notons par $Top(DST)$ l'identifiant de la destination au sommet de la pile DST. L'empilement d'un identifiant ou une liste d'identifiants D est notée par $Push(DST, D)$ et l'opération inverse, *i.e.,* dépilement est notée par $Pop(DST)$.

1.4 Problèmes rencontrés dans les réseaux multicouches

Dans cette section, nous introduisons informellement quelques problèmes intéressants dans les réseaux multicouches. Nous expliquons également l'incomplétude des différentes solutions proposées dans ce contexte. Enfin, nous soulignons brièvement certaines idées qui permettent de résoudre l'intégralité de ses problèmes.

1.4.1 Établissement automatique de tunnels

Dans la section 1.2, nous avons vu quelques exemples d'encapsulation dans les réseaux. Nous avons vu également que la présence d'encapsulation dans un réseau rend le processus de routage (*i.e.,* le calcul de chemins) plus complexe, en particulier la présence de tunnels. Malheureusement, le calcul des chemins dans les réseaux avec des tunnels n'est pas encore entièrement automatisé. Bien que plusieurs tentatives d'automatisation aient eu lieu (*ex.,* Teredo [44], 6over4 [10], 6to4 [11], TSP [8], ISATAP [104], etc.), elles ne résolvent pas entièrement le problème. En fait, la plupart de ces méthodes nécessitent l'accès à un système de noms de domaine (DNS) ou à un autre serveur dédié. Dans de tels serveurs, les routes sont pré-calculées. De plus, ils ne sont pas nécessairement optimaux en termes de coût de trajet. Surtout, aucun d'entre eux n'est en mesure de déterminer automatiquement les points de sortie des différents tunnels. Cette tâche est effectuée soit par configuration manuelle, soit par pré-calcul. De plus, même si l'établissement dynamique de tunnels et la configuration (TSP) Tunnel Setup Protocol [8] automatisent une partie de la négociation des paramètres, ils sont très limités et le choix des extrémités du tunnel n'est pas géré par le protocole de routage. Au vu du nombre énorme et croissant d'appareils connectés, cette approche n'est plus durable.

À ce stade, nous préconisons que la solution la plus simple consiste à calculer les chemins les plus courts avec la prise en compte des tunnels (y compris les points d'entrée et de sortie) directement par le protocole de routage. Une telle solution permettrait

une automatisation complète et un recalcul rapide du chemin. Les instances de protocole de routage, qui s'exécutent en continu, pourraient s'adapter aux événements de changements de topologie. Dans ce contexte, les protocoles de routage existants s'appuient sur les algorithmes classiques de calcul de chemins (*ex.,* Dijkstra [20], Bellman-Ford [5], Floyd-Warshall [109], etc.). Or, ces derniers ne fonctionnent plus dans un réseau multicouche, car ils ne peuvent gérer les encapsulations et les piles de protocoles correspondantes. Il se trouve que des problèmes simples de routage dans un réseau classique deviennent très difficiles dans un réseau multicouche. Par exemple, le calcul d'un chemin sous contrainte de bande passante est trivialement polynomial dans un réseau classique, mais est NP-difficile dans un réseau multicouche [58], de même, la longueur d'un chemin dans un tel réseau peut être exponentielle [60]. Pour cela, une approche efficace consiste à concevoir des algorithmes de calcul de chemin qui prennent en compte les encapsulations et les conversions. Ces algorithmes peuvent être basés sur des techniques classiques (une généralisation aura lieu) ou sur des nouvelles techniques. Le chapitre 2 permet de traiter les problématiques citées précédement, en proposant des algorithmes de calcul de chemins faisables dans un réseau hétérogène et multicouche.

La définition d'un protocole de routage s'appuie sur deux étapes importantes. La première étape ou conception consiste à spécifier la manière dont les nœuds d'un réseau communiquent (les algorithmes de routage) via des en-têtes spécifiques (le format des PDU). La seconde étape ou validation est l'étape la plus cruciale. Cette dernière doit assurer le bon fonctionnement du protocole conçu sur les deux plans théorique et pratique, en particulier sa correction et sa convergence. L'analyse classique des protocoles de routage est trop liée aux spécificités d'un système particulier, ceci peut entraver une large compréhension de ce qui peut et ne peut pas être accompli avec ces protocoles en termes de convergence et de caractéristiques des chemins vers lesquels convergent les protocoles. À titre d'exemple, nous citons le problème de routage avec des métriques composées (QoS) [24], le problème de convergence absolue en mode asynchrone des protocoles de routage à vecteurs [18].

Une piste intéressente pour faire face à ses problèmes est la séparation entre le mécanisme de routage (comment les routes sont échangées et mises à jour) et la politique de routage (comment les routes sont décrites et comparées). L'outil mathématique qui permet une telle séparation est connu sous le nom d'algèbre de routage. Cette dernière permet de modéliser un protocole de routage par une structure algébrique. À ce niveau, il existe trois styles d'algèbre de routage (semi-anneau [12, 77, 24], algèbre de Sobrinho [101, 100, 37] et algèbre avec fonctions [17, 18]). Jusqu'à aujourd'hui, les travaux existants sur l'algèbre de routage se sont principalement concentrés sur l'application de cette approche aux protocoles de routage qui sont généralement utilisés dans les réseaux qui ont un seul protocole de communication. Or, ces travaux ne sont pas adaptés aux réseaux hétérogènes et multicouches et à la prise en compte des tunnels. Il se trouve que des problèmes difficiles dans un réseau classique, comme cité précedemment, deviennent très complexes et même incompréhensibles dans un réseau multicouche. Pour cela, développer une algèbre de routage pour les réseaux multicouches sera bénifique pour la compréhension de ce genre de problèmes. Dans le chapitre 3, nous étudions l'application

de l'algèbre de routage dans un réseau multicouche où les tunnels sont omniprésents.

1.4.2 Disponibilité de topologies de réseaux multicouches

Dans la section précédente, nous avons cité le modèle théorique (algèbre de routage) le plus répandu pour étudier la correction et la convergence d'un protocole de routage. Ici, nous nous intéressons à la validation d'un protocle de routage sur le plan pratique. Autrement dit, le tester et l'évaluer via des scénarios de simulations. À ce niveau, le premier pas à réaliser est la préparation de l'ensemble des données (*i.e.,* topologies) nécessaires pour effectuer les simulations. Il existe deux façons pour préparer l'ensemble des topologies, par la collecte des cartes de réseaux réelles [114, 9, 108] ou par la génération des cartes de réseaux synthétiques [111]. Or, collecter des cartes hétérogènes et multicouches est une tâche complexe et coûteuse, puisque on ne connait pas à l'avance les nœuds hétérogènes ou multiprotocolaires. Par exemple, l'identification des nœuds en double-pile dans le cas d'un réseau IPv4/IPv6 s'appuie sur la surveillance des messages DNS ou avec l'utilisation des paquets ICMPv6 via l'adresse multicast IPv6 [115]. L'une des techniques utilisée est basée sur une hiérarchie de serveurs de noms à deux niveaux dans laquelle le serveur de premier niveau, accessible via IPv4, renvoie les enregistrements du serveur de second niveau. Dans sa réponse DNS, il encode également l'adresse IPv4 du client contactant. Chaque requête arrivant au serveur de noms de second niveau via IPv6 révèle la requête IPv4 initiale. Cette technique a découvert 674K paires de candidats sur une période de six mois [6].

D'où l'intéret de la génération des cartes de réseaux synthétiques. Il exite différentes méthodes pour générer une topologie réseau tout dépend du méchanisme de création de liens utilisé. Dans ce contexte, nous citons les générateurs de type *topologique plat* [111], *hiérarchique* [112, 22, 113], *lois de puissance* [28, 74, 35, 2, 87, 1], *échantillonnage cartographique* [68], *autres* [52, 70, 94]. Tous les générateurs cités précédemment permettent uniquement de générer des topologies réseaux monocouches, et ils ne sont pas adaptés aux réseaux hétérogènes et multicouches. Rappelons qu'un réseau hétérogène et multicouche est modélisé par un graphe sous-jaçent représentant la topologie où chaque nœud dispose d'un ensemble de fonctions d'adaptation. Pour cela, l'idée la plus simple pour créer un générateur de topologies réseaux multicouches est de séparer le processus en deux étapes. La première étape consiste à générer le graphe sous-jacent. Cela peut être réalisé par l'un des générateurs de topologie classiques (plus généralement générateurs de graphes). La seconde étape permet de distribuer les fonctions d'adaptation sur les nœuds de la topologie générée, *i.e.* construire l'ensemble des fonctions de chaque nœud.

En général, placer des fonctions d'adaptation dans un réseau multicouche est une tâche difficile car les chercheurs n'ont généralement pas accès à de telles informations privilégiées du monde réel. Les instantanés de topologie d'Internet sont généralement principalement cartographiés à partir de l'espace d'adressage public, car les zones situées derrière les NAT ou à l'intérieur des opérateurs de réseau ne peuvent pas être

facilement explorées. De plus, Internet est un système distribué qui n'appartient à aucune entité administrative unique. Par conséquent, il n'est pas possible d'utiliser des algorithmes centralisés pour optimiser le placement des convertisseurs de protocole et des tunnels. Ce problème de placement est un problème d'optimisation classique de la classe NP-difficile [57], qui consiste à trouver l'emplacement optimal pour un ensemble d'objets, étant donné un ensemble de contraintes. Diverses approches existent pour résoudre ce problème, telles que la programmation linéaire, l'ensemble dominant et les approches d'apprentissage automatique. Dans notre cas de génération de topologie de réseau multicouche, nous visons à trouver un emplacement pour ces fonctions d'adaptation qui augmente l'accessibilité, c'est-à-dire la faisabilité du chemin, dans le réseau multicouche. Autrement dit, nous visons à produire des réseaux multicouches où, à tout le moins, n'importe quel nœud peut atteindre n'importe quel autre nœud, quelle que soit sa pile de protocoles. Cependant, la contrainte d'accessibilité ou faisabilité du chemin n'est pas facile à résoudre dans un tel réseau. Dans le chapitre 4, nous abordons ces problématiques, en proposant différents mécanismes de génération de topologies de réseaux hétérogènes et multicouches.

1.5 Conclusion

Dans un réseau hétérogène et multicouche, plusieurs protocoles de communication sont omniprésents. Afin d'assurer la communication dans de tels réseaux, le mécanisme d'encapsulation a été mis en place. Cette opération consiste à mettre les unités d'informations d'un protocole à l'intérieur des unités d'informations d'un autre protocole. La présence des encapsulations dans un chemin d'une source à une destionation permet de créer un ou plusieurs tunnels. À ce niveau, le calcul de chemin avec la présence de tunnels nécessite de prendre en compte les contraintes de compatibilité.

Dans ce chapitre, nous avons d'abord défini le principe d'encapsulation et nous avons cité quelques exemples d'encapsulation dans les réseaux de télécommunication. Nous avons également expliqué la faisabilité de chemins avec la prise en compte des tunnels, et nous avons cité la plupart des techniques d'établissement de tunnels dans les réseaux actuels. Ensuite, nous avons présenté le modèle formel des réseaux multicouches utilisé tout au long de ce document. Enfin, nous avons expliqué quelques problèmes rencontrés dans les réseaux multicouches.

Chapitre 2

Routage dans les réseaux hétérogènes et multicouches

Si vous pouvez le résoudre, c'est un exercice ; sinon c'est un problème de recherche.

– Richard Bellman

Sommaire

2.1 Introduction

Jusqu'à présent, l'automatisation complète du routage dans les réseaux avec tunnels, demeure une tâche complexe. Malgré les diverses tentatives d'automatisation telles que Teredo, 6over4, TSP, etc., elles n'ont pas encore résolu intégralement cette problématique. En réalité, dans la plupart de ces approches, les routes sont pré-établies et ne sont pas toujours les plus optimales en termes de coût de trajet. De plus, aucune de ces méthodes ne parvient à déterminer automatiquement les points de sortie des divers tunnels. Ainsi, cette tâche requiert soit une configuration manuelle, soit une pré-détermination. Cela découle de la complexité inhérente à la conception d'algorithmes de calcul de chemins qui tiennent compte des encapsulations et des piles de protocoles correspondantes.

Pour l'instant, nous ne disposons que d'un seul algorithme distribué connu sous le nom de *Stack-Vector*, basé sur une généralisation de l'algorithme Bellman-Ford en remplaçant le vecteur de distance par un vecteur de piles de protocoles. Malheureusement, cet algorithme présente plusieurs inconvénients, en particulier en ce qui concerne son temps de convergence, qui s'avère très élevé. Une approche efficace consiste à concevoir un algorithme de calcul de chemins avec décomposition de chemins ou simplement fermeture transitive : deux chemins sont concaténés pour n'en former qu'un seul. L'algorithme de fermeture transitive le plus connu est celui de Floyd-Warshall. Cependant, son application directe à notre contexte se heurte à un obstacle majeur : la compatibilité des chemins. Par exemple, il peut arriver qu'un premier chemin se termine avant la fin d'un tunnel tandis que le second commence par deux tunnels imbriqués, le tout sur un même nœud. Ainsi, la conception d'un tel algorithme exige une identification préalable des cas dans lesquels la concaténation de deux chemins avec tunnels est réalisable.

Dans ce chapitre, nous étudions l'opération de fermeture transitive des chemins faisables et des chemins valides et nous proposons des algorithmes de calcul de chemins avec tunnels en utilisant cette nouvelle technique. La section 2.2 résume les travaux connexes et détaille le problème en question. Dans la section 2.3, nous étudions d'abord la fermeture transitive des chemins faisables. En se basant sur cela, nous présentons une amélioration de l'algorithme Stack-Vector avec fermeture transitive. Enfin, nous généralisons cette opération et nous proposons un algorithme de fermeture transitive complet *à la* Floyd-Warshall. La section 2.4 présente les résultats de simulations effectuées par nos algorithmes et deux implémentations parallélisées de l'algorithme Stack-Vector d'origine. Dans la section 2.5, nous décrivons quelques éléments pour implémenter un protocole de routage basé sur nos algorithmes. Enfin, la section 2.6 conclut le chapitre.

Ce chapitre est basé sur nos travaux sur les algorithmes de calcul de chemins avec établissement automatique de tunnels dans les réseaux hétérogènes et multicouches, qui ont été publiés dans la conférence internationale IFIP Networking 2022 [80], et dans la conférence nationale AlgoTel [82]. La version complète est actuellement en révision pour le journal international IEEE/ACM Transactions on Networking (ToN) [81]. L'implémentation des algorithmes est disponible gratuitement sur le dépôt Gitlab de l'Université de Bordeaux [79].

2.2 Autour du calcul de chemins dans les réseaux multicouches

Dans cette section, nous définissons d'abord le problème de calcul de chemins dans les réseaux multicouches. Nous discutons également les propriétés importantes de ce problème. Enfin, nous citons la plupart des solutions algorithmiques proposées pour résoudre ce dernier problème.

2.2.1 État de l'art

Les auteurs de [58] proposent le premier algorithme s'attaquant à ce problème. Ils proposent un algorithme Breadth-First Search (BFS) qui explore tous les chemins possibles dans une approche recherche exhaustive ou brute-force. Cet algorithme est exponentiel dans le pire des cas (indépendamment de la longueur du chemin). Les auteurs de [62] proposent un algorithme polynomial (par rapport à la taille du réseau et à la longueur du chemin le plus court) dans le cas sans contrainte de bande passante. Leur solution est une approche basée sur la théorie des langages et permet de calculer le chemin optimal entre deux nœuds en minimisant soit le nombre de sauts dans le chemin, soit le nombre de fonctions d'adaptation. Dans [49], les auteurs proposent un modèle matriciel et quelques algorithmes associés pour résoudre le problème des plus courts chemins ayant une longueur d'au plus k. Les algorithmes proposés sont exponentiels et ne permettent pas le calcul de chemins contenant des cycles. Dans [64], les auteurs généralisent largement le travail de [62]. Leur algorithme calcule le chemin le plus court en fonction de toute métrique additive choisie par l'utilisateur (liens pondérés, fonctions d'adaptation pondérées, etc.). Ils proposent également des heuristiques (polynomiales) pour calculer le plus court chemin sous des contraintes de bande passante. Ils généralisent les travaux de [107] pour proposer un algorithme exponentiel qui calcule le plus court chemin possible sous plusieurs contraintes de *Quality of Service* (QoS).

Tous ces travaux cités précédemment proposent des algorithmes centralisés. De plus, ces algorithmes calculent des chemins mais pas des tables de routage. Ainsi, ils ne conviennent qu'au routage source. Le premier algorithme entièrement distribué pour résoudre le problème est proposé dans [59]. C'est une généralisation de l'algorithme de Bellman-Ford, un algorithme à vecteur de distance, d'où le nom d'algorithme *Stack-Vector*. L'idée principale est de propager la distance (comme dans Bellman-Ford) mais aussi la pile de protocoles. Chaque nœud envoie un message à ses voisins pour les informer qu'il peut atteindre une destination à une certaine distance, et si le paquet reçu possède une certaine pile de protocoles. Les cycles étant possibles, la terminaison de l'algorithme est garantie par la hauteur maximale de la pile de protocoles (voir section 2.2.2.2). Cet algorithme est adapté au routage saut par saut puisqu'il construit des tables de routage pour chaque nœud. Les extrémités des tunnels dans certains chemins faisables sont automatiquement calculées pendant le processus de calcul de chemins et

elles sont stockées dans les tables de routage.

2.2.2 Problème de calcul de chemins faisables

2.2.2.1 Définition et complexité du problème

Étant donné un réseau multicouche $\mathcal{N}$, le problème qu'on cherche à résoudre est de calculer le plus court (optimal) chemin faisable $\mathcal{P}$ entre chaque paire de nœuds S et D (l'existence d'un tel chemin n'est pas garantie même si le graphe sous-jacent $\mathcal{G}$ est fortement connexe). Ceci correspond au problème *All-Pairs Shortest Path* (APSP) dans un réseau impliquant des fonctions d'adaptation. Plus précisément, on veut calculer la table de routage de chaque nœud, afin qu'un paquet entre la source et la destination emprunte le chemin le plus court. Plus formellement, le problème est le suivant :

$$\min \omega(\mathcal{P}) \;=\; \sum_{(U,f,V)\in\mathcal{P}} \omega(U,f,V) \quad \text{où} \quad \mathcal{P} \text{ est un chemin faisable entre } S \text{ et } D$$

Ce problème a été prouvé NP-difficile sous la contrainte de bande passante par Kuipers et (Freek) Dijkstra [58]. Mais cette preuve ne fonctionne pas sur les graphes orientés symétriques. Cependant, la plupart des réseaux de communication sont symétriques. Lamali et al. [60] ont montré que la version décisionnelle du problème reste NP-difficile même avec deux protocoles et dans un graphe symétrique.

2.2.2.2 Propriétés des chemins faisables avec tunnels

Rappelons que le calcul de chemins dans les réseaux englobant des fonctions d'adaptation ne peut pas être effectué en utilisant des algorithmes de calcul de chemins classiques (*ex.,* Dijkstra, Bellman-Ford, Floyd-Warshall, etc.). Ces algorithmes ne prennent pas en compte les fonctions d'adaptation. De plus, les plus courts chemins dans de tels réseaux ont plusieurs propriétés non triviales. Dans la suite, nous citons quelques propriétés des chemins faisables.

Chemins avec circuits. Un plus court chemin faisable (voir l'unique chemin faisable) peut avoir des circuits [21, 58, 62]. Le cas trivial est le passage par une suite d'encapsulations, afin de construire une pile de protocoles nécessaire pour atteindre une destination. Dans la figure 2.1, le chemin direct de S à D en passant par les nœuds U_1, U_2, U_5, U_6 et en appliquant la fonction d'adaptation de chaque nœud n'est pas faisable, car le nœud U_5 ne peut pas effectuer sa fonction d'adaptation (extraire le protocole x du y) sur la pile de protocoles y. En revanche, la figure 2.2 décrit l'unique chemin faisable entre S et D dans le même réseau. Ce chemin passe par tous les nœuds du réseau en formant un circuit sur les nœuds U_2, U_3, U_4, U_2. Ce circuit permet de construire la pile de protocoles nécessaire afin de traverser les nœuds U_5 et U_6.

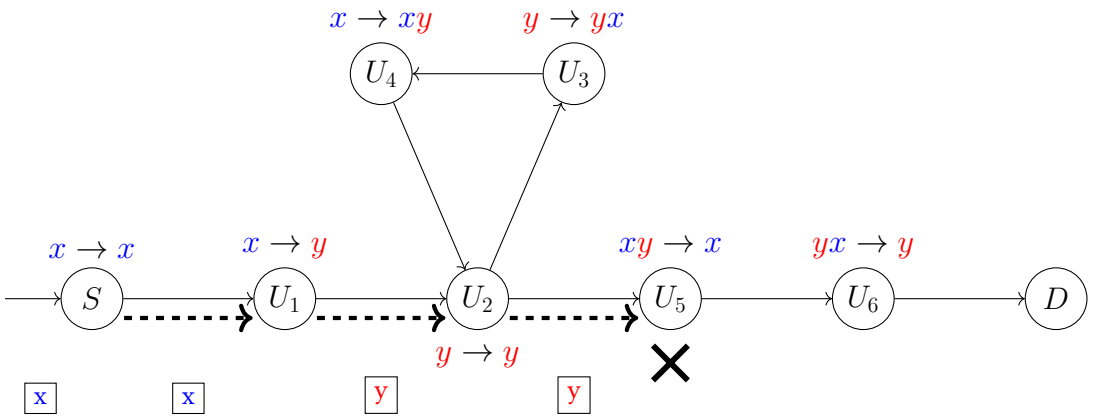


FIGURE 2.1 – Exemple d'un chemin direct mais non faisable dans un réseau multicouche.

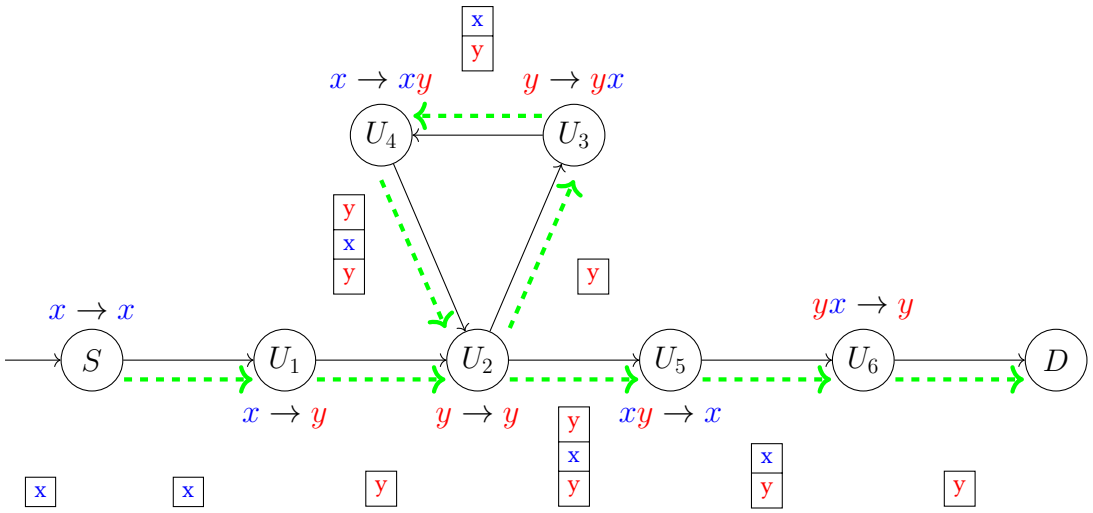


FIGURE 2.2 – Exemple d'un chemin faisable avec circuit dans un réseau multicouche.

Sous-structure optimale. La programmation dynamique est un outil important pour résoudre des problèmes d'optimisation. Elle permet de trouver la solution d'un problème en combinant des solutions de sous-problèmes. Cette méthode peut être appliquée uniquement si le problème possède une sous-structure optimale, autrement dit, si la solution optimale à ce problème peut se décomposer en sous-solutions optimales aux sous-problèmes composant le problème global. Par exemple, le calcul des plus courts chemins dans les graphes où tous les sous-chemins d'un plus court chemin sont plus courts. Dans un contexte multicouche, cette propriété n'est pas toujours satisfaite, car un sous-chemin d'un chemin faisable n'est pas forcément faisable. Dans la figure 2.2, le chemin de S à U_5 est un sous-chemin du chemin faisable de S à D. Ce sous-chemin n'est pas faisable, car la pile d'arrivée au niveau du nœud U_5 n'est pas de hauteur 1 et elle contient encore des protocoles encapsulés. Le seul cas où un chemin faisable peut avoir des sous chemins faisables est lorsqu'il contient des vallées. Cependant, un sous-chemin d'un chemin valide (sans-vallée ou pas) est toujous un chemin valide. En prenant en compte le même exemple précédent, le chemin de S à U_5 est un sous-chemin valide du chemin valide de S

à D. Cette dernière propriété permet l'utilisation de la programmation dynamique pour résoudre notre problème.

Chemins de longueur exponentielle. En général, il est évident que la longueur d'un chemin sans circuits ne dépasse pas le nombre de nœuds n. Or, nous avons vu dans la propriété précédente qu'un chemin multicouche peut comporter des circuits. La figure 2.3 représente un réseau multicouche avec deux protocoles x et y, dont la longueur du plus court chemin faisable $\mathcal{P}$ entre S et D n'est pas linéaire en fonction de n. Ce chemin permet à la source S et la destination D de communiquer via le protocole x. Il est composé de deux parties de longueur k, où $k = n^2 - 1$ (on suppose que n est pair). La première est un circuit formé par les nœuds $(U_1, \ldots, U_k)$. Tous ces nœuds sont uniquement capables de retransmettre un protocole tel qu'il est reçu, sauf le nœud U_k qui est capable d'encapsuler n'importe quel protocole dans y. La deuxième partie est un chemin composée par les nœuds $(V_1, \ldots, V_k)$. Ils sont seulement capables de désencapsuler n'importe quel protocole du protocole y. Pour atteindre la destination D, le seul chemin faisable qui existe doit traverser la séquence des nœuds $(V_1, \ldots, V_k)$ et enlever une occurrence du protocole y sur chaque nœud. Ainsi, en arrivant au nœud V_1, la pile de protocoles de ce chemin faisable doit contenir k occurrences de y à son sommet. Or, il n'existe qu'un seul nœud capable d'encapsuler des y, et il se trouve dans le circuit (le nœud U_k). Donc le chemin doit traverser le circuit k fois avant d'atteindre V_1. La longueur d'un tel chemin faisable est au moins n^2, $|\mathcal{P}| = k^2 + k + 2 = \frac{n^2}{4} - \frac{n^2}{2} + 2$.

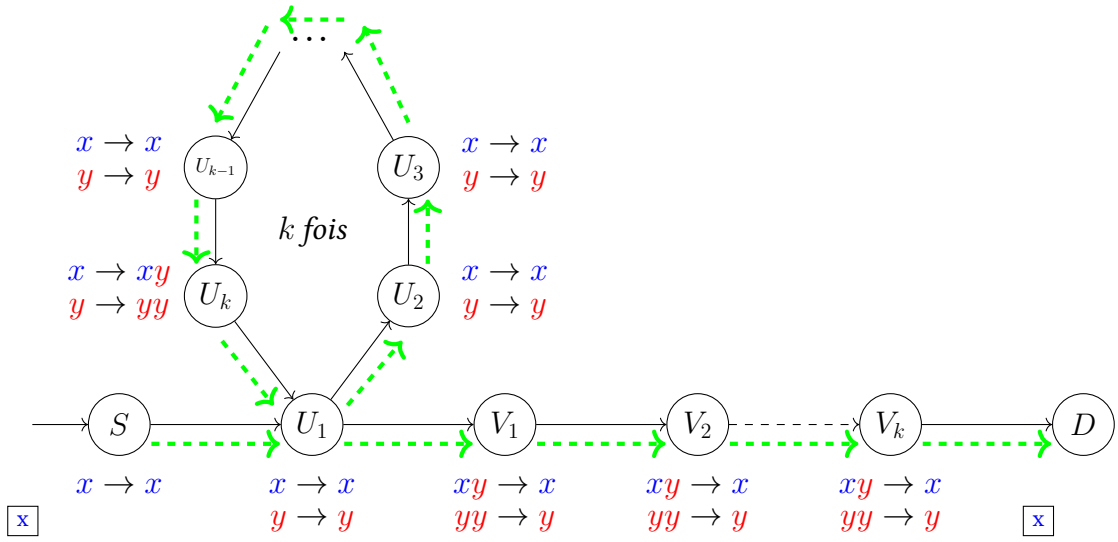


FIGURE 2.3 – Exemple de réseau multicouche où le plus court chemin faisable de S à D comporte au moins n^2 liens [60].

Dans [60], les auteurs ont prouvé des bornes inférieures et supérieures pour la longueur d'un plus court chemin faisable dans un réseau multicouche. Ils ont montré que dans un réseau multicouche avec λ protocoles, la longueur d'un plus court chemin faisable est exponentielle en fonction du nombre de protocoles mais polynomiale en fonction du nombre de nœuds. Cette longueur peut être superpolynomiale avec uniquement

deux protocoles. Ils ont montré également que la longueur d'un plus court chemin faisable dans n'importe quel réseau multicouche ne dépasse jamais $2^{(\lambda+1)\lambda^2 n^2}$. Les preuves sont basées sur la théorie des langages et elles sont détaillées dans [60].

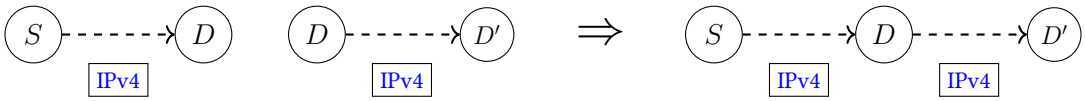
Hauteur maximale de la pile de protocoles. Dans les sections précédentes, nous avons vu qu'un plus court chemin faisable peut comporter des circuits et il peut être même de longueur exponentielle. Pour ces raisons, la pile de protocoles peut croître indéfiniment. Or, un algorithme de calcul de chemins doit avoir une contrainte de terminaison ou convergence. Pour cela, les auteurs de [59] ont prouvé que la hauteur de la pile de protocoles d'un plus court chemin faisable ne dépasse pas λn^2. Leur preuve est liée au lemme de l'étoile (itération ou pompage) de la théorie des langages et elle est détaillée dans [59]. Plus précisement, en modélisant le calcul de chemins faisables par une grammaire alégbrique, un chemin est représenté par un mot de la grammaire. En se basant sur cela, l'idée est que tout mot assez long du langage engendré par la grammaire possède un ou des facteurs qui peuvent être enlevés ou répétés, tout en restant à l'intérieur du langage.

2.2.3 Notre approche

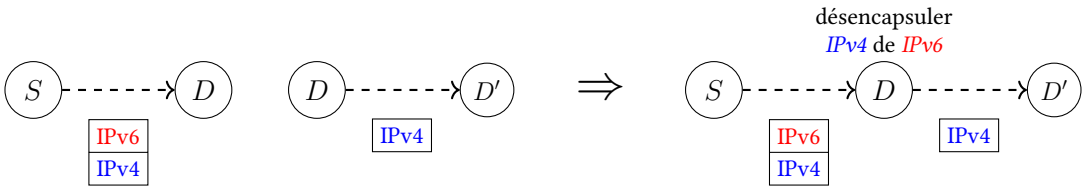
Rappelons que le problème de base qu'on cherche à résoudre est de calculer le plus court chemin faisable entre chaque paire de nœuds. Pour cela, notre idée est de calculer les plus courts chemins faisables en utilisant la fermeture transitive, *i.e.,* concaténer deux sous-chemins pour obtenir un plus long. Autrement dit, si un nœud source S a une route vers une destination D, et que D a une route vers D', alors nous pouvons essayer de construire une route de S vers D' . Mais cela nécessite d'abord que S ait accès à la table de routage de D afin de savoir quelles destinations D peut atteindre. Si S et D ne sont pas voisins, cette opération est très coûteuse à réaliser en envoyant des messages dans un contexte totalement distribué. Mais dans un contexte parallèle, plusieurs processeurs peuvent calculer indépendamment et peuvent communiquer via la transmission de messages et la mémoire partagée. Ainsi, nous pouvons concevoir un algorithme où n'importe quel nœud peut accéder aux tables de routage des autres nœuds. Cela est possible en tirant parti de la technologie SDN basée sur une architecture hautement parallèle de contrôleurs.

Cependant, outre les problèmes de synchronisation habituels inhérents à tout algorithme parallèle, nous devons faire face aux conditions permettant la concaténation de deux sous-chemins. La figure 2.4 [1] illustre certains de ces problèmes. Comme le montre la figure 2.4a, si le réseau n'utilise qu'un seul protocole, la concaténation est possible. Ainsi un chemin de S à D et un autre de D à D' conduisent à un chemin de S à D'. Il s'agit d'une fermeture transitive classique utilisée en théorie des graphes, par exemple par l'algorithme Floyd-Warshall. La figure 2.4b décrit un cas où, dans la route de S à D, ce dernier reçoit un paquet avec la pile de protocoles (IPv4.IPv6), *i.e.,* un paquet IPv4

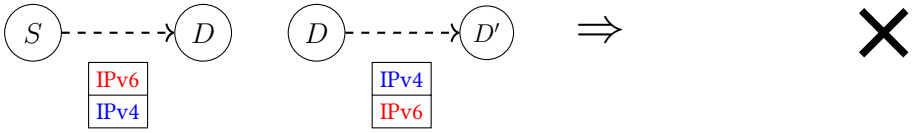
1. Notez que ces exemples ne décrivent pas toujours des chemins faisables. Nous les présentons uniquement pour l'illustration des problèmes de concaténation des sous-chemins avec les tunnels.



(a) Concaténation de sous-chemins dans un réseau avec un protocole unique.



(b) Concaténation possible de sous-chemins dans un réseau avec IPv4 et IPv6.



(c) Concaténation impossible de sous-chemins dans un réseau avec IPv4 et IPv6.

FIGURE 2.4 – Quelques cas de concaténation (im)possible de sous-chemins.

encapsulé dans un IPv6. Et dans la route de D à D', le nœud D doit émettre un paquet IPv4 (sans autres paquets imbriqués) afin d'atteindre D'. Heureusement, D a la fonction d'adaptation qui désencapsule un paquet IPv4 d'un paquet IPv6. Ainsi, lors de la réception du paquet avec la pile de protocoles (IPv4.IPv6), il peut extraire le paquet IPv4 interne et ainsi le transmettre à D'. Encore une fois, la route de S à D' est obtenue par concaténation. Cependant, il convient de préciser dans la table de routage de D que lorsqu'il reçoit un paquet avec la pile de protocoles (IPv4.IPv6) à destination de D', il doit effectuer la désencapsulation appropriée. En revanche, la figure 2.4c illustre un cas où la concaténation est impossible. Dans la route de S à D, le nœud D reçoit un paquet avec une pile de protocoles (IPv6.IPv4), tandis que dans la route de D à D', le nœud D doit émettre un paquet avec la pile de protocoles (IPv4.IPv6) afin d'atteindre D'. Le nœud D n'est pas capable de convertir la pile de protocoles (IPv6.IPv4) en (IPv4.IPv6). Ainsi, la concaténation de ces deux sous-chemins est impossible.

Afin de construire un chemin faisable par fermeture transitive, il existe deux façons différentes de décomposition d'un chemin faisable en sous-chemins en fonction de la hauteur des piles de protocoles induites tout au long du chemin. La première méthode consiste à décomposer le chemin faisable en $(k + 1)$ sous-chemins faisables où k est le nombre de vallées dans le chemin faisable. Autrement dit, un tel chemin est représenté par la concaténation de k chemins faisables sans-vallées. Cette technique s'applique uniquement aux chemins faisables comportant au moins une vallée. La figure 2.5 représente une telle décomposition. Contrairement à cette décomposition restreinte aux chemins faisables avec vallées, la seconde méthode permet une décomposition complète du chemin faisable avec k liens en k sous-chemins valides. Autrement dit, chaque lien représente un chemin valide entre ces extrémités. La figure 2.6 illustre une décomposition

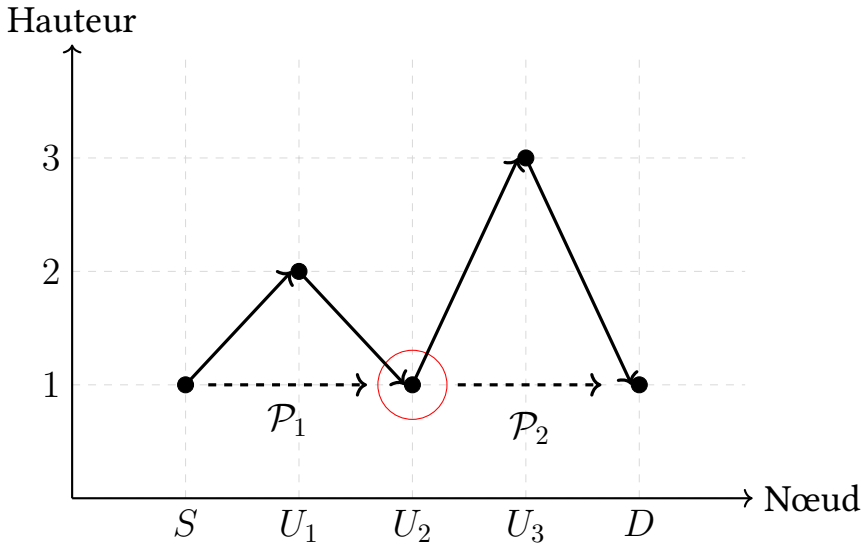


FIGURE 2.5 – Décomposition de chemin faisable en sous-chemins faisables sans-vallées.

complète du même chemin faisable représenté dans la figure 2.5. Dans la prochaine section, nous étudions les conditions permettant d'effectuer une fermeture transitive entre les sous-chemins avec tunnels dans les deux cas de décomposition mentionés ci-dessus. De plus, nous verrons que l'association de la fermeture transitive à certaines idées trouvées dans l'algorithme Stack-Vector peut être très bénéfique. Enfin, nous proposons une généralisation de l'algorithme Floyd-Warshall.

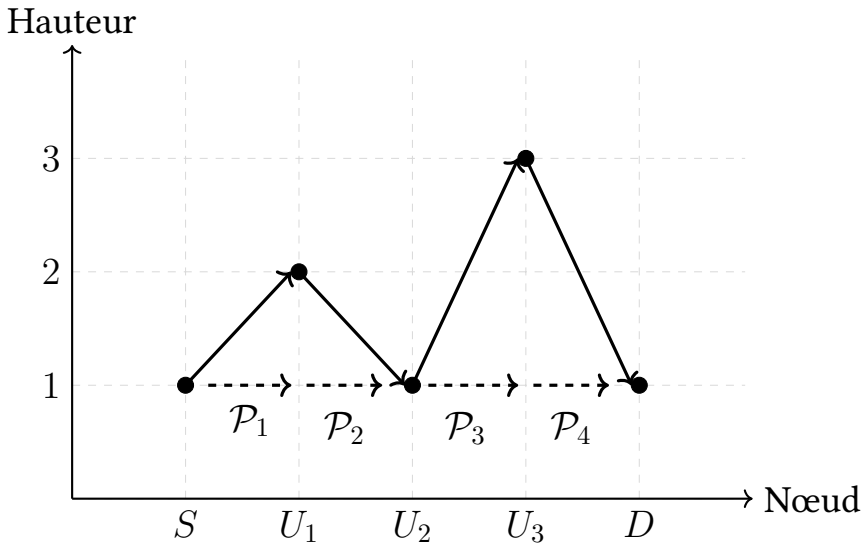


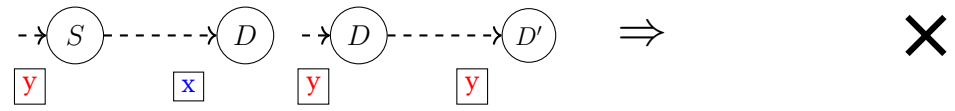
FIGURE 2.6 – Décomposition de chemin faisable en sous-chemins valides.

2.3 Algorithmes de calcul de chemins avec fermeture transitive

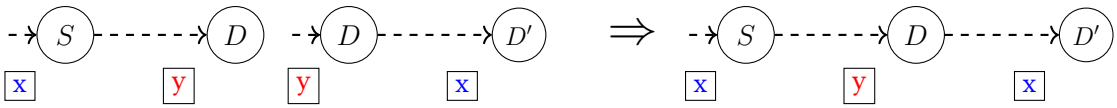
Dans cette section, nous étudions d'abord la fermeture transitive des chemins faisables. Ensuite, nous améliorons l'algorithme Stack-Vector en utilisant cette opération. Enfin, nous généralisons la fermeture transitive pour les chemins valides et nous proposons une généralisation de l'algorithme Floyd-Warshall.

2.3.1 Fermeture transitive de chemins faisables

Rappelons qu'un chemin faisable est un chemin qui commence et qui se termine par une pile de protocoles de hauteur 1, *i.e.,* un paquet sans aucune encapsulation. Dans cette section, nous étudions la concaténation de deux chemins faisables afin d'obtenir un chemin faisable plus long. La figure 2.7 décrit les deux cas possibles auxquels nous pouvons être confrontés dans le processus de concaténation de deux chemins faisables. Dans le premier cas (figure 2.7a), le protocole entrant dans D dans le chemin de S à D est x, tandis que le protocole entrant dans D pour atteindre D' vaut y. Ces chemins ne peuvent pas être concaténés. Par contre (figure 2.7b), le protocole entrant dans D est le même dans le chemin de S à D et dans le chemin de D à D'. Ici la concaténation est possible.



(a) Concaténation impossible de chemins faisables.



(b) Concaténation possible de chemins faisables.

FIGURE 2.7 – Fermeture transitive de chemins faisables.

Plus formellement, soit le chemin faisable $\mathcal{P} = H_{in}Sf_0U_1f_1 \ldots U_if_iH_{out}D$ et le chemin faisable, $\mathcal{P}' = H'_{in}Df'_0U'_1f'_1 \ldots U'_jf'_jH'_{out}D'$. On dit que la concaténation de $\mathcal{P}$ et $\mathcal{P}'$ est possible si est seulement si, $H_{out} = H'_{in}$. Et on définit le chemin résultant par le chemin faisable $\mathcal{P}'' = H_{in}Sf_0U_1f_1 \ldots U_if_iDf'_0U'_1f'_1 \ldots U'_jf'_jH'_{out}D'$. Dans la section suivante, nous utilisons cette nouvelle opération afin d'améliorer le calcul de l'algorithme Stack-Vector.

2.3.2 Amélioration de l'algorithme Stack-Vector

Comme mentionné dans la section 2.2.1, l'algorithme Stack-Vector est le premier et le seul algorithme distribué permettant de calculer des tables de routage dans un réseau multicouche. Malheureusement, cet algorithme souffre de plusieurs problèmes, en particulier son temps de convergence. Afin d'améliorer ce dernier, nous résumons d'abord les idées principales de l'algorithme Stack-Vector.

Principe. Il commence par une étape d'initialisation où chaque nœud U envoie à ses voisins le message $(U, x, 0)$. Cela signifie que le nœud U peut atteindre lui-même (destination = U) s'il reçoit un paquet de n'importe quel protocole $x \in In(U)$. C'est une étape d'initialisation classique comme dans l'algorithme de Bellman-Ford, sauf que le protocole pouvant être reçu est spécifié.

La forme générale des messages est (D, H, ω), ce qui signifie que l'émetteur V peut atteindre la destination D au coût ω s'il reçoit un paquet avec la pile de protocoles H. Lors de la réception d'un tel message, un récepteur U essaie de savoir s'il est capable d'envoyer un paquet avec la pile de protocoles H. Il n'est capable de le faire que s'il a une fonction d'adaptation $f \in \mathcal{F}(U)$ et s'il reçoit un paquet avec une pile de protocoles H' telle que $f(H') = H$. En d'autres termes, il peut lui-même atteindre D au coût $\omega + \omega(U, f, V)$ s'il reçoit un paquet avec une pile $\overline{f}(H)$.

Ainsi, chaque ligne de la table de routage $\mathcal{T}_U$ du nœud U est un 5-uplet (D, H, ω, V, f). Cela signifie que, dans le processus de routage, si U reçoit un paquet avec une pile de protocoles H et une destination D, il applique f à H puis il envoie le paquet au voisin suivant V. Le coût de cet itinéraire est de ω. La clé d'accès à la table de routage est le couple (D, H). Ainsi, lorsque U reçoit un message (D, H, ω), il sélectionne d'abord les fonctions d'adaptation $f \in \mathcal{F}(U)$ telles que $\overline{f}(H) \neq \phi$. S'il n'y a pas d'entrée dans sa table de routage avec la clé $(D, \overline{f}(H))$, il ajoute la ligne $(D, \overline{f}(H), \omega + \omega(U, f, V), V, f)$. Sinon, il compare au coût ω' trouvé dans l'entrée existante et le remplace par la nouvelle ligne uniquement si $\omega' > \omega + \omega(U, f, V)$, *i.e.,* le coût de la nouvelle route est meilleur que l'ancienne route. La terminaison de cet algorithme est garantie par le fait que dans un plus court chemin entre deux nœuds (s'il existe), la pile de protocoles le long du chemin ne dépasse jamais λn^2 (voir section 2.2.2.2). Ainsi, les nœuds n'envoient pas de messages (D, H, ω) où $Hei(H) > \lambda n^2$, et l'algorithme s'arrête à une certaine ronde.

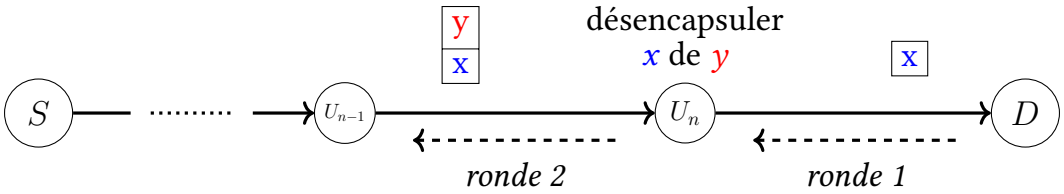


FIGURE 2.8 – Propagation de chemin dans l'algorithme Stack-Vector.

La figure 2.8 illustre l'algorithme Stack-Vector. À la première ronde (l'étape d'initialisation) le nœud D envoie un message à ses voisins, par exemple U_n, l'informant qu'il

peut se joindre au coût 0 s'il reçoit un paquet de protocole x, *i.e.,* le message $(D, x, 0)$. À la deuxième ronde (l'étape de construction de table de routage), U_n vérifie ses fonctions d'adaptation, $f = (xy \rightarrow x)$ dans cet exemple. Ainsi, il peut atteindre D s'il reçoit la pile $\overline{f}(x) = xy$. Puis, il envoie à ses voisins le message $(D, H', \omega(U_n, f, D))$ où $H' = xy$ et ainsi de suite. Cet algorithme est entièrement distribué, et donc *de facto* parallélisable. Dans la section 2.4, nous l'implémentons en deux versions. Dans les deux versions, l'algorithme local correspondant à chaque nœud est un thread. Dans la première version, la communication s'effectue par le biais de messages passant entre les threads. Dans la seconde, chaque nœud accède à la table de routage de ses voisins via une mémoire partagée (au lieu de recevoir des messages).

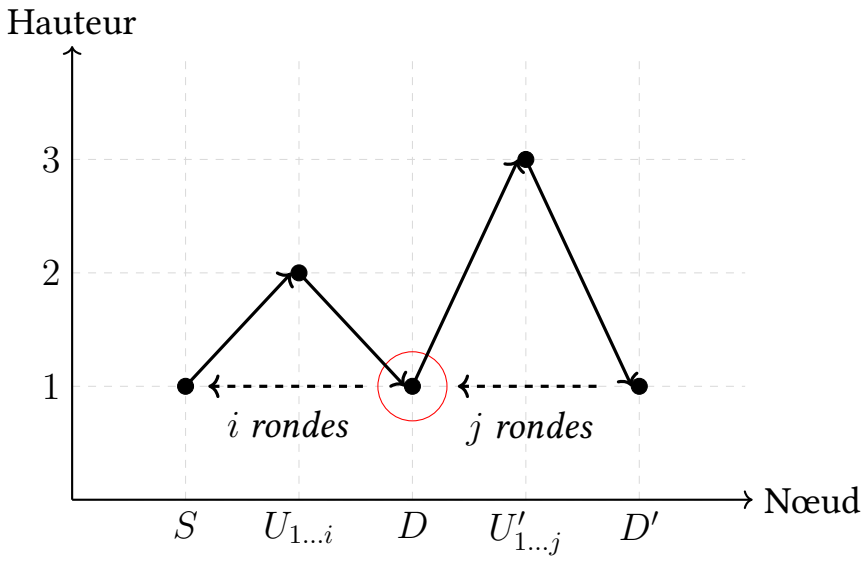


FIGURE 2.9 – Construction de chemin faisable par propagation de l'algorithme Stack-Vector et fermeture transitive de sous-chemins.

Notre idée principale pour améliorer le temps de convergence de l'algorithme Stack-Vector est de limiter la propagation des routes. Cela se fait en ajoutant un saut à chaque ronde à travers les messages envoyés par chaque nœud à ses voisins, nous visons à concaténer directement les sous-chemins. Pour cela, notre idée est d'utiliser l'algorithme Stack-Vector pour calculer uniquement des chemins faisables sans-vallées, puis d'utiliser la fermeture transitive de ces chemins pour calculer ceux qui restent. Autrement dit, un chemin faisable avec k vallées est calculé par la fermeture transitive de $(k+1)$ chemins faisables sans-vallées qui le composent. Dans la figure 2.9, le chemin de S à D est un chemin faisable avec une seule vallée au niveau du nœud D. Ce chemin est calculé par la fermeture transitive des deux sous-chemins faisables (sans-vallées) de S à D et de D à D'. Les deux sous-chemins sont calculés en utilisant la propagation de l'algorithme Stack-Vector. Ils sont construit après $max(i, j)$ itérations de l'algorithme Stack-Vector. Ainsi, le chemin de S à D' est obtenu directement après une seule itération de fermeture transitive, ce qui fait au total $max(i, j) + 1$ itérations. Contrairement à la construction sans fermeture transitive qui fait $(i + j)$ itérations où dans chaque itération les routes

sont propagées. Dans la suite, nous adaptons l'algorithme Stack-Vector afin de calculer uniquement les chemins faisables sans-vallées.

Dans la prochaine section, nous proposons un algorithme pour calculer la table de routage de chaque nœud dans un réseau multicouche *i.e.,* le plus court chemin faisable entre chaque paire de nœuds. Notre algorithme est constitué en une étape d'initialisation, une étape de l'algorithme Stack-Vector pour construire les chemins faisables sans-valléees et une étape de fermeture transitive pour concaténer les chemins faisables. Cet algorithme peut être déployé en centralisé, ou bien dans un contexte distribué avec mémoire partagée.

2.3.2.1 Algorithme

Initialisation. L'algorithme 1 montre le processus d'initialisation de chaque nœud. Il est à peu près le même que celui de l'algorithme Stack-Vector, sauf que le nœud a un accès direct aux protocoles $x \in In(V)$ pour tous ses voisins V grâce à la mémoire partagée. Il calcule également $\overline{f}(x)$ afin de savoir que s'il reçoit un paquet avec la pile de protocoles de départ $H_{in} = \overline{f}(x)$ et la destination V, alors il doit appliquer f pour atteindre V. La pile d'arrivée H_{out} est le protocole x reçu par V dans cette route. La pile destination DST est initialisée par l'unique destination V.

Algorithme 1 : Initialisation de la table de routage d'un nœud U.

(1) **Pour tout** $V \in Nei(U)$ **faire**
(2) **Pour tout** $x \in In(V)$ **faire**
(3) **Pour tout** $f \in \mathcal{F}(U)$ **faire**
(4) $(H_{in}, H_{out}) \leftarrow (\overline{f}(x), x)$
(5) **Si** $H_{in} \neq \phi$ **alors**
(6) $\omega \leftarrow \omega(U, f, V)$
(7) $DST \leftarrow Push(DST, V)$
(8) Ajouter la route $(V, H_{in}, H_{out}, V, f, \omega, DST)$ à la table $\mathcal{T}_U$

Lorsqu'un nœud tente d'ajouter une nouvelle ligne $(D, H_{in}, H_{out}, V, f, \omega, DST)$ à sa table de routage $\mathcal{T}$, il vérifie si le tuple (D, H_{in}, H_{out}) est déjà dans sa table de routage. Si ce n'est pas le cas, il insère la nouvelle ligne dans la table. Sinon, il compare le coût de la nouvelle route avec celui de l'ancienne. La table est mise à jour en remplaçant l'ancienne route par la nouvelle si le nouveau coût est inférieur. Cette étape se fait selon l'algorithme 2. Cet algorithme est le même que celui de l'algorithme Stack-Vector en utilisant la nouvelle définition de la table de routage.

Construction de chemins faisables sans-vallées. L'algorithme 3 construit les chemins faisables sans-vallées en utilisant la propagation de l'algorithme Stack-Vector mais avec des changements importants :

— Le nœud exécutant l'algorithme local accède directement à la table de routage de ses voisins.

Algorithme 2 : Ajout d'une ligne à une table de routage d'un nœud U.

(1) Soit la route $(D, H_{in}, H_{out}, V, f, \omega, DST)$
(2) **Si** $(D, H_{in}, H_{out} \notin \mathcal{T}_U$ **alors**
(3) Ajouter la route $(D, H_{in}, H_{out}, V, f, \omega, DST)$ à la table $\mathcal{T}_U$
(4) **Sinon**
(5) **Si** $\mathcal{T}(D, H_{in}, H_{out}).cout > \omega$ **alors**
(6) $\mathcal{T}(D, H_{in}, H_{out}).cout \leftarrow \omega$
(7) $\mathcal{T}(D, H_{in}, H_{out}).prochain\_vosin \leftarrow V$
(8) $\mathcal{T}(D, H_{in}, H_{out}).fonction\_adaptation \leftarrow f$
(9) $\mathcal{T}(D, H_{in}, H_{out}).pile\_destinations \leftarrow DST$

— Si $Hei(H_{in}) = 1$, cela signifie que U est le point de départ d'un chemin faisable sans-vallée. Ainsi, il ne considère pas les routes où $Hei(H_{in}) = 1$ (ligne 4) et arrête de les propager. Nous pouvons voir cette différence sur la figure 2.9. Le nœud D arrête de propager la route vers S lors de l'utilisation de l'algorithme 3, alors qu'il la propage lors de l'utilisation de l'algorithme Stack-Vector d'origine.

— Lorsqu'une ligne est ajoutée à la table de routage, l'itération i est stockée dans la ligne elle-même (ligne 9), afin d'optimiser les calculs et de vérifier uniquement les routes ajoutées ou mises à jour à l'itération précédente.

Algorithme 3 : Construction de chemins faisables sans-vallées sur le nœud U.

(1) $i \leftarrow 1$
(2) **Répéter**
(3) **Pour tout** $V \in Nei(U)$ **faire**
(4) **Pour tout** $(D, H_{in}, H_{out}, V, f, \omega, DST) \in \mathcal{T}_V$ *et* $Hei(H_{in}) \neq 1$ **faire**
(5) **Pour tout** $f' \in \mathcal{F}(U)$ **faire**
(6) $(H'_{in}, H'_{out}) \leftarrow (\overline{f}'(H_{in}), H_{out})$
(7) **Si** $H'_{in} \neq \phi$ *et* $Hei(H'_{in}) \leq \lambda n^2$ **alors**
(8) $\omega' \leftarrow \omega + \omega(U, f, V)$
(9) Ajouter la route $(D, H'_{in}, H'_{out}, V, f', \omega', DST)$ à la table $\mathcal{T}_U$
(10) *# si la route n'existe pas ou si la précédente a un coût supérieur*
(11) $i \leftarrow i + 1$
(12) **Jusqu'à ce qu'** *aucune table de routage n'a été modifiée*;

Fermeture transitive de tous les chemins faisables. L'algorithme 4 décrit l'opération de fermeture transitive des chemins faisables. Il permet de concaténer deux routes de S à D et de D à D' au niveau d'un nœud commun D si la pile d'arrivée H_{out} à D dans la première route est la même que la pile de départ H'_{in} dans la seconde route (ligne 5). La pile de départ de la nouvelle route est la pile H_{in} du nœud S et la pile d'arrivée est la pile H'_{out} du nœud D' (ligne 7). La somme des coûts ω et ω' définit le coût de la nouvelle route (ligne 6). La pile des destinations DST contient toutes les destinations dans la première route, suivi par les destinations dans la deuxième route (ligne 8).

Algorithme 4 : Fermeture transitive de chemins faisables sur le nœud S.

(1) $i \leftarrow 1$
(2) **Répéter**
(3) **Pour tout** $(D, H_{in}, H_{out}, V, f, \omega, DST) \in \mathcal{T}_S$ *et* $Hei(H_{in}) = 1$ **faire**
(4) **Pour tout** $(D', H'_{in}, H'_{out}, V', f', \omega', DST') \in \mathcal{T}_D$ *et* $Hei(H'_{in}) = 1$ **faire**
(5) **Si** $H_{out} = H'_{in}$ **alors**
(6) $\omega'' \leftarrow \omega + \omega'$
(7) $(H''_{in}, H''_{out}) \leftarrow (H_{in}, H'_{out})$
(8) $DST'' \leftarrow Push(DST', DST)$
(9) Ajouter la route $(D', H''_{in}, H''_{out}, V, f, \omega'', DST'')$ à la table $\mathcal{T}_S$
(10) *# si la route n'existe pas ou si la précédente a un coût supérieur*
(11) $i \leftarrow i + 1$
(12) **Jusqu'à ce qu'** *aucune table de routage n'a été modifiée*;

2.3.2.2 Complexité et correction

Complexité. L'algorithme 1 vérifie chaque lien (U, V) avec chaque protocole d'entrée et fonction d'adaptation (lignes 1-3). Dans le pire des cas, un nœud peut avoir λ protocoles d'entrée et $3\lambda^2$ fonctions d'adaptation. Concernant l'algorithme 3, nous avons $\mathcal{O}(m\,|\mathcal{T}|\,\lambda^2)$ opérations (lignes 3-9), comme une table de routage $\mathcal{T}$ peut contenir $n\,(\frac{1-\lambda^{h_{max}+1}}{1-\lambda} - 1)$ routes dans le pire des cas (ligne 4) incluant $n\lambda$ routes avec une hauteur de pile égale à 1. La boucle "répéter" se termine après $\mathcal{O}(\lambda n^2 \text{diam}\,\mathcal{N})$ itérations. L'algorithme 4 s'arrête après $\mathcal{O}(log_2\,(\text{diam}\,\mathcal{N}))$ itérations, de sorte qu'à chaque itération, on a $\mathcal{O}(n^3\lambda^2)$ opérations (lignes 3-10). Le tableau 2.1 suivant résume les bornes de complexité des algorithmes.

Algorithme	Complexité
Algorithme 1	$\mathcal{O}(m\,\lambda^3)$
Algorithme 3	$\mathcal{O}(\lambda n^2\ \text{diam}\,\mathcal{N}\,m\,\lvert\mathcal{T}\rvert\,\lambda^2)$
Algorithme 4	$\mathcal{O}(log_2\,(\text{diam}\,\mathcal{N})\ n^3\lambda^2)$

TABLE 2.1 – Complexité temporelle de l'algorithme Stack-Vector avec fermeture transitive dans le pire des cas.

Correction. Nous allons prouver que l'algorithme 4 converge correctement. La preuve de l'algorithme 3 peut être trouvée dans [59].

On définit par $\mathcal{N}' = (\mathcal{G}', \mathcal{A}', \mathcal{F}', \omega')$ le réseau obtenu à partir des tables de routage du réseau $\mathcal{N} = (\mathcal{G}, \mathcal{A}, \mathcal{F}, \omega)$ après l'application de l'algorithme 3 tels que :

$$\mathcal{G}' = (\mathcal{V}, \mathcal{E}') \qquad \text{et} \qquad (\mathcal{A}, \mathcal{F}, \omega) = (\mathcal{A}', \mathcal{F}', \omega')$$

Où $\mathcal{E}'$ est l'ensemble des chemins faisables sans-vallées calculés par l'algorithme 3 . Il faut noter que le diamètre $\text{diam}\,\mathcal{N}' \leq \text{diam}\,\mathcal{N}$. En se basant sur cette définition, nous

prouvons la proposition suivante.

Proposition 1. *Pour tout réseau multicouche $\mathcal{N}'$, l'algorithme 4 calcule tous les chemins faisables optimaux (décomposables en k sous-chemins faisables où $k \leq$ diam $\mathcal{N}'$) après $\mathcal{O}(\log_2(\text{diam}\,\mathcal{N}'))$ itérations.*

Démonstration. Nous démontrons la proposition ci-dessus en deux étapes :

1. Après i itérations, tous les chemins faisables optimaux de longueur $\leq 2^i$ sont calculés, $i \geq 0$,
2. Soit k le numéro de la dernière itération, on a $k \leq log_2(\text{diam}\,\mathcal{N}')$.

Nous prouvons par récurrence qu'après i itérations, tous les chemins faisables optimaux de longueur $\leq 2^i$ sont calculés, $i \geq 0$.

- Si $i = 0$, tous les chemins faisables optimaux de longueur ≤ 1 sont calculés (à la fin de l'algorithme 3).
- Soit $i > 0$. Nous supposons que tous les chemins faisables optimaux de longueur $\leq 2^i$ sont calculés à l'itération i et nous prouvons que tous les chemins faisables optimaux de longueur $\leq 2^{i+1}$ sont calculés à l'itération $i+1$.
  Soit $\mathcal{P}$ défini comme $\mathcal{P} = \mathcal{P}_1 \cdot \mathcal{P}_2$, un chemin calculé par fermeture transitive à l'itération $i+1$. Ainsi $\mathcal{P}_1$ et $\mathcal{P}_2$ ont été calculés à l'itération i, de sorte que par hypothèse de récurrence, nous avons $|\mathcal{P}_1| \leq 2^i$ et $|\mathcal{P}_2| \leq 2^i$, et, comme par définition $|\mathcal{P}| = |\mathcal{P}_1| + |\mathcal{P}_2|$ donc $|\mathcal{P}| \leq 2^{i+1}$.

Il reste à prouver la convergence après k itérations, avec $k \leq log_2(\text{diam}\,\mathcal{N}')$.
Soit k_{max} le numéro de la dernière itération, où $k_{max} = log_2(\text{diam}\,\mathcal{N}')$, en particulier diam $\mathcal{N}' \leq 2^{k_{max}}$. Soit $\mathcal{P}$ un chemin faisable optimal, par définition de diam $\mathcal{N}'$, on a $|\mathcal{P}| \leq$ diam $\mathcal{N}'$ et donc $|\mathcal{P}| \leq 2^{k_{max}}$. Par conséquent, à l'étape 1 (correction), $\mathcal{P}$ est calculé avant l'itération k_{max}. Donc la convergence est atteinte après k itérations, $k \leq k_{max} = log_2(\text{diam}\,\mathcal{N}')$. □

2.3.2.3 Routage des paquets

Rappelons que notre algorithme permet de limiter la propagation des routes dans l'algorithme Stack-Vector en utilisant la fermeture transitive des chemins sans-vallées. Par exemple, dans la figure 2.10 et pendant la construction du chemin de S à D' le nœud D arrête la propagation de la route vers D'. À ce moment, tous les nœuds entre S et D n'auront pas la route pour atteindre la destination finale D' et ils n'ont que la route pour atteindre la destination intermédiaire D. Ainsi, le routage saut par saut n'est pas suffisant pour le routage dans des chemins construits par l'algorithme de fermeture transitive et il ne peut être utilisé que dans des chemins calculés par l'algorithme Stack-Vector. Dans

cette situation, le routage par segments peut être utilisé parallèlement avec le routage saut par saut, où les chemins (sous-chemins dans un chemin de fermeture transitive) calculés par l'algorithme Stack-Vector représentent les segments. Pour cela, il est nécessaire d'implémenter une pile "DST" pour les chemins de fermeture transitive qui contient les identifiants de destination intermédiaires pour connaître les sous-chemins utilisés afin d'atteindre la destination finale [2]. Dans la figure 2.10, si S émet un paquet à D' avec la pile "DST" $= \{D', D\}$ alors les nœuds $U_1, \ldots, U_i$ entre S et D routent le paquet vers la destination D, et à l'arrivée à D la pile "DST" ne contient que D', donc les nœuds $U'_1, \ldots, U'_j$ du second sous-chemin acheminent le paquet vers la destination finale D'.

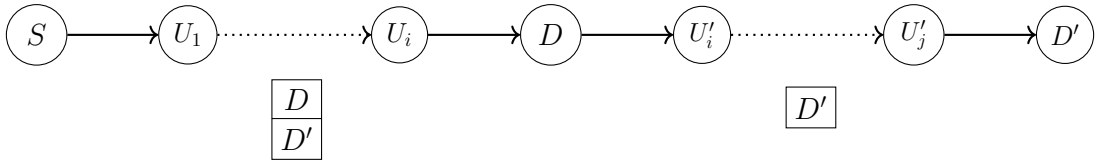


FIGURE 2.10 – routage par segments dans les chemins de fermeture transitive en utilisant la pile DST.

Une fois les tables de routage construites, si un nœud U reçoit un paquet de la forme $(DST, H_{in}, H_{out}, data)$, il vérifie s'il existe une route vers la première destination D dans "DST". Si la réponse est non alors D est inaccessible avec le couple de piles (H_{in}, H_{out}). Sinon, si la route correspondante est $(D, H_{in}, H_{out}, V, f, \omega, DST)$, alors la pile DST est mise à jour si elle est la fin d'un sous-chemin afin d'émettre le paquet $(DST, f(H_{in}), H_{out}, data)$ au successeur V. Cette étape est donnée par l'algorithme 5.

Algorithme 5 : Routage par sauts et par segments d'un paquet sur le nœud U.

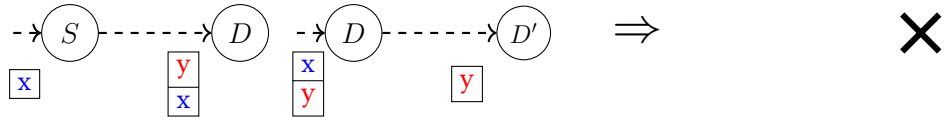
(1) Recevoir un paquet $(DST, H_{in}, H_{out}, data)$
(2) $D \leftarrow Top(DST)$
(3) **Si** $(D, H_{in}, H_{out}) \notin \mathcal{T}_U$ **alors**
(4) Pas de route, jetez le paquet
(5) **Sinon**
(6) Soit la route $(D, H_{in}, H_{out}, V, f, \omega, DST) \in \mathcal{T}_U$
(7) **Si** $(D = U)$ **alors**
(8) $DST \leftarrow Pop(DST)$
(9) Transmettre le paquet $(DST, f(H_{in}), H_{out}, data)$ à V

2.3.3 Fermeture transitive de chemins valides

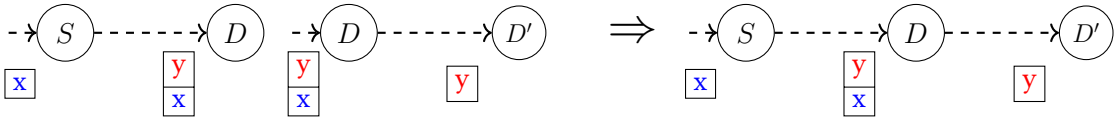
Dans la section 2.3.1, nous avons défini une opération de fermeture transitive qui ne s'applique qu'aux chemins faisables. Cette opération peut être généralisée sur tous les

2. La pile de destination "DST" peut avoir au plus $\mathcal{O}(\operatorname{diam} \mathcal{N})$ identifiants (chemin contenant uniquement des conversions).

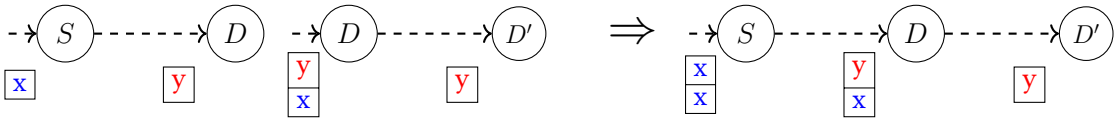
chemins valides afin de concevoir un algorithme de fermeture transitive complet. Cet algorithme sera donc une généralisation de l'algorithme de Floyd-Warshall en ajoutant les piles de protocoles. Pour cela, on généralise d'abord l'opération de fermeture transitive pour la concaténation des chemins valides. La figure 2.11 suivante décrit les différents situations possibles pour concaténer deux chemins valides.



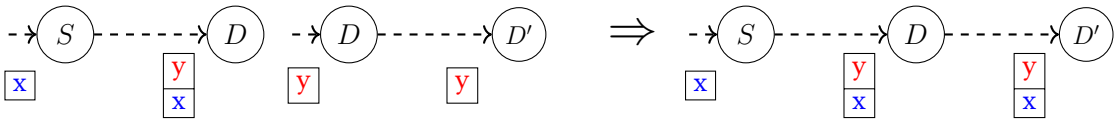
(a) Concaténation impossible de chemins valides.



(b) Concaténation possible de chemins valides ($H_{out} = H'_{in}$).



(c) Concaténation possible de chemins valides (H_{out} est un suffixe de H'_{in}).



(d) Concaténation possible de chemins valides (H'_{in} est un suffixe de H_{out}).

FIGURE 2.11 – Fermeture transitive de chemins valides.

Soit le chemin valide de S à D, $\mathcal{P} = H_{in}Sf_0U_1f_1 \dots U_if_iH_{out}D$ et le chemin valide de D à D', $\mathcal{P}' = H'_{in}Df'_0U'_1f'_1 \dots U'_jf'_jH'_{out}D'$. Le cas le plus simple de concaténation de ces deux chemins est lorsque $H_{out} = H'_{in}$. Ici, la concaténation est directe (voir figure 2.11b). Mais ce n'est pas le seul cas où on peut appliquer une concaténation. Par exemple, la figure 2.11c, illustre un deuxième cas où $H_{out} = y$ et $H'_{in} = xy$. Cela dit, en considérant les deux piles comme des mots, on peut constater que H_{out} est un suffixe de H'_{in}. Posons $H'_{in} = H.H_{out}$ où "." dénote la concaténation (de piles) et H est la pile qui correspond à H'_{in} privée de son suffixe H_{out}. Le troisième cas où la concaténation est possible est symétrique au précédent et il est représenté par la figure 2.11d, *i.e.*, H'_{in} est suffixe de H_{out}. Dans ce cas, on peut construire le chemin valide $H_{in}S \dots H.H'_{out}D'$ où $H = H_{out} - H'_{in}$. En dehors de ces trois cas où il n'existe pas de relation entre les deux piles H_{out} et H'_{in}, il est impossible de concaténer les deux chemins valides $\mathcal{P}$ et $\mathcal{P}'$. Un exemple de ce dernier est représenté dans la figure 2.11a. Pour résumer, à partir de deux chemins valides $\mathcal{P}$ et $\mathcal{P}'$, on peut donc obtenir un chemin valide de S à D' en passant via D, $\mathcal{P}'' = H_{in}Sf_0U_1f_1 \dots U_if_iDf'_0U'_1f'_1 \dots U'_jf'_jH'_{out}D'$ de la manière suivante :

$$(H''_{in}, H''_{out}) = \begin{cases} (H_{in}, H'_{out}) & \text{si} \quad H_{out} = H'_{in} \\ (H.H_{in}, H'_{out}) \text{ où } H = H'_{in} - H_{out} & \text{si} \quad H_{out} \text{ est un suffixe de } H'_{in} \\ (H_{in}, H.H'_{out}) \text{ où } H = H_{out} - H'_{in} & \text{si} \quad H'_{in} \text{ est un suffixe de } H_{out} \end{cases} \tag{2.1}$$

Dans la suite, nous utilisons cette opération afin de concevoir un autre algorithme de fermeture transitive qui permet de calculer les plus courts chemins valides (en particulier les chemins faisables) entre chaque paire de nœuds dans un réseau multicouche.

2.3.4 Généralisation de l'algorithme Floyd-Warshall

Dans la section 2.3.1, nous avons défini une fermeture transitive qui s'applique uniquement sur des chemins faisables. Cette opération a été utilisée à côté de l'algorithme Stack-Vector afin de calculer tous les plus courts chemins faisables entre chaque paire de nœuds dans un réseau multicouche. Ici, ce n'est plus le cas, autrement dit, nous avons une opération de fermeture transitive plus générale et elle permet de calculer la concaténation de tous types de chemins valides (en particulier les chemins faisables) entre chaque paire de nœuds dans un réseau multicouche.

Dans la suite, nous proposons un algorithme de fermeture transitive complet *à la* Floyd-Warshall, où la concaténation est réalisée en fonction de l'existence d'un nœud en commun et de la compatibilité des piles de protocoles incidents à ce nœud. Cet algorithme peut être déployé en centralisé, ou bien dans un contexte distribué avec mémoire partagée.

2.3.4.1 Algorithme

Initialisation. L'algorithme 6 montre le processus d'initialisation de chaque nœud. Pour chaque voisin V dans $Nei(U)$, le nœud U ajoute une ou plusieurs routes avec la ou les fonctions d'adaptation correspondantes. Les piles de protocoles (H_{in}, H_{out}) de chaque route sont initialisées selon le type de la fonction d'adaptation f utilisée. Le coût ω de la route est le poids du lien et la fonction correspondante $\omega(U, f, V)$.

Fermeture transitive de tous les chemins valides. Durant l'étape de construction de table de routage, on vérifie la compatibilité de toutes les combinaisons de chemins valides, *i.e.,* toutes les lignes des tables de routage $(D, H_{in}, H_{out}, V, f, \omega)$ de $\mathcal{T}_S$ et $(D', H'_{in}, H'_{out}, V', f', \omega')$ de $\mathcal{T}_D$, pour un couple de nœuds S et D' quelconque. Si les deux routes sont compatibles (selon les conditions de (2.1)) alors on ajoute la ligne $(D', H''_{in}, H''_{out}, V, f, \omega'')$ à $\mathcal{T}_S$ (où H''_{in} et H''_{out} sont également définies selon la formule (2.1)). Le poids de cette nouvelle route ω'' est la somme des deux poids ω et ω'. Lorsqu'un nœud S tente d'ajouter une nouvelle ligne $(D', H''_{in}, H''_{out}, V, f, \omega'')$ à sa table de routage $\mathcal{T}_S$, il vérifie si le triplet $(D', H''_{in}, H''_{out})$ est déjà présent dans sa table. Si ce

Algorithme 6 : Initialisation de la table de routage d'un nœud U.

(1) **Pour tout** $V \in Nei(U)$ **faire**
(2) **Pour tout** $f \in \mathcal{F}(U)$ **faire**
(3) **Si** $f = (x \to y)$ **alors**
(4) $(H_{in}, H_{out}) = (x, y)$
(5) **Si** $f = (x \to xy)$ **alors**
(6) $(H_{in}, H_{out}) = (x, xy)$
(7) **Si** $f = (xy \to x)$ **alors**
(8) $(H_{in}, H_{out}) = (xy, x)$
(9) $\omega \leftarrow \omega(U, f, V)$
(10) Ajouter la route $(V, H_{in}, H_{out}, V, f, \omega)$ à la table $\mathcal{T}_U$

n'est pas le cas, la nouvelle ligne est insérée dans la table. Sinon, le coût $\omega+\omega'$ du nouveau chemin est comparé à l'ancien. La table est mise à jour en remplaçant l'ancien chemin par le nouveau si le nouveau coût de celui-ci est inférieur et si la hauteur de H''_{in} (resp. H''_{out}) $\leq \lambda n^2$, car il est prouvé dans [59] que la pile de protocole d'un plus court chemin faisable (ou valide) ne peut dépasser cette valeur (voir section 2.2.2.2). Selon cette dernière valeur et la ligne 6, si aucune route n'a été ajoutée ou modifiée alors le calcul des tables de routage est fini. Afin d'obtenir tous les chemins faisables, il suffit de prendre les routes où $Hei(H_{in}) = 1$ et $Hei(H_{out}) = 1$. Le reste des routes sont des chemins valides et ils sont uniquement nécessaires pour le routage. L'algorithme 7 décrit cette étape.

Algorithme 7 : Fermeture transitive de chemins valides sur le nœud S.

(1) **Répéter**
(2) **Pour tout** $(D, H_{in}, H_{out}, V, f, \omega) \in \mathcal{T}_S$ **faire**
(3) **Pour tout** $(D', H'_{in}, H'_{out}, V', f', \omega') \in \mathcal{T}_D$ **faire**
(4) Calculer (H''_{in}, H''_{out}) en fonction de (H_{in}, H_{out}) et (H'_{in}, H'_{out})
(5) *# selon la formule* (2.1)
(6) **Si** $Hei(H''_{in}) \leq \lambda n^2$ *et* $Hei(H''_{out}) \leq \lambda n^2$ **alors**
(7) $\omega'' \leftarrow \omega + \omega'$
(8) Ajouter la route $(D', H''_{in}, H''_{out}, V, f, \omega'')$ à la table $\mathcal{T}_S$
(9) *# si elle n'existe pas ou si la précédente a un coût supérieur*
(10) **Jusqu'à ce qu'** *aucune table de routage n'a été modifiée*;

2.3.4.2 Complexité et correction

Complexité. L'étape d'initialisation vérifie chaque lien (U, V) avec chaque fonction d'adaptation f dans l'ensemble $\mathcal{F}(U)$. Le nombre de liens est m et le nombre de fonctions d'adaptation est $3\lambda^2$. Ainsi, la complexité temporelle de l'algorithme 6 d'initialisation est $\mathcal{O}(m\,\lambda^2)$. L'algorithme 7 de fermeture transitive complet a une complexité temporelle de $\mathcal{O}(log_2(\text{diam}\,\mathcal{N})\,n\,|\mathcal{T}|^2)$. Le tableau 2.2 suivant résume les bornes de complexité des

algorithmes.

Algorithme	Complexité
Algorithme 6	$\mathcal{O}(m\,\lambda^3)$
Algorithme 7	$\mathcal{O}(log_2\,(\text{diam}\,\mathcal{N})\;n^3\lambda^2)$

TABLE 2.2 – Complexité temporelle de l'algorithme Floyd-Warshall avec piles de protocoles dans le pire des cas.

Correction. Afin de prouver que l'algorithme 7 converge correctement, nous pouvons appliquer directement la proposition 1. Dans ce cas, étant donné un réseau multicouche $\mathcal{N}$, l'algorithme 7 converge correctement après $\mathcal{O}(log_2\,(\text{diam}\,\mathcal{N}))$.

2.3.4.3 Routage des paquets

L'algorithme défini dans la section précédente permet de construire pas à pas des chemins valides. Lors de chaque étape de construction, il ajoute le voisin suivant dans le chemin à calculer. Dans cette situation, le routage saut par saut est suffisant. Lorsqu'un nœud U reçoit un paquet $(D, H_{in}, H_{out}, data)$, il vérifie s'il existe une route vers la destination D. Si la réponse est non, alors D est inaccessible avec les piles (H_{in}, H_{out}). Sinon, si la route correspondante est $(D, H_{in}, H_{out}, V, f, \omega)$, alors le nœud U émet le paquet $(D, f(H_{in}), H_{out}, data)$ au prochain voisin V. Cette étape peut être accomplie par l'algorithme 8. Cet algorithme de routage est le même que l'algorithme de routage proposé dans la section 2.3.2.3 en supprimant la pile de destinations "DST". Autrement dit, l'algorithme de routage saut par saut proposé dans l'algorithme Stack-Vector [59].

Algorithme 8 : Routage par sauts d'un paquet sur le nœud U.

(1) Recevoir un paquet $(D, H_{in}, H_{out}, data)$
(2) **Si** $(D, H_{in}, H_{out}) \notin \mathcal{T}_U$ **alors**
(3) Pas de route, jetez le paquet
(4) **Sinon**
(5) Soit la route $(D, H_{in}, H_{out}, V, f, \omega) \in \mathcal{T}_U$
(6) Transmettre le paquet $(D, f(H_{in}), H_{out}, data)$ à V

2.4 Evaluation de performances des algorithmes

Dans cette section, nous avons effectué des simulations afin d'évaluer les performances des algorithmes définis dans la section 2.3. Nous avons utilisé à la fois des topologies aléatoires et réalistes (*i.e.,* des cartes échantillonnées). Les trois algorithmes suivants ont été implémentés [3] :

3. Chaque algorithme local sur un nœud est implémenté en tant que thread.

1. L'algorithme Stack-Vector [59] a été implémenté avec un passage de message entre les threads et nommé SV-MP,
2. L'algorithme Stack-Vector a été implémenté avec une mémoire partagée et nommé SV-SM,
3. L'algorithme Stack-Vector avec fermeture transitive a été implémenté avec de la mémoire partagée et nommé TC.

La complexité relativement élevée de l'algorithme complet de fermeture transitive *à la* Floyd-Warshall est due au produit cartésien des tables de routage, ce qui le rend lent par rapport aux autres algorithmes centralisés. Nous avons l'intention de trouver un moyen plus efficace d'optimiser le calcul du produit cartésien des tables et paralléliser l'implémentation de l'algorithme.

2.4.1 Méthodologie

L'implémentation a été développée en ISO C++14 à l'aide de la bibliothèque igraph [16] pour générer des graphes aléatoires. L'implémentation de la version distribuée est donnée comme suit : chaque nœud est simulé par un thread, et un lien orienté (U, V) est implémenté comme une file d'attente où U ne peut qu'écrire et V ne peut que lire. Le code source de nos algorithmes est librement disponible sur un dépôt Gitlab hébergé par l'Université de Bordeaux [79]. Les simulations ont été effectuées sur le cluster Curta du MCIA [4] qui est équipé de 336 nœuds de calcul Lenovo ThinkSystem SD530, chacun ayant deux processeurs Intel Xeon Gold SKL-hyperthreadés à 16 cœurs. Processeurs 6130 à 2,1 GHz avec 96 Go de RAM. Dans l'implémentation, chaque nœud du réseau a son propre thread et lit/écrit des données partagées en utilisant des mutex et des variables de condition. Un nombre suffisant de nœuds de calcul a été réservé dans le cluster afin de garantir que le thread de chaque nœud simulé obtienne un cœur physique. Les réservations des nœuds de calcul ont été faites comme suit :

— Pour les topologies aléatoires à 100 nœuds : 4×32 cœurs
— Pour les topologies aléatoires à 300 nœuds : 10×32 cœurs
— Pour les topologies aléatoires à 500 nœuds : 16×32 cœurs
— Pour les topologies réalistes à 1000 nœuds : 32×32 cœurs

Les paramètres d'entrée de chaque algorithme sont les suivants : le nombre de nœuds n dans le réseau, la probabilité p de disponibilité d'une fonction d'adaptation, le nombre λ de protocoles et la hauteur maximale de pile $h_{\max}$, *i.e.,* le nombre maximal autorisé de protocoles imbriqués en même temps. Les métriques de sortie sont les suivantes : le % de chemins faisables trouvés, l'exploration de chemins, le temps de convergence, la longueur moyenne des chemins, le diamètre faisable du réseau, la taille des tables de routage et la taille de la pile de destinations (*i.e.,* segments). Toutes les valeurs de résultat présentées dans les figures suivantes sont moyennées sur les valeurs de résultat de 100 d'exécutions.

4. https ://www.mcia.fr/

Toutes les topologies de réseau aléatoires utilisées pour l'expérimentation sont générées selon les étapes suivantes :

1. Nous générons un graphe aléatoire par le mécanisme d'attachement préférentiel défini par Barabàsi et Albert dans [3], où chaque nouveau nœud ajouté est attaché à 3 nœuds existants,
2. Nous convertissons ensuite le graphe généré en un graphe orienté symétrique. Chaque lien non orienté est converti en deux liens orientés,
3. Nous distribuons chaque fonction d'adaptation sur un nœud avec une probabilité donnée p. Pour un nombre donné λ de protocoles, il y a $3\lambda^2$ fonctions d'adaptation possibles.

Afin de construire des topologies réalistes, nous avons superposé deux cartes de *Autonomous System* (AS) collectées par *Route Views* [108] le 1er novembre 2019. Une carte contient tous les AS des nœuds IPv4, tandis que l'autre contient tous les AS des nœuds IPv6. Les AS superposés donnent une carte multiprotocole IPv4/IPv6. La topologie de niveau AS assemblée compte 67381 nœuds et 162369 liens, où 73 % des nœuds sont des AS contenant uniquement des nœuds IPv4, 1 % des nœuds sont des AS contenant uniquement des nœuds IPv6 et 26 % des nœuds sont des AS contenant des nœuds double pile IPv4/IPv6. De plus, comme cette carte assemblée au niveau AS est actuellement trop grande pour exécuter directement le code d'implémentation de notre expérimentation, nous avons échantillonné cette carte pour en produire de plus petites de l'ordre de quelques milliers de nœuds. Par conséquent, les réseaux réalistes sont générés à partir de la topologie assemblée décrite ci-dessus, selon les étapes suivantes :

1. Nous générons deux sous-graphes appelés T1 et T2 de 1000 nœuds en utilisant un algorithme qui échantillonne la topologie assemblée (définie comme la topologie source). Cet algorithme est implémenté dans l'outil Network Topology Analysis and Internet Modeling appelé *nem* [68]. disponible en tant que logiciel libre et open source [5]. L'algorithme est conçu pour que les caractéristiques des topologies échantillonnées soient similaires aux cartes d'origine à l'exception des distances, des excentricités et des diamètres qui sont légèrement réduits par rapport à ceux d'origine.
2. Nous distribuons les fonctions d'adaptation de sorte que les nœuds IPv4 (resp. IPv6) ne puissent avoir que des retransmissions classiques IPv4 ou IPv6, v4 $\rightarrow$ v4 (resp. v6 $\rightarrow$ v6). Les nœuds en double pile IPv4/IPv6 peuvent avoir un seul ensemble parmi : l'ensemble de conversions IPv4 et IPv6 (v4 $\rightarrow$ v6, v6 $\rightarrow$ v4), l'ensemble d'encapsulation IPv4 dans IPv6 (v4 $\rightarrow$ v4v6, v4v6 $\rightarrow$ v4), l'ensemble d'encapsulation IPv6 dans IPv4 (v6 $\rightarrow$ v6v4, v6v4 $\rightarrow$ v6).

Les pourcentages moyens de la répartition des deux protocoles réseau dans les deux topologies générées sont les suivants :

— Pour la topologie de réseau T1 : 69,90 % des nœuds sont IPv4 uniquement (resp. 0,6 % IPv6 uniquement) et 29,50 % des nœuds sont IPv4 et IPv6.

5. https ://www.labri.fr/perso/magoni/nem/

— Pour la topologie de réseau T2 : 47,60 % des nœuds sont IPv4 uniquement (resp. 0,7 % IPv6 uniquement) et 51,70 % des nœuds sont IPv4 et IPv6.

Les topologies générées n'ont pas la même distribution de protocole en raison du comportement de l'algorithme d'échantillonnage.

2.4.2 Temps de convergence et exploration de chemins

2.4.2.1 Temps de convergence

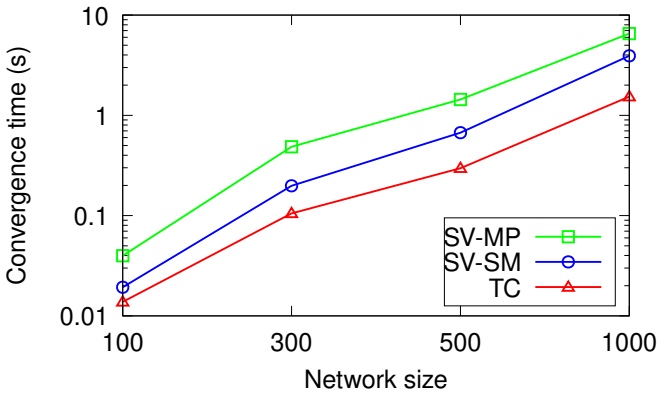


FIGURE 2.12 – Temps de convergence des algorithmes selon une probabilité $p = 0.05$. L'axe y utilise une échelle logarithmique.

Les figures 2.12, 2.13, 2.14 montrent le temps de convergence des trois algorithmes sur les topologies aléatoires en fonction de la taille du réseau et la probabilité p de disponibilité d'une fonction d'adaptation sur chaque nœud.

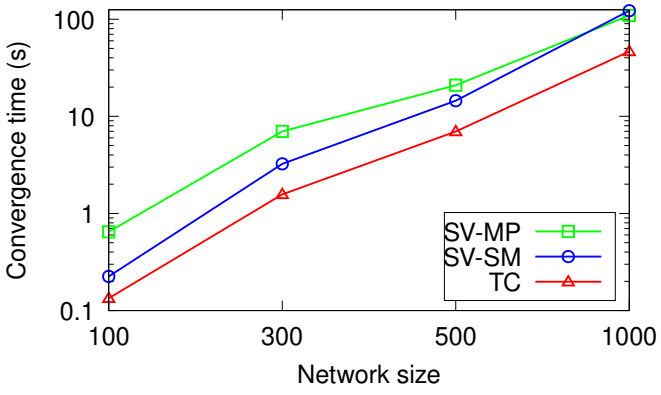


FIGURE 2.13 – Temps de convergence des algorithmes selon une probabilité $p = 0.10$. L'axe y utilise une échelle logarithmique.

Nous pouvons voir que TC est l'algorithme le plus rapide pour tous les paramètres, suivi de l'algorithme SV-MP puis de l'algorithme SV-SM.

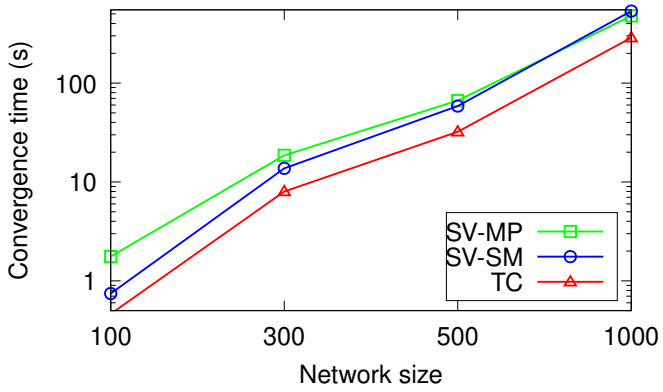


FIGURE 2.14 – Temps de convergence des algorithmes selon une probabilité $p = 0.15$. L'axe y utilise une échelle logarithmique.

Par exemple, lorsque $n = 1000$, $p = 0.10$ et $h_{max} = 5$, le temps de traitement de l'algorithme TC est de $287s$ tandis que celui de SV-MP est de $477s$ et celui de SV-SM est de 534.

Les figures 2.15 et 2.16 montrent le temps de convergence des algorithmes implémentés sur les topologies T1 et T2 respectivement. Ils présentent des résultats légèrement différents.

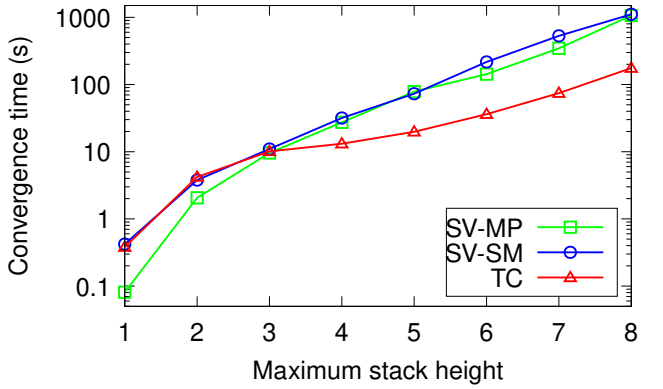


FIGURE 2.15 – Temps de convergence dans les topologies de réseau T1.

Nous pouvons voir que dans la topologie T1, TC est l'algorithme le plus rapide lorsque la hauteur de pile maximale est égale ou supérieure à 3. Il est d'un ordre de grandeur plus rapide lorsque la taille de pile maximale est égale ou supérieure à 6. En dessous de 3, TC est similaire à SV-SM en temps de calcul. Ce n'est pas un problème car nous avons besoin d'une hauteur de pile maximale supérieure à 3 pour trouver plus de chemins faisables.

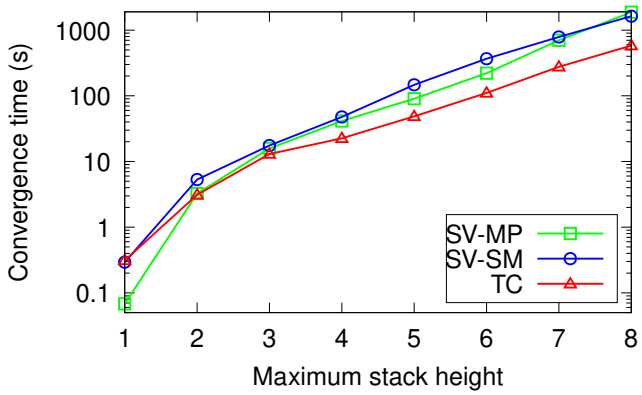


FIGURE 2.16 – Temps de convergence dans les topologies de réseau T2.

2.4.2.2 Exploration de chemins

Les figures 2.17, 2.18 montrent le pourcentage de chemins faisables trouvés par les algorithmes dans la topologie de réseau aléatoire 1k et les topologies réelles T1 et T2 en fonction de différents paramètres. La figure 2.17 montre le pourcentage de chemins faisables trouvés par les algorithmes dans la topologie de réseau aléatoire 1k en fonction de la hauteur maximale de la pile et la probabilité p. Nous pouvons voir que, pour les mêmes valeurs de n et p, seulement 54% de paires ordonnées de nœuds sont liées par un chemin faisable lorsque $h_{max} = 3$. Le pourcentage de paires liées augmente à 60% lorsque $h_{max} = 4$. Cela signifie que de nouvelles routes sont trouvées si nous autorisons un maximum de 4 tunnels imbriqués en même temps au lieu de seulement 3.

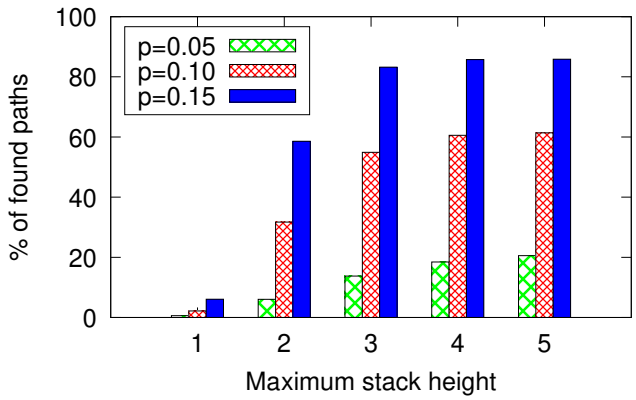


FIGURE 2.17 – Pourcentage de chemins trouvés dans la topologie de réseau aléatoire 1k.

La figure 2.18 montre le pourcentage de chemins trouvés dans les topologies T1 et T2 en fonction de la hauteur maximale de pile. Nous pouvons voir que l'augmentation de la hauteur aide à trouver plus de chemins faisables dans les topologies T1 et T2. On observe un rendement décroissant lorsque la hauteur est égale ou supérieure à 6. Il est important de noter que pour une hauteur maximale de pile de 6 ou plus, l'algorithme TC prend environ 20 secondes pour T1 et 100 secondes pour T2 pour calculer les chemins

faisables et les tables de routage, tandis que les deux algorithmes SV prennent environ 110 secondes pour T1 et 300 secondes pour T2. Comme le % de chemins trouvés continue d'augmenter jusqu'à 60% pour T2 et 40% pour T1 pour des hauteurs de 6 ou plus, être capable de calculer rapidement à ces paramètres est un avantage.

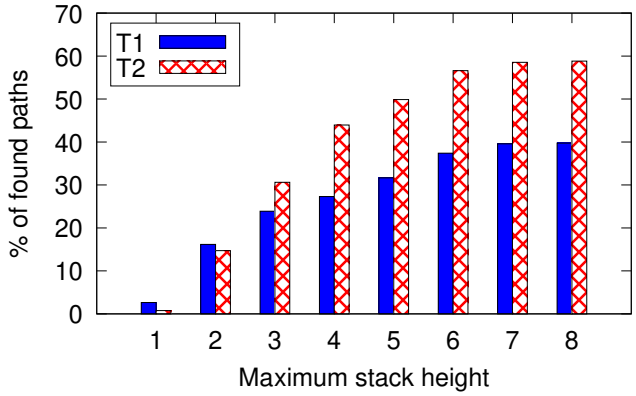


FIGURE 2.18 – Pourcentage de chemins trouvés dans les topologies de réseau T1 et T2.

La figure 2.19 montre le nombre de chemins explorés (faisables et non faisables) par chaque algorithme sur la topologie de réseau aléatoire 1k en fonction de la hauteur maximale de pile h_{max}. Nous pouvons voir que TC explore moins de chemins que SV-MP et SV-SM sur la topologie de réseau aléatoire 1k. Par exemple, lorsque h_{max} est fixé à 3, le % des chemins explorés par l'algorithme TC est d'environ 6% alors que celui par SV-MP est d'environ 39% et celui par SV-SM est 17%. Le pourcentage de chemins explorés passe à 18% pour l'algorithme TC et à 100% et 44% pour les algorithmes SV-MP et SV-SM, respectivement. Notez que les valeurs élevées de l'algorithme SV-MP peuvent être liées à la transmission de messages ou à l'asynchronisme.

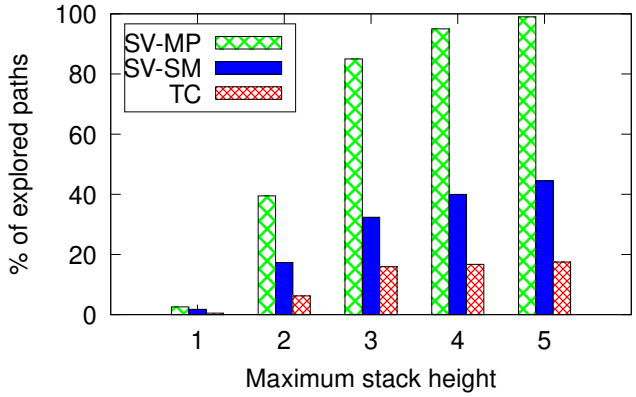


FIGURE 2.19 – Pourcentage de chemins explorés dans la topologie de réseau aléatoire 1k.

Les figures 2.20, 2.21 montrent le nombre de chemins explorés (faisables et non faisables) par chaque algorithme dans les topologies T1 et T2, et selon la hauteur maximale de pile h_{max}.

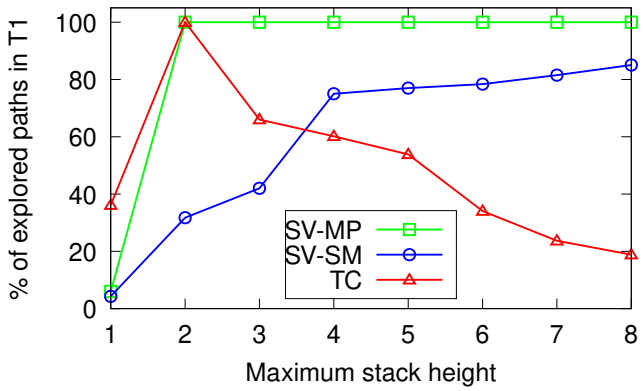


FIGURE 2.20 – Pourcentage de chemins explorés dans les topologies de réseau T1.

Nous pouvons voir que TC explore moins de chemins que SV-MP et SV-SM. Par exemple, lorsque h_{max} vaut 5, le % des chemins explorés par l'algorithme TC est d'environ 50% (resp. 60%) alors que celui par SV-MP est d'environ 100% (resp. 100%) et celle par SV-SM est de 70% (resp. 90%) dans le réseau T1 (resp. T2). Le pourcentage de chemins explorés diminue à 20% (resp. 30%) pour l'algorithme TC dans le réseau T1 (resp. T2) et reste entre 80% et 100% à la fois pour le SV-MP et les algorithmes SV-SM, respectivement dans les deux réseaux T1 et T2. Notez que les valeurs élevées produites par l'algorithme SV-MP peuvent être liées à la transmission de messages ou à l'asynchronisme.

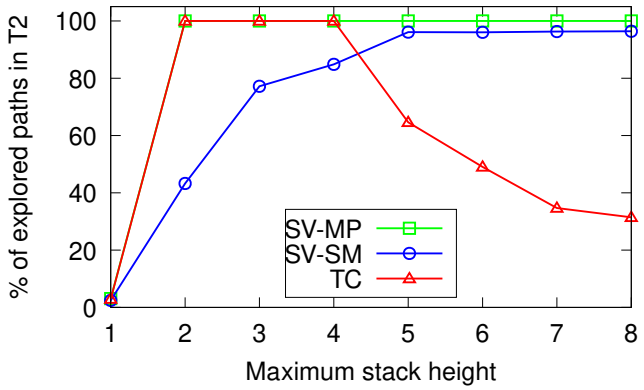


FIGURE 2.21 – Pourcentage de chemins explorés dans les topologies de réseau T2.

Ces résultats montrent que lorsque l'exploration de SV augmente, l'exploration de TC diminue. Rappelons que notre algorithme TC est composé de deux étapes principales (l'étape SV modifiée et l'étape de fermeture transitive). La première étape (algorithme 3) assure le calcul des chemins faisables sans-vallées (segments). Cette étape est basée sur une version modifiée de l'algorithme SV en limitant la propagation des routes infaisables ou des routes de chemins faisables avec vallées (ces chemins seront obtenus directement par l'étape de fermeture transitive). La deuxième étape (algorithme 4) permet de calculer la fermeture transitive de tous les chemins faisables qui restent. Dans ce contexte, explorer davantage dans l'algorithme SV d'origine [59] permet encore moins d'explorer

dans la version modifiée de SV (algorithm 3).

2.4.3 Longueur et diamètre des chemins faisables

2.4.3.1 Longueur des chemins faisables

Les figures 2.22, 2.23, 2.24 montrent la longueur moyenne du chemin en fonction de la taille du réseau, la probabilité p et la hauteur maximale de la pile h_{max}.

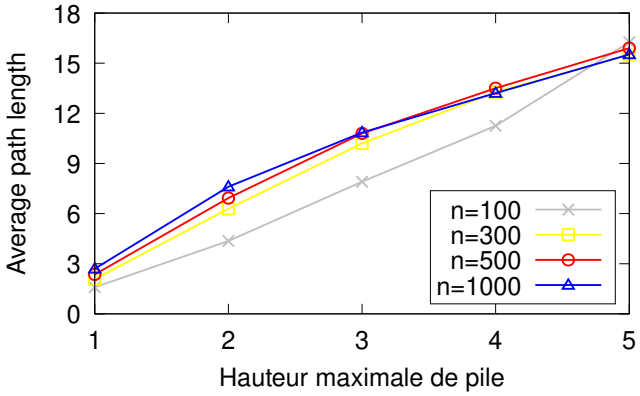


FIGURE 2.22 – Longueur moyenne de chemin dans des topologies aléatoires selon une probabilité $p = 0.05$.

Nous pouvons voir que la hauteur maximale de la pile a un impact énorme sur la longueur des chemins multicouches (*i.e.,* les chemins avec des piles plus élevées utilisent plus de fonctions que les autres), contrairement à la probabilité p et à la taille du réseau.

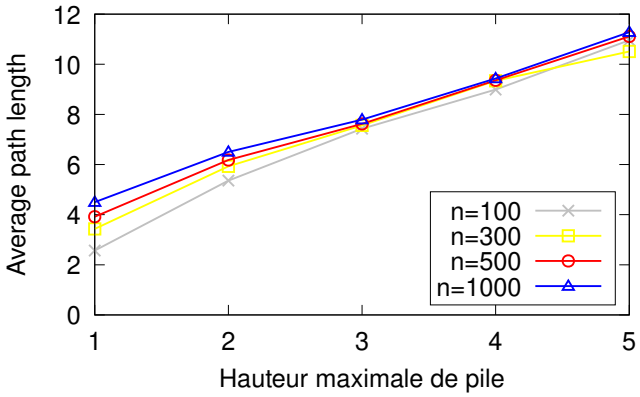


FIGURE 2.23 – Longueur moyenne de chemin dans des topologies aléatoires selon une probabilité $p = 0.10$.

Par exemple, lorsque $p = 0.10$ et $h_{max} = 2$, la longueur moyenne du chemin est d'environ 5 (resp. 7) sauts dans un réseau de 100 (resp. 1000) nœuds. Alors que pour $h_{max} = 5$, la longueur moyenne du chemin atteint 11 sauts pour toutes les tailles de réseau. Nous pouvons voir également qu'avec une faible probabilité, *i.e.*, $p = 0.05$ la longueur moyenne du chemin atteint 16 sauts au lieu de 11 et 9 sauts pour les deux autres probabilités. Ces résultats montrent qu'avec une faible probabilité, le nombre de fonctions par nœud est également faible, ce qui nécessite de passer par plus de nœuds pour calculer des chemins faisables.

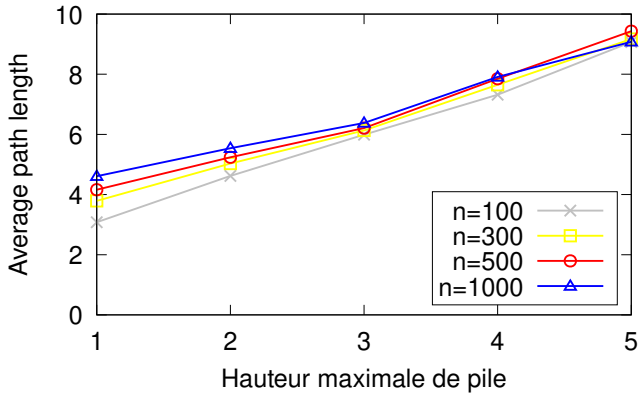


FIGURE 2.24 – Longueur moyenne de chemin dans des topologies aléatoires selon une probabilité $p = 0.15$.

Les figures 2.25, 2.26, 2.27 montrent la longueur maximale du chemin en fonction de la taille du réseau, la probabilité p et la hauteur maximale de la pile h_{max}.

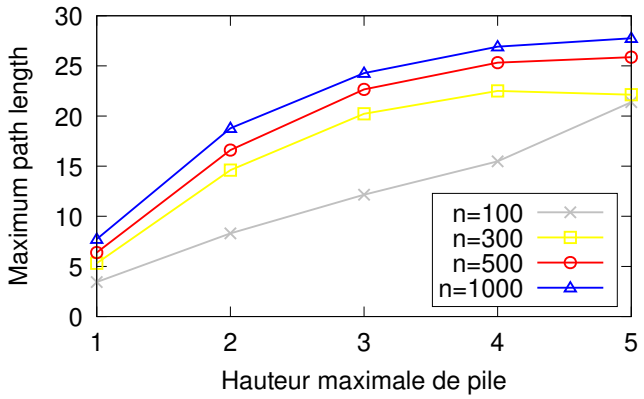


FIGURE 2.25 – Longueur maximale de chemin dans des topologies aléatoires selon une probabilité $p = 0.05$.

Nous pouvons voir que la longueur maximale du chemin dans les réseaux aléatoires peut atteindre 28 sauts lorsque $p = 0,05$ et 19 (resp. 14) sauts pour la probabilité $p = 0,10$ (resp. $p = 0,15$).

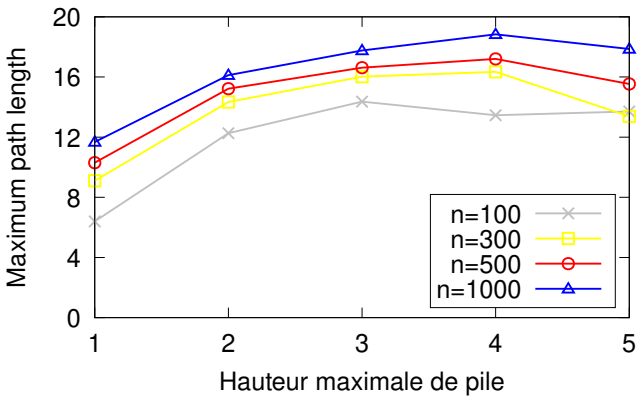


FIGURE 2.26 – Longueur maximale de chemin dans des topologies aléatoires selon une probabilité $p = 0.10$.

De manière similaire aux résultats ci-dessus, plus la probabilité p est faible, plus la longueur du chemin est élevée car il y a moins de fonctions d'adaptation dans le réseau.

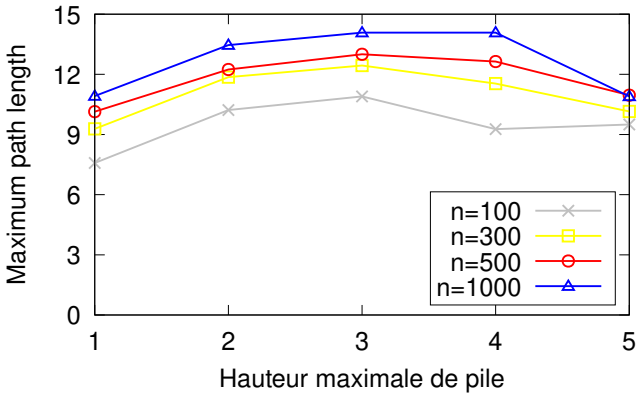


FIGURE 2.27 – Longueur maximale de chemin dans des topologies aléatoires selon une probabilité $p = 0.15$.

La figure 2.28 montre la longueur moyenne du chemin dans les topologies de réseau T1 et T2. Nous pouvons voir que la longueur moyenne du chemin dépend fortement de la hauteur maximale de la pile. La longueur moyenne dans les chemins qui ont une pile avec 3 protocoles est d'environ 12 sauts en T1 et d'environ 10 sauts en T2. Alors que pour les chemins qui ont une pile avec 6 protocoles, la longueur moyenne du chemin est de 29 en T1 et de 18 en T2. La différence entre T1 et T2 peut être liée à la structure du graphe sous-jacent, au taux de faisabilité du chemin, et en particulier à la répartition des fonctions d'adaptation sur chaque topologie (T2 contient plus de nœuds hétérogènes v4/v6 que T1).

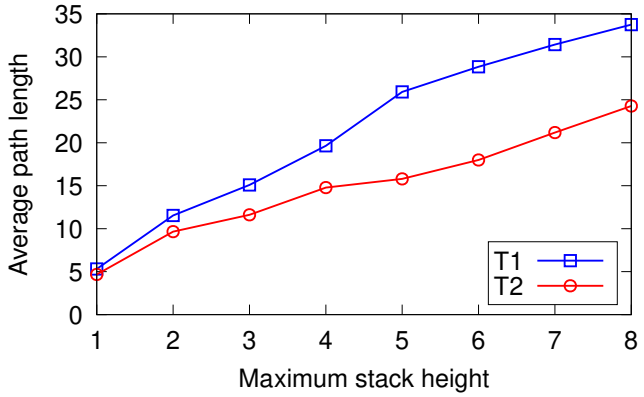


FIGURE 2.28 – Longueur moyenne de chemins dans les topologies de réseau T1 et T2.

La figure 2.29 montre la longueur maximale du chemin dans les topologies de réseau T1 et T2. Nous pouvons voir que la longueur maximale dépend fortement de la hauteur maximale de pile. La longueur maximale dans les chemins qui ont une pile avec 2 protocoles est d'environ 30 sauts en T1 et d'environ 25 sauts en T2. Alors que pour les chemins qui ont une pile avec 5 protocoles, la longueur maximale des chemins est de 53 en T1 et de 35 en T2. Nous pouvons voir également que la longueur maximale est de 47 sauts dans T1 et de 32 sauts dans T2. Ceci est lié au nombre d'encapsulations et de conversions impliquées dans le chemin. Autrement dit, augmenter la hauteur maximale de pile offre la possibilité d'utiliser plus d'encapsulations et moins de conversions (s'il est possible) dans le chemin. Par exemple, si l'unique chemin faisable pour $h_{max} = 3$ doit passer par k conversions, il est possible qu'il existe un autre chemin optimal qui utilise moins de k encapsulations (au lieu de k conversions) pour une hauteur $h_{max} > 3$.

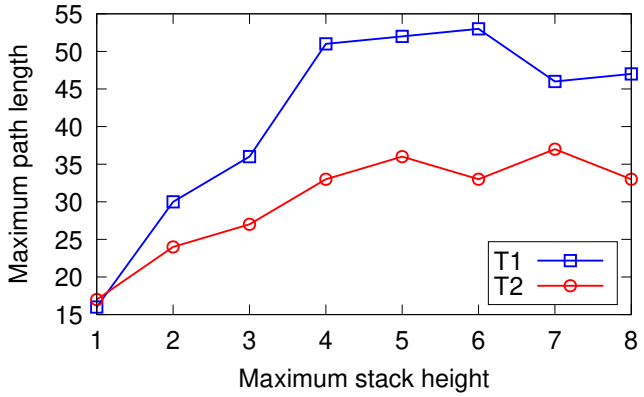


FIGURE 2.29 – Longueur maximale de chemins dans les topologies de réseau T1 et T2.

2.4.3.2 Diamètre faisable

Rappelons que les chemins faisables dans un réseau multicouche peuvent avoir des longueurs exponentielles. La figure 2.30 montre la longueur du diamètre du réseau en

fonction de la probabilité p et la taille du réseau avec une hauteur maximale de pile $h_{max} = 5$. Nous pouvons voir que la taille et la probabilité du réseau ont un impact sur la longueur du diamètre du réseau. Notez que le diamètre avec $p = 0.0$ est le diamètre du graphe sous-jacent, *i.e.,* le réseau sans aucune fonction d'adaptation. Par exemple, lorsque $p = 0.05$, le diamètre est de 21 (resp. 28) sauts dans un réseau aléatoire de 100 (resp. 1000) nœuds. Alors que pour $p = 0.10$, le diamètre est de 14 (resp. 19) sauts dans un réseau aléatoire de 100 (resp. 1000) nœuds. Cependant si $p = 0.15$ le diamètre est de 11 (resp. 14) sauts dans un réseau aléatoire de 100 (resp. 1000) nœuds. Nous savons que la probabilité p a un impact direct sur le nombre de fonctions par nœud, et les résultats précédents montrent que l'augmentation de la probabilité p augmente le pourcentage de chemins faisables. Par conséquent, avoir plus de chemins faisables offre plus de choix pour trouver un chemin optimal avec moins de sauts.

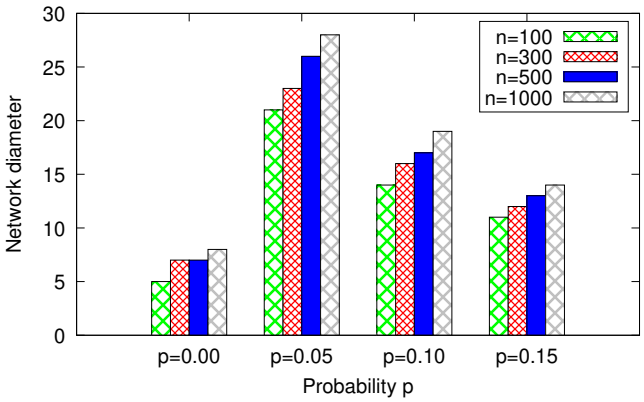


FIGURE 2.30 – Diamètre faisable du réseau dans les topologies aléatoires.

Le tableau 2.3 montre la longueur du diamètre faisable du réseau multicouche dans les topologies T1 et T2 avec $h_{max} = 8$. On voit que le diamètre du graphe sous-jaçent T1 (resp. T2) est de 23 (resp. 21) sauts. Le diamètre du réseau multicouche en T1 (resp. T2) atteint 53 (resp. 36) sauts. Ces résultats montrent l'impact de la proportion de nœuds en double pile IPv4/IPv6 dans le réseau. En particulier, plus le nombre de nœuds hétérogènes est élevé, plus la faisabilité du chemin est élevée. En d'autres termes, avoir plus de chemins faisables offre plus de possibilités pour trouver un chemin optimal plus court.

Topologie	**Diamètre ($\mathcal{G}$)**	**Diamètre($\mathcal{N}$)**
T1	23	53
T2	21	36

TABLE 2.3 – Diamètre faisable du réseau dans les topologies T1 et T2.

2.4.4 Nombre de segments et taille des tables de routage

2.4.4.1 Nombre de segments

Les figures 2.31, 2.32, 2.33 montrent la hauteur de la pile de segments "DST" en fonction de la taille du réseau, de la probabilité p et de la hauteur maximale de pile h_{max}.

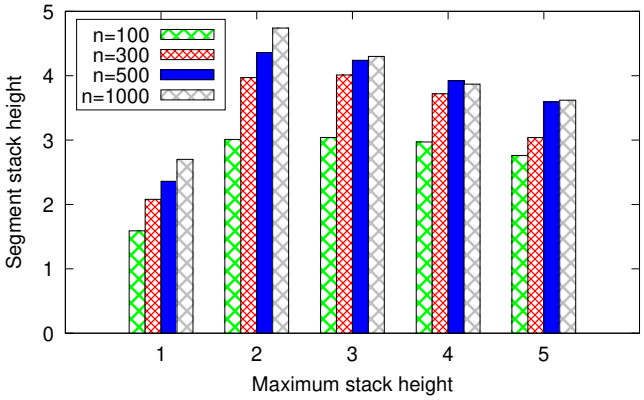


FIGURE 2.31 – Taille de la pile de segments dans des réseaux aléatoires selon une probabilité $p = 0.05$.

Nous pouvons voir que, dans toutes les configurations, le nombre de segments *i.e.,* le nombre de chemins sans vallées dans le "DST" ne dépasse pas 5 identifiants de destination. En particulier, la hauteur de la pile "DST" est égale à 5 (voir diamètre du graphe sous-jacent) dans les chemins où la pile de protocoles contient au plus deux protocoles (y compris les chemins avec uniquement des conversions). Alors que pour les chemins contenant entre 3 et 5 tunnels imbriqués, la hauteur du "DST" est comprise entre 2 et 4.

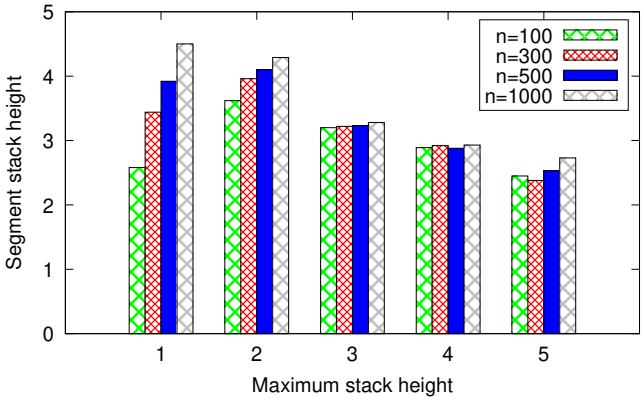


FIGURE 2.32 – Taille de la pile de segments dans des réseaux aléatoires selon une probabilité $p = 0.10$.

Ces résultats montrent que la hauteur de la pile "DST" est liée au nombre de conversions et d'encapsulations impliquées dans le chemin. En d'autres termes, plus le nombre

de conversions est élevé, plus le nombre de segments est élevé.

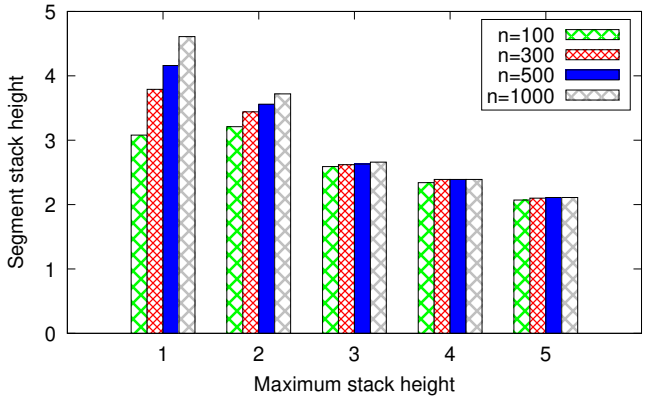


FIGURE 2.33 – Taille de la pile de segments dans des réseaux aléatoires selon une probabilité $p = 0.15$.

La figure 2.34 montre la hauteur de pile de segments dans les topologies de réseau T1 et T2 en fonction de la hauteur maximale de pile de protocoles h_{max}. On peut voir que le nombre de segments ne dépasse pas 9 pour le réseau T1 et 6 pour T2. Par exemple, la hauteur de pile des segments est de 7 (resp. 5) dans le réseau T1 (resp. T2) si $h_{max} = 3$. Lorsque $h_{max} = 8$, la hauteur de pile des segments est de 5 (resp. 3) dans le réseau T1 (resp. T2). Ces résultats montrent que le nombre de segments diminue lorsque la valeur h_{max} augmente.

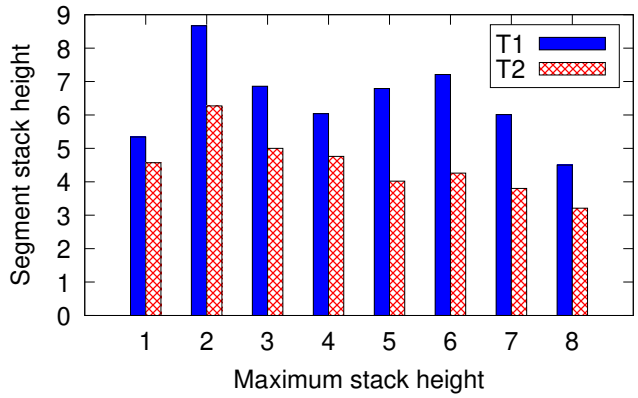


FIGURE 2.34 – Taille de la pile de segments dans les topologies de réseau T1 et T2.

Ces résultats montrent que la hauteur de la pile "DST" est peut être liée à la nature des segments, notamment le fait d'avoir plus de segments *plats* (sans encapsulation/décapsulation, *i.e.,* des tunnels) augmente la hauteur de la pile de segments et vice versa.

2.4.4.2 Taille des tables de routage

La complexité spatiale du problème dans le pire des cas, c'est-à-dire la taille de la table de routage dans le pire des cas, peut atteindre $n\,(\frac{1-\lambda^{h_{max}+1}}{1-\lambda} - 1)$ entrées (*i.e.,* des lignes), qui peuvent être assez élevées. Les figures 2.35, 2.36, 2.37 montrent la taille de la table de routage, *i.e.,* le nombre de lignes en fonction de la taille du réseau, la probabilité p et la hauteur maximale de pile h_{max}.

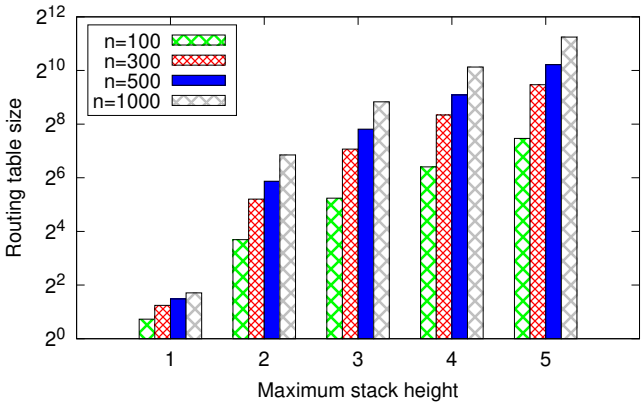


FIGURE 2.35 – Taille de la table de routage dans des réseaux aléatoires selon une probabilité $p = 0.05$.

Nous pouvons voir que tous les paramètres d'entrée ont un impact sur la taille de la table de routage. Par exemple, si $p = 0.05$ alors la taille de la table de routage est d'environ 2^4 (resp. 2^7) lignes dans un réseau de 100 (resp. 1000) nœuds et h_{max} fixé à 2, et est d'environ 2^7 (resp. 2^{11}) dans un réseau de 100 (resp. 1000) nœuds et $h_{max} = 5$. Alors que pour $p = 0.15$, la taille de la table de routage est d'environ 2^8 (resp. 2^{11}) lignes dans un réseau de 100 (resp. 1000) nœuds et h_{max} fixé à 2, et est d'environ 2^{11} (resp. 2^{14}) dans un réseau de 100 (resp. 1000) nœuds et h_{max} fixé à 5.

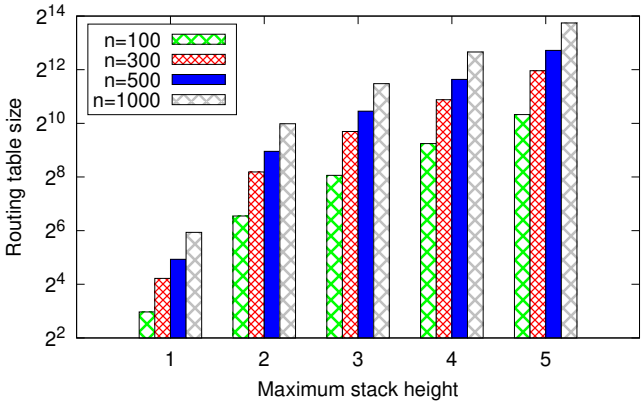


FIGURE 2.36 – Taille de la table de routage dans des réseaux aléatoires selon une probabilité $p = 0.10$.

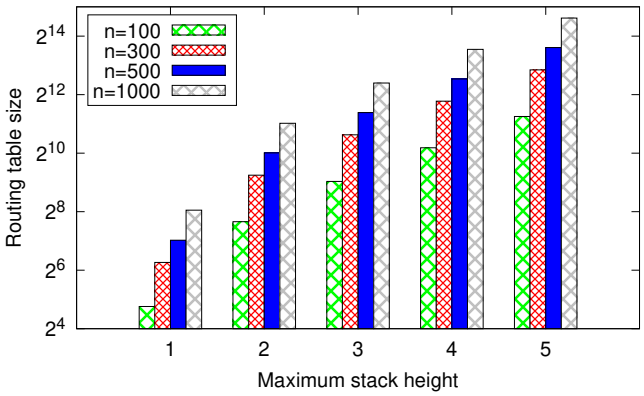


FIGURE 2.37 – Taille de la table de routage dans des réseaux aléatoires selon une probabilité $p = 0.15$.

Ces résultats montrent que, d'une part, augmenter la taille du réseau n permet à un nœud d'atteindre plus de destinations, *i.e.,* d'avoir plus de destinations possibles dans sa table de routage. D'autre part, augmenter la probabilité p et h_{max}, *i.e.,* avoir plus de fonctions d'adaptation par nœud (y compris les encapsulations), permet d'augmenter le taux de faisabilité, et aussi d'augmenter les piles de protocoles possibles pour chaque paire d'extrémités. Cela permet d'atteindre la même destination avec différentes piles de protocoles.

La figure 2.38 montre la taille de la table de routage dans les topologies de réseau T1 et T2 en fonction de la hauteur maximale de la pile de protocoles h_{max}. Nous pouvons voir que la hauteur maximale de la pile de protocoles h_{max} a un impact significatif sur le nombre de lignes dans les tables de routage. Par exemple, pour $h_{max} = 3$, la taille de la table de routage est d'environ 2^{12} lignes pour les réseaux T1 et T2. Alors que pour $h_{max} = 8$, la taille de la table de routage est d'environ 2^{16} lignes pour les réseaux T1 et T2.

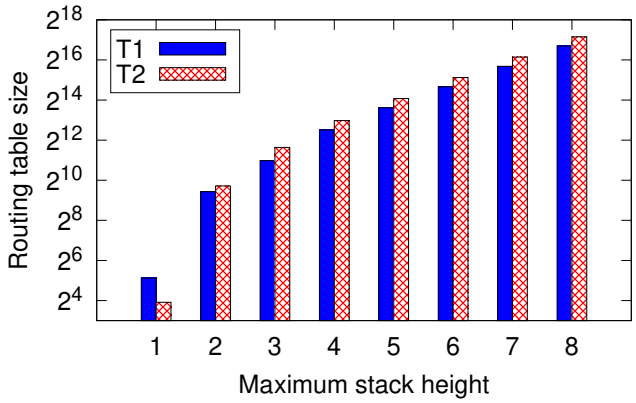


FIGURE 2.38 – Taille de la table de routage dans les topologies de réseau T1 et T2.

Ces résultats montrent que l'augmentation de la hauteur maximale de pile permet d'augmenter le taux de faisabilité, et également d'augmenter les piles de protocoles possibles pour chaque couple source-destination. Cela permet d'atteindre la même destination avec différentes piles de protocoles.

2.5 Limitations pratiques et adaptation des algorithmes.

Transformer les algorithmes proposés en un protocole de routage à part entière nécessite de définir des messages, des en-têtes, des structures de données, etc. Bien que cette partie soit laissée pour des travaux futurs, nous présentons ici quelques éléments de mise en œuvre.

2.5.1 Exigences

Comme déjà souligné dans [59], le problème principal pour une implémentation d'un protocole de routage avec tunnels est que la hauteur de la pile de protocoles et les limites de longueur du message sont polynomiales, ce qui peut rendre les en-têtes trop volumineux par rapport à la charge utile des données. La solution, proposée dans [59] et également appliquée ici, est de borner la hauteur maximale de la pile par une constante h_{max}. Cette valeur devient donc un paramètre du protocole. En pratique, les plus courts chemins peuvent généralement être calculés pour de petites valeurs de h_{max}.

De plus, l'algorithme de fermeture transitive de chemins faisables nécessite également de prendre en compte une pile de destionations DST dans l'en-tête de paquet. Bien que cette pile soit limitée par le diamètre du réseau, elle peut encore être trop longue pour l'en-tête. Nos résultats de simulation, présentés dans la section 2.4, ont montré que cette taille de pile de destinations ne dépasse jamais 5 pour les topologies aléatoires et 9 pour les topologies réelles.

Un autre problème important est l'utilisation du chiffrement sur les paquets. De nombreux réseaux sécurisés, tels que les réseaux privés virtuels (VPN), la périphérie du service d'accès sécurisé (SASE), etc. utilisent le cryptage, ce qui rend les en-têtes de paquet et les données illisibles pour les protocoles fonctionnant dans d'autres couches. Dans de tels cas, il est impossible de calculer des chemins sur le réseau multicouche, car les réseaux sécurisés cachent leur topologie. Inversement, dans notre proposition, les routeurs peuvent voir toutes les piles présentes dans le réseau multicouche, ce qui peut être considéré comme un problème dans les réseaux sensibles. De plus, les piles dans les en-têtes doivent également être lisibles par n'importe quel routeur du réseau multicouche.

Une implémentation nécessitera que tous les nœuds et les protocoles aient un identifiant unique. Les identifiants de protocole peuvent être codés sur un octet (comme le

champ de protocole IP). La longueur de l'adresse des nœuds dépend du type de protocole et peut varier de deux octets (adresses courtes IEEE 802.15.4) à seize octets (adresses IPv6). Les messages de protocole de routage qui annoncent des itinéraires incluront le nœud de destination, la pile de protocoles nécessaire pour qu'un paquet atteigne la destination et le coût de l'itinéraire. Les tables de routage stockeront les entrées indexées par leurs destinations et leurs piles de protocoles. Chaque entrée contiendra à son tour le coût, le saut suivant et la fonction d'adaptation à appliquer au paquet transmis.

2.5.2 Format de paquet

La figure 2.39 illustre une mise en œuvre possible d'un PDU *i.e.,* paquet réseau multicouche. L'en-tête commence par la pile de destinations, qui est composée de d adresses de destination, implémentées sous la forme d'une pile de d étiquettes MPLS correspondantes, de la première rencontrée à la dernière étant l'adresse de destination finale. L'entête comprend alors l'adresse source (non utilisée pour le routage) suivie de la hauteur de la pile de protocoles, indiquant le nombre de protocoles dans la pile de protocoles qui suit, *i.e.,* la pile des identifiants de chaque protocole, correspondant aux en-têtes imbriqués (par évitant l'accès à l'ensemble de la pile d'en-têtes, cette pile de protocoles accélère le processus de routage). Vient ensuite la pile d'en-tête composée de chaque en-tête encapsulé (précédé de sa longueur). Enfin est jointe les données du paquet. Les longueurs d'adresse dans la figure 2.39 sont indiquées comme n pour chacune d'entre elles, mais elles peuvent être de différentes tailles compte tenu du protocole utilisé au niveau de leur couche correspondante. De même, les longueurs d'en-tête dans la figure sont indiquées comme m pour chacun d'eux, mais elles peuvent être de différentes tailles compte tenu du protocole utilisé au niveau de leur couche correspondante.

Comme indiqué précédemment, la pile de segments, nommée DST, peut être implémentée avec le routage par segments (SR). En SR, un segment est identifié par un identifiant pour tout type d'instruction. Ainsi, dans un routeur, un identifiant peut pointer vers une action de transfert ou une action de service [30]. Une telle action de service peut être une encapsulation ou une désencapsulation. Dans le cas de SR-MPLS, chaque destination serait remplacée par son identifiant d'étiquette MLPS correspondant de 20 bits. Le mappage entre les étiquettes et les adresses de destination devrait être conservé dans une table supplémentaire pour qu'une source soit capable de coder la pile. La taille de pile maximale prise en charge par SR est de 6 pour les équipements Huawei [43], de 10 pour les équipements Cisco [14], de 11 pour les équipements Nokia [86] et de 16 pour les équipements Juniper [54].

SR-MPLS Étiquette 1 (4 octets)
SR-MPLS Étiquette 2 (4 octets)
…
SR-MPLS Étiquette d (4 octets)
Source (n octets)
Hauteur de pile (1 octet) = k
Protocole 1 (1 octet)
Protocole 2 (1 octet)
…
Protocole k (1 octet)
Longueur de l'en-tête du protocole 1
En-tête du protocole 1 (m octets)
…
Longueur de l'en-tête du protocole k
En-tête du protocole k (m octets)
Données

FIGURE 2.39 – Format possible d'en-tête et données d'un paquet réseau multicouche.

2.5.3 Cas pratiques

Les cas réalistes où le routage automatique dans les réseaux multiprotocoles à grande échelle sont requis incluent les grands réseaux maillés ou de capteurs IoT où plusieurs protocoles peuvent être disponibles et où la virtualisation du réseau est nécessaire pour les clients multi-locataires. Cela peut également être le cas pour les réseaux cellulaires 5G+ qui mettent en œuvre le découpage du réseau. La dynamique des réseaux maillés et la mobilité des terminaux dans les réseaux cellulaires nécessitent un établissement automatique de tunnels, car leur configuration manuelle serait ingérable.

Notre solution est conçue pour les réseaux multicouches, ce qui se produit généralement lorsque plusieurs réseaux virtuels (VPN) et/ou protocoles (IPv4/IPv6) sont superposés. Il peut être utilisé dans le contexte intra-domaine du réseau d'un grand opérateur. Il ne sera probablement pas en mesure de gérer les informations de routage public mondial en raison de sa taille. Cela signifie que ce système de routage multicouche ne peut pas être déployé pour couvrir tous les routeurs BGP de l'internet (car sa complexité de calcul est $> O(n^3)$). Mais les routeurs (*i.e.,* les commutateurs SDN) participant à un réseau multicouche de taille gérable peuvent être dispersés entre les domaines tant qu'ils sont accessibles à leur contrôleur SDN. Deux options sont alors possibles. Si un seul contrôleur principal est utilisé (avec un ou plusieurs contrôleurs de secours), cela signifie que les opérateurs des routeurs sont disposés à partager les informations de routage à l'intérieur de ce contrôleur commun. Si chaque opérateur possède son propre contrôleur alors un échange de données entre contrôleurs doit être mis en place pour que les contrôleurs puissent partager la même vue du réseau multicouche.

La configuration ou l'établissement automatique de tunnels dépend de l'architecture disponible. Dans une architecture standard basée sur des routeurs exécutant des protocoles entièrement distribués, alors chaque routeur doit exécuter un daemon spécifique capable *i)* d'émettre des requêtes de construction de tunnel, et capable de *ii)* recevoir et traiter de telles requêtes de construction. Dans une architecture SDN, le contrôleur est en charge du calcul centralisé des tunnels dynamiques car il connaît la topologie complète du réseau. Une fois définies, les règles de tunnel peuvent être poussées dans les commutateurs SDN associés pour configurer les tunnels.

Comme mentionné ci-dessus, nos solutions proposées peuvent être déployées dans un environnement SDN. Dans les réseaux SDN, un commutateur SDN possède une table de règles SDN (ou table de flux). Une telle table possède une nombre de règles limitées. La taille maximale de la table de flux autorisée est 32k règles pour les équipements Cisco [13], 65k règles pour les équipements Juniper [53] et HPE [42], et 160k règles pour les équipements Dell [19].

2.6 Conclusion

Les réseaux englobant plusieurs protocoles de communication nécessitent l'utilisation de fonctions d'adaptation (conversion, encapsulation, décapsulation) dans certains nœuds spécifiques pour assurer l'accessibilité à l'intérieur du réseau. Aujourd'hui, cela est principalement effectué par une configuration manuelle sur ces nœuds spécifiques. Il n'existe actuellement aucun protocole de routage capable de calculer automatiquement les plus courts chemins qui incluent les conversions et les tunnels.

Dans ce chapitre, nous proposons des algorithmes parallèles capables de calculer des tables de routage en utilisant l'opération de fermeture transitive. Nos algorithmes offrent un degré élevé de parallélisation et permettent d'utiliser la concaténation de chemins (*i.e.,* la fermeture transitive) chaque fois que possible. Ils sont particulièrement adaptés aux réseaux gérés par SDN où le contrôleur détient la connaissance complète du réseau. Les résultats montrent que nos algorithmes surpassent les autres solutions en termes de temps de traitement tout en étant aussi efficaces pour le calcul de chemin.

Chapitre 3

Métaroutage dans les réseaux hétérogènes et multicouches

Le but de l'abstraction n'est pas d'être vague, mais de créer un nouveau niveau sémantique dans lequel on peut être absolument précis.

– Edsger Dijkstra

Sommaire

3.1 Introduction

Le métaroutage utilise une structure algébrique, appelée algèbre de routage, pour modéliser les protocoles de routage. Cela facilite la conception et la validation de ces protocoles, en particulier la convergence. Une algèbre de routage se définit par un ensemble d'éléments et deux opérations binaires : l'une pour composer les chemins, l'autre pour les comparer et les choisir. Les algorithmes classiques des plus courts chemins reposent généralement sur deux opérations binaires : "min" pour comparer les poids des chemins et "+" pour calculer le poids d'un chemin à partir des poids de ses composants. Les structures algébriques de base employées dans ces contextes sont les semi-anneaux, les algèbres de fonctions et les algèbres de Sobrinho.

Jusqu'à présent, la recherche sur les algèbres de routage a principalement porté sur leur application aux protocoles de routage des réseaux utilisant un seul protocole d'adressage et de transmission. Cependant, dans le contexte des réseaux multicouches, l'utilisation des algèbres de routage pour déduire des propriétés devient essentielle. Dans un réseau multicouche, une algèbre de routage doit être capable d'énumérer tous les ensembles des plus courts chemins valides entre chaque paire de nœuds du réseau. Ces chemins peuvent être représentés par des couples de piles de protocoles (ou une composition de fonctions d'adaptation qui rendent un chemin valide) ainsi que leurs poids associés. Pour définir ces nouvelles structures, il est nécessaire de mettre en place deux nouvelles opérations binaires sur l'ensemble des plus courts chemins valides. La première opération permet de concaténer les plus courts chemins valides (tout en préservant la validité) tandis que la seconde opération est utilisée pour choisir les plus courts chemins valides.

Dans ce chapitre, nous proposons la modification de ces algèbres de routage existantes pour traiter les réseaux qui contiennent plusieurs protocoles de transmission où les tunnels sont omniprésents. Nous étudions également certaines propriétés de convergence sur ces algèbres proposées. La section 3.2 passe en revue les travaux existants sur les algèbres de routage et présente les fondements algébriques utilisés tout au long de ce chapitre. Ceci est suivi dans la section 3.3 par trois définitions des structures algébriques (semi-anneau, algèbre de fonctions et algèbre de Sobrinho) nécessaires pour le problème des plus courts chemins valides. Enfin, nous prouvons un théorème dans la section 3.4 qui fournit une borne supérieure sur la longueur de chaque plus court chemin valide dans un réseau multicouche libre. La section 3.6 conclut le chapitre.

Ce chapitre est basé sur nos travaux sur les algèbres de routage avec tunnels dans les réseaux hétérogènes et multicouches, qui ont été publiés dans le Workshop New IP and Beyond de la conférence internationale IEEE ICNP 2022 [83]. La version complète est actuellement en révision pour le journal international Journal of Network and Computer Applications (JNCA) [85].

3.2 Autour de l'algèbre de routage

Dans cette section, nous citons la plupart des recherches sur l'algèbre de routage sans tunnels. Nous définissons par la suite les fondements algébriques nécessaires à la compréhension de ce chapitre. Nous définissons également les différents styles de l'algèbre de routage. Enfin, nous expliquons comment modéliser la solution optimale du problème de calcul de chemins en utilisant une algèbre de routage.

3.2.1 État de l'art

L'algèbre de routage a été initialement étudiée par [12]. L'auteur a proposé une structure algébrique basée sur des semi-anneaux pour la formulation du problème de calcul de chemins. Il a également proposé des solutions génériques basées sur des méthodes classiques d'algèbre linéaire telles que la méthode de Jacobi et Gauss-Seidel et la méthode d'élimination de Gauss et Jordan. Plusieurs structures algébriques et algorithmes ont été décrits par [7, 40, 34, 75] pour résoudre divers problèmes d'optimisation et de routage dans les réseaux. [77] a proposé de nouvelles structures algébriques et des algorithmes génériques pour le problème du plus court chemins et le problème des $k-$plus courts chemins. Plus récemment, des travaux sur les algèbres de routage ont été appliqués aux protocoles de routage existants. Une première application de ceci était sur les protocoles de routage à vecteur de chemin proposés par [101, 100] basé sur une structure dit algèbre de Sobrinho. Non seulement les auteurs ont proposé une condition suffisante pour garantir la correction du protocole iBGP, mais ils ont également montré que si l'algèbre est *monotone*[1] alors le protocole converge dans n'importe quel réseau, mais pas nécessairement vers une solution optimale globale. Si l'algèbre est *isotone*, alors le protocole converge vers une solution optimale globale.

Dans [37], les auteurs ont proposé une approche appelée *metarouting* qui est basée sur l'algèbre de Sobrinho et utilisée pour définir des protocoles de routage de manière déclarative et de haut niveau. Ils ont également introduit un produit "scoped" pour modéliser la métrique combinée d'iBGP et d'eBGP. Dans [37, 38, 24], les auteurs ont proposé le produit lexicographique pour combiner plusieurs métriques de QoS en une seule métrique composite. Ils ont également montré quelles propriétés sont requises pour garantir des solutions optimales globales et locales dans le cas des produits lexicographiques. Un autre produit dit fonctionnel a été utilisé par [56] pour modéliser des métriques non lexicales EIGRP. Un important travail de [18] a montré que les conditions de Sobrinho sont suffisantes pour qu'un protocole de routage converge vers une solution unique, notamment dans le cas de l'algorithme asynchrone de Bellman-Ford. Ils ont utilisé une algèbre de routage avec fonctions[2]. Plus récemment, [102] a étudié le problème du routage avec

1. [102] utiliser le terme *inflation*, tandis que [18] utilise le terme *croissant* pour désigner la propriété de monotonie définie par [100]. Dans cet article, nous utilisons le terme monotone.

2. Une algèbre de fonctions est une généralisation d'une algèbre d'endomorphismes d'un monoïd, voir [76] pour plus d'informations.

des métriques non isotones et a proposé de nouveaux protocoles de routage basés sur un ordre partiel afin de préserver l'isotonie.

3.2.2 Modèles d'algèbre de routage

Comme mentionné dans la section précédente, il existe trois styles d'algèbre de routage : un semi-anneau tel que décrit par [12, 77, 24], l'algèbre de Sobrinho telle que proposée par [101, 100, 37] et une algèbre de fonctions comme proposée par [17, 18]. Un modèle de quadrant a été proposé par [38] pour résumer les différents styles existants et leurs relations. Dans la suite, nous définissons d'abord les notations et les propriétés algébriques utilisées tout au long de ce chapitre. Ensuite, nous définissons les différents styles d'algèbre de routage avec des exemples de problèmes de routage traditionnels.

3.2.2.1 Préliminaires algébriques

- Nous notons $S = \{a, b, c, \dots\}$ l'ensemble fini d'éléments a, b, c, etc.
- Nous notons $\mathcal{P}(S) = \{\emptyset, \{a\}, \{a, b\}, \dots, S, \}$ l'ensemble des parties de S.
- Nous notons $\oplus$ et $\otimes$ deux opérations binaires sur les éléments de l'ensemble S.
- Nous notons $(\oplus \times \otimes)$ le produit direct des deux opérations binaires $\oplus$ et $\otimes$.
- Nous notons $\overline{0}$ et $\overline{1}$ l'élément neutre ou absorbant d'une opération binaire $\oplus$ ou $\otimes$.
- Nous notons $\preceq$ ou $\prec$ une relation d'ordre binaire sur les éléments de S.
- Nous notons $\mathbb{R}^{\infty}$ l'ensemble $\mathbb{R} \cup \{-\infty, +\infty\}$.

Soient un ensemble S et deux opérations binaires $\oplus$ et $\otimes$. Pour tout a, b, c dans S, on définit les propriétés algébriques de base dans le tableau 3.1 suivant.

Proprieté	Définition
Sélectivité $(\oplus)$	$a \oplus b \in \{a, b\}$
Idempotence $(\oplus)$	$a \oplus a = a$
Associativité $(\oplus)$	$a \oplus (b \oplus c) = (a \oplus b) \oplus c$
Commutativité $(\oplus)$	$a \oplus b = b \oplus a$
Distributivité-Droite $(\oplus, \otimes)$	$(b \oplus c) \otimes a = (b \otimes a) \oplus (c \otimes a)$
Distributivité-Gauche $(\oplus, \otimes)$	$a \otimes (b \oplus c) = (a \otimes b) \oplus (a \otimes c)$
Neutre $(\oplus, \overline{0})$	$a \oplus \overline{0} = \overline{0} \oplus a = a$
Absorbant $(\oplus, \overline{1})$	$a \oplus \overline{1} = \overline{1} \oplus a = \overline{1}$

TABLE 3.1 – Définition des propriétés algébriques.

Soit une relation d'ordre $\preceq$ sur un ensemble S. Pour tout a, b, c dans S, on dit que la relation $\preceq$ est un ordre *partiel* si est seulement si $\preceq$ elle satisfait :

— *Réflexivité* : $a \preceq b$
— *Transitivité* : $a \preceq b$ et $b \preceq c$, alors $a \preceq c$
— *antisymétrie* : $a \preceq b$ et $b \preceq a$, alors $a = b$

Cet ordre est dit *total*, *i.e.,* S est totalement ordonné par $\preceq$ si est seulement si $\preceq$ satisfait :

— *Totalalité* : $a \preceq b$ ou $b \preceq a$

La relation $\leq$ sur les nombres réels définit un ordre total sur $\mathbb{R}$. Un exemple d'une relation d'ordre partiel est la relation $\subseteq$ définie sur un ensemble des parties $\mathcal{P}(S)$ de S,

$$\text{Pour tout } S_1, S_2 \in \mathcal{P}(S),\ S_1 \subseteq S_2 \text{ alors } S_1 \cup S_2 = S_2$$
$$\text{Pour tout } S_1, S_2, S_3 \in \mathcal{P}(S),\ S_1 \subseteq S_2 \text{ alors } S_1 \cup S_3 \subseteq S_2 \cup S_3$$

Et on peut représenter une telle relation par un diagramme de Hasse (voir les treillis).

3.2.2.2 Structures algébriques

Semi-anneau. Un semi-anneau est une structure $SM = (S, \oplus, \otimes, \overline{0}, \overline{1})$, où :

— $\oplus$ et $\otimes$ sont deux opérations binaires associatives sur les éléments de S
— $\oplus$ est commutative
— $\otimes$ est distributive à gauche et à droite sur $\oplus$
— $\overline{0}$ est l'élément neutre de $\oplus$ et l'absorbant de $\otimes$
— $\overline{1}$ est l'élément neutre de $\otimes$

Si $\oplus$ est idempotente alors la relation $\preceq_\oplus$ est un ordre partiel sur S, tel que :

$$(a \preceq_\oplus b) \equiv (a = a \oplus b)$$
$$(a \prec_\oplus b) \equiv (a = a \oplus b) \text{ et } (a \neq b)$$

Cet ordre est total si l'opération $\oplus$ est sélective. Le tableau 3.2 suivant résume quelques exemples de semi-anneaux pour des problèmes de routage connus.

Problème	**Structure**	S	$\oplus$	$\otimes$	$\overline{0}$	$\overline{1}$
chemins plus courts	SM_{sp}	$\mathbb{R}^\infty$	min	+	$-\infty$	0
chemins plus larges	SM_{wp}	$\mathbb{R}^\infty$	max	min	$-\infty$	$-\infty$
chemins plus fiables	SM_{rp}	$[0, 1]$	max	$\times$	0	1

TABLE 3.2 – Exemples de semi-anneaux pour les problèmes de routage classiques.

Algèbre de fonctions. Une algèbre avec fonctions est modélisée par la structure algébrique $AF = (S, \oplus, F, \overline{0}, \overline{1})$, où :

— $\oplus$ est une opération binaire associative, commutative et sélective sur les éléments de l'ensemble S

- F est une ensemble de fonctions de $S \to S$
- $\overline{0}$ est l'élément neutre de $\oplus$ et le point fixe pour toute $f \in F,\ f(\overline{0}) = \overline{0}$
- $\overline{1}$ est l'élément absorbant de $\oplus$

Dans le cas d'une opération arbitraire $\otimes$, l'ensemble des fonctions F est défini par :

$$F_{\otimes} = \{f_a(b) = a \otimes b \mid a \in S\}$$

Le tableau 3.3 suivant résume quelques exemples d'algèbres avec fonctions pour des problèmes de routage connus.

Problème	**Structure**	S	$\oplus$	$F_{\otimes}$	$\overline{0}$	$\overline{1}$
chemins plus courts	AF_{sp}	$\mathbb{R}^{\infty}$	min	F_{+}	$+\infty$	$-\infty$
chemins plus larges	AF_{wp}	$\mathbb{R}^{\infty}$	max	F_{min}	$-\infty$	$+\infty$
chemins plus fiables	AF_{rp}	$[0,1]$	max	$F_{\times}$	0	1

TABLE 3.3 – Exemples d'algèbres avec fonctions pour les problèmes de routage classiques.

Algèbre de Sobrinho. Une algèbre de Sobrinho est une structure algébrique pondérée. Elle est modélisée par le septuplet $AS = (W, \leq, S, L, \otimes, \omega, \overline{1})$, où :

- W est un ensemble de poids
- $\leq$ est un ordre total sur W
- S est un ensemble de signatures
- L est un ensemble d'étiquettes
- $\otimes : S \times L \to S$ est une opération binaire associant pour toute paire (signature, étiquette) une signature
- $\omega : S \to W$ est une fonction de poids associant les signatures aux poids
- $\overline{1}$ est la signature absorbante de $\otimes$ avec un poids $\omega(\overline{1})$ maximal, *i.e.,* pour toute signature s dans l'ensemble $S - \{\overline{1}\},\ \omega(s) \leq \omega(\overline{1})$

Le tableau 3.4 suivant résume quelques exemples d'algèbres de Sobrinho pour des problèmes de routage connus.

Problème	**Structure**	W	$\leq$	S	L	$\otimes$	ω	$\overline{1}$
chemins plus courts	AS_{sp}	$\mathbb{R}^{\infty}$	$\leq$	$\mathbb{R}^{\infty}$	$\mathbb{R}$	$+$	$id_{\mathbb{R}^{\infty}}$	$+\infty$
chemins plus larges	AS_{wp}	$\mathbb{R}^{\infty}$	$\geq$	$\mathbb{R}^{\infty}$	$\mathbb{R}$	min	$id_{\mathbb{R}^{\infty}}$	$-\infty$
chemins plus fiables	AS_{rp}	$[0,1]$	$\geq$	$[0,1]$	$[0,1]$	$\times$	$id_{[0,1]}$	0

TABLE 3.4 – Exemples d'algèbres de Sobrino pour les problèmes de routage classiques.

3.2.3 Problème de routage généralisé

Dans cette section, nous définissons le problème de routage généralisé en utilisant la structure semi-anneau. Nous montrons également comment adapter cette définition

aux autres styles d'algèbre de routage. Enfin, nous illustrons par un exemple simple le caclul des plus courts chemins avec le semi-anneau SM_{sp}.

3.2.3.1 Définition du problème

Soit le semi-anneau $(S, \oplus, \otimes, \overline{0}, \overline{1})$, on définit le semi-anneau des matrices carrées de taille n par la structure $\Big(\mathbf{M}_n(S), \oplus, \otimes, \mathbf{N}, \mathbf{I}\Big)$, où les deux matrices $\mathbf{N}$ et $\mathbf{I}$ sont de la forme suivante :

- $\mathbf{N}_{i,j} = \overline{0}$
- $\mathbf{I}_{i,j} = \begin{cases} \overline{1} & \text{si } (i = j) \\ \overline{0} & \text{sinon} \end{cases}$

Et pour toutes matrices $\mathbf{X}, \mathbf{Y} \in \mathbf{M}_n(S)$, les deux opérations $\oplus$ et $\otimes$ sont définies comme suit :

- $(\mathbf{X} \oplus \mathbf{Y})_{i,j} = \mathbf{X}_{i,j} \oplus \mathbf{Y}_{i,j}$
- $(\mathbf{X} \otimes \mathbf{Y})_{i,j} = \bigoplus_{k=1}^{n} \mathbf{X}_{i,k} \otimes \mathbf{Y}_{k,j}$

Par définition, le semi-anneau $\Big(\mathbf{M}_n(S), \oplus, \otimes, \mathbf{N}, \mathbf{I}\Big)$ des matrices carrées de taille n vérifie les mêmes propriétés algébriques du semi-anneau $(S, \oplus, \otimes, \overline{0}, \overline{1})$.

Étant donné un semi-anneau $(S, \oplus, \otimes, \overline{0}, \overline{1})$, un graphe pondéré $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \omega)$ avec n nœuds et une fonction de poids $\omega : \mathcal{E} \to S$ sur les liens de $\mathcal{G}$ tel que $\omega_{i,j}$ est le poids du lien (U_i, U_j). Soit $\mathbf{A} \in \mathbf{M}_n(S)$ la matrice d'adjacence des poids de $\mathcal{G}$ de taille carrée n telle que :

$$\mathbf{A}_{i,j} = \begin{cases} \omega_{i,j} & \text{si } (U_i, U_j) \in \mathcal{E} \\ \overline{0} & \text{sinon} \end{cases}$$

Pour un chemin $\mathcal{P} = U_0, U_1, U_2, ..., U_{k-1}, U_k$, le poids $\omega(\mathcal{P})$ est :

$$\begin{aligned} \omega(\mathcal{P}) &= \mathbf{A}_{0,1} \otimes \mathbf{A}_{1,2} \otimes ... \otimes \mathbf{A}_{k-1,k} \\ &= \omega_{0,1} \otimes \omega_{1,2} \otimes ... \otimes \omega_{k-1,k} = \bigotimes_{i=0}^{k-1} \omega_{i,i+1} \end{aligned}$$

On définit la puissance de la matrice $\mathbf{A} \in \mathbf{M}_n(S)$ récursivement par :

$$\mathbf{A}^k = \begin{cases} \mathbf{I} & \text{si } k = 0 \\ \mathbf{A} \otimes \mathbf{A}^{k-1} & \text{sinon} \end{cases}$$

Soit $\mathfrak{P}_{i,j}^{k}$ est l'ensemble de tous les chemins entre U_i et U_j de longueur exactement k. On note par $\mathfrak{P}_{i,j}^{(k)}$ l'ensemble de tous les chemins entre U_i et U_j de longueur au plus k. On

définit le poids du chemin optimal entre U_i et U_j de longueur au plus k comme suit :

$$\mathbf{A}_{i,j}^{(k)} \;=\; \omega_{i,j}^{(k)} \;=\; \bigoplus_{\mathcal{P}\in\mathfrak{P}_{i,j}^{(k)}} \omega(\mathcal{P})$$

D'où :

$$\mathbf{A}^{(k)} \;=\; \bigoplus_{\mathcal{P}\in\mathfrak{P}^{(k)}} \omega(\mathcal{P})$$

La solution optimale globale consiste à calculer la matrice $\mathbf{A}^*$ (si elle existe) où :

$$\mathbf{A}^* \;=\; \bigoplus_{k\geq 0} \mathbf{A}^{(k)} \;=\; \bigoplus_{\mathcal{P}\in\mathfrak{P}^{(k)}} \omega(\mathcal{P})$$

Dans le cas d'une algèbre avec fonctions, une fonction de poids est définie de $\mathcal{E} \rightarrow F$. Cette fonction est représentée par une matrice d'adjacence $\mathbf{A}$ de taille carrée n où $\mathbf{A}_{i,j} \in F$. Nous définissons cette matrice d'adjacence comme suit :

$$\mathbf{A}_{i,j} = \begin{cases} f_{i,j} & \text{si } (U_i, U_j) \in \mathcal{E} \\ f_{\overline{1}} & \text{sinon} \end{cases}$$

Où : $f_{i,j}$ représente la fonction du lien (U_i, U_j) et $f_{\overline{1}}$ représente la fonction constante pour les liens manqués avec $f_{\overline{1}}(a) = \overline{0}$ pour tout $a \in S$. Nous définissons le calcul global par une matrice $\mathbf{X} \in \mathbb{M}_n(S)$ où $\mathbf{X}_{i,j}$ est le poids optimal de U_i à U_j. On définit maintenant l'application de $\mathbf{A}$ sur $\mathbf{X}$ avec :

$$\mathbf{A}(\mathbf{X}_{i,j}) = \bigoplus_{k} \mathbf{A}_{i,k}(\mathbf{X}_{k,j})$$

Pour l'algèbre de Sobrinho, la fonction de poids est définie de $\mathcal{E} \rightarrow L$. Cette fonction permet de définir la matrice d'adjacence $\mathbf{A}$ de taille $n \times n$ où $\mathbf{A}_{i,j} \in L$. Nous définissons cette matrice d'adjacence comme suit :

$$\mathbf{A}_{i,j} = \begin{cases} l_{i,j} & \text{si } (U_i, U_j) \in \mathcal{E} \\ \overline{1} & \text{sinon} \end{cases}$$

Où : $l_{i,j}$ représente l'étiquette du lien (U_i, U_j) et $\overline{1}$ représente la signature spéciale pour les liens manqués.

3.2.3.2 Exemple d'un problème plus courts chemins

Afin d'expliquer comment calculer la solution d'un problème de calcul de chemins, nous illustrons un exemple d'une solution optimale pour le problème classique des plus

courts chemins en utilisant le semi-anneau $SM_{sp} = (\mathbb{R}^{\infty}, min, +, -\infty, 0)$. Soit le graphe orienté et pondéré $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \omega)$ défini dans la figure 3.1 avec la fonction de poids $\omega : \mathcal{E} \to \mathbb{R}^{\infty}$.

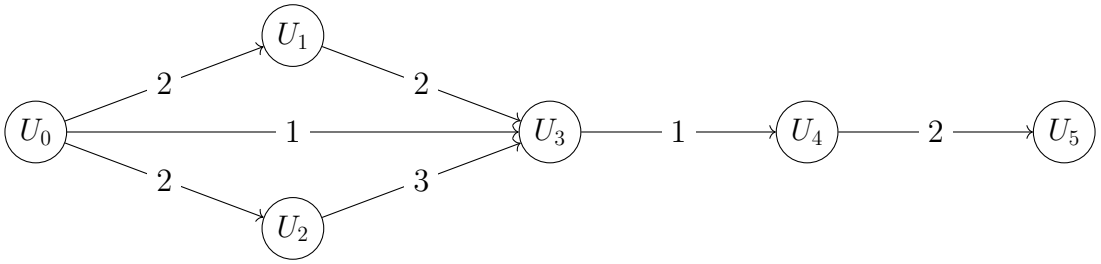


FIGURE 3.1 – Exemple de graphe pour le calcul des plus courts chemins avec le semi-anneau SM_{sp}.

On définit $\mathbf{A}$ la matrice d'adjacence des poids du graphe représenté dans la figure 3.1. Comme le graphe est orienté dans un sens unique, la partie diagonale inférieure de la matrice ne contient que des poids $(-\infty)$ *i.e.,* absence de liens (voir tableau 3.5).

$$\mathbf{A} = \begin{array}{c} \begin{array}{cccccc} U_0 & U_1 & U_2 & U_3 & U_4 & U_5 \end{array} \\ \begin{bmatrix} -\infty & 2 & 2 & 1 & -\infty & -\infty \\ & -\infty & -\infty & 2 & -\infty & -\infty \\ & & -\infty & 3 & -\infty & -\infty \\ & & & -\infty & 1 & -\infty \\ & & & & -\infty & 2 \\ & & & & & -\infty \end{bmatrix} \end{array} \begin{array}{c} U_0 \\ U_1 \\ U_2 \\ U_3 \\ U_4 \\ U_5 \end{array}$$

TABLE 3.5 – Exemple de la matrice d'adjacence $\mathbf{A}$ du graphe représenté dans la figure 3.1.

Dans cet exemple, nous nous intéressons à calculer le plus court chemin entre chaque paire de nœuds, en particulier entre le nœud U_0 et U_5. Pour cela, on calcule la matrice $\mathbf{A}^*$ des poids optimaux en utilisant les opérations min et $+$ du semi-anneau SM_{sp}. Il est déjà connu que dans un graphe sans circuits absorbants (tous les circuits ont des poids positifs), le calcul de la matrice $\mathbf{A}^*$ converge après $(n-1)$ itérations *i.e.,* la longueur maximale d'un chemin élémentaire [12]. Et donc :

$$\mathbf{A}^* = \mathbf{A}^{(n-1)} = \mathbf{A}^{(5)} = \mathbf{I}\ min\ \mathbf{A}\ min\ \mathbf{A}^2\ min\ \mathbf{A}^3\ min\ \mathbf{A}^4\ min\ \mathbf{A}^5$$

D'où : D'après l'élément $\mathbf{A}^*_{0,5}$ dans le tableau 3.6, le plus court chemin entre U_0 et U_5 a un poids de 4, et il croise les nœuds U_3 et U_4. Les deux autres chemins ayant les poids 7

$$\mathbf{A}^* = \begin{array}{c} \begin{array}{cccccc} U_0 & U_1 & U_2 & U_3 & U_4 & U_5 \end{array} \\ \left[\begin{array}{cccccc} 0 & 2 & 2 & 1 & 2 & 4 \\ & 0 & -\infty & 2 & 3 & 5 \\ & & 0 & 3 & 4 & 6 \\ & & & 0 & 1 & 3 \\ & & & & 0 & 2 \\ & & & & & 0 \end{array}\right] \end{array} \begin{array}{c} U_0 \\ U_1 \\ U_2 \\ U_3 \\ U_4 \\ U_5 \end{array}$$

TABLE 3.6 – Exemple de la matrice $\mathbf{A}^*$ des plus courts chemins du graphe représenté dans la figure 3.1.

et 8 en passant par U_1 et U_2 respectivement sont ignorés par le calcul, et ils ne sont donc pas des chemins optimaux.

3.2.4 Notre approche

Rappelons que le problème de routage qu'on cherche à résoudre dans un réseau hétérogène et multicouche est de calculer les plus courts chemins faisables (ou plus généralment valides) entre chaque paire de nœuds du réseau. Afin de construire une structure algèbrique qui couvre les contraintes de ce dernier problème, *i.e.,* la faisabilité et l'optimalité, nous proposons de découper le problème en deux sous-problèmes (chemins plus courts et chemins valides). Pour cela, nous disposons déjà de la structure de chemins plus courts (sp) dans les différents styles et il ne nous reste qu'à définir la structure chemins valides (vp). Cette nouvelle structure doit avoir la capacité d'énumérer tous les ensembles des chemins valides entre chaque paire de nœuds du réseau. Ces chemins peuvent être représentés par des couples de piles de protocoles ou par la composition de fonctions d'adaptation qui rendent le chemin valide. Dans cette situation, il est nécessaire de construire deux opérations sur l'ensemble des chemins valides. La première opération est utilisée pour concaténer les chemins valides (concaténation de piles de protocoles ou composition de fonctions d'adaptation) et la seconde est utilisée pour fusionner les chemins valides (l'union des chemins valides pour chaque paire de nœuds).

Une fois la structure de chemins valides est définie, nous définissons la structure de chemins valides et plus courts (vsp) par un produit semi-direct des deux structures sp et vp. La structure résultante (vsp) doit avoir la capacité d'énumérer tous les ensembles des plus courts chemins valides entre chaque paire de nœuds du réseau. Pour ce faire, l'opération de concaténation est le produit direct de l'addition des poids de la structure (sp) et la concaténation des piles (ou composition de fonctions) de la nouvelle structure (vp). Cette opération permet de calculer la somme des poids d'un chemin sous condition de préserver sa validité. Cependant, la deuxième opération (énumération sous condition)

doit conserver tous les chemins valides. En cas d'égalité, *i.e.,* les chemins valides ayant la même pile de protocoles (tunnel) ou la même composition de fonctions, la seconde opération doit conserver la plus courte. Cette opération est définie par un nouveau produit dit *union-min* qui combine les deux opérations minmum de la structure (sp) et l'opération union de la structure (vp).

3.3 Algèbres de routage dans les réseaux multicouches

Dans cette section, nous définissons d'abord l'opération de composition de fonctions d'adaptation, la concaténation de piles de protocoles et le produit d'opérations union-min. Ensuite, nous définissons trois structures algébriques (semi-anneau, algèbre de fonctions et algèbre de Sobrinho) pour le problème de chemins valides (vp). En se basant sur ça, nous étudions le produit semi-direct des structures chemins valides (vp) avec les structures chemins plus courts (sp) afin de modéliser les plus courts chemins valides (vsp). Les tableaux 3.7,3.10, 3.11 résument toutes les structures du problème de chemins valides proposés dans ce document.

3.3.1 Composition de fonctions et concaténation de piles

Rappelons qu'un chemin est représenté par une suite de fonctions d'adaptation et une suite de piles de protocoles. Afin de définir la concaténation de chemins valides, nous définissons par la suite deux opérations binaires sur les fonctions et sur les piles.

3.3.1.1 Composition de fonctions d'adaptation

Nous avons vu dans la section 1.3.2 que l'application successive de fonctions d'adaptation sur une pile de protocoles de départ induit une pile d'arrivée. Cette application peut être représentée par une composition de fonctions. Plus formellement, soit $\mathcal{H} = \{\phi, x, y, yx, xxx, \dots\}$ l'ensemble de toutes les piles de protocoles possibles. Notez que cet ensemble peut être fini pour un réseau multicouche donné dans lequel le nombre de protocoles dans n'importe quelle pile ne dépasse pas λn^2 (voir section 2.2.2.2). Plus précisement, le nombre de piles de protocoles possibles est donné par :

$$|\mathcal{H}| = 1 + \lambda + \lambda^2 + ... + \lambda^{\lambda n^2} = \frac{1 - \lambda^{\lambda n^2}}{1 - \lambda}$$

Dans ce contexte, on peut définir une fonction d'adaptation élémentaire (*i.e.,* non composée) $f \in \mathcal{F}$ comme une fonction f de $\mathcal{H}_a \to \mathcal{H}_b$ où $\mathcal{H}_a$ (resp. $\mathcal{H}_b$) est un ensemble de piles de protocoles dans $\mathcal{H}$ représentant le domaine (resp. image) de f. Par exemple, si la fonction est la désencapsulation $(xy \to x)$, alors $f : \mathcal{H}_{xy} \to \mathcal{H}_x$ où $\mathcal{H}_{xy}$ (resp. $\mathcal{H}_x$) est l'ensemble de toutes les piles de protocoles commençant par la sous-pile de protocoles xy

(resp. le protocole x). En utilisant cette formalisation, nous pouvons définir l'application successive de deux fonctions d'adaptation par une composition notée $\odot$ de ces deux fonctions.

Soient $f : \mathcal{H}_a \rightarrow \mathcal{H}_b$ et $f' : \mathcal{H}'_a \rightarrow \mathcal{H}'_b$ deux fonctions d'adaptation élémentaires dans $\mathcal{F}$. On définit la composition par la nouvelle fonction $f'' = f' \odot f$ comme suit :

$$f'' = \begin{cases} \mathcal{H}''_a \rightarrow H''_b & \text{si } (\mathcal{H}''_a \neq \emptyset) \text{ et } (\mathcal{H}''_b \neq \emptyset) \\ \{\phi\} \rightarrow \{\phi\} & \text{sinon} \end{cases}$$

Où :

- $\mathcal{H}''_a = \left\{ H \in \mathcal{H}_a \mid f(H) \in \mathcal{H}'_a \right\}$
- $\mathcal{H}''_b = \left\{ f'(H) \mid H \in (\mathcal{H}_b \cap \mathcal{H}'_a) \right\}$

Cette composition $\odot$ est associative et non commutative. La fonction $\{\phi\} \rightarrow \{\phi\}$ est notée par la fonction interdite $(\phi \rightarrow \phi)$. Il est clair que l'ensemble des fonctions d'adaptation élémentaires $\mathcal{F}$ n'est pas clos par composition. Par exemple, si la composition est $(x \rightarrow xy) \odot (y \rightarrow yx)$ alors la fonction composée est $(y \rightarrow yxy)$. Cette nouvelle fonction n'appartient pas à l'ensemble $\mathcal{F}$. Pour cela, on définit un nouvel ensemble $\hat{\mathcal{F}}$ de toutes les fonctions d'adaptation clos par composition. Chaque composition de deux fonctions d'adaptation est représentée par la fonction résultante. L'impossibilité de composition est représentée par la fonction interdite $(\phi \rightarrow \phi)$. Soit $\hat{\mathcal{F}} = \{\phi \rightarrow \phi, x \rightarrow xxx, xyx \rightarrow x, \dots\}$ l'ensemble de toutes les fonctions clos par composition. On dit que deux fonctions $\hat{f}_1$ et $\hat{f}_2$ de $\hat{\mathcal{F}}$ sont équivalentes si et seulement si elles ont le même domaine et la même image. Par exemple, les deux fonctions $(x \rightarrow xx)$ et $(y \rightarrow x) \odot (x \rightarrow xy)$ sont équivalentes et elle peuvent s'appliquer sur les piles de protocoles commençant par x et renvoient les piles de protocoles commençant par xx. Dans la suite du document, on notera par $\mathcal{F}_{id}$ l'ensemble de toutes les fonctions passives, *i.e.,* retransmissions classiques $\{x \rightarrow x, y \rightarrow y, \dots\}$ appelées fonctions d'adaptation identitées.

En utlisant cette nouvelle opération de composition de fonctions, un chemin valide $\mathcal{P} = H_{in} S f_0 U_1 f_1 \dots U_k f_k H_{out} D$ est représenté par la composition valide de ces fonctions d'adaptation $f_k \odot \dots \odot f_1 \odot f_0$. Cette dernière composition est définie par la fonction $(H_{in} \rightarrow H_{out})$ de l'ensemble $\hat{\mathcal{F}}$.

3.3.1.2 Concaténation de piles de protocoles

Dans la section 2.3.3, nous avons défini la fermeture transitive de chemins valides. Cette technique permet de concaténer deux chemins valides en se basant sur leur piles de protocoles de départ et d'arrivée. Dans ce contexte, nous définissons cette technique comme une nouvelle opération binaire notée $\diamond$ sur les couples de piles de protocoles. Plus formellement, soit deux couples de piles de protocoles (H_{in}, H_{out}) et (H'_{in}, H'_{out}) dans $\mathcal{H}^2$. On définit la concaténation des deux couples par le nouveau couple de piles

$(H''_{in}, H''_{out}) = (H_{in}, H_{out}) \diamond (H'_{in}, H'_{out})$ comme suit :

$$(H''_{in}, H''_{out}) = \begin{cases} (H_{in}, H'_{out}) & \text{si } H_{out} = H'_{in} \\ (H.H_{in}, H'_{out}) & \text{si } H'_{in} - H_{out} = H \\ (H_{in}, H.H'_{out}) & \text{si } H_{out} - H'_{in} = H \\ (\phi, \phi) & \text{sinon} \end{cases}$$

Cette opération de concaténation est associative et non commutative. Son élément absorbant est le couple de piles interdites (ϕ, ϕ), *i.e.,* la concaténation avec le couple de piles interdites est toujours le couple de piles interdites. Il faut noter que cette opération n'admet pas d'élément neutre, puisque pour un couple de piles (H_{in}, H_{out}) l'élément neutre doit être de la forme (H_{out}, H_{out}). Ceci est l'élement neutre spécifique à chaque couple de piles et ne peut être commun pour tout couple de piles. Autrement dit, tous les chemins valides n'ont pas la même pile d'arrivée H_{out}.

En utilisant cette nouvelle opération de concaténation de piles, un chemin valide est représenté par la concaténation de couples de piles des liens qui le compose.

3.3.2 Produit union-min

Afin d'énumérer l'ensemble de tous les plus courts chemins valides entre chaque paire de nœuds dans un réseau multicouche, nous définissons le produit union-min des deux opérations union et min sur un ensemble d'éléments pondérés [3]. Nous définissons également la relation d'ordre correspondante à cette nouvelle opération union-min.

3.3.2.1 Opération union-min

Étant donné un ensemble fini d'éléments $S = \{a, b, c \dots\}$. Nous définissons l'ensemble pondéré de S comme le produit d'ensembles $S \times \mathbb{R}^{\infty}$ et nous notons $\mathcal{P}(S \times \mathbb{R}^{\infty})$ son ensemble des parties. Chaque sous-ensemble de l'ensemble des parties contient une paire unique pour chaque élément. Soient S_i et S_j deux sous-ensembles de $\mathcal{P}(S \times \mathbb{R}^{\infty})$. Nous définissons l'opération *union-min* de S_i et S_j comme suit :

$$S_i \underset{min}{\cup} S_j = \left\{ (a, \omega_a) \;\middle|\; (a, \omega_a) \in S_i \wedge \forall\, (b, \omega_b) \in S_j, (a = b) \Rightarrow \omega_a = \omega_a \, min \, \omega_b \right\}$$

Cette nouvelle opération, *i.e.,* union-min, introduit l'idée consistant à énumérer différents chemins pondérés d'une source à une destination. Et dans le cas d'égalité de chemins (avec les mêmes éléments), elle garde le chemin avec le poids le plus courts. Il est facile de vérifier que l'opération union-min est idempotente, associative et commu-

3. Ces éléments peuvent être les piles de protocoles ou les compositions de fonctions d'adaptation. Cela dépend de la nature de la structure algébrique.

tative, mais non sélective (elle peut retourner un nouveau sous-ensemble différent des deux sous-ensembles initiaux). L'élément neutre de cette opération est le sous-ensemble vide et l'élément absorbant est le sous-ensemble d'éléments uniques de poids $(-\infty)$, *i.e.,* $\big\{(a,-\infty),(b,-\infty),(c,-\infty),\dots\big\}$.

3.3.2.2 Relation d'ordre union-min

Soient S_i et S_j deux sous-ensembles d'éléments pondérés de $\mathcal{P}(S\times\mathbb{R}^{\infty})$. Nous définissons la relation d'ordre comme suit :

$$S_i \subseteq_{min} S_j \equiv \forall\,(a,\omega_a)\in S_i \Rightarrow \exists\,(b,\omega_b)\in S_j,\ (a=b)\ \wedge\ (\omega_a\leq\omega_b)$$

Il est clair que cet ordre est un ordre partiel, *i.e.,* il existe des sous-ensembles incomparables. Par exemple, les deux sous-ensembles $\big\{(a,1),(b,5)\big\}$ et $\big\{(a,4),(b,2)\big\}$ sont incomparables.

Dans les sections suivantes, nous utiliserons cette opération afin de définir notre produit semi-direct pour les semi-anneaux et les algèbres de fonctions. Dans le cas de l'algèbre de Sobrinho, l'opération union-min est capturée par la relation d'ordre prédéfinie dans la structure de Sobrinho.

3.3.3 Semi-anneau avec tunnels

Dans cette section, nous définissons deux semi-anneaux (sur l'ensemble de compositions et sur l'ensemble des piles) pour le calcul des chemins valides. Ensuite, nous définissons le produit avec le semi-anneau plus courts chemins afin de modéliser le problème des plus courts chemins valides. Le tableau 3.7 résume ces nouvelles structures.

Problème	**Structure**	S	$\oplus$	$\otimes$	$\overline{0}$	$\overline{1}$
chemins valides	SM_{vp}	$\mathcal{P}(\hat{\mathcal{F}})$	$\cup$	$\odot$	$\emptyset$	$\mathcal{F}_{id}$
chemins valides	SM_{vp}	$\mathcal{P}(\mathcal{H}^2)$	$\cup$	$\diamond$	$\emptyset$	$\{(\epsilon,\epsilon)\}$
chemins valides et plus courts	SM_{vsp}	$\mathcal{P}(\hat{\mathcal{F}}\times\mathbb{R}^{\infty})$	$\underset{min}{\cup}$	$(\odot\times+)$	$\emptyset$	$(\mathcal{F}_{id}\times 0)$
chemins valides et plus courts	SM_{vsp}	$\mathcal{P}(\mathcal{H}^2)$	$\underset{min}{\cup}$	$(\diamond\times+)$	$\emptyset$	$(\{(\epsilon,\epsilon)\}\times 0)$

Table 3.7 – Semi-anneaux pour le problème de routage avec tunnels.

3.3.3.1 Structure sur l'ensemble de fonctions

Rappelons qu'un chemin valide est représenté par une composition valide de fonctions d'adaptation. Afin de calculer l'ensemble de tous les chemins valides, *i.e.,* compositions valides, nous étendons la définition de la composition des fonctions d'adaptation sur les ensembles de compositions.

Soit $\hat{\mathcal{F}}$ l'ensemble de toutes les fonctions d'adaptation clos par composition, et $\mathcal{P}(\hat{\mathcal{F}})$ son ensemble des parties. Si $\hat{F}_1$ et $\hat{F}_2$ sont deux sous-ensembles de $\mathcal{P}(\hat{\mathcal{F}})$, alors nous définissons l'ensemble des compositions par paires :

$$\hat{F}_1 \odot \hat{F}_2 = \left\{ \hat{f}_1 \odot \hat{f}_2 \,|\, \hat{f}_1 \in \hat{F}_1 \text{ et } \hat{f}_2 \in \hat{F}_2 \right\}$$

Notez qu'en cas de composition non valide, la fonction interdite résultante sera retirée de l'ensemble résultant. La composition avec un ensemble vide est toujours un ensemble vide. Sur la base de cette opération, nous définissons notre semi-anneau chemins valides SM_{vp} qui énumère tous les chemins valides comme suit :

$$SM_{vp} = \left(\mathcal{P}(\hat{\mathcal{F}}), \cup, \odot, \emptyset, \mathcal{F}_{id} \right)$$

Où, $\emptyset$ est l'ensemble vide de compositions et $\mathcal{F}_{id}$ est l'ensemble des fonctions d'adaptation identitées $\{x \to x, y \to y, \dots\}$. Il est facile de vérifier que la composition $\odot$ est associative et non commutative avec l'ensemble vide $\emptyset$ comme élément absorbant. Nous vérifions par la suite l'élément neutre avec l'ensemble de fonctions identitées $\mathcal{F}_{id}$ et la distributivité de $\odot$ sur $\cup$.

Élément neutre de $\odot$. Soit $\hat{F} = \{\hat{f}_i, \hat{f}_{i+1}, \dots, \hat{f}_{j-1}, \hat{f}_j\}$ un ensemble de fonctions composées dans $\mathcal{P}(\hat{\mathcal{F}})$. Nous voulons montrer que :

$$\hat{F} \odot \mathcal{F}_{id} = \mathcal{F}_{id} \odot \hat{F} = \hat{F} \quad \forall\, \hat{F} \in \mathcal{P}(\hat{\mathcal{F}})$$

Soit $\hat{f}_k : \mathcal{H}_a \to \mathcal{H}_b$, $i \leq k \leq j$, une fonction dans $\hat{F}$. Il y a deux situations possibles :

1. $\mathcal{H}_a$ est un ensemble de piles de protocoles commençant par x. Dans ce cas, $\hat{f}_k \odot (x \to x) = \hat{f}_k$ et pour toute $f \in \mathcal{F}_{id} - \{(x \to x)\}$ nous avons $\hat{f}_k \odot f = (\phi \to \phi)$.
2. $\mathcal{H}_b$ est un ensemble de piles de protocoles commençant par x. Dans ce cas, $(x \to x) \odot \hat{f}_k = \hat{f}_k$ et pour toute $f \in \mathcal{F}_{id} - \{(x \to x)\}$ nous avons $f \odot \hat{f}_k = (\phi \to \phi)$.

Notez que, par définition de $\odot$, les fonctions interdites sont supprimées de l'ensemble résultant.

Distributivité de $\odot$ sur $\cup$. Soient $\hat{F}_1$, $\hat{F}_2$, $\hat{F}_3$ sont trois ensembles de fonctions composées dans $\mathcal{P}(\hat{\mathcal{F}})$. Par définition de $\odot$ sur les ensembles de fonctions composées, qui calculent l'ensemble des compositions par paires, nous pouvons voir que :

$$\hat{F}_1 \odot (\hat{F}_2 \cup \hat{F}_3) = (\hat{F}_1 \odot \hat{F}_2) \cup (\hat{F}_1 \odot \hat{F}_3)$$

Et :

$$(\hat{F}_2 \cup \hat{F}_3) \odot \hat{F}_1 = (\hat{F}_2 \odot \hat{F}_1) \cup (\hat{F}_3 \odot \hat{F}_1)$$

Maintenant, nous pouvons définir le semi-anneau plus courts chemins valides SM_{vsp}

par le produit semi-direct suivant de SM_{vp} et SM_{sp} :

$$SM_{vsp} = \left(\mathcal{P}\big(\hat{\mathcal{F}} \times \mathbb{R}^{\infty}\big), \underset{min}{\cup}, (\odot \times +), \emptyset, (\mathcal{F}_{id} \times 0)\right)$$

Il est bien connu que le produit direct de semi-groupes conserve la propriété d'associativité, les éléments identités et les absorbants, comme indiqué par [39]. Dans la plupart des situations où les structures algébriques violent certains axiomes des semi-anneaux, elles ne satisfont généralement pas la distributivité de $\otimes$ sur $\oplus$. Nous vérifions cette propriété afin de nous assurer que notre structure définit bien un semi-anneau.

Distributivité du produit $(\odot \times +)$ sur le produit $\underset{min}{\cup}$. Nous voulons montrer que pour toutes $\hat{F}_1,\ \hat{F}_2,\ \hat{F}_3$ ensembles de fonctions composées pondérées dans $\mathcal{P}(\hat{\mathcal{F}} \times \mathbb{R}^{\infty})$ nous avons $lhs = rhs$, où :

$$lhs = \hat{F}_1 \,(\odot \times +)\, (\hat{F}_2 \underset{min}{\cup} \hat{F}_3)$$
$$rhs = (\hat{F}_1 \,(\odot \times +)\, \hat{F}_2) \underset{min}{\cup} (\hat{F}_1 \,(\odot \times +)\, \hat{F}_3)$$

En se basant sur la définition de l'opération union-min, nous distinguons les deux cas suivants :

1. Les deux ensembles $\hat{F}_2$ et $\hat{F}_3$ sont strictement différents et n'ont aucune fonction composée commune :
$$\hat{F}_2 \underset{min}{\cup} \hat{F}_3 = \hat{F}_2 \cup \hat{F}_3$$
Dans ce cas, nous pouvons voir que :
$$lhs = rhs = (F_1 \,(\odot \times +)\, \hat{F}_2) \cup (F_1 \,(\odot \times +)\, \hat{F}_3)$$

2. Les deux ensembles ont certaines fonctions composées communes :
$$\hat{F}_2 \underset{min}{\cup} \hat{F}_3 = \hat{F}_2^* \cup \hat{F}_3^*$$
Où : $\hat{F}_2 = \hat{F}_2^* \cup \hat{F}$ et $\hat{F}_3 = \hat{F}_3^* \cup \hat{F}'$ et $\hat{F}$ (resp. $\hat{F}'$) est l'ensemble non vide des fonctions composées communes et non optimales de $\hat{F}_2$ (resp. $\hat{F}_3$). Dans cette situation, nous avons :
$$\hat{F}_1(\odot \times +)\hat{F}_2 = (\hat{F}_1(\odot \times +)\hat{F}_2^*) \cup (\hat{F}_1(\odot \times +)\hat{F})$$
$$\hat{F}_1(\odot \times +)\hat{F}_3 = (\hat{F}_1(\odot \times +)\hat{F}_3^*) \cup (\hat{F}_1(\odot \times +)\hat{F}')$$
Et :
$$(\hat{F}_1(\odot \times +)\hat{F}_2^*) \underset{min}{\cup} (\hat{F}_1(\odot \times +)\hat{F}') = (\hat{F}_1(\odot \times +)\hat{F}_2^*)$$
$$(\hat{F}_1(\odot \times +)\hat{F}_3^*) \underset{min}{\cup} (\hat{F}_1(\odot \times +)\hat{F}) = (\hat{F}_1(\odot \times +)\hat{F}_3^*)$$

Comme on peut le voir, les deux ensembles $\hat{F}_2^*$ et $\hat{F}_3^*$ sont différents. On peut donc appliquer le cas 1 :

$$lhs = rhs = (F_1 (\odot \times +) \hat{F}_2^*) \cup (F_1 (\odot \times +) \hat{F}_3^*)$$

Il faut noter qu'on peut vérifier la distributivité à droite de la même manière.

3.3.3.2 Structure sur l'ensemble de piles

Une autre façon de définir un semi-anneau sur l'ensemble de piles de protocoles est basée sur l'idée de l'algorithme généralisé de Floyd-Warshall proposé dans la section 2.3.4. Cet algorithme est basé sur l'opération de concaténation des couples de piles de protocoles définie dans la section 3.3.1.2. Or, cette nouvelle opération n'admet pas d'élément neutre. Une solution possible à ce problème est de rajouter une pile vide (ϵ) dans l'ensemble des piles $\mathcal{H}$. Dans ce contexte, le chemin vide représenté par le couple de piles vides (ϵ, ϵ) est l'élément neutre de la concaténation des piles tel que :

$$(H_{in}, H_{out}) \diamond (\epsilon, \epsilon) = (\epsilon, \epsilon) \diamond (H_{in}, H_{out}) = (H_{in}, H_{out}) \qquad \forall (H_{in}, H_{out}) \in \mathcal{H}^2$$

En se basant sur ça, nous pouvons définir un semi-anneau chemins valides SM_{vp} sur l'ensemble de piles :

$$SM_{vp} = \Big(\mathcal{P}(\mathcal{H}^2), \cup, \diamond, \emptyset, \{(\epsilon, \epsilon)\}\Big)$$

Maintenant, nous pouvons définir le semi-anneau SM_{vsp} sur l'ensemble de piles par le produit semi-direct du deux semi-anneaux SM_{vp} et SM_{sp} :

$$SM_{vsp} = \Big(\mathcal{P}\big(\mathcal{H}^2\big), \underset{min}{\cup}, (\diamond \times +), \emptyset, (\{(\epsilon, \epsilon)\} \times 0)\Big)$$

Il faut noter qu'on peut vérifier les propriétés de cette structure de la même manière que le semi-anneau précédent sur l'ensemble des compositions.

3.3.3.3 Exemple d'un problème plus courts chemins valides

Afin d'expliquer comment calculer les plus courts chemins valides avec le semi-anneau SM_{vsp} sur l'ensemble des compositions de fonctions, nous reprenons le même réseau de la figure 3.1 en ajoutant des fonctions d'adaptation sur les nœuds pour créer un réseau multicouche. Ce nouveau réseau est présenté dans la figure 3.2.

Nous définissons maintenant la matrice d'adjacence $\mathbf{A}$ du réseau présenté dans la figure 3.2.

Dans cet exemple, nous nous intéressons au calcul de l'ensemble des plus courts chemins valides, *i.e.,* les compositions valides les plus courtes du nœud U_0 au nœud U_5. Notez que les chemins intermédiaires seront obtenus directement en calculant la matrice

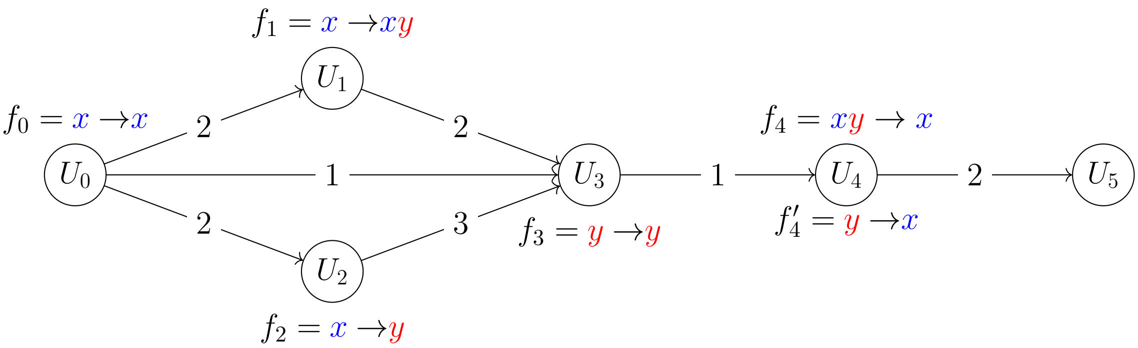


FIGURE 3.2 – Exemple de réseau multicouche pour le calcul des plus courts chemins valides avec le semi-anneau SM_{vsp}.

$$\mathbf{A} = \begin{array}{c} \\ \end{array}\begin{array}{cccccc} U_0 & U_1 & U_2 & U_3 & U_4 & U_5 \\ \end{array}$$

$$\mathbf{A} = \begin{bmatrix} \emptyset & \{(f_0,2)\} & \{(f_0,2)\} & \{(f_0,1)\} & \emptyset & \emptyset \\ & \emptyset & \emptyset & \{(f_1,2)\} & \emptyset & \emptyset \\ & & \emptyset & \{(f_2,3)\} & \emptyset & \emptyset \\ & & & \emptyset & \{(f_3,1)\} & \emptyset \\ & & & & \emptyset & \{(f_4,2),(f'_4,2)\} \\ & & & & & \emptyset \end{bmatrix} \begin{matrix} U_0 \\ U_1 \\ U_2 \\ U_3 \\ U_4 \\ U_5 \end{matrix}$$

TABLE 3.8 – Exemple de la matrice d'adjacence $\mathbf{A}$ du réseau multicouche présenté dans la figure 3.2.

$\mathbf{A}^*$. Pour cela, nous utilisons la formule définie dans la section 3.2.3 pour calculer la matrice $\mathbf{A}^*$. Le résultat final de ce dernier est montré dans le tableau 3.9.

$$\mathbf{A}^* = \begin{bmatrix} \mathcal{F}_{id} \times 0 & \{(f_0,2)\} & \{(f_0,2)\} & \{(f_0,1),(f_1 \odot f_0,4),(f_2 \odot f_0,5)\} & \{(f_3 \odot f_1 \odot f_0,5),(f_3 \odot f_2 \odot f_0,6)\} & \{(f_4 \odot f_3 \odot f_1 \odot f_0,7),(f'_4 \odot f_3 \odot f_1 \odot f_0,7)\} \\ & \mathcal{F}_{id} \times 0 & \emptyset & \{(f_1,2)\} & \{(f_3 \odot f_1,3)\} & \{(f_4 \odot f_3 \odot f_1,5),(f'_4 \odot f_3 \odot f_1,5)\} \\ & & \mathcal{F}_{id} \times 0 & \{(f_2,3)\} & \{(f_3 \odot f_2,4)\} & \{(f_4 \odot f_3 \odot f_2,6),(f'_4 \odot f_3 \odot f_2,6)\} \\ & & & \mathcal{F}_{id} \times 0 & \{(f_3,1)\} & \{(f_4 \odot f_3,3),(f'_4 \odot f_3,3)\} \\ & & & & \mathcal{F}_{id} \times 0 & \{(f_4,2),(f'_4,2)\} \\ & & & & & \mathcal{F}_{id} \times 0 \end{bmatrix} \begin{matrix} U_0 \\ U_1 \\ U_2 \\ U_3 \\ U_4 \\ U_5 \end{matrix}$$

(colonnes : U_0, U_1, U_2, U_3, U_4, U_5)

TABLE 3.9 – Exemple de la matrice $\mathbf{A}^*$ des plus courts chemins valides du réseau multicouche représenté dans la figure 3.2.

3.3.4 Algèbre de fonctions avec tunnels

Dans cette section, nous définissons des algèbres avec fonctions (sur l'ensemble de compositions et sur l'ensemble des piles) pour le calcul des chemins valides. Ensuite, nous définissons le produit avec l'algèbre de fonctions plus courts chemins afin de modéliser le problème plus courts chemins valides. Le tableau 3.10 résume ces nouvelles structures.

Problème	**Structure**	S	$\oplus$	$F_{\otimes}$	$\overline{0}$	$\overline{1}$
chemins valides	AF_{vp}	$\mathcal{P}(\hat{\mathcal{F}})$	$\cup$	$\mathcal{P}(F_{\odot})$	$\emptyset$	$\hat{\mathcal{F}}$
chemins valides	AF_{vp}	$\mathcal{P}(\mathcal{H}^2)$	$\cup$	$\mathcal{P}(F_{\diamond})$	$\emptyset$	$\mathcal{H}^2$
chemins valides	AF_{vp}	$\mathcal{P}(\mathcal{H})$	$\cup$	$\mathcal{P}(\mathcal{F})$	$\emptyset$	$\mathcal{H}$
chemins valides et plus courts	AF_{vsp}	$\mathcal{P}(\hat{\mathcal{F}} \times \mathbb{R}^{\infty})$	$\underset{min}{\cup}$	$\mathcal{P}(F_{\odot} \times F_{+})$	$\emptyset$	$(\hat{\mathcal{F}} \times -\infty)$
chemins valides et plus courts	AF_{vsp}	$\mathcal{P}(\mathcal{H}^2 \times \mathbb{R}^{\infty})$	$\underset{min}{\cup}$	$\mathcal{P}(F_{\diamond} \times F_{+})$	$\emptyset$	$(\mathcal{H}^2 \times -\infty)$
chemins valides et plus courts	AF_{vsp}	$\mathcal{P}(\mathcal{H} \times \mathbb{R}^{\infty})$	$\underset{min}{\cup}$	$\mathcal{P}(\mathcal{F} \times F_{+})$	$\emptyset$	$(\mathcal{H} \times -\infty)$

TABLE 3.10 – Algèbres avec fonctions pour le problème de routage avec tunnels.

3.3.4.1 Structure sur l'ensemble de fonctions

Afin de définir une algèbre de fonctions dans laquelle les plus courts chemins valides sont représentés par les compositions valides et plus courtes de fonctions d'adaptation, nous pouvons directement transformer le semi-anneau sur l'ensemble des compositions de fonctions SM_{vp} et obtenir la structure suivante :

$$AF_{vp} = \left(\mathcal{P}\left(\hat{\mathcal{F}}\right), \cup, \mathcal{P}(F_{\odot}), \emptyset, \hat{\mathcal{F}}\right)$$

Maintenant, nous définissons l'algèbre avec fonctions plus courts chemins valides AF_{vsp} par le produit semi-direct de AF_{vp} et AF_{sp} comme suit :

$$AF_{vsp} = \left(\mathcal{P}\left(\hat{\mathcal{F}} \times \mathbb{R}^{\infty}\right), \underset{min}{\cup}, \mathcal{P}(F_{\odot} \times F_{+}), \emptyset, (\hat{\mathcal{F}} \times -\infty)\right)$$

Où $(F_{\odot} \times F_{+})$ est l'ensemble des fonctions qui s'appliquent sur les compositions pondérées définies par l'ensemble produit :

$$\left\{\left(\hat{f}_a, f_{\omega_a}\right)(b, \omega_b) \;=\; (a \odot b, \omega_a + \omega_b), \mid b \in \hat{\mathcal{F}}, \omega_b \in \mathbb{R}^{\infty}\right\}$$

Notez que l'application d'une fonction $(\hat{f}_a, f_{\omega_a})$ sur l'ensemble vide de compositions pondérées est l'ensemble vide de compositions pondérées. Cela définit un point fixe pour toutes les fonctions de l'ensemble des parties. L'ensemble de toutes les compositions de poids $(-\infty)$ définit l'ensemble absorbant pour l'opération union-min.

3.3.4.2 Structure sur l'ensemble de piles

Nous définissons ici une algèbre sur les piles de protocoles basée sur la concaténation de chemins valides. Pour cela, on peut transformer directement le semi-anneau sur l'ensemble des couples de piles SM_{vp} et obtenir la structure suivante :

$$AF_{vp} = \Big(\mathcal{P}\big(\mathcal{H}^2\big), \cup, \mathcal{P}(F_\diamond), \emptyset, \mathcal{H}^2\Big)$$

En utilisant le produit semi-direct avec la structure AF_{sp} on obtient :

$$AF_{vsp} = \Big(\mathcal{P}\big(\mathcal{H}^2 \times \mathbb{R}^\infty\big), \underset{min}{\cup}, \mathcal{P}(F_\diamond \times F_+), \emptyset, (\mathcal{H}^2 \times -\infty)\Big)$$

Où $(F_\diamond \times F_+)$ est l'ensemble des fonctions qui s'appliquent sur les couples de piles de protocoles pondérées définies par l'ensemble de produit :

$$\Big\{\big(f_H, f_\omega\big)(H, \omega) \; = \; (H \diamond H', \omega + \omega'_H), \; \mid H' \in \mathcal{H}^2, \omega'_H \in \mathbb{R}^\infty\Big\}$$

Notez que l'application d'une fonction (f_H, f_ω) sur l'ensemble vide de couples de piles de protocoles pondérées donne l'ensemble vide. Cela définit un point fixe pour toutes les fonctions de l'ensemble des parties. L'ensemble de toutes les paires de piles de protocoles avec un poids $(-\infty)$ définit l'ensemble absorbant pour l'opération union-min.

Une autre façon pour définir une algèbre sur les piles de protocoles est la généralisation directe de l'algorithme Stack-Vector [59]. Rappelons que nos fonctions d'adaptation sont définies de $\mathcal{H} \to \mathcal{H}$, nous pouvons donc définir une algèbre de fonctions sur l'ensemble de toutes les piles de protocoles $\mathcal{H}$. Pour cela, nous définissons l'application d'un ensemble de fonctions d'adaptation sur un ensemble de piles de protocoles.

Soit $\mathcal{F}' = \{f_1, ..., f_\ell\}$ un ensemble de fonctions d'adaptation dans $\mathcal{P}(\mathcal{F})$ et $\mathcal{H}' = \{H_1, ..., H_k\}$ un ensemble de piles de protocoles dans $\mathcal{P}(\mathcal{H})$. On définit l'application de $\mathcal{F}'$ sur $\mathcal{H}'$:

$$\mathcal{F}'(\mathcal{H}') = \Big\{f_i(H_j) \mid f_i \in \mathcal{F}' \text{ et } H_j \in \mathcal{H}'\Big\}$$

Plus particulièrement, si l'ensemble $\mathcal{H}'$ ou $\mathcal{F}'$ est l'ensemble vide, alors $\mathcal{F}'(\mathcal{H}')$ est aussi l'ensemble vide. On définit l'algèbre AF_{vp} comme suit :

$$AF_{vp} = \Big(\mathcal{P}(\mathcal{H}), \cup, \mathcal{P}(\mathcal{F}), \emptyset, \mathcal{H}\Big)$$

En utilisant le produit semi-direct avec l'algèbre AF_{sp} on obtient :

$$AF_{vsp} = \Big(\mathcal{P}\big(\mathcal{H} \times \mathbb{R}^\infty\big), \underset{min}{\cup}, \mathcal{P}(\mathcal{F} \times F_+), \emptyset, (\mathcal{H} \times -\infty)\Big)$$

Où $(\mathcal{F} \times F_+)$ est l'ensemble des fonctions qui s'appliquent aux piles de protocoles pon-

dérées définies par l'ensemble de produit :

$$\Big\{\big(f, f_\omega\big)(H, \omega_H) \;=\; (f(H), \omega + \omega_H),\; \mid H \in \mathcal{H}, \omega_H \in \mathbb{R}^\infty\Big\}$$

Notez que l'application d'une fonction (f, f_ω) sur l'ensemble vide de piles de protocoles pondérées est l'ensemble vide de piles de protocoles pondérées. Cela définit un point fixe pour toutes les fonctions de l'ensemble des parties. L'ensemble de toutes les piles de protocoles avec un poids $(-\infty)$ définit l'ensemble absorbant pour l'opération union-min.

La violation la plus courante parmi toutes nos algèbres avec fonctions est qu'elles manquent de sélectivité, ce qui est lié à la nature de l'opération union-min.

3.3.5 Algèbre de Sobrinho avec tunnels

Dans cette section, nous définissons des algèbres de Sobrinho (sur l'ensemble de compositions et sur l'ensemble des piles) pour le calcul des chemins valides. Ensuite, nous définissons le produit avec l'algèbre de Sobrinho plus courts chemins afin de modéliser le problème plus courts chemins valides. Le tableau 3.11 résume ces nouvelles structures.

Problème	**Structure**	W	$\leq$	S	L	$\otimes$	ω	$\overline{1}$
chemins valides	AS_{vp}	$\mathcal{P}(\hat{\mathcal{F}})$	$\subseteq$	$\mathcal{P}(\hat{\mathcal{F}})$	$\mathcal{P}(\mathcal{F})$	$\odot$	$\mathrm{id}_{\mathcal{P}(\hat{\mathcal{F}})}$	$\emptyset$
chemins valides	AS_{vp}	$\mathcal{P}(\mathcal{H})$	$\subseteq$	$\mathcal{P}(\mathcal{H})$	$\mathcal{P}(\mathcal{F})$	$\circledast$	$\mathrm{id}_{\mathcal{P}(\mathcal{H})}$	$\emptyset$
chemins valides	AS_{vp}	$\mathcal{P}(\mathcal{H}^2)$	$\subseteq$	$\mathcal{P}(\mathcal{H}^2)$	$\mathcal{P}(\mathcal{H}^2)$	$\diamond$	$\mathrm{id}_{\mathcal{P}(\mathcal{H}^2)}$	$\emptyset$
chemins valides et plus courts	AS_{vsp}	$\mathcal{P}(\mathbb{R}^\infty)$	$\subseteq$	$\mathcal{P}(\mathcal{H})$	$\mathcal{P}(\mathcal{F})$	$\circledast$	$\mathcal{P}(\mathcal{H}) \to \mathcal{P}(\mathbb{R}^\infty)$	$\emptyset$
chemins valides et plus courts	AS_{vsp}	$\mathcal{P}(\mathbb{R}^\infty)$	$\subseteq$	$\mathcal{P}(\mathcal{H}^2)$	$\mathcal{P}(\mathcal{H}^2)$	$\diamond$	$\mathcal{P}(\mathcal{H}^2) \to \mathcal{P}(\mathbb{R}^\infty)$	$\emptyset$
chemins valides et plus courts	AS_{vsp}	$\mathcal{P}(\mathbb{R}^\infty)$	$\subseteq$	$\mathcal{P}(\hat{\mathcal{F}})$	$\mathcal{P}(\mathcal{F})$	$\odot$	$\mathcal{P}(\hat{\mathcal{F}}) \to \mathcal{P}(\mathbb{R}^\infty)$	$\emptyset$
chemins valides et plus courts	AS_{vsp}	$\mathcal{P}(\hat{\mathcal{F}} \times \mathbb{R}^\infty)$	$\subseteq_{min}$	$\mathcal{P}(\hat{\mathcal{F}} \times \mathbb{R}^\infty)$	$\mathcal{P}(\mathcal{F} \times \mathbb{R})$	$(\odot \times +)$	$\mathrm{id}_{\mathcal{P}(\hat{\mathcal{F}} \times \mathbb{R}^\infty)}$	$\emptyset$
chemins valides et plus courts	AS_{vsp}	$\mathcal{P}(\mathcal{H} \times \mathbb{R}^\infty)$	$\subseteq_{min}$	$\mathcal{P}(\mathcal{H} \times \mathbb{R}^\infty)$	$\mathcal{P}(\mathcal{F} \times \mathbb{R})$	$(\circledast \times +)$	$\mathrm{id}_{\mathcal{P}(\mathcal{H} \times \mathbb{R}^\infty)}$	$\emptyset$
chemins valides et plus courts	AS_{vsp}	$\mathcal{P}(\mathcal{H}^2 \times \mathbb{R}^\infty)$	$\subseteq_{min}$	$\mathcal{P}(\mathcal{H}^2 \times \mathbb{R}^\infty)$	$\mathcal{P}(\mathcal{H}^2 \times \mathbb{R})$	$(\diamond \times +)$	$\mathrm{id}_{\mathcal{P}(\mathcal{H}^2 \times \mathbb{R}^\infty)}$	$\emptyset$

Table 3.11 – Algèbres de Sobrino pour le problème de routage avec tunnels.

3.3.5.1 Structure sur l'ensemble de fonctions

Afin de modéliser le problème de chemins valides avec une algèbre de Sobrinho sur l'ensemble des compositions de fonctions, nous définissons la structure suivante :

$$AS_{vp} = \Big(\mathcal{P}(\hat{\mathcal{F}}), \subseteq, \mathcal{P}(\mathcal{F}), \mathcal{P}(\hat{\mathcal{F}}), \emptyset, \odot, \mathrm{id}_{\mathcal{P}(\hat{\mathcal{F}})}\Big)$$

Les signatures sont les ensembles de compositions valides de fonctions d'adaptation et les étiquettes sont définies comme des ensembles de fonctions d'adaptation. L'opération d'extension est la composition d'ensembles de compositions. La signature spéciale est l'ensemble vide de compositions et la fonction de poids est la fonction d'identité de l'ensemble des parties de compositions. La relation d'ordre est donnée par l'ordre partiel $\subseteq$ défini sur des sous-ensembles de compositions. Notez que les deux ensembles $\{x \to x\}$

et $\{x \to y\}$ sont incomparables. En utilisant cette algèbre, nous pouvons définir l'algèbre de Sobrinho AS_{vsp} pour le problème plus courts chemins valides comme le produit semi-direct de AS_{vp} et AS_{sp} :

$$AS_{vsp} = \Big(\mathcal{P}\big(\hat{\mathcal{F}} \times \mathbb{R}^{\infty}\big), \subseteq_{min}, \mathcal{P}\big(\mathcal{F} \times \mathbb{R}\big), \mathcal{P}\big(\hat{\mathcal{F}} \times \mathbb{R}^{\infty}\big), \emptyset, (\odot \times +), \mathrm{id}_{\mathcal{P}(\hat{\mathcal{F}} \times \mathbb{R}^{\infty})}\Big)$$

L'ensemble des signatures est l'ensemble des parties de toutes les fonctions de composition pondérées et l'ensemble des étiquettes est l'ensemble des parties de toutes les fonctions d'adaptation pondérées. Notez que la relation d'ordre est définie sur des ensembles de compositions pondérées. Cette relation d'ordre est la même que l'ordre partiel défini dans la section 3.3.2.2 pour l'opération union-min.

Une autre idée serait de définir directement une algèbre de Sobrinho avec une fonction de poids spécifique, qui associe un ensemble de compositions à un ensemble de poids. Nous définissons cette algèbre comme suit :

$$AS_{vsp} = \Big(\mathcal{P}(\mathbb{R}^{\infty}), \subseteq, \mathcal{P}(\mathcal{F}), \mathcal{P}(\hat{\mathcal{F}}), \emptyset, \odot, \omega\Big)$$

Où $\omega : \mathcal{P}(\hat{\mathcal{F}}) \to \mathcal{P}(\mathbb{R}^{\infty})$ est la fonction de poids.

3.3.5.2 Structure sur l'ensemble de piles

Rappelons que l'application de fonctions d'adaptation sur un ensemble de piles de protocoles donne un nouvel ensemble de piles. Cette application peut être utilisée pour définir une algèbre de Sobrinho sur des ensembles de piles de protocoles, avec une opération spécifique pour modéliser l'application de fonctions d'adaptation sur des piles de protocoles. Nous définissons cette algèbre comme suit :

$$AS_{vp} = \Big(\mathcal{P}(\mathcal{H}), \subseteq, \mathcal{P}(\mathcal{F}), \mathcal{P}(\mathcal{H}), \emptyset, \circledast, \mathrm{id}_{\mathcal{P}(\mathcal{H})}\Big)$$

L'opération $\circledast : \mathcal{P}(\mathcal{H}) \times \mathcal{P}(\mathcal{F}) \to \mathcal{P}(\mathcal{H})$ est l'opération binaire qui modélise l'application d'un ensemble de fonctions d'adaptation sur un ensemble de piles de protocoles. La fonction de poids ω est la fonction d'identité de l'ensemble des parties de toutes les piles de protocoles. En utilisant cette algèbre, nous définissons l'algèbre plus courts chemins valides par le produit direct suivant :

$$AS_{vsp} = \Big(\mathcal{P}\big(\mathcal{H} \times \mathbb{R}^{\infty}\big), \subseteq_{min}, \mathcal{P}\big(\mathcal{F} \times \mathbb{R}\big), \mathcal{P}\big(\mathcal{H} \times \mathbb{R}^{\infty}\big), \emptyset, (\circledast \times +), \mathrm{id}_{\mathcal{P}(\mathcal{H} \times \mathbb{R}^{\infty})}\Big)$$

L'ensemble des signatures est l'ensemble des parties de toutes les piles de protocoles pondérées et l'ensemble des étiquettes est l'ensemble des parties de toutes les fonctions d'adaptation pondérées. Notez que la relation d'ordre est définie par un ordre partiel sur les ensembles de piles de protocoles pondérées.

Une autre possibilité consiste à définir directement une algèbre de Sobrinho avec une fonction de poids spécifique, qui associe un ensemble de piles de protocoles à un ensemble de poids. Nous définissons cette algèbre comme suit :

$$AS_{vsp} = \Big(\mathcal{P}(\mathbb{R}^{\infty}), \subseteq, \mathcal{P}(\mathcal{F}), \mathcal{P}(\mathcal{H}), \emptyset, \circledast, \omega\Big)$$

Où $\omega : \mathcal{P}(\mathcal{H}) \to \mathcal{P}(\mathbb{R}^{\infty})$ est la fonction de poids.

Dans la section 3.3.1.2, nous avons défini une opération de concaténation binaire de paires de piles de protocoles afin de modéliser l'opération de fermeture transitive des chemins valides. En utilisant cette opération, nous pouvons définir une algèbre de Sobrinho sur les ensembles de toutes les couples de piles de protocoles. Cette algèbre est définie comme suit :

$$AS_{vp} = \Big(\mathcal{P}(\mathcal{H}^2), \subseteq, \mathcal{P}(\mathcal{H}^2), \mathcal{P}(\mathcal{H}^2), \emptyset, \diamond, \mathrm{id}_{\mathcal{P}(\mathcal{H}^2)}\Big)$$

On définit maintenant l'algèbre de Sobrinho AS_{vsp} pour le problème plus courts chemins valides sur les couples de piles de protocoles par le produit direct suivant :

$$AS_{vsp} = \Big(\mathcal{P}\big(\mathcal{H}^2 \times \mathbb{R}^{\infty}\big), \subseteq_{min}, \mathcal{P}\big(\mathcal{H}^2 \times \mathbb{R}\big), \mathcal{P}\big(\mathcal{H}^2 \times \mathbb{R}^{\infty}\big), \emptyset, (\diamond \times +), \mathrm{id}_{\mathcal{P}(\mathcal{H}^2 \times \mathbb{R}^{\infty})}\Big)$$

L'ensemble de signatures et l'ensemble d'étiquettes sont l'ensemble des parties de toutes les couples pondérées de piles de protocoles, et la relation d'ordre est définie par un ordre partiel sur les ensembles de couples pondérées de piles de protocoles.

Une autre idée serait de définir directement une algèbre de Sobrinho avec une fonction de poids spécifique, qui associe un ensemble de couples de piles de protocoles à un ensemble de poids. Nous définissons cette algèbre comme suit :

$$AS_{vsp} = \Big(\mathcal{P}(\mathbb{R}^{\infty}), \subseteq, \mathcal{P}(\mathcal{H}^2), \mathcal{P}(\mathcal{H}^2), \emptyset, \diamond, \omega\Big)$$

Où $\omega : \mathcal{P}(\mathcal{H}^2) \to \mathcal{P}(\mathbb{R}^{\infty})$ est la fonction de poids.

La violation la plus courante parmi toutes nos algèbres de Sobrinho est que la relation d'ordre n'est pas totale, car elle est définie sur des sous-ensembles de l'ensemble des parties.

3.4 Propriétés de convergence des algèbres de routage

Dans cette section, nous montrons quelques propriétés de convergence sur les algèbres de routage proposées dans la section précédente. Ensuite, nous prouvons la convergence itérative de la matrice $\mathbf{A}^*$ en utilisant un point fixe sur la longueur des plus courts chemins valides.

3.4.1 Propriétés de monotonie et isotonie

En général, la convergence des protocoles de routage (*ex.,* protocoles de vecteur de distance et de vecteur de chemin) est basée sur les propriétés de monotonie et isotonie des algèbres de routage correspondantes [101, 100].

3.4.1.1 Monotonie

La monotonie garantit que le protocole de routage converge dans n'importe quel réseau, mais pas nécessairement vers une solution optimale globale, comme le prouve [100]. Plus formellement, soit $\otimes$ l'opération d'extension de chemins avec la relation d'ordre $\preceq_{\oplus}$. On dit que l'opération $\otimes$ est *monotone* si et seulement si,

$$a \preceq_{\oplus} a \otimes b \equiv a = a \oplus (a \otimes b) \, \forall \, a, b \in S$$

Nous pouvons voir que toutes les algèbres proposées dans ce chapitre ne satisfont pas cette propriété. La composition de deux ensembles peut donner un nouvel ensemble (éventuellement l'ensemble vide) qui est incomparable aux premiers ensembles. Et l'application d'un ensemble de fonctions d'adaptation sur un ensemble de piles de protocoles peut donner un nouvel ensemble de piles de protocoles (éventuellement l'ensemble vide) qui est incomparable au premier ensemble. Pour cela, nous prouvons la proposition suivante :

Proposition 2. *Le produit direct des opérations* $(\odot \times +)$ *sur l'ensemble des parties* $\mathcal{P}\left(\hat{\mathcal{F}} \times \mathbb{R}^{\infty}\right)$ *n'est pas monotone.*

Démonstration. Nous démontrons la proposition en utilisant le contre-exemple de composition suivant,

$$\Big\{(x \to xyx, 3), \, (y \to x, 2)\Big\} \, (\odot \times +) \, \Big\{(x \to xy, 1)\Big\} = \Big\{(x \to xx, 3)\Big\}$$

On voit que les deux ensembles $\{(x \to xyx, 3), \, (y \to x, 2)\}$ et $\{(x \to xx, 3)\}$ sont incomparables en utilisant la relation d'ordre partiel $\subseteq_{min}$ définie dans la section 3.3.2.2. □

3.4.1.2 Isotonie

L'isotonie garantit que le protocole de routage converge vers une solution globale optimale, comme le prouve [100]. Plus formellement, soit $\otimes$ l'opération d'extension de chemins avec la relation d'ordre $\preceq_{\oplus}$. On dit que l'opération $\otimes$ est *isotone* si et seulement si :

$$a \preceq_{\oplus} b \implies a \otimes c \preceq_{\oplus} b \otimes c \, \forall \, a, b, c \in S$$

Afin de montrer que nos algèbres sont isotones, nous prouvons la proposition suivante pour les semi-anneaux avec composition. La même preuve peut être adaptée aux autres styles d'algèbres.

Proposition 3. *Le produit direct des opérations* $(\odot \times +)$ *sur l'ensemble des parties* $\mathcal{P}\big(\hat{\mathcal{F}} \times \mathbb{R}^{\infty}\big)$ *est isotone.*

Démonstration. Nous démontrons la proposition par contradiction. Supposons qu'il existe trois ensembles de fonctions composées pondérées $\hat{F}_1$, $\hat{F}_2$ et $\hat{F}_3$, et que la propriété suivante est vraie :

$$\Big(\hat{F}_1 \subseteq_{min} \hat{F}_2\Big) \wedge \Big(\hat{F}_1(\odot \times +)\hat{F}_3 \not\subseteq_{min} \hat{F}_2(\odot \times +)\hat{F}_3\Big)$$

Par la définition de la relation d'ordre de l'opération union-min :

$$\hat{F}_1 \subseteq_{min} \hat{F}_2 \equiv \exists \hat{F},\ (\hat{F}_2 = \hat{F}_1^* \cup \hat{F}) \wedge (\hat{F}_1 \underset{min}{\cup} \hat{F}_1^* = \hat{F}_1)$$

Maintenant, nous calculons la composition de $\hat{F}_2$ et $\hat{F}_3$:

$$\hat{F}_2(\odot \times +)\hat{F}_3 = \big(\hat{F}_1^* \cup \hat{F}\big)(\odot \times +)\hat{F}_3 = \big(\hat{F}_1^*(\odot \times +)\hat{F}_3\big) \cup \big(\hat{F}(\odot \times +)\hat{F}_3\big)$$

Donc, nous pouvons voir que :

$$\big(\hat{F}_1(\odot \times +)\hat{F}_3\big) \subseteq_{min} \big(\hat{F}_1^*(\odot \times +)\hat{F}_3\big) \cup \big(\hat{F}(\odot \times +)\hat{F}_3\big)$$

Par conséquent, les ensembles $\hat{F}_1$, $\hat{F}_2$ et $\hat{F}_3$ ne peuvent pas exister. □

3.4.2 Thèorème de convergence itérative

La violation de monotonie signifie qu'il est possible que notre algèbre ne converge pas dans tout réseau multicouche. Pour cette raison, nous prouvons la convergence de notre algèbre par la présence d'un point fixe dans tout réseau mutlicouche. Dans ce contexte, il est déjà connu que dans les réseaux classiques sans circuits absorbants (avec uniquement des poids positifs) appelés réseaux libres, la solution optimale globale $\mathbf{A}^*$ converge vers la matrice $\mathbf{A}^{(n-1)}$ où $(n-1)$ est la longueur maximale d'un chemin élémentaire, comme le prouve [12, 33]. Cela signifie que les circuits ne font qu'augmenter le poids des chemins et seront donc ignorés par le calcul. Dans notre cas, ce n'est pas la même situation, et comme nous l'avons vu dans le modèle multicouche, des circuits sont autorisés et parfois nécessaires pour certains chemins, *i.e.,* construisant la pile de protocoles nécessaire pour que le chemin soit valide. Pour cette raison, nous définissons d'abord le circuit valide et le chemin valide élémentaire dans le modèle de réseau multicouche afin de généraliser le théorème de convergence cité ci-dessus.

3.4.2.1 Circuit valide et chemin valide élémentaire

En théorie des graphes, un circuit est un chemin qui commence et qui se termine sur le même nœud. Dans notre cas, les chemins sont composés de piles de protocoles en plus des nœuds et des liens. Pour cela, on définit un circuit valide comme étant un circuit classique dont la pile de protocoles de départ et la pile protocoles d'arrivée sont identiques. Plus formellement, un chemin valide $\mathcal{P} = H_{in}Sf_0U_1f_1 \dots U_kf_kH_{out}D$ est un *circuit valide* si et seulement si :

- Le nœud S est le même que le nœud D, *i.e.*, $S = D$
- La pile de départ H_{in} et la pile d'arrivée H_{out} sont les mêmes, *i.e.*, $H_{in} = H_{out}$

En utilisant la définition ci-dessus, nous définissons maintenant un *chemin valide élémentaire* comme étant un chemin valide dans lequel ses circuits (s'ils existent) ne sont pas des circuits valides. Autrement dit, il ne contient que des circuits commençant et se terminant par une pile de protocoles différente.

Un réseau multicouche est dit *libre* si tous ses circuits multicouches ont des poids positifs.

3.4.2.2 Existence d'un point fixe

Nous avons vu dans la section 2.2.2.2 que la longueur d'un plus court chemin faisable (ou plus généralement valide) peut être exponentielle en fonction du nombre de protocoles, et polynomial en fonction du nombre de nœuds. Plus précesiment, cette longueur ne dépasse jamais $2^{(\lambda+1)\lambda^2n^2}$ dans n'importe quel réseau multicouhe. En utilisant ce résultat, nous prouvons le théorème suivant.

Théorème 1. *Dans un réseau multicouche libre $\mathcal{N}$ nous avons :*

$$\mathbf{A}^* = \mathbf{A}^{(k)} = \mathbf{I} \oplus \mathbf{A} \oplus \mathbf{A}^2 \oplus \cdots \oplus \mathbf{A}^k$$

Où k est la longueur maximale des chemins valides élémentaires dans $\mathcal{N}$, et elle est égale à $2^{(\lambda+1)\lambda^2n^2} - 1$.

Démonstration. Soit $\mathcal{N}$ un réseau multicouche libre avec des chemins élémentaires de longueur au plus k, $k \leq 2^{(\lambda+1)\lambda^2n^2} - 1$. Supposons qu'il existe un entier $\ell > 0$, où $(k+\ell)$ est un point fixe pour $\mathbf{A}^*$:

$$\mathbf{A}^* = \mathbf{A}^{(k+\ell)} = \mathbf{I} \oplus \mathbf{A} \oplus \cdots \oplus \mathbf{A}^k \oplus \mathbf{A}^{k+1} \oplus \cdots \oplus \mathbf{A}^{k+\ell}$$

Soit $\mathbf{A}_{i,j}^{(k+\ell)}$ l'ensemble des fonctions composées pondérées valides du nœud U_i au nœud U_j de taille au plus $(k+\ell)$, *i.e.*, avec au plus $(k+\ell)$ fonctions d'adaptation. Supposons qu'il existe un chemin valide représenté par une fonction composée pondérée $(\hat{f}, \omega) \in \mathbf{A}_{i,j}^{p}$

de p fonctions d'adaptation telles que $k+1 \leq p \leq k+\ell$. On dit que $(\hat{f}, \omega)$ est optimale si et seulement si l'une des conditions suivantes est vraie :

- **Hypothèse 1 :** Supposons qu'il n'y ait pas de fonction composée pondérée $(\hat{f}', \omega') \in \mathbf{A}_{i,j}^{q}$ avec $1 \leq q \leq k$ telle que $\hat{f}' = \hat{f}$.
  Cependant, d'après la longueur maximale des chemins valides élémentaires, s'il existe un plus court chemin valide avec au moins $(k+1)$ fonctions d'adaptation, alors il existe un autre plus court chemin valide avec au plus k fonctions d'adaptation. Plus précisément, il existe une fonction composée pondérée $(\hat{f}', \omega') \in \mathbf{A}_{i,j}^{q}$, tels que $\hat{f}' = \hat{f}$ et $\omega' \leq \omega$. Et par la définition de l'opération $\oplus$ (union-min), la fonction composée $\hat{f}$ est ignorée.

  $$\big\{(\hat{f}, \omega), \dots\big\} \underset{min}{\cup} \big\{(\hat{f}', \omega'), \dots\big\} = \big\{(\hat{f}', \omega'), \dots\big\}$$

  Par conséquent, le chemin valide représenté par $\hat{f}$ n'est pas élémentaire et il contient un circuit valide.
- **Hypothèse 2 :** Supposons que pour toutes les fonctions composées pondérées $(\hat{f}', \omega') \in \mathbf{A}_{i,j}^{q}$ de q fonctions d'adaptation avec $1 \leq q \leq k$ tels que $\hat{f}' = \hat{f}$ et $\omega' > \omega$.
  Cependant, d'après la longueur maximale des chemins valides élémentaires, s'il existe un plus court chemin valide avec au moins $(k+1)$ fonctions d'adaptation, alors il existe un autre plus court chemin valide avec au plus k fonctions d'adaptation. Plus précisément, il existe une fonction composée pondérée $(\hat{f}', \omega') \in \mathbf{A}_{i,j}^{q}$, tels que $\hat{f}' = \hat{f}$ et $\omega' \leq \omega$. Cela signifie que la fonction composée $\hat{f}$ représente un chemin valide qui contient au plus un circuit valide ayant un poids négatif. Par conséquent, le réseau multicouche $\mathcal{N}$ n'est pas libre.

Les deux hypothèses ci-dessus ne tiennent pas, donc $(\hat{f}, \omega)$ ne peut pas être optimal. Par conséquent, la longueur maximale k des chemins valides élémentaires ($k \leq 2^{(\lambda+1)\lambda^2 n^2} - 1$) dans un réseau multicouche libre $\mathcal{N}$ est un point fixe pour la solution optimale globale $\mathbf{A}^*$. □

3.4.3 Taille de la solution optimale

Rappelons que dans un réseau multicouche, il peut y avoir plusieurs plus courts valides chemins entre une source et une destination avec différentes compositions de fonctions d'adaptation, qui peuvent induire différentes piles de protocoles. Dans cette situation, le nombre des plus courts valides chemins entre chaque paire de nœuds (U_i, U_j) dépend du nombre de piles de protocoles possibles dans le réseau. Dans ce qui suit, nous définissons cette valeur afin de calculer la taille de la solution optimale globale $\mathbf{A}^*$ du problème des plus courts chemins valides.

Soit λ le nombre de protocoles et h_{max} la hauteur maximale de la pile de protocoles atteinte dans un réseau multicouche. Le nombre de piles de protocoles possibles est $\lambda^{h_{max}}$. Cela signifie que, pour une paire de nœuds (U_i, U_j), le nombre des plus courts chemins valides avec chaque pile de protocoles possible peut être donné par :

$$|\mathbf{A}^*_{i,j}| = \lambda + \lambda^2 + \lambda^3 + \cdots + \lambda^{h_{max}} = (\frac{1-\lambda^{h_{max}+1}}{1-\lambda} - 1)$$

Par conséquent, dans le pire des cas, le nombre des plus courts chemins valides pour toutes les paires est donné par :

$$|\mathbf{A}^*| = n^2(\frac{1-\lambda^{h_{max}+1}}{1-\lambda} - 1)$$

Cette taille peut induire un nombre exponentiel d'opérations (composition de fonctions ou concaténation de piles de protocoles) dans un algorithme de routage multicouche. Cette limite est étroite, *i.e.,* il est possible de trouver un réseau où le plus court chemin possible atteint une hauteur de pile de λn^2 protocoles comme le prouve Lamali *et al.* [59]. Ainsi, la longueur maximale d'un plus court chemin valide peut être exponentielle comme mentionné dans la section 2.2.2.2. Enfin, ces limitations théoriques ne sont pas dues à l'algorithme de routage multicouche mais sont inhérentes à la nature du problème. Dans la section suivante, nous montrons les limites de valeur de chaque borne en pratique.

3.5 Évaluation des propriétés de convergence

Dans cette section, nous avons effectué des simulations afin de valider les limites théoriques données dans la section précédente. Pour cela, nous avons utilisé l'algorithme Stack-Vector [59] en abrégé SV et l'algorithme Transitive-Closure 2.3.1 en abrégé TC. Dans les deux algorithmes, nous n'autorisons pas les circuits multicouches dans le calcul de la solution optimale.

3.5.1 Méthodologie

Les simulations ont été réalisées sur une station de travail équipée d'un processeur Intel i9-11900 hyperthreadé à 8 cœurs capable d'atteindre 5,2 GHz avec 128 Go de RAM. L'implémentation a été écrite en ISO C++14 à l'aide de la bibliothèque igraph [16] pour générer des topologies de réseaux multicouches aléatoires. Toutes les topologies de réseaux utilisées pour l'expérimentation sont des graphes générés aléatoirement par un mécanisme d'attachement préférentiel défini par Barabàsi et Albert dans [3], où chaque nœud nouvellement ajouté est attaché à 3 nœuds existants. Pour un nombre λ de pro-

tocoles donné, il y a $3\lambda^2$ fonctions d'adaptation possibles. Chaque fonction d'adaptation est rendue disponible sur un nœud avec une probabilité fixe $p = 0,05$. Les paramètres d'entrée de chaque simulation sont : le nombre de nœuds n dans le réseau et le nombre de protocoles $\lambda = 2$. Les métriques de sortie sont : la longueur maximale du chemin, la hauteur maximale de la pile de protocoles, la taille optimale de la solution et l'efficacité de l'algorithme de calcul de chemins. Toutes les valeurs de résultat présentées dans les figures suivantes sont moyennées sur les valeurs de résultat de 100 exécutions.

3.5.2 Longueur maximale du chemin

La figure 3.3 montre la longueur maximale du chemin obtenue par la solution optimale calculée par les algorithmes SV et TC en fonction de la taille du réseau. Nous pouvons voir que dans tous les contextes, la longueur des chemins calculés ne dépasse jamais la longueur maximale des chemins multicouches élémentaires indiquée ci-dessus. Cela signifie que sans circuits multicouches, le calcul de la solution des plus courts chemins valides $\mathbf{A}^*$ converge après au plus la longueur maximale des chemins multicouches élémentaires, comme indiqué dans la section 3.4. Les résultats sont les mêmes pour les algorithmes SV et TC car ils calculent tous les deux la solution optimale.

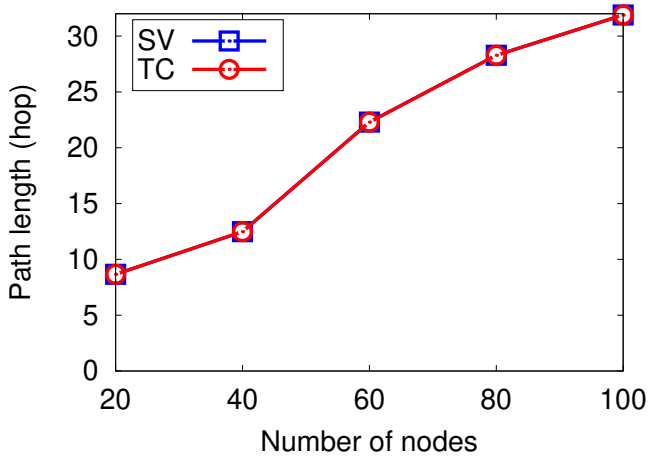


FIGURE 3.3 – La longueur maximale du chemin atteinte dans la solution optimale calculée par les algorithmes SV et TC en fonction de la taille du réseau.

3.5.3 Hauteur maximale de la pile

La figure 3.4 montre la hauteur maximale de la pile de protocoles obtenue par la solution optimale calculée par les algorithmes SV et TC en fonction de la taille du réseau. Les résultats sont les mêmes pour les algorithmes SV et TC car ils calculent tous les deux la solution optimale. Ces résultats montrent que la hauteur maximale de la pile a un

impact énorme sur la longueur maximale du chemin, et plus précisément sur le temps de convergence et la taille de la solution des plus courts chemins valide $\mathbf{A}^*$.

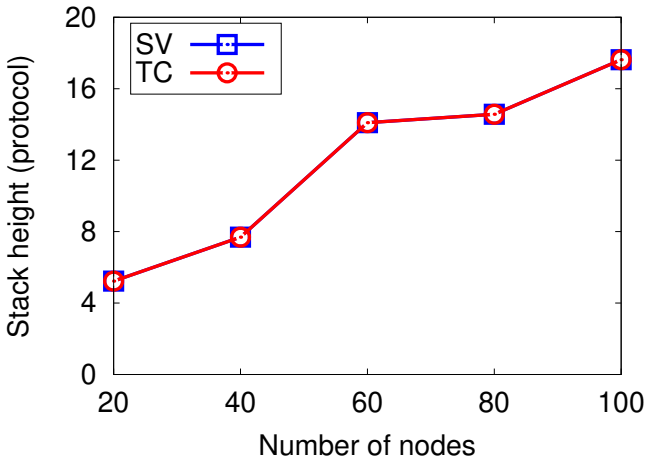


FIGURE 3.4 – La hauteur maximale de la pile de protocoles atteinte dans la solution optimale calculée par les algorithmes SV et TC en fonction de la taille du réseau.

3.5.4 Efficacité de l'algorithme de calcul de chemins

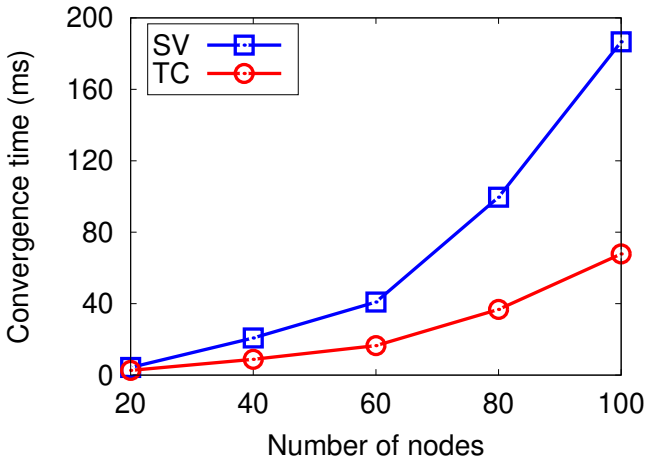


FIGURE 3.5 – Le temps de convergence des algorithmes SV et TC en fonction de la taille du réseau.

La figure 3.5 montre le temps total requis par les algorithmes SV et TC pour calculer les tables de routage en fonction de la taille du réseau. Nous l'appelons le temps de convergence de l'algorithme. Nous pouvons voir que l'algorithme TC est plus efficace que l'algorithme SV et peut être plus rapide jusqu'à 63% lorsqu'il est exécuté sur des réseaux à 100 nœuds. Ces résultats montrent que même si ces deux algorithmes

sont capables de trouver la solution optimale pour le routage dans les réseaux multicouches (comme le prouvent les algèbres et montré dans les sous-sections précédentes), ils peuvent néanmoins présenter des performances disparates en raison de leurs complexités temporelles différentes.

3.6 Conclusion

L'algèbre de routage est une abstraction puissante pour étudier les protocoles de routage (*ex.,* protocoles de vecteur de distance, vecteur de chemin), en particulier les propriétés de validation et de convergence. Ces propriétés deviennent plus complexes dans le cas des protocoles de routage multicouches où les tunnels sont omniprésents. Il est donc important d'avoir des modèles algébriques pour les problèmes de routage avec tunnels. Dans ce contexte, les modèles algébriques utilisés sont les semi-anneaux, l'algèbre avec fonctions et l'algèbre de Sobrinho. Or, ces modèles ne sont pas adaptés aux réseaux multicouches et ne prennent pas en compte les fonctions d'adaptation et les tunnels.

Dans ce chapitre, nous avons étudié la modification des trois styles d'algèbre de routage existants dans le cas des réseaux multicouches. Pour cela, nous avons défini trois algèbres de routage pour le calcul des plus courts chemins valides (vsp) : un semi-anneau, une algèbre de fonctions et une algèbre de Sobrinho. Ces structures permettent d'énumérer l'ensemble des plus courts chemins valides entre chaque paire de nœuds dans le réseau. Toutes nos structures sont basées sur un produit semi-direct des deux structures, la nouvelle structure chemins valides (vp) et la structure chemins plus courts (sp). Nous avons montré que les algèbres de routage proposées sont isotones et non monotones avec une relation d'ordre partiel. Enfin, nous avons proposé un point fixe pour ces algèbres et nous avons prouvé la convergence itérative vers la solution optimale du problème des plus courts chemins valides (vsp).

Chapitre 4

Génération de topologies de réseaux hétérogènes et multicouches

La meilleure théorie s'inspire de la pratique.

– Donald Knuth

Sommaire

4.1 Introduction

La simulation en communication et réseaux informatiques est un outil puissant pour concevoir et analyser des protocoles. Elle offre l'avantage de ne pas être limitée par des contraintes matérielles, nécessitant simplement des logiciels d'application abordables et prêts à l'emploi. Cependant, une configuration appropriée de la topologie du réseau est essentielle. Les topologies des couches OSI un et deux sont généralement simples et faciles à modéliser (*ex.,* bus, anneau, étoile). En revanche, pour les couches OSI trois et supérieures, les topologies peuvent être plus vastes et complexes graphiquement. Les chercheurs utilisent différents modèles pour les générer. Depuis les années 90, plusieurs générateurs ont été développés pour reproduire celles observées dans les réseaux réels de couche trois (et au-delà), comme Internet. On distingue différentes classes de générateurs de topologies monocouches, notamment les topologies plates, hiérarchiques, suivant des lois de puissance, basées sur l'échantillonnage cartographique, et autres.

Néanmoins, ces générateurs ne peuvent produire qu'une topologie de réseau à une seule couche, représentant un protocole spécifique à une couche donnée de la pile de protocoles. Avec l'avènement des réseaux virtuels et la coexistence protocolaire, les protocoles se sont multipliés et les piles de protocoles sont devenues plus complexes. Des protocoles de routage dynamique capables de calculer des routes à travers plusieurs couches ont été proposés et leur évaluation nécessite l'utilisation de topologies de réseaux multicouches. A notre connaissance, un seul générateur de ce type, nommé MulNeG, existe actuellement. MulNeG se concentre davantage sur les réseaux de couche application tels que les réseaux sociaux et n'est donc pas tout à fait approprié pour générer des topologies de réseau de couche inférieure. Pour combler ce vide, il est nécessaire de concevoir et mettre en œuvre un nouveau générateur de topologie de réseau multicouche.

Dans ce chapitre, nous proposons un générateur de topologie de réseau multicouche appelé *Multilayer Network Topology Generator* (MNTG). Notre générateur offre trois modes à partir desquels il est possible de créer des topologies de réseaux multicouches. Il s'agit de topologies uniques, de plusieurs topologies synthétiques ou de plusieurs cartes de réseaux réels. La section 4.2 passe en revue les travaux existants sur le sujet et présente les méthodes de génération de graphes aléatoires et de distribution de fonctions d'adaptation utilisés dans ce chapitre. Ceci est suivi dans la section 4.3 par trois algorithmes de génération de topologie de réseau multicouche afin de définir l'architecture et l'implémentation de notre générateur. Enfin, nous évaluons les performances de notre générateur dans la section 4.4 par la création de différentes topologies multicouches. La section 4.5 conclut le chapitre.

Ce chapitre est basé sur nos travaux sur la génération de topologies de réseaux hétérogènes et multicouches, qui ont été publiés dans la conférence internationale IEEE VCC 2023 [84], et disponible dans le dépôt Gitlab de l'Université de Bordeaux sous forme de logiciel libre et open source [78].

4.2 Autour de la génération des topologies de réseaux multicouches

Dans cette section, nous citons la plupart des recherches sur la génération de topologies de réseaux multicouches. Nous citons brièvement les méthodes de génération de graphes aléatoires utilisées dans ce chapitre. Nous expliquons également notre méthode pour distribuer des fonctions d'adaptation dans un réseau multiouche. Enfin, nous décrivons notre approche pour concevoir un générateur de topologies de réseaux multiouches.

4.2.1 État de l'art

Les réseaux multicouches ont récemment reçu beaucoup d'attention dans la science des réseaux. En particulier, les chercheurs se sont concentrés sur les réseaux multicouches stochastiques. Un tel réseau est créé par l'agrégation de plusieurs réseaux (un par couche) où chacun est un sous-graphe d'un réseau fondamental. Ils peuvent coder différents types de connexions et/ou des connexions dépendant du temps sur le même ensemble de nœuds. En 2018, Jiang *et al.* ont proposé un modèle pour de tels réseaux et ont pu dériver des distributions de probabilité pour le modèle [51]. Dans le même ordre d'idées, Pan *et al.* ont proposé, lors de la même année, une une approche d'agrégation pour détecter les communautés dans les réseaux multicouches [88], mais non limité aux réseaux multicouches stochastiques. En 2023, Kawase *et al.* a introduit un nouveau modèle d'optimisation pour la découverte de sous-graphes denses dans les réseaux multicouches [55].

Concernant la génération de topologie de réseau multicouche, un seul outil, nommé MulNeG, a été proposé par Popiel *et al.* en 2015 [93]. Cet outil se concentre sur la modélisation des réseaux sociaux. Comme pour les réseaux stochastiques, MulNeG génère des réseaux où chaque nœud a son homologue dans chaque couche et où seul l'ensemble des liens de chaque couche diffère les uns des autres. Bien que chaque nœud (représentant généralement un utilisateur) appartienne à plusieurs couches (*i.e.,* les réseaux sociaux), il n'y a pas de concept de définition de chemins pour déplacer des données d'un nœud à un autre en utilisant des conversions ou des tunnels.

4.2.2 Génération de graphes aléatoires

La génération de topologies de réseau, de par sa nature même, est liée à la génération de graphes aléatoires. Les modèles de graphes aléatoires ont été proposés pour la première fois par Erdös et Rényi [27] en 1959. Les graphes ont été générés en plaçant des liens aléatoires entre les nœuds en utilisant une probabilité p. Dans l'approche de Erdös et Rényi, la distribution des degrés de nœuds est binomiale et le nombre de liens dépend

du nombre de nœuds. Watts et Strogatz [110] ont proposé une autre classe de modèles de graphes aléatoires appelés réseaux de petit monde. L'idée principale de leur modèle consistait à construire un anneau pour un nombre donné de nœuds, dans lequel chaque nœud est connecté à k voisins. Les arêtes sont ensuite sélectionnées par un processus de "recâblage" selon une probabilité p. Ce modèle produit des graphes qui avait de petites longueurs moyennes du plus court chemin et des coefficients de regroupement élevés.

Modèle de génération	**Référence**
Erdös-Rényi	[27]
Watts-Strogatz	[110]
Barabási-Albert	[4]

TABLE 4.1 – Méthodes de génération de graphes aléatoires.

La dernière classe pertinente de modèles de graphes aléatoires sont ceux qui génèrent des réseaux sans échelle, *i.e.,* des réseaux qui ont des distributions de degré de loi de puissance. Ils ont été introduits pour la première fois par Barabási et Albert [4] en 1999. Dans leur méthode proposée, les mécanismes de croissance et d'attachement préférentiel sont mélangés. Le réseau commence petit, puis de nouveaux nœuds sont ajoutés. Ces nouveaux nœuds sont déjà connectés aux nœuds existants, qui ont une probabilité p proportionnelle au degré du nœud courant. La distribution des degrés de nœud d'un tel graphe suit une loi de puissance et la longueur moyenne du plus court chemin croît comme une fonction logarithmique. Le tableau 4.1 décrit les recherches dans lesquelles ces méthodes ont été présentées pour la première fois et où plus de détails sur chacune des méthodes de génération peuvent être trouvés. Certaines méthodes de génération produiront les mêmes paramètres, tandis que d'autres produiront des différents paramètres.

4.2.3 Placement des fonctions dans les réseaux

Comme mentioné déjà dans la section 1.4.2, le placement de fonctions (*ex.,* VNF, convertisseurs WDM, NAT dans un réseau IPv4/IPv6, etc.) dans un réseau ou plus généralement dans un graphe est un problème d'optimisation difficile. Dans un contexte de réseau multicouche, ce problème consiste à trouver l'emplacement optimal des fonctions d'adaptation qui augmente l'accessibilité, *i.e.,* la faisabilité du chemin, dans le réseau multicouche. Autrement dit, l'idée est de produire des réseaux multicouches dans lesquels, au minimum, n'importe quel nœud peut atteindre n'importe quel autre nœud, quelle que soit sa pile de protocoles. Cependant, la contrainte d'accessibilité/faisabilité du chemin n'est pas facile à résoudre dans un tel réseau. Pour cela, notre objectif est d'utiliser deux méthodes pour placer des fonctions dans un réseau multicouche. La première méthode, dite méthode probabiliste, consiste à utiliser une probabilité p fournie par l'utilisateur qui représente la probabilité de disponibilité d'une fonction sur un nœud. La deuxième méthode, appelée méthode split, est plus réaliste que la première, et elle consiste à répartir les fonctions de retransmission traditionnelles sur les nœuds monoprotocoles et

les fonctions de conversion ou d'encapsulation sur les nœuds multiprotocoles.

Cependant, la méthode probabiliste est limitée puisqu'elle permet de produire des réseaux avec certains nœuds isolés *i.e.,* qui n'ont aucune fonction d'adaptation. Autrement dit, cette méthode manque de connectivité et d'accessibilité où il n'existe pas toujours un chemin (au sens classique) entre chaque paire de nœuds. Ceci est dû à la probabilité étant donné que plus elle est faible, moins le nœud a de chance d'avoir la fonction et vice versa. Pour cela, il faut connaître le nombre moyen de nœuds effectifs (ou nœuds perdus) et le nombre moyen de fonctions d'adaptation par nœud, selon la probabilité p. Plus formellement, la distribution aléatoire des $3\lambda^2$ fonctions (avec les λ protocoles, nous avons $3\lambda^2$ fonctions possibles) sur un nœud est $3\lambda^2$ d'expériences binomiales et chaque expérience est un essai de Bernoulli. Ainsi, la probabilité d'obtenir exactement k fonctions est donnée par :

$$P(X = k) = C_k^{3\lambda^2}\, p^k\, (1-p)^{3\lambda^2 - k}$$

En se basant sur ça, la probabilité d'obtenir un nœud vide (sans aucune fonction) est :

$$P(X = 0) = (1-p)^{3\lambda^2}$$

Et la probabilité d'obtenir au moins une fonction par nœud est :

$$P(X \geq 1) = 1 - P(X = 0)$$

Enfin, le nombre moyen de fonctions par nœud utilisant la probabilité p est :

$$E(X) = 3\lambda^2\, p$$

Le tableau 4.2 résume toutes ces valeurs dans le cas de réseaux avec 2 protocoles, *i.e.,* 12 fonctions d'adaptation et une probabilité p comprise entre 0.05 et 0.30.

Probabilité	**Nœuds effectifs**	**Fonctions/Nœud**
0.05	46%	0.6
0.10	72%	1.2
0.20	93%	2.4
0.30	99%	3.6

TABLE 4.2 – Le pourcentage moyen de nœuds effictifs et le nombre moyen de fonctions d'adaptation par nœud selon la probabilité de distribution p dans un réseau avec 2 protocoles.

4.2.4 Notre approche

Dans la section 1.3.1, nous avons défini le réseau multicouche comme étant un graphe orienté avec un ensemble de protocoles et un ensemble de fonctions d'adaptation.

Chaque nœud possède un sous-ensemble des fonctions parmi l'ensemble des fonctions disponibles dans le réseau. Afin de créer un générateur de topologies de réseau multicouche l'idée est de séparer la génération en deux étapes. La première étape consiste à créer le graphe sous-jacent, tandis que la deuxième étape se charge de répartir les fonctions d'adaptation sur les nœuds du graphes généré à l'étape précédente.

Dans ce contexte, la création du graphe sous-jacent peut être réalisée de plusieurs manières. La méthode la plus simple consiste à générer un ou plusieurs graphes aléatoires, en fonction du nombre de couches, et en utilisant l'un des générateurs existants mentionnés dans la section précédente. Une autre approche possible serait de collecter des cartes de réseaux réels, une par protocole ou par couche, et de les fusionner tout en identifiant les nœuds multiprotocoles. Une fois le graphe sous-jacent créé, la distribution des fonctions d'adaptation sur les nœuds peut être effectuée selon deux méthodes, en fonction du choix de l'utilisateur. La première méthode est purement aléatoire, permettant ainsi de répartir les fonctions d'adaptation sur les nœuds en suivant une probabilité donnée. Cependant, la deuxième méthode repose sur la capacité de chaque nœud. Plus précisément, les nœuds monoprotocoles sont uniquement en mesure de transmettre le protocole qui leur est attribué, tandis que les nœuds multiprotocoles ont la capacité de convertir ou d'encapsuler/désencapsuler les protocoles en fonction de la demande. Cette dernière méthode s'avère plus réaliste que la précédente, car elle tient compte des compétences spécifiques de chaque nœud.

4.3 Générateur de topologies multicouches

Dans cette section, nous présentons les différentes méthodes de génération de topologies de réseau multicouche. Nous proposons également un algorithme pour chaque méthode. Enfin, nous décrivons l'architecture et l'implémentation des différentes méthodes dans notre générateur MNTG.

4.3.1 Algorithme de génération mono-aléatoire

La méthode mono-aléatoire est une méthode de génération entièrement aléatoire, où un seul graphe est généré ou chargé avec une distribution probabiliste de fonctions d'adaptation sur les nœuds (les nœuds hétérogènes et leurs protocoles sont déterminés par les fonctions d'adaptation qui leur sont attribuées). L'algorithme 9 permet d'accomplir cette tâche. Pour cela, il distribue, sur chaque nœud du graphe, un ensemble de fonctions d'adaptation à partir d'un ensemble de protocoles communiqués. Afin d'utiliser toutes les fonctions de la distribution, cet ensemble est initialisé avec l'ensemble de tous les protocoles.

La distribution de fonctions d'adaptation est réalisée par l'algorithme 10. Ce dernier permet de distribuer aléatoirement les fonctions d'adaptation une par une, et vérifier si chaque fonction peut gérer les protocoles communiqués des nœuds correspondants. Si

Algorithme 9 : Génération de nœuds mono-aléatoires

(1) **Pour tout** $U \in \mathcal{V}$ **faire**
(2) $\quad \mathcal{A} \leftarrow$ Tous_Protocoles ()
(3) $\quad$ Distribution_Aléatoire_Fonctions $\left(U, \mathcal{A}\right)$

c'est le cas, la fonction est attachée au nœud avec une probabilité p.

Algorithme 10 : Distribution aléatoire des fonctions d'adaptation

(1) **Pour tout** $f \in \mathcal{F},\ f = (x \rightarrow y)$ *ou* $(x \rightarrow xy)$ *ou* $(xy \rightarrow x)$ **faire**
(2) $\quad$ **Si** $(x, y) \in \mathcal{A}$ **alors**
(3) $\quad\quad$ **Si** *Nombre_Aléatoire_Réel (0, 1)* $<$ *Probabilité (*f*)* **alors**
(4) $\quad\quad\quad \mathcal{F}(U) \leftarrow \mathcal{F}(U) \cup \{f\}$

4.3.2 Algorithme de génération multi-aléatoire

La méthode multi-aléatoire utilise plusieurs graphes, plus précisément un par couche de protocole disponible. Les nœuds hétérogènes sont générés par une fusion aléatoire de nœuds à partir des graphes sous-jacents. La répartition des fonctions d'adaptation sur chaque nœud dépend de son ensemble de protocoles communiqués et en utilisant la probabilité de chaque fonction. L'algorithme 11 permet de créer une topologie multi-aléatoire. En d'autres termes, il permet de créer des nœuds multi-protocoles. Cette étape est basée sur le nombre de nœuds qui existent en fonction du nombre de protocoles (spécifié par l'utilisateur). Afin de créer un nœud communicant j protocoles, l'algorithme choisit aléatoirement j nœuds parmi j graphes distincts. L'ensemble des protocoles communiqués d'un tel nœud est l'ensemble des protocoles de chaque graphe sélectionné. Une fois le nœud créé, l'algorithme distribue les fonctions d'adaptation à l'aide de l'algorithme 10.

Algorithme 11 : Génération de nœuds multi-aléatoires

(1) **Pour tout** $k = 1, 2, \ldots, |\mathcal{V}_j|,\ j \in \{1, 2, \ldots, \lambda\}$ **faire**
(2) $\quad$ **Pour tout** $l = 1, 2, \ldots, j$ **faire**
(3) $\quad\quad \mathcal{G}_i \leftarrow$ Sélectionner_Graphe $\left(\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_\lambda\right)$
(4) $\quad\quad V_i \leftarrow$ Sélectionner_Nœud $\left(\mathcal{V}(\mathcal{G}_i)\right)$
(5) $\quad\quad \mathcal{A} \leftarrow$ Ajouter_Protocole $\left(i\right)$
(6) $\quad\quad U \leftarrow$ Fusionner_Nœud $\left(V_i\right)$
(7) $\quad$ Distribution_Aléatoire_Fonctions $\left(U, \mathcal{A}\right)$

4.3.3 Algorithme de génération multi-réel

La méthode multi-réelle n'est disponible qu'avec des graphes prédéfinis. Les nœuds hétérogènes sont générés à partir d'un fichier d'entrée multi-stack, dans lequel chaque ligne représente l'ensemble des nœuds à fusionner entre les différents graphes sous-jacents. Les fonctions d'adaptation sont distribuées de manière plus réaliste, contrairement aux deux premières méthodes. Les nœuds mono-protocoles ne peuvent avoir qu'une retransmission classique et les nœuds multi-protocoles peuvent avoir soit une conversion dans les deux sens (ce que l'on trouve typiquement dans les réseaux réels), soit l'encapsulation et sa désencapsulation correspondante (un nœud s'il encapsule un protocole dans un autre est capable de faire l'opération inverse de désencapsulation), ou les deux capacités en même temps, *i.e.,* convertir et encapsuler. Cette technique est implémentée par l'algorithme 12. Il permet de créer des nœuds multiprotocoles en utilisant des données de cartographie réseau réelles fournies par l'utilisateur dans un fichier multi-stack. Pour chaque tuple de nœuds présents ou non dans chaque couche, l'algorithme crée un seul nœud avec le protocole communiqué de chaque couche réseau, *i.e.,* graph. Ensuite, il distribue des fonctions d'adaptation sur ce nœud.

Algorithme 12 : Génération de nœuds multi-réels

```
(1) Pour tout (V_1, V_2, ..., V_λ) ∈ Fichier Multi Stack faire
(2)     Pour tout V_i ∈ (V_1, V_2, ..., V_λ) faire
(3)         Si Non_Vide(V_i) alors
(4)             A ← Ajouter_Protocole (i)
(5)             U ← Fusionner_Nœud (V_i)
(6)     Distribution_Réelle_Fonctions (U, A)
```

La distribution des fonctions d'adaptation sur les nœuds est implémentée dans l'algorithme 13. Ce dernier attribue, à partir de l'ensemble des protocoles communiqués, un ensemble de fonctions d'adaptation à chaque nœud. Cet ensemble de fonctions ne peut avoir que la retransmission classique (dans le cas de nœuds monoprotocoles), la conversion dans les deux sens, ou l'encapsulation et sa désencapsulation correspondante. D'autres options incluent les deux fonctions, *i.e.,* les conversions et les encapsulations.

4.3.4 Architecture et implémentation

La figure 4.1 montre comment fonctionne la génération de réseau multicouche par notre générateur MNTG. Le processus commence par l'analyse des paramètres à partir d'un fichier de spécification d'entrée. Ensuite, les graphes sont soit générés selon un paramètre booléen (basé sur le modèle de génération de graphes), soit chargés (avec les fichiers de graphes d'entrée correspondants). Une fois les graphes disponibles, l'étape suivante consiste à générer le réseau multicouche. Cette étape commence par générer toutes les fonctions d'adaptation possibles à partir d'un ensemble de protocoles (selon le

Algorithme 13 : Distribution réaliste des fonctions d'adaptation

(1) **Si** $|\mathcal{A}| = 1$, $\mathcal{A} = \{x\}$ **alors**
(2) $\quad \mathcal{F}(U) \leftarrow \mathcal{F}(U) \cup \{x \rightarrow x\}$
(3) **Sinon**
(4) $\quad$ **Pour tout** $(x, y) \in \mathcal{A}$ **faire**
(5) $\quad\quad$ **Selon** Nombre_Aléatoire_Réel (1, 7) **faire**
(6) $\quad\quad\quad$ **Cas 1 :** $\mathcal{F}(U) \leftarrow \mathcal{F}(U) \cup \{x \rightarrow y, y \rightarrow x\}$
(7) $\quad\quad\quad$ **Cas 2 :** $\mathcal{F}(U) \leftarrow \mathcal{F}(U) \cup \{x \rightarrow xy, xy \rightarrow x\}$
(8) $\quad\quad\quad$ **Cas 3 :** $\mathcal{F}(U) \leftarrow \mathcal{F}(U) \cup \{y \rightarrow yx, yx \rightarrow y\}$
(9) $\quad\quad\quad$ **Cas 4 :** Appliquer les cas 1 et 2
(10) $\quad\quad\quad$ **Cas 5 :** Appliquer les cas 2 et 3
(11) $\quad\quad\quad$ **Cas 6 :** Appliquer les cas 1 et 3
(12) $\quad\quad\quad$ **Cas 7 :** Appliquer les cas 1 et 2 et 3

paramètre "nombre de protocoles"). Rappelons que, pour λ protocoles, nous avons $3\lambda^2$ fonctions d'adaptation possibles. La génération des nœuds est l'étape la plus importante, à partir de laquelle les nœuds multiprotocoles sont créés. À ce stade, trois méthodes de génération sont possibles : mono-aléatoire, multi-aléatoire et multi-réelle.

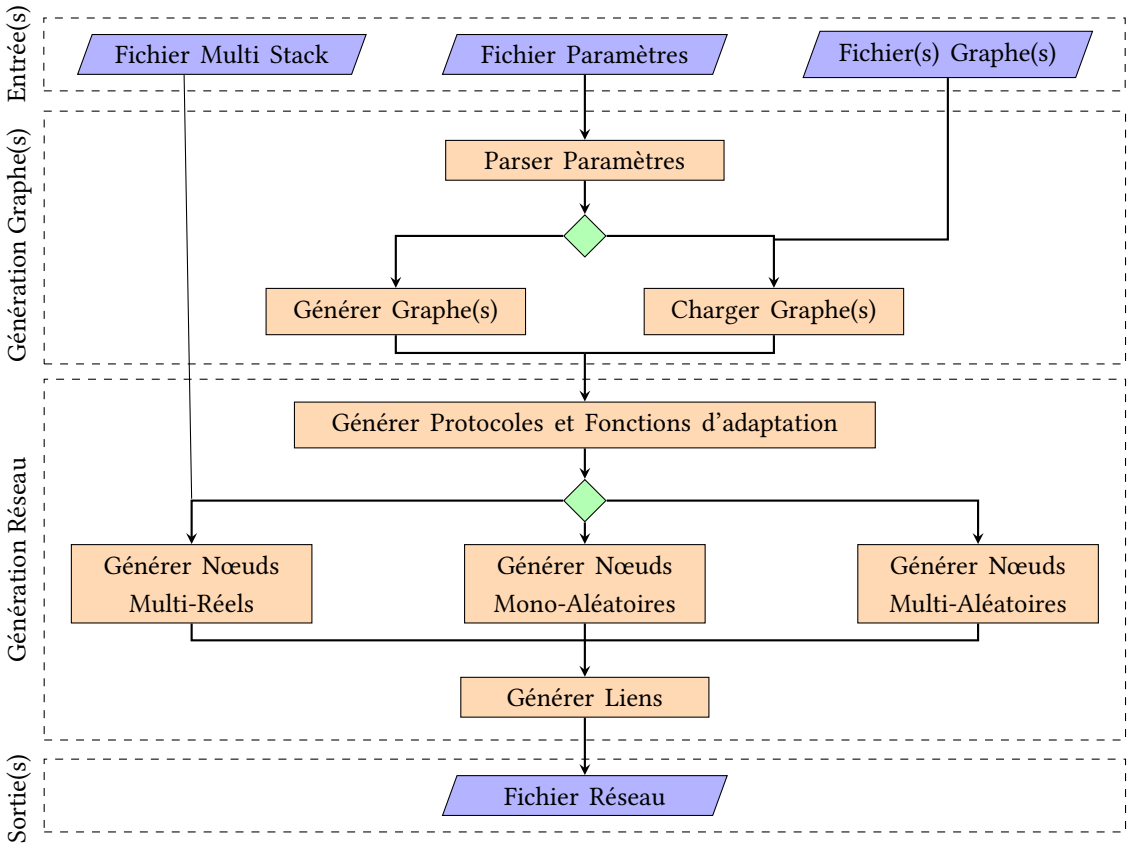


FIGURE 4.1 – Diagramme de flux du générateur MNTG.

Afin de gérer efficacement le code source, nous avons choisi d'utiliser le langage de programmation orienté objet C++. La génération de graphes aléatoires classiques (*i.e.,* Barabasi-Albert, Erdos-Renyi et Watts-Strogatz) est réalisée par la bibliothèque igraph [16]. Le code source de notre générateur est librement disponible sur un dépôt Gitlab hébergé par l'Université de Bordeaux [78].

La génération d'une topologie de réseau multicouche se fait à l'aide de trois types de fichiers d'entrée : le fichier de paramètres, les fichiers de graphes sous-jacents et le fichier multi-stack. Les fichiers de graphes sous-jacents suivent le format Pajek' NET (qui est facile à manipuler avec la bibliothèque igraph). Ceux-ci contiennent la description des nœuds et des liens. Le fichier de paramètres est un fichier texte codé en ASCII, qui agit comme un fichier de spécifications et contient des paramètres communs pour toutes les méthodes de génération. Ces paramètres incluent le nom du fichier de sortie, le nombre de nœuds, le nombre de protocoles, le nombre de graphes d'entrée (si les graphes sont chargés), les coûts des fonctions d'adaptation et la graine du générateur de nombres pseudo-aléatoires.

Les deux méthodes aléatoires (mono et multi) nécessitent ces paramètres supplémentaires : la taille de chaque graphe généré, le nombre de nœuds en fonction du nombre de protocoles communiqués, le modèle de génération (BA, ER ou WS), la probabilité de disponibilité de chaque fonction d'adaptation, les paramètres d'attachement pour le modèle BA, la probabilité de connexion pour le modèle ER et le degré moyen et la probabilité bêta pour le modèle WS. La méthode multi-réel utilise ces paramètres communs, les graphes d'entrées et le fichier multi-stack. Le fichier multi-stack est un fichier ASCII qui décrit les nœuds communs (les identifiants d'AS ou les adresses de routeurs) qui existent entre les différents graphes d'entrée, *i.e.,* les nœuds multi-protocoles qui supportent plusieurs protocoles ou ceux présents dans différentes couches du réseau.

La plupart des formats de représentation de graphes ne prennent pas en charge les graphes multicouches. Pour résoudre ce problème, nous utilisons deux nouveaux formats. Le premier format est un format texte codé en ASCII facilement lisible qui décrit les informations du réseau (nombre de nœuds, nombre de liens et nombre de protocoles ou de couches), les informations sur chaque nœud (l'identifiant unique, l'ensemble des protocoles communiqués et le ensemble des fonctions d'adaptation pondérées), et l'ensemble des liens (identifiant source, identifiant destination et le poids des liens sans fonctions). Le deuxième format est basé sur le format DOT de Graphviz [36] avec la possibilité d'ajouter des attributs et des clusters (sous-graphes) afin de représenter la topologie produite par notre générateur. Le fichier de sortie comprend des informations typiques sur la topologie du réseau (telles que le nombre de nœuds, le nombre de liens, la liste des nœuds et la liste des liens pondérés ou non), ainsi que des informations supplémentaires liées à l'hétérogénéité des protocoles et à la multiplicité des couches (telles que le nombre de protocoles ou les fonctions d'adaptation de chaque nœud avec leurs coûts d'utilisation).

4.4 Évaluation des performances du générateur

Dans cette section, nous avons effectué des simulations afin d'évaluer les performances de notre générateur. Pour cela, nous avons détaillé la complexité temporelle et spatiale de notre générateur. Nous avons également mesuré le temps de génération dans différents modes (mono et multi). Enfin, nous avons montré l'impact des topologies multicouches sur les simulations.

4.4.1 Méthodologie

L'impact des topologies de réseau monocouche sur les résultats de simulation de protocole s'est avéré assez significatif il y a vingt ans [69]. Afin d'évaluer la consommation de ressources du MNTG et l'impact des topologies générées sur les simulations, nous avons généré des topologies avec MNTG et nous avons simulé l'algorithme Stack-Vector [59] dans le même environnement que celui utilisé dans le chapitre 2. Toutes les expériences ont été réalisées sur un Dell Latitude 5300 équipé d'un processeur Intel i7-8665U 4 cœurs hyperthreadé cadencé jusqu'à 4,80 GHz avec 32 Go de RAM.

Toutes les topologies de réseau utilisées pour les expériences sont des graphes aléatoires générés par les méthodes Barabàsi-Albert [4], Erdös-Rényi [27] et Watts-Strogatz [110]. Les paramètres de simulation incluent le mode de génération (mono ou multi), la méthode des graphes aléatoires (Barabàsi-Albert avec une valeur de paramètre d'attachement de 3, Erdös-Rényi avec une probabilité de connexion de 0,05, et Watts-Strogatz avec un degré moyen de 2 et une probabilité bêta de 0,01), la taille du réseau (nombre de nœuds n), la probabilité p de disponibilité d'une fonction d'adaptation sur un nœud, et le nombre de couches (*i.e.,* le nombre de protocoles λ). Il faut noter que pour les simulations de l'algorithme Stack-Vector, la taille du réseau est fixée à 50 de nœuds et la hauteur maximale de la pile est fixée à 3. Les principales mesures de sortie sont le temps de traitement, la consommation de mémoire, le pourcentage des chemins faisables et la longueur moyenne des chemins. Tous les points indiqués sur les courbes sont les résultats de valeurs moyennes sur 100 exécutions.

Concernant les topologies réelles, nous reprenons les mêmes topologies T1 et T2 de taille 1000 nœuds avec 2 protocoles IPv4 et IPv6, précédemment utilisées pour évaluer les algorithmes de calcul de chemins dans le chapitre 2. Nous rappelons par la suite les caractéristiques de chaque topologie.

- Pour la topologie de réseau T1 : 69,90 % des nœuds sont IPv4 uniquement (resp. 0,6 % IPv6 uniquement) et 29,50 % des nœuds sont IPv4 et IPv6.
- Pour la topologie de réseau T2 : 47,60 % des nœuds sont IPv4 uniquement (resp. 0,7 % IPv6 uniquement) et 51,70 % des nœuds sont IPv4 et IPv6.

4.4.2 Complexité et consommation de ressources

La méthode de génération mono-aléatoire distribue les fonctions d'adaptation sur tous les nœuds, *i.e.,* $\mathcal{V}$ un par un. Rappelons que pour λ protocoles, nous avons $3\lambda^2$ fonctions d'adaptation possibles. Cette génération est donc effectuée après $n \times 3\lambda^2$ opérations. Dans la méthode de génération multi-aléatoire, l'ensemble des nœuds est décomposé en λ sous-ensembles $\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_\lambda$ selon le nombre des protocoles utilisés par les nœuds de chaque sous-ensemble. Dans ce cas, l'ensemble des fonctions d'adaptation à distribuer sur les nœuds varie d'un sous-ensemble à l'autre. Le traitement général de cette méthode peut être écrit comme une somme d'opérations $\sum_{k=1}^{\lambda} n_k \times (k+3k^2)$, où n_k est le nombre de nœuds qui communiquent en utilisant k protocoles et $3k^2$ est le nombre de fonctions d'adaptation à distribuer sur les k nœuds fusionnés. Dans la méthode de génération multi-réelle, le nombre de nœuds qui communiquent avec k protocoles est représenté par le nombre de tuples de taille k, *i.e.,* le nombre de lignes contenant k colonnes dans le fichier multi-stack. La distribution des fonctions est réalisée par un couple de protocoles de sorte que, pour chaque couple, un ensemble de fonctions est distribué selon les différents cas possibles. Ainsi, le nombre total d'opérations peut être écrit sous la forme $n_1 + \sum_{k=2}^{\lambda} n_k \times (k + k^2)$. La complexité temporelle de la borne supérieure dans le pire des cas, pour chaque algorithme, peut être réduite à $O(n\lambda^2)$.

Complexité	**Borne supérieure**
Temporelle	$O(n\lambda^2)$
Spatiale	$O(\lambda n(\lambda + \log n))$

Table 4.3 – Complexité temporelle et spatiale du générateur MNTG.

Concernant la consommation d'espace en mode multi-*, pour λ couches, au plus n nœuds par couche, au plus λ protocoles par nœud, et au plus m liens par couche, le nombre d'états stockés en mémoire est de l'ordre de $O(\lambda(\lambda n + m))$. Si nous supposons des graphes creux où $m \propto n \log n$, alors ce nombre revient à $O(\lambda n(\lambda + \log n))$. Le tableau 4.3 résume la complexité temporelle et spatiale de notre générateur MNTG.

Les figures 4.2,4.3 montrent le temps d'exécution du générateur MNTG dans les deux modes mono et multi-aléatoires en fonction de la taille du réseau, le nombre de couches et la probabilité p.

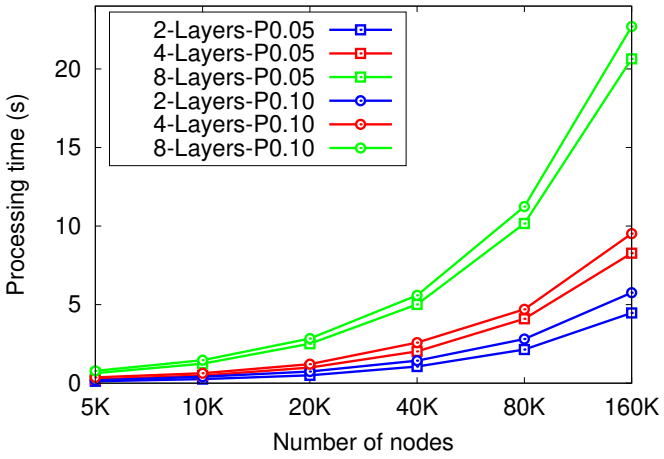


FIGURE 4.2 – Temps d'exécution du générateur MNTG dans le mode mono-aléatoire.

Nous pouvons voir que dans les deux méthodes de génération, le nombre de couches impacte davantage le temps de génération que la taille du réseau et la probabilité p. Par exemple, le temps de génération dans le mode mono-aléatoire (resp. multi-aléatoire) est d'environ $5s$ (resp. $10s$) pour un réseau de 4 couches et avec 80k nœuds. En revanche, pour un réseau de 8 couches et avec 160k nœuds, le temps de génération est d'environ $22s$ (resp. $44s$) dans le mode mono-aléatoire (resp. multi-aléatoire).

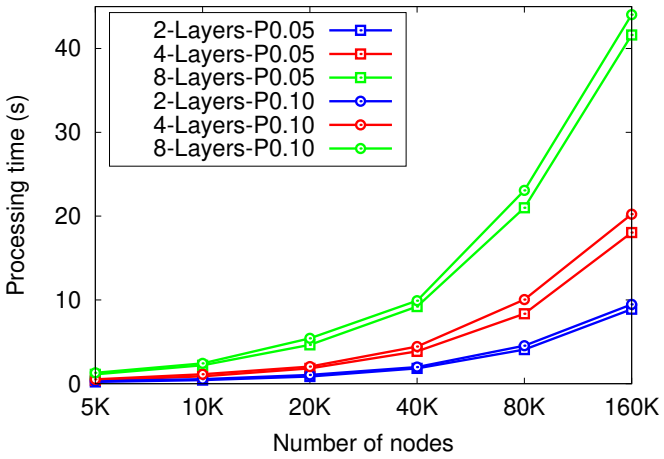


FIGURE 4.3 – Temps d'exécution du générateur MNTG dans le mode multi-aléatoire.

Les figures 4.4, 4.5 montrent la consommation de mémoire du générateur MNTG dans les deux modes mono et multi-aléatoires en fonction de la taille du réseau, le nombre de couches et la probabilité p.

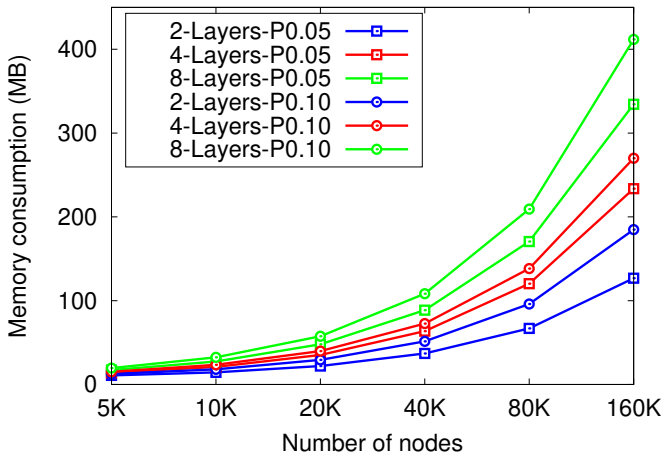


FIGURE 4.4 – Consommation de mémoire du générateur MNTG dans le mode mono-aléatoire.

Nous pouvons voir que dans les deux modes de génération, la consommation de mémoire dépend de tous les paramètres de génération. Par exemple, la consommation de mémoire dans le mode mono-aléatoire (resp. multi-aléatoire) avec une probabilité $p = 0.10$ est d'environ 138MB (resp. 224MB) pour un réseau de 4 couches et avec 80k nœuds. Toutefois, pour un réseau de 8 couches et avec 160k nœuds, la consommation de mémoire est d'environ 412MB (resp. 620MB) dans le mode mono-aléatoire (resp. multi-aléatoire).

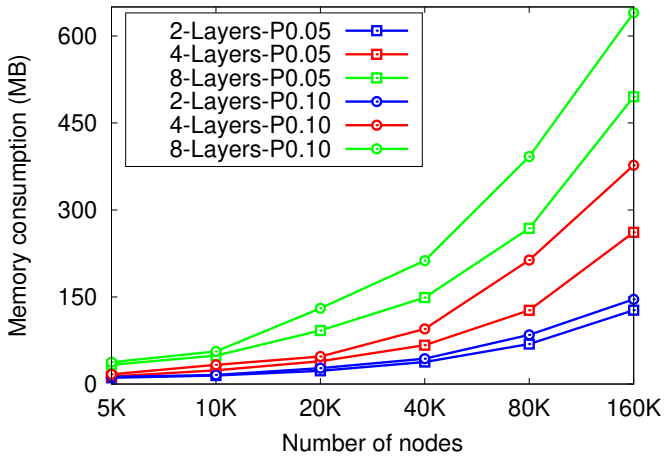


FIGURE 4.5 – Consommation de mémoire du générateur MNTG dans le mode multi-aléatoire.

Tous ces résultats montrent l'impact des différents paramètres sur les performances du générateur MNTG et elles correspondent bien aux résultats analytiques présentés dans le texte ci-dessus.

4.4.3 Impact des topologies multicouches sur les simulations

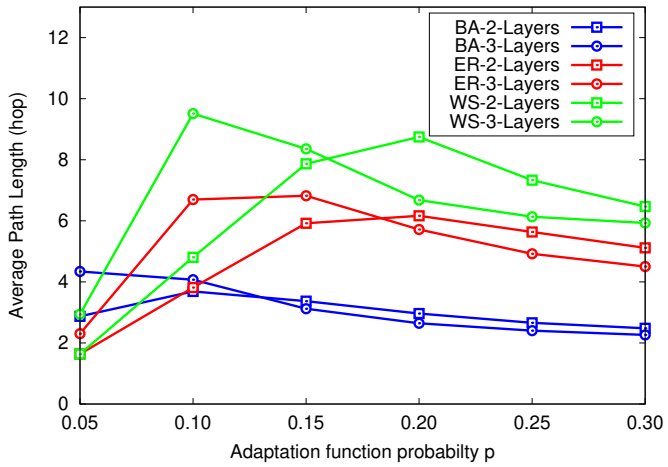


FIGURE 4.6 – Impact des topologies mono-aléatoires sur la longueur des chemins faisables.

Les figures 4.6,4.7,4.8 montrent la longueur moyenne des chemins dans les réseaux multicouches générés par MNTG en fonction de la méthode de génération (mono ou multi), le modèle de graphe aléatoire, le nombre de couches et la probablité p de distribution des fonctions.

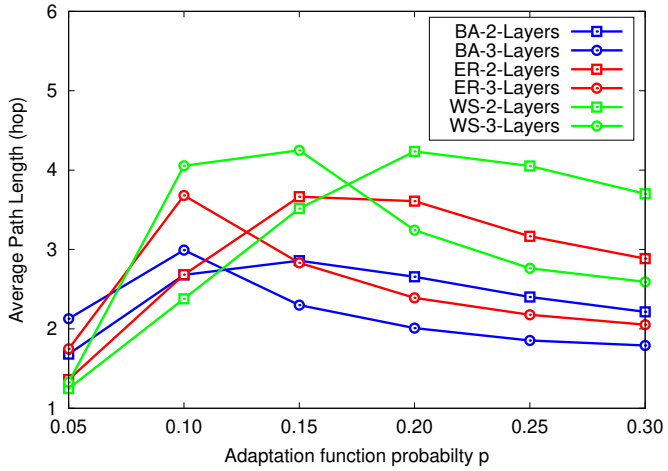


FIGURE 4.7 – Impact des topologies multi-aléatoires sur la longueur des chemins faisables.

Nous pouvons voir que dans le mode mono-aléatoire, la longueur moyenne de chemins varie entre 2 et 4 sauts pour le modèle BA, entre 2 et 7 pour le modèle ER et entre 2 et 10 pour le modèle WS.

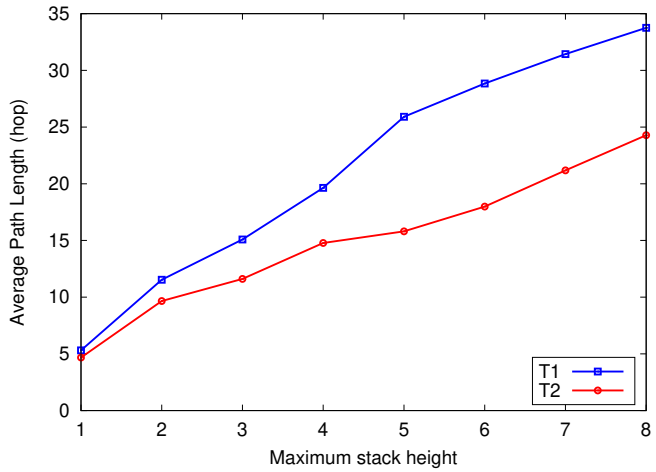


FIGURE 4.8 – Impact des topologies multi-réelles sur la longueur des chemins faisables.

Tandis que dans le mode multi-aléatoire, la longueur moyenne de chemins varie entre 2 et 3 sauts pour le modèle BA, entre 2 et 4 pour le modèle ER et entre 1 et 4 pour le modèle WS. Enfin, pour le mode multi-reél, la longueur moyenne de chemins varie entre 5 et 34 sauts pour la topologie T1 et entre 4 et 24 pour la topologie T2.

Les figures 4.9, 4.10, 4.11 montrent le pourcentage de chemins faisables dans les réseaux multicouche générés par MNTG en fonction de la méthode de génération (mono ou multi), le modèle de graphe aléatoire, le nombre de couches et la probablité p de distribution des fonctions.

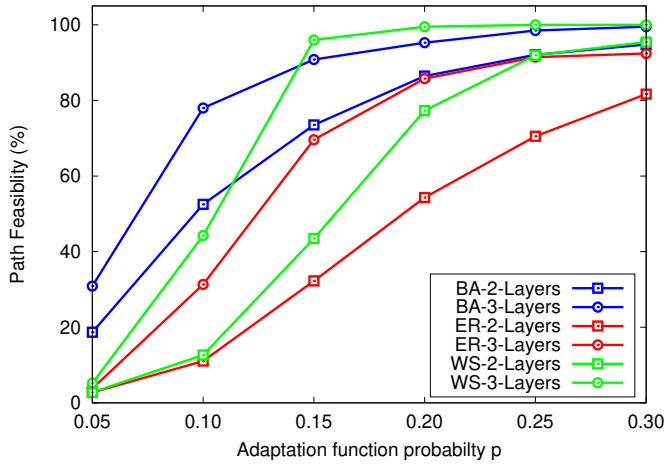


FIGURE 4.9 – Impact des topologies mono-aléatoires sur le pourcentage des chemins faisables.

Nous pouvons voir que dans le mode mono-aléatoire, le pourcentage de chemins faisables varie de 5% à 100% en fonction des différents paramètres.

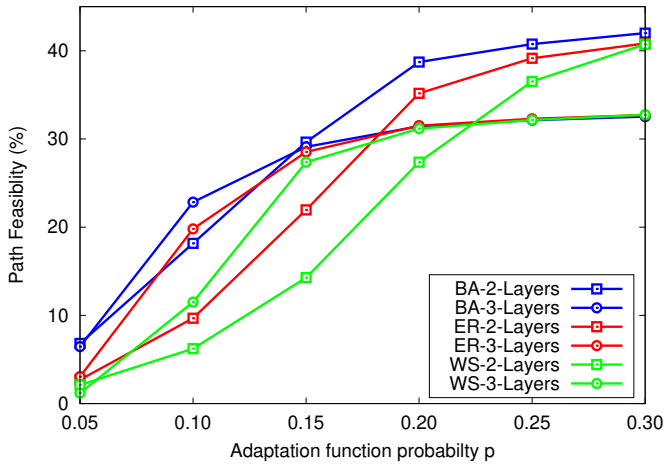


FIGURE 4.10 – Impact des topologies multi-aléatoires sur le pourcentage des chemins faisables.

Tandis que pour le mode multi-aléatoire, le pourcentage de chemins faisables varie de 5% à 40% en fonction des différents paramètres. Enfin, pour le mode multi-reél, le pourcentage de chemins faisables varie de 3% à 40% pour la topologie T1 et de 1% à 59% pour la topologie T2.

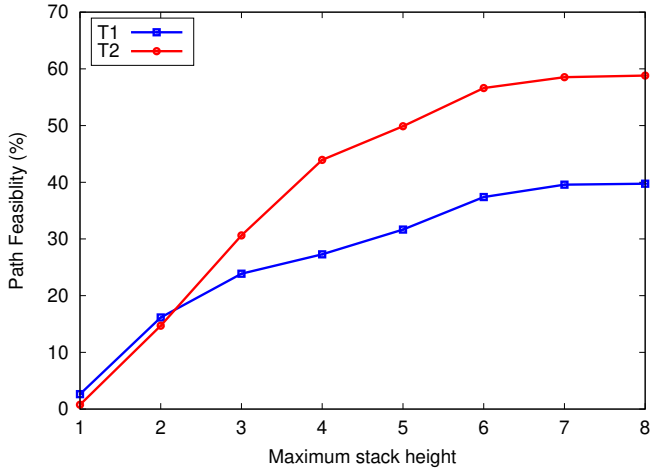


FIGURE 4.11 – Impact des topologies multi-réelles sur le pourcentage des chemins faisables.

Tous ces résultats montrent qu'ils sont fortement influencés par les paramètres du générateur. Par conséquent, il est important de simuler un protocole sur plusieurs types de topologies générées par divers jeux de paramètres pour capturer pleinement son comportement.

4.5 Conclusion

La génération de topologie de réseau est un élément essentiel pour l'évaluation des recherches en matière de réseaux par simulation. Dans ce contexte, tous les modèles de génération de topologie de réseau existants permettant uniquement de générer des topologies de réseau à une seule couche ou avec un seul protocole de communication. Or, les réseaux d'aujourd'hui sont de plus en plus des réseaux hétérogènes et multicouches (*ex.,* les réseaux virtuels, les réseaux IPv4/IPv6, etc.) où plusieurs protocoles de communication cœxistent.

Dans ce chapitre, nous avons proposé un nouveau générateur de topologies de réseau multicouches nommé MNTG permettant de créer des topologies multicouches appropriées à la simulation de protocoles réseau en prenant en compte les conversions et les encapsulations de protocoles. À notre connaissance, nous sommes les premiers à proposer un tel générateur. Il est écrit en ISO C++, libre, open source et disponible à partir d'un dépôt Gitlab de l'Université de Bordeaux. Nous avons présenté à la fois sa structure interne ainsi que ses algorithmes en pseudo-code. Nous avons également évalué ses performances du point de vue de la consommation de ressources et nous avons montré comment les topologies générées ont un impact significatif sur les résultats de simulation du protocole de routage multicouche.

Conclusion générale

La réalisation d'un objectif doit être le point de départ d'un autre.

– Alexander Graham Bell

Principales contributions

Aujourd'hui, la majorité des réseaux de communication sont caractérisés par leur hétérogénéité et leurs multiples couches, où différents protocoles coexistent et où des encapsulations sont fréquentes. Ces dernières permettent la création de tunnels (parfois imbriqués). Les protocoles de routage actuels et leurs algorithmes associés ne permettent pas d'établir automatiquement ces tunnels. Dans cette thèse, nous avons abordé le problème du routage au sein de réseaux hétérogènes et multicouches, en particulier le calcul de chemins avec établissement automatique de tunnels.

Nous avons étudié le calcul de chemins avec fermeture transitive, définissant une opération de concaténation de chemins incluant des tunnels. À partir de cela, nous avons proposé deux algorithmes permettant d'établir les tables de routage, *i.e.,* déterminer le plus court chemin entre toutes les paires de nœuds au sein d'un réseau hétérogène et multicouche. Ces algorithmes peuvent être soit totalement centralisés, soit distribués avec une mémoire partagée, et offrent à la fois des méthodes de routage par sauts et par segments. Tous nos algorithmes ont été prouvés et analysés afin de donner des bornes supérieures à leur complexité.

Dans le but de concevoir un protocole de routage basé sur les algorithmes de routage proposés, nous avons étudié la validation théorique de nos algorithmes par les algèbres de routage. Nous avons proposé la modification des algèbres de routage existants (semi-anneau, algèbre de fonctions et algèbre de Sobrinho) afin de modéliser le problème de routage avec tunnels dans les réseaux hétérogènes et multicouches. En se basant sur cela, nous avons étudié des propriétés de convergence sur les algèbres proposés.

L'évaluation de nos algorithmes de routage par des simulations nous a mené à étudier la génération de topologie de réseaux hétérogènes et multicouches. Nous avons défini trois méthodes de génération de réseaux hétérogènes et multicouches (mono-aléatoire, multi-aléatoires et multi-réels). En se basant sur cela, nous avons proposé un générateur de topologies de réseaux hétérogènes et multicouches.

Perspectives de recherche

Le sujet de routage avec établissement automatique de tunnels étant vaste et encore mal compris, beaucoup de pistes de travaux futurs sont envisagées.

— Dans le chapitre 2, nous avons proposé des algorithmes pour le routage avec tunnels. Néanmoins, nos algorithmes ne peuvent pas être transformés directement en un protocole de routage. Il est d'abord important de définir le format des entêtes, de la table de routage, des messages, des diagrammes d'états, etc. Puis, il faut implémenter les algorithmes dans un simulateur à évènements discrets (*ex.,* ns, OMNeT++, etc.). Une autre piste intéressante serait d'étudier l'adaptation de nos algorithmes pour le routage multicast avec tunnels. Pour cela, la première partie du travail serait de définir et étudier la notion d'arbre dans un réseau multicouche. Ensuite, il faudrait concevoir un algorithme de calcul de tels arbres. Ce calcul étant déjà NP-difficile dans un réseau classique (problème de Steiner), on se concentrera sur des approches approximatives et heuristiques.

— Les algèbres de routage proposés dans le chapitre 3 ont été utilisés dans le cas des problèmes de routage avec une seule métrique additive, en particulier le problème des plus courts chemins avec tunnels. Une perspective intéressante serait de modéliser d'autres problèmes de routage avec des métriques QoS composées (*ex.,* plus courts chemins avec tunnels sous contrainte de bande passante, etc.). De plus, les propriétés de convergence prouvées dans le cadre de cette thèse sont liées à la convergence itérative des algorithmes. Une autre perspective intéressante serait d'utiliser ces modèles pour prouver la convergence des algorithmes distribués asynchrones.

— Dans le chapitre 4, nous avons proposé un générateur de topologies de réseaux hétérogènes et multicouches. Ce dernier permet de générer des topologies de réseaux dont le placement des fonctions (conversions et encapsulations) est aléatoire. Il serait intéressant de proposer une méthode d'optimisation de placement des fonctions. Une telle solution ferait intervenir des algorithmes de placement basés sur la programmation linéaire, l'ensemble dominant, l'apprentissage, etc. Une autre perspective intéressante serait d'utiliser des outils de la combinatoire analytique, en particulier le calcul de la fonction génératrice et la formule analytique pour calculer la probabilité qu'une topologie donnée contienne des chemins faisables selon une certaine distribution de probabilité des fonctions d'adaptation sur les nœuds du réseau.

— Enfin, le modèle de réseaux utilisé et les algorithmes de routage proposés sont très génériques et nécessitent une application sur des cas réels de réseaux hétérogènes et multicouches. Une application naturelle de nos algorithmes est l'interopérabilité IPv4/IPv6. D'autres applications possibles seraient les réseaux IoT et VPN.

Bien que le routage avec établissement automatique de tunnels dans les réseaux hétérogènes et multicouches est un problème dans le domaine des réseaux, il fait intervenir des outils théoriques très variés des mathématiques discrètes. Nous citons :

- En théorie des graphes : la généralisation des algorithmes de calcul de chemins Bellman-Ford [59, 63] et Floyd-Warshall [82, 80, 81], le calcul d'arbre et le problème de Steiner, la génération de cartes de réseaux avec graphes aléatoires [84], le placement des convertisseurs et le problème ensemble dominant [31].
- En théorie des langages : la modélisation avec automate à pile et grammaire algébrique [62, 64, 60].
- En théorie de la complexité : les bornes sur la longueur des chemins faisbles [60, 61], la réduction du problème de calcul de chemins avec tunnels sous contrainte de bande passante vers le problème chemin hamiltonien [60], la réduction du problème de placement de convertisseurs vers le problème SAT [31].
- En combinatoire : la modélisation avec les chemins de Dyck et Motzkin [90, 89].
- En algèbre : la modélisation avec les algèbres tropicales (algèbre min-plus) [85].
- En théorie de l'information : le chiffrement des tunnels.

Publications personnelles

Journaux internationaux

[81] Noureddine Mouhoub, Mohamed Lamine Lamali et Damien Magoni. *A Transitive Closure Algorithm for Routing with Automatic Tunneling*. En cours de révision pour les IEEE/ACM Transactions on Networking (ToN), 2023.

[85] Noureddine Mouhoub, Maria Moloney et Damien Magoni. *Metarouting with Automatic Tunneling in Multilayer Networks*. En cours de révision pour le Journal of Network and Computer Applications (JNCA), 2023.

Conférences internationales

[80] Noureddine Mouhoub, Mohamed Lamine Lamali et Damien Magoni. *A Highly Parallelizable Algorithm for Routing With Automatic Tunneling*. Publié dans les actes du 21st IFIP Networking Conference (Networking), 2022.

[84] Noureddine Mouhoub, Maria Moloney et Damien Magoni. *A New Flexible Multilayer Network Topology Generator*. Publié dans les actes du 1st IEEE Virtual Conference on Communications (VCC), 2023.

Atelier international

[83] Noureddine Mouhoub, Mohamed Lamine Lamali et Damien Magoni. *Semiring Algebraic Structure for Metarouting with Automatic Tunneling*. Publié dans les actes du 1st Workshop on New IP and Beyond of the IEEE International Conference on Network Protocols (ICNP), 2022.

Conférence nationale

[82] Noureddine Mouhoub, Mohamed Lamine Lamali et Damien Magoni. *(Presque) Floyd-Warshall sous stéroïdes : (encore) un algorithme de routage pour l'établissement automatique de tunnels*. Publié dans les actes des 24èmes Rencontres Francophones sur les Aspects Algorithmiques des Télécommunications (AlgoTel), 2022.

Logiciels libres et open source

[78] Noureddine Mouhoub. *Multilayer Network Topology Generator (MNTG)*. 2023. URL : https://gitub.u-bordeaux.fr/hera/random-generator.

[79] Noureddine Mouhoub. *Multilayer Routing Algorithms*. 2022. URL : https://gitub.u-bordeaux.fr/hera/multilayer-routing-algorithms.

Bibliographie

[1] William Aiello, Fan Chung et Linyuan Lu. « A Random Graph Model for Massive Graphs ». In : *ACM STOC*. 2000, 171–180.

[2] Rita Albert et Albert-Laszlo Barabasi. « Topology of Evolving Networks : Local Events and Universality ». In : *Phys. rev. letters* 85 (2000), p. 5234-7.

[3] Albert-László Barabási et Réka Albert. « Emergence of Scaling in Random Networks ». In : *Science* 286.5439 (1999), p. 509-512. issn : 0036-8075.

[4] Albert-László Barabási et Réka Albert. « Emergence of Scaling in Random Networks ». In : *Science* 286.5439 (1999), p. 509-512.

[5] Richard Bellman. « On a routing problem ». In : *Quarterly of applied mathematics* 16.1 (1958), p. 87-90.

[6] Arthur Berger et al. « Internet Nameserver IPv4 and IPv6 Address Relationships ». In : *Proceedings of the 2013 Conference on Internet Measurement Conference*. IMC '13. Barcelona, Spain : Association for Computing Machinery, 2013, 91–104. isbn : 9781450319539.

[7] Carré Bernard. *Graphs and networks / Bernard Carré*. eng. Oxford applied mathematics and computing science series. Oxford New York : Clarendon Press Oxford University Press, 1979. isbn : 0-19-859615-4.

[8] M. Blanchet et F. Parent. *RFC 5572 : IPv6 Tunnel Broker with the Tunnel Setup Protocol (TSP)*. 2010.

[9] CAIDA. *Center for Applied Internet Data Analysis (CAIDA)*. 1997. url : https://www.caida.org/.

[10] B. Carpenter et C. Jung. *RFC2529 : Transmission of IPv6 over IPv4 Domains without Explicit Tunnels*. USA, 1999.

[11] B. Carpenter et K. Moore. *RFC 3056 : Connection of IPv6 Domains via IPv4 Clouds*. 2001.

[12] B. A. Carré. « An Algebra for Network Routing Problems ». In : *IMA Journal of Applied Mathematics* 7.3 (juin 1971), p. 273-294. issn : 0272-4960.

[13] Cisco. *cisco-nexus-9000-series-nx-os-verified-scalability*. 2023. url : https://www.cisco.com/c/en/us/td/docs/switches/datacenter/nexus9000/sw/93x/scalability/guide-938/cisco-nexus-9000-series-nx-os-verified-scalability-guide-938.html.

[14] Cisco ASR 9000 Series Aggregation Services Routers. *Segment Routing Configuration Guide for Cisco ASR 9000 Series Routers, IOS XR Release 7.3.x*. 2023. URL : https://www.cisco.com/c/en/us/td/docs/routers/asr9000/software/asr9k-r7-3/segment-routing/configuration/guide/b-segment-routing-cg-asr9000-73x/b-segment-routing-cg-asr9000-71x_chapter_01000.html.

[15] Dr. Matt Crawford. *Transmission of IPv6 Packets over Ethernet Networks*. RFC 2464. Déc. 1998.

[16] Gábor Csárdi et Tamás Nepusz. « The Igraph Software Package for Complex Network Research ». In : *InterJournal Complex Systems* (2005), p. 1695.

[17] Matthew L. Daggitt et Timothy G. Griffin. « Rate of Convergence of Increasing Path-Vector Routing Protocols ». In : *2018 IEEE 26th International Conference on Network Protocols (ICNP)* (2018), p. 335-345.

[18] Matthew L. Daggitt, Alexander J. T. Gurney et Timothy G. Griffin. « Asynchronous Convergence of Policy-Rich Distributed Bellman-Ford Routing Protocols ». In : SIGCOMM '18. New York, NY, USA : Association for Computing Machinery, 2018, 103–116. ISBN : 9781450355674.

[19] Dell. *Max OF Scaling Numbers in FDB Mode*. 2023. URL : https://dl.dell.com/manuals/all-products/esuprt_ser_stor_net/esuprt_networking/esuprt_net_netwrkng_sw/force10-sw-defined-ntw_deployment%20guide4_en-us.pdf.

[20] E. W. Dijkstra. « A Note on Two Problems in Connexion with Graphs ». In : *Numer. Math.* 1.1 (1959), 269–271. ISSN : 0029-599X.

[21] F. Dijkstra et al. « A multi-layer network model based on ITU-T G.805 ». In : *Comput. Netw.* (2008).

[22] M.B. Doar. « A better model for generating test networks ». In : *IEEE GLOBECOM*. 1996, p. 86-93.

[23] Jason A. Donenfeld. « WireGuard : Next Generation Kernel Network Tunnel ». In : *Network and Distributed System Security Symposium*. 2017.

[24] Seweryn Dynerowicz et Timothy G. Griffin. « On the forwarding paths produced by Internet routing algorithms ». In : *2013 21st IEEE International Conference on Network Protocols (ICNP)*. 2013, p. 1-10.

[25] Nasser El-Aawar et al. *Encapsulation Methods for Transport of Asynchronous Transfer Mode (ATM) over MPLS Networks*. RFC 4717. Déc. 2006.

[26] *Encapsulation Methods for Transport of Frame Relay over Multiprotocol Label Switching (MPLS) Networks*. RFC 4619. Sept. 2006.

[27] P. Erdös et A. Rényi. « On Random Graphs I ». In : *Publicationes Mathematicae Debrecen* 6 (1959), p. 290.

[28] Michalis Faloutsos, Petros Faloutsos et Christos Faloutsos. « On Power-Law Relationships of the Internet Topology ». In : *SIGCOMM CCR* 29.4 (1999), 251–262.

[29] Dino Farinacci et al. *Generic Routing Encapsulation (GRE)*. RFC 1701. 1994.

[30] Clarence Filsfils et al. *SRv6*. 2017. URL : https://www.segment-routing.net/tutorials/2017-12-05-srv6-introduction/.

[31] Rohan Fossé, Mohamed Lamine Lamali et Paul Ouvrard. « SAT EST IN BONO LOCO Oui mais combien ? et où ? » In : *ALGOTEL 2020–22èmes Rencontres Francophones sur les Aspects Algorithmiques des Télécommunications*. 2020.

[32] Robert E. Gilligan et Erik Nordmark. *Basic Transition Mechanisms for IPv6 Hosts and Routers*. RFC 4213. 2005.

[33] M. Gondran. « Algèbre linéaire et cheminement dans un graphe ». fr. In : *RAIRO - Operations Research - Recherche Opérationnelle* 9.V1 (1975), p. 77-99.

[34] M. Gondran et M. Minoux. *Graphes et algorithmes*. Paris, 3e édition : Eyrolles, 1995.

[35] R. Govindan et H. Tangmunarunkit. « Heuristics for Internet map discovery ». In : *IEEE INFOCOM*. T. 3. 2000, p. 1371-1380.

[36] Graphviz. *graph visualization software (Graphviz)*. URL : https://graphviz.org/.

[37] Timothy G. Griffin et Joäo Luís Sobrinho. « Metarouting ». In : *Proceedings of the 2005 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*. SIGCOMM '05. New York, NY, USA : Association for Computing Machinery, 2005, 1–12. ISBN : 1595930094.

[38] Alexander J. T. Gurney et Timothy G. Griffin. « Lexicographic products in metarouting ». In : *2007 IEEE International Conference on Network Protocols*. 2007, p. 113-122.

[39] John Charles Harden. « Direct and Semidirect Products of Semigroups ». Mém. de mast. Knoxville, 1949.

[40] U. Hebisch et H.J. Weinert. *Semirings : Algebraic Theory and Applications in Computer Science*. Series in algebra. World Scientific, 1998. ISBN : 9789810236014.

[41] Charles Hornig. *A Standard for the Transmission of IP Datagrams over Ethernet Networks*. RFC 894. Avr. 1984.

[42] HPE. *OpenFlow Configuration*. 2023. URL : https://techhub.hpe.com/eginfolib/networking/docs/switches/5940/5200-1028b_openflow_cg/content/491966867.htm#370558467.

[43] Huawei. *segment-routing-configuration*. 2023. URL : https://support.huawei.com/enterprise/fr/doc/EDOC1100138131/ceb6d155/segment-routing-configuration-commands.

[44] Christian Huitema. *Teredo : Tunneling IPv6 over UDP through Network Address Translations (NATs)*. RFC 4380. Fév. 2006.

[45] « IEEE Standard for information technology-Telecommunications and information exchange between systems-Local and metropolitan area networks-Specific requirements-Part 11 : Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications : Amendment 6 : Medium Access Control (MAC) Security Enhancements ». In : *IEEE Std 802.11i-2004* (2004), p. 1-190. DOI : 10.1109/IEEESTD.2004.94585.

[46] « IEEE Standard for Local and Metropolitan Area Network–Bridges and Bridged Networks ». In : *IEEE Std 802.1Q-2018 (Revision of IEEE Std 802.1Q-2014)* (2018), p. 1-1993. DOI : 10.1109/IEEESTD.2018.8403927.

[47] « IEEE Standard for Local and metropolitan area networks – Virtual Bridged Local Area Networks Amendment 7 : Provider Backbone Bridges ». In : *IEEE Std 802.1ah-2008 (Amendment to IEEE Std 802.1Q-2005)* (2008), p. 1-110. DOI : 10.1109/IEEESTD.2008.4602826.

[48] « IEEE Standard for Local and Metropolitan Area Networks—Virtual Bridged Local Area Networks—Amendment 4 : Provider Bridges ». In : *IEEE Std 802.1ad-2005 (Amendment to IEEE Std 802.1Q-2005)* (2006), p. 1-74. DOI : 10.1109/IEEESTD.2006.6044678.

[49] Farabi Iqbal, Jeroen van der Ham et Fernando Kuipers. « Technology-aware Multi-domain Multi-layer Routing ». In : *Comput. Commun.* 62.C (mai 2015), p. 85-96. ISSN : 0140-3664.

[50] ISO. *Information processing systems – OSI reference model, international standards organization*. Rapp. tech. 7498, ISO. 1984.

[51] Bo Jiang et al. « On a Class of Stochastic Multilayer Networks ». In : *ACM Meas. Anal. Comput. Syst.* (2018).

[52] Cheng Jin, Qian Chen et Sugih Jamin. « Inet : Internet Topology Generator ». In : *University of Michigan, Tech. Rep. CSE-TR-433-00* (2000).

[53] Juniper. *OpenFlow User Guide*. 2023. URL : https://www.juniper.net/documentation/us/en/software/junos/sdn-openflow/topics/ref/command/show-openflow-statistics-tables.html.

[54] Juniper Networks TechLibrary. *source-packet-routing*. 2023. URL : https://www.juniper.net/documentation/us/en/software/junos/bgp/topics/ref/statement/autogen-protocols-source-packet-routing.html.

[55] Yasushi Kawase, Atsushi Miyauchi et Hanna Sumita. « Stochastic Solutions for Dense Subgraph Discovery in Multilayer Networks ». In : *Proceedings of the Sixteenth ACM International Conference on Web Search and Data Mining*. 2023, 886–894.

[56] Hussein Khayou et Bakr Sarakbi. « A Validation Model for Non-Lexical Routing Protocols ». In : *Journal of Network and Computer Applications* 98.C (2017), 58–64. issn : 1084-8045.

[57] Jakob Krarup et Peter Mark Pruzan. « The simple plant location problem : Survey and synthesis ». In : *European Journal of Operational Research* 12.1 (1983), p. 36-81. issn : 0377-2217.

[58] Fernando Kuipers et Freek Dijkstra. « Path Selection in Multi-layer Networks ». In : *Comput. Commun.* 32.1 (jan. 2009), p. 78-85. issn : 0140-3664.

[59] M. L. Lamali et al. « A stack-vector routing protocol for automatic tunneling ». In : *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*. 2019, p. 1675-1683.

[60] Mohamed Lamine Lamali, Nasreddine Fergani et Johanne Cohen. « Algorithmic and complexity aspects of path computation in multi-layer networks ». In : *IEEE/ACM Transactions on Networking* 26.6 (2018).

[61] Mohamed Lamine Lamali, Nasreddine Fergani et Johanne Cohen. « Quelques bornes sur les chemins dans les réseaux multicouches ». In : *ALGOTEL 2019-21èmes Rencontres Francophones sur les Aspects Algorithmiques des Télécommunications*. 2019.

[62] Mohamed Lamine Lamali, Hélia Pouyllau et Dominique Barth. « Path computation in multi-layer multi-domain networks : A language theoretic approach ». In : *Computer Communications* 36.5 (2013), p. 589-599.

[63] Mohamed Lamine Lamali et al. « Bellman-Ford sous stéroïdes : Un algorithme de routage pour l'établissement automatique des tunnels ». In : *ALGOTEL 2020–22èmes Rencontres Francophones sur les Aspects Algorithmiques des Télécommunications*. 2020.

[64] Mohamed Lamine Lamali et al. « Path computation in multi-layer networks : Complexity and algorithms ». In : *IEEE INFOCOM 2016*. 2016, p. 1-9.

[65] Marc Lasserre et Vach Kompella. *Virtual Private LAN Service (VPLS) Using Label Distribution Protocol (LDP) Signaling*. RFC 4762. 2007.

[66] Tony Li et al. *Generic Routing Encapsulation (GRE)*. RFC 2784. 2000.

[67] Chris M. Lonvick et Tatu Ylonen. *The Secure Shell (SSH) Protocol Architecture*. RFC 4251. 2006.

[68] Damien Magoni. « Network Topology Analysis and Internet Modelling with Nem ». In : *International Journal of Computers and Applications* 27.4 (2005), p. 252-259.

[69] Damien Magoni et Jean-Jacques Pansiot. « Influence of Network Topology on Protocol Simulation ». In : *Int. Conference on Networking*. 2001, p. 762-770.

[70] Priya Mahadevan et al. « Orbis : Rescaling Degree Correlations to Generate Annotated Internet Topologies ». In : *SIGCOMM CCR* 37.4 (2007), 325–336.

[71] Eric Mannie. *Generalized Multi-Protocol Label Switching (GMPLS) Architecture*. RFC 3945. Nov. 2004.

[72] Luca Martini et al. *Encapsulation Methods for Transport of Ethernet over MPLS Networks*. RFC 4448. Avr. 2006.

[73] Jim McManus et al. *Requirements for Traffic Engineering Over MPLS*. RFC 2702. 1999.

[74] Alberto Medina, Ibrahim Matta et John Byers. « On the Origin of Power Laws in Internet Topologies ». In : *SIGCOMM CCR* 30.2 (2000), 18–28.

[75] Michel Minoux. *Graphes, Dioides et Semi-Anneaux : Nouveaux modèles et algorithmes*. TEC and DOC, 2001.

[76] Michel Minoux et Michel Gondran. *Graphs, Dioids and Semirings. New Models and Algorithms*. T. 41. Operations Research/Computer Science Interfaces Series. Springer, 2008.

[77] Mehryar Mohri. « Semiring Frameworks and Algorithms for Shortest-Distance Problems ». In : *J. Autom. Lang. Comb.* 7.3 (2002), 321–350. issn : 1430-189X.

[86] Nokia. *segment-routing-configuration*. 2023. url : https://infocenter.nokia.com/public/7750SR160R4A/index.jsp?topic=%2Fcom.sr.mpls%2Fhtml%2Fmpls_cli.html.

[87] C.R. Palmer et J.G. Steffan. « Generating network topologies that obey power laws ». In : *IEEE GLOBECOM*. 2000, p. 434-438.

[88] Zhisong Pan, Guyu Hu et Dong Li. « Detecting Communities from Multilayer Networks : An Aggregation Approach ». In : *Proceedings of the International Conference on Intelligent Science and Technology*. 2018, 6–11.

[89] Élie de Panafieu, Mohamed Lamine Lamali et Michael Wallner. « De la probabilité de creuser un tunnel ». In : *ALGOTEL 2019-21èmes Rencontres Francophones sur les Aspects Algorithmiques des Télécommunications*. 2019.

[90] Élie de Panafieu, Mohamed Lamine Lamali et Michael Wallner. « Combinatorics of nondeterministic walks of the dyck and motzkin type ». In : *2019 Proceedings of the Sixteenth Workshop on Analytic Algorithmics and Combinatorics (ANALCO)*. SIAM. 2019, p. 1-12.

[91] Prayson Pate et Stewart Bryant. *Pseudo Wire Emulation Edge-to-Edge (PWE3) Architecture*. RFC 3985. Mars 2005. doi : 10.17487/RFC3985. url : https://www.rfc-editor.org/info/rfc3985.

[92] Charles E. Perkins. *IP Encapsulation within IP*. RFC 2003. Oct. 1996.

[93] Adrian Popiel, Przemysław Kazienko et Tomasz Kajdanowicz. « MuNeG : The Framework for Multilayer Network Generator ». In : *Proceedings of the IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*. 2015, 1316–1323.

[94] B. Quoitin. *Towards more representative internet topologies*. Rapp. tech. Université Catholique de Louvain, 2010.

[95] Eric Rescorla. *The Transport Layer Security (TLS) Protocol Version 1.3*. RFC 8446. Août 2018. doi : 10.17487/RFC8446. url : https://www.rfc-editor.org/info/rfc8446.

[96] Max Riegel. *Requirements for Edge-to-Edge Emulation of Time Division Multiplexed (TDM) Circuits over Packet Switching Networks*. RFC 4197. Oct. 2005.

[97] Karen Seo et Stephen Kent. *Security Architecture for the Internet Protocol*. RFC 4301. Déc. 2005.

[98] Himanshu C. Shah et al. *IP-Only LAN Service (IPLS)*. RFC 7436. 2015.

[99] W. Simpson. *IP in IP Tunneling*. RFC 1853. Oct. 1995.

[100] J.L. Sobrinho. « An algebraic theory of dynamic network routing ». In : *IEEE/ACM Transactions on Networking* 13.5 (2005), p. 1160-1173.

[101] João Luis Sobrinho. « Network Routing with Path Vector Protocols : Theory and Applications ». In : *Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*. SIGCOMM '03. New York, NY, USA : Association for Computing Machinery, 2003, 49–60. isbn : 1581137354.

[102] João Luís Sobrinho et Miguel Alves Ferreira. « Routing on Multiple Optimality Criteria ». In : SIGCOMM '20. New York, NY, USA : Association for Computing Machinery, 2020, 211–225. isbn : 9781450379557.

[103] *TDM-IP interworking – User plane interworking*. ITU-T Recommendation Y.1453. Mars 2006. url : https://www.itu.int/rec/T-REC-Y.1453-200603-I/en.

[104] Fred Templin. *RFC 5579 : Transmission of IPv4 Packets over Intra-Site Automatic Tunnel Addressing Protocol (ISATAP) Interfaces*. 2010.

[105] G. Tsirtsis et P. Srisuresh. *RFC2766 : Network Address Translation - Protocol Translation (NAT-PT)*. USA, 2000.

[106] Andrew J. Valencia et al. *Layer Two Tunneling Protocol "L2TP"*. RFC 2661. 1999.

[107] Piet Van Mieghem et Fernando A Kuipers. « Concepts of exact QoS routing algorithms ». In : *IEEE/ACM Transactions on networking* 12.5 (2004), p. 851-864.

[108] Route Views. *University of Oregon RouteViews Project*. url : http://www.routeviews.org/routeviews/.

[109] Stephen Warshall. « A Theorem on Boolean Matrices ». In : *J. ACM* 9.1 (jan. 1962), p. 11-12. issn : 0004-5411.

[110] Duncan J. Watts et Steven H. Strogatz. « Collective dynamics of 'small-world' networks ». In : *Nature* 393.6684 (1998), p. 440-442.

[111] B.M. Waxman. « Routing of multipoint connections ». In : *IEEE Journal on Selected Areas in Communications* 6.9 (1988), p. 1617-1622.

[112] E.W. Zegura, K.L. Calvert et S. Bhattacharjee. « How to model an internetwork ». In : *IEEE INFOCOM*. T. 2. 1996, p. 594-602.

[113] E.W. Zegura, K.L. Calvert et M.J. Donahoo. « A quantitative comparison of graph-based models for Internet topology ». In : *IEEE/ACM Transactions on Networking* 5.6 (1997), p. 770-783.

[114] Beichuan Zhang et al. « Collecting the Internet AS-Level Topology ». In : *SIGCOMM Comput. Commun. Rev.* 35.1 (2005), 53–61. issn : 0146-4833.

[115] Qinhua Zheng et al. « A New Worm Exploiting IPv4-IPv6 Dual-Stack Networks ». In : *Proceedings of the 2007 ACM Workshop on Recurring Malcode*. WORM '07. Alexandria, Virginia, USA : Association for Computing Machinery, 2007, 9–15. isbn : 9781595938862.

[116] Glen Zorn, Gurdeep-Singh Pall et Kory Hamzeh. *Point-to-Point Tunneling Protocol (PPTP)*. RFC 2637. 1999.

Algorithmes pour le routage dans les réseaux multicouches

Résumé : Les réseaux hétérogènes et multicouches sont des réseaux où différents protocoles coexistent et où des encapsulations ont lieu. Ces réseaux sont de plus en plus répandus (coexistence IPv4/IPv6, tunnels IPsec, TLS, etc.). Plus généralement, n'importe quel réseau permettant l'établissement de tunnels peut être considéré comme un réseau hétérogène ou multicouche. De nos jours, les tunnels sont omniprésents dans les réseaux. Or, les modèles basés sur les graphes et les algorithmes qui en découlent ne sont pas adaptés à la prise en compte des tunnels. Il se trouve que des problèmes simples dans un réseau classique, comme par exemple le calcul de chemins et le routage, deviennent très difficiles dans un réseau multicouche (le calcul d'un chemin sous contrainte de bande passante est trivialement polynomial dans un réseau classique, mais est NP-difficile dans un réseau multicouche, de même, la longueur d'un chemin dans un tel réseau peut être exponentielle). Ces problèmes sont encore très mal compris et il est nécessaire de développer de nouveaux modèles et de nouveaux algorithmes pour les résoudre. Le but de cette thèse est d'étudier les problèmes de communication dans de tels réseaux, en particulier le calcul de chemins centralisé et distribué, la communication multipoint via le calcul d'arbres de diffusion, ainsi que l'adaptation de ces algorithmes en protocoles de routage. Cette thèse s'inscrit dans le cadre du projet ANR JCJC HÉtérogénéité et algorithmes de RoutAge dans les réseaux (HÉRA), dont le but est de mieux comprendre les fondements théoriques structurels et algorithmiques des réseaux multicouches.
Mots-clés : Réseaux multicouches, Tunneling, Algorithmes de routage, Algèbres de routage

---

Algorithms for routing in multilayer networks

Abstract: Heterogeneous and multilayer networks are networks where different protocols coexist and where encapsulations take place. These networks are becoming more widespread (IPv4/IPv6 coexistence, IPsec tunnels, TLS, etc.). More generally, any network allowing the establishment of tunnels can be considered as an heterogeneous or multilayer network. Today, tunnels are ubiquitous in networks. However, the models and the algorithms that can be applied on them are not adapted to take tunnels into account. It turns out that simple problems in a conventional network, such as path computation and routing, become very difficult in a multilayer network (the computation of a path under bandwidth constraint is trivially polynomial in a conventional network, but is NP-hard in a multilayer network. Similarly, the length of a path in such a network can be exponential). These problems are not well understood and it is necessary to develop new models and new algorithms to solve them. The aim of this thesis is to study the communication problems in such networks, in particular the centralized and distributed path computation, the multicast communication via the computation of diffusion trees, as well as the adaptation of these algorithms in routing protocols. This thesis is part of the ANR project JCJC HEterogeneity and Routing Algorithms in networks (HERA), whose goal is to better understand the theoretical structural and algorithmic foundations of multilayer networks.
Keywords: Multilayer networks, Tunneling, Routing algorithms, Routing algebras

---

LaBRI - UMR 5800 - CNRS
Université de Bordeaux, 33405 Talence, France.