

THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ FÉDÉRALE TOULOUSE MIDI-PYRÉNÉES

Délivré par :

l'Institut National des Sciences Appliquées de Toulouse (INSA de Toulouse)

Présentée et soutenue le *04/07/2018* **par :**

ARMEL FRANCKLIN SIMO TEGUEU

Vers les réseaux guidés par et pour les applications hautement dynamiques

---

JURY

NAZIM AGOULMINE	Professeur d'Université	Rapporteur
TOUFIK AHMED	Professeur d'Université	Rapporteur
GUILLAUME URVOY-KELLER	Professeur d'Université	Examinateur
CHRISTOPHE CHASSOT	Professeur d'Université	Examinateur
THIERRY VILLEMUR	Professeur d'Université	Directeur de thèse
SLIM ABDELLATIF	Maître de Conférences	Directeur de thèse

École doctorale et spécialité :

EDSYS : Informatique 4200018

Unité de Recherche :

Laboratoire d'analyse et d'architecture des systèmes

Directeur(s) de Thèse :

Thierry VILLEMUR et *Slim ABDELLATIF*

Rapporteurs :

Nazim AGOULMINE et *Toufik AHMED*

Résumé

Les applications modernes s'appuient sur des architectures qui combinent de plus en plus de composants logiciels émanant de plusieurs éditeurs, implantant des fonctions de plus en plus spécialisées, et très souvent déployés dans le « cloud ». De ce fait, ces applications nécessitent une dynamique et une adaptation certaines vis-à-vis des flux de données que leurs composants échangent et des besoins de qualité de service (QdS) que ces flux requièrent. Pour la majorité des applications, il s'avère difficile d'identifier à l'avance l'ensemble des flux et/ou d'exprimer précisément les besoins de QdS associés. Ainsi, fournir un service de communication réseau capable de répondre et de suivre les besoins de ces applications sans gaspiller, par surdimensionnement, l'utilisation des ressources réseau, pose plusieurs défis aux réseaux de communication supports, notamment un haut degré de flexibilité, largement au-delà des possibilités des réseaux de communication actuels.

L'objectif de ce travail de thèse est de développer le concept de réseau guidé par les applications (ADN : Application Driven Networking), réseau capable d'offrir des services de communication personnalisés et dynamiques aux applications. Le qualificatif personnalisé signifie que le service ADN répond à des besoins de communication applicatifs exprimés avec un niveau de granularité très fin pouvant aller jusqu'aux flux élémentaires. Ces besoins peuvent être exprimés explicitement par l'application ou inférés par le réseau par analyse du trafic. L'aspect dynamique signifie que le service ADN est reprogrammé et ajusté pour suivre l'évolution des besoins de l'application dans le temps.

Les contributions de ce travail de thèse couvrent plusieurs points. Nous avons défini l'architecture générale d'un réseau ADN bâti sur une infrastructure réseau de type SDN (Software Defined Network) en explicitant ses composants fonctionnels et en spécifiant les interfaces entre composants. Nous avons développé les algorithmes de ses principaux composants, notamment deux algorithmes d'allocation de ressources réseau qui calculent les chemins de données et les ressources réseau à y réserver pour satisfaire les exigences de bande passante et de délai des services ADN, tout en optimisant l'utilisation de ressources. Nous avons mis au point deux heuristiques de migration des services ADN afin de répartir aux mieux la charge du réseau et d'augmenter l'admissibilité des requêtes des services ADN à venir. Nous avons développé un prototype démonstrateur de réseau ADN qui fournit un ensemble de services ADN à des applications dynamiques basées sur le middleware temps réel DDS (Data Distribution Service).

Mots clés : Applications modernes, Qualité de Service dynamique, réseaux définis par logiciel, réseaux guidés par les applications

Abstract

Modern applications are typically composed of lots of software components that tend to implement self-contained specialized functions. These components are often supplied by many software editors and provisioned and accessed via the cloud. As a consequence, the data flows that are exchanged between applications' components and their QoS requirements vary over time. Moreover, in many situations, it is quite difficult to pre-identify this set of data flows and/or express precisely the associated QoS. Hence, providing a network service that meets application requirements and dynamically evolves with their needs without uselessly wasting network resources poses several challenges to the underlying communication network. Notably, the necessity of a high level of flexibility, far beyond the capabilities of today's communication networks.

The aim of this PhD is to develop the Application Driven Networking (ADN) concept, which is able to provide tailored and dynamic network services to applications. Tailored means that the ADN service captures a fine-grained description of application needs, which can consider elementary flows. These needs can be expressed, either, explicitly by the application or inferred by the network by traffic analysis. The dynamic facet means that the ADN service is reprogrammed and adjusted to fit to evolving application needs.

The main contributions of this thesis are the following. First, a general architecture of the ADN network built on top of a Software Defined Network (SDN) infrastructure is proposed. Algorithms related to the ADN functional components are also proposed, in particular two network resource allocation algorithms that calculate the optimal (in terms of network resource utilization) data paths and the required network resources that meet application requirements. Two ADN service migration heuristics are also proposed to efficiently distribute the network load and increase the acceptance of forthcoming network service requests. An ADN network prototype is developed as proof of concept. It provides ADN services to dynamic applications with QoS requirements built on top of the DDS (Data Distribution Service) middleware.

Keywords : modern applications, dynamic Quality of Service, Software-Defined Networks, Application Driven Networking

Table des matières

Résumé **i**

Abstract **iii**

Introduction générale **1**

1 Notions préliminaires, État de l'art & Positionnement **5**
1.1 Software-Defined Networking 5
1.2 Le protocole OpenFlow 6
1.3 L'interface Nord 10
1.4 Problématique 11
1.4.1 La portée de la problématique 12
1.4.2 Les constats 13
1.5 État de l'art des solutions existantes 14
1.5.1 Taxonomie 14
1.5.2 Quelques travaux connexes existants dans la littérature 16
1.6 Notre approche générale 24
1.6.1 Notre vision 24
1.6.2 Définition d'un réseau ADN et positionnement 24
1.7 Conclusion 26

2 Architecture générale d'un réseau ADN **29**
2.1 Conception d'un réseau ADN 29
2.1.1 Les applications et les opérateurs au centre de la conception 29
2.1.2 Principes de conception 31
2.2 Architecture générale du réseau ADN 33
2.3 Architecture fonctionnelle du réseau ADN 34
2.3.1 Request Handler 36
2.3.2 Flow Aggregator 38
2.3.3 Virtual NETwork Resource Allocator 40
2.3.4 Virtual NETwork Deployer 41
2.3.5 Application Classifier 45
2.3.6 Autonomic Manager 47
2.4 Synthèse 51
2.5 Conclusion 52

3 Algorithme d'allocation de ressources **53**
3.1 Problèmes d'allocation de ressources dans un contexte de virtualisation réseau . . 54
3.1.1 L'allocation de ressources pour des réseaux virtuels 54
3.1.2 Cas particulier de l'allocation de ressources pour des liens virtuels 54

3.1.3 Les requêtes . . . 55
3.1.4 Les différents aspects des problèmes d'allocation de ressources pour réseaux virtuels . . . 55
3.2 État de l'art de la littérature existante . . . 58
3.2.1 L'allocation de ressources pour des réseaux virtuels . . . 58
3.2.2 L'allocation de ressources pour des liens virtuels . . . 59
3.2.3 Synthèse . . . 59
3.3 Positionnement de notre problème . . . 61
3.3.1 Définition . . . 61
3.3.2 Hypothèses . . . 61
3.4 Modélisation . . . 63
3.4.1 Données en entrée . . . 63
3.4.2 Variables . . . 64
3.4.3 Contraintes . . . 65
3.4.4 Fonction objective . . . 66
3.4.5 Variation sur le calcul de la fonction objective . . . 66
3.4.6 Apport du modèle par rapport à l'existant . . . 67
3.5 Modèle linéaire . . . 67
3.5.1 Variables . . . 67
3.5.2 Contraintes . . . 68
3.5.3 Fonction objective . . . 69
3.5.4 Complexité . . . 70
3.6 Évaluation expérimentale . . . 70
3.6.1 Modèle de simulation considéré . . . 70
3.6.2 Métriques de performance . . . 74
3.6.3 Méthodes heuristiques existantes . . . 75
3.6.4 Résultats expérimentaux sur le réseau GÉANT . . . 76
3.6.5 Résultats expérimentaux sur le réseau de Campus . . . 86
3.7 Conclusion . . . 94

4 Mécanismes de défragmentation de ressources 95
4.1 Description contextuelle du problème . . . 96
4.2 Problème de fragmentation de ressources dans un contexte de virtualisation réseau 97
4.2.1 Sous-problème de sélection et réallocation des réseaux virtuels . . . 98
4.2.2 Les différents aspects du problème de fragmentation de ressources . . . 99
4.3 État de l'art de la littérature existante . . . 100
4.3.1 Taxonomie des approches de défragmentation de ressource . . . 100
4.3.2 Synthèse des travaux existants . . . 101
4.4 Définition du problème . . . 105
4.4.1 Le réseau physique . . . 105
4.4.2 Les allocations d'un lien virtuel actif . . . 105
4.4.3 Objectif de la défragmentation . . . 106
4.5 Mécanisme de défragmentation proactif . . . 106
4.5.1 Contrôle des reconfigurations . . . 106

4.5.2 Sélection des liens virtuels candidats à la migration . . . 107
4.5.3 Réallocation des liens virtuels sélectionnés . . . 110
4.6 Mécanisme de défragmentation réactif . . . 111
4.6.1 Pré-sélection des liens virtuels candidats à la migration . . . 112
4.6.2 Sélection et Réallocation . . . 112
4.7 Évaluation expérimentale . . . 113
4.7.1 Modèle de simulation considéré . . . 114
4.7.2 Métriques de performance . . . 116
4.7.3 Méthodes existantes . . . 118
4.7.4 Résultats expérimentaux sur le réseau GÉANT . . . 118
4.7.5 Résultats expérimentaux sur le réseau de Campus . . . 123
4.8 Conclusion . . . 123

Conclusion générale **127**

A Mécanismes de défragmentation de ressources **131**
A.1 Résultats expérimentaux sur le réseau de Campus . . . 131

B Démonstrateur ADN **135**
B.1 Introduction . . . 135
B.2 Application dynamique DDS considérée . . . 136
B.2.1 Description des topics . . . 136
B.2.2 Composante dynamique de l'application considérée . . . 137
B.3 Infrastructure réseau considérée . . . 138
B.4 Scénario applicatif et résultats obtenus . . . 139
B.4.1 Description du scénario applicatif . . . 139
B.4.2 Résultats du démonstrateur . . . 141
B.5 Conclusion . . . 142

Bibliographie **143**

Table des figures

1 Priorités des administrateurs réseaux (Source : ZK Research 2014) 2

1.1 Architectures réseau traditionnelle et architecture d'un Software Defined Network (inspiré de [Nunes 2014]) 6
1.2 Architecture de SDN vue par l'ONF [ONF 2013c], vision classique et haut-niveau de l'architecture SDN. 7
1.3 Composants d'un commutateur OpenFlow [ONF 2013a] 7
1.4 Pipeline de traitement d'un paquet dans un commutateur OpenFlow [ONF 2013a] 8
1.5 Aperçu des interfaces Nord avec leur niveau d'abstraction [ONF 2013b] 10
1.6 Environnements réseau distincts à considérer (Source : Cisco) 13
1.7 Taxonomie 16
1.8 Vue haut niveau de l'approche ADN 25

2.1 Cas d'utilisation du système ADN 30
2.2 Architecture générale ADN 34
2.3 Choix de conception relatifs à l'application de provisionnement 35
2.4 Architecture fonctionnelle ADN 35
2.5 Diagramme de composants de l' *ADN Service Provisioning* 36
2.6 L'API Nord du réseau ADN 37
2.7 Modèle de service ADN et interface de service du *Request Handler* 38
2.8 Fonctionnement du système lors d'une requête de « création de service » 39
2.9 Interface fournie par le *VNET Resource Allocator* et modèle d'un chemin de données 40
2.10 Interface fournie par le *VNETDeployer* 41
2.11 Interface fournie par l'*Autonomic Manager* 48
2.12 Boucle de contrôle autonomique MAPE-K (IBM 2006) 49
2.13 Relation entre les entités manipulées par le système 51

3.1 Modèles de réseau considérés 71
3.2 Comparaison de différents algorithmes d'allocation 77
3.3 Influence du path splitting et du type de requête sur le temps de calcul 77
3.4 Taux d'acceptation et d'utilisation des liens en fonction du temps pour des requêtes Multicast et Unicast -$\beta = 150$, $PS_{ratio} = 0.45$ 78
3.5 Taux d'acceptation des requêtes de type **unicast** en fonction de leur taux d'arrivée et des paramètres de simulation 80
3.6 Taux d'acceptation des requêtes de type **multicast** en fonction de leur taux d'arrivée et des paramètres de simulation 81
3.7 Charge du réseau en fonction du taux d'arrivée pour des réquêtes de type unicast et multicast - $\beta = 150$ 82
3.8 Fonction de répartition du taux d'utilisation des liens et des nœuds pour des réquêtes de type unicast et multicast - $\beta = 150, PS_{ratio} = 45\%$ 84

3.9 Comparaison de différents algorithmes d'allocation 87
3.10 Influence du path splitting et du type de requête sur le temps de calcul 87
3.11 Taux d'acceptation et d'utilisation des liens en fonction du temps pour des requêtes Multicast et Unicast -$\alpha = 150$, $PS_{ratio} = 0.45$ 88
3.12 Taux d'acceptation des requêtes de type **unicast** en fonction de leur taux d'arrivée et des paramètres de simulation 90
3.13 Taux d'acceptation des requêtes de type **multicast** en fonction de leur taux d'arrivée et des paramètres de simulation 91
3.14 Charge du réseau en fonction du taux d'arrivée pour des réquêtes de type unicast et multicast - $\alpha = 150$ 92
3.15 Fonction de répartition du taux d'utilisation des liens et nœuds pour des réquêtes de type unicast et multicast - $\alpha = 150, PS_{ratio} = 45\%$ 93

4.1 Illustration 96
4.2 Taxonomie des approches de défragmentation de ressources 99
4.3 Structure générale d'un mécanisme de défragmentation de ressources 100
4.4 Comparaison de différentes méthodes de défragmentation 119
4.5 Taux utilisation maximal en fonction du temps et du type de requêtes - $\lambda = 0.04$ 120
4.6 Charge du réseau en fonction du temps et du type de requêtes - $\lambda = 0.04$ 121
4.7 Taux d'acceptation des requêtes en fonction de leur taux d'arrivée et des paramètres de simulation. 122
4.8 Nombre total de reconfigurations en fonction du taux d'arrivée des requêtes et des paramètres de simulation. 124

A.1 Comparaison de différents algorithmes de défragmentation 131
A.2 Taux d'acceptation des requêtes en fonction de leur taux d'arrivée et des paramètres de simulation. 132
A.3 Nombre total de reconfigurations en fonction du taux d'arrivée des requêtes et des paramètres de simulation. 133

B.1 Application de simulation considérée 135
B.2 Topics DDS relatifs à l'application considérée 137
B.3 Topologie du réseau de campus du LAAS 138
B.4 Plateforme mise en place 139
B.5 Illustation du scénario à l'intant t0 140
B.6 Illustation du scénario à l'intant t1 141
B.7 Résultats allocations à l'instant t0 141
B.8 Résultats allocations à l'instant t1 141

Liste des tableaux

1.1 Composants d'une entrée de la table de flux dans un commutateur OpenFlow . . 8
1.2 Composants d'une entrée de la table de groupe dans un commutateur OpenFlow 8
1.3 Composants d'une entrée de la table de mesure (en haute) et bande de mesure qui la compose (en bas) 9

3.1 Récapitulatif des différents types d'allocation et contraintes étudiés par chaque article 60

4.1 Classification des approches de défragmentation de ressource 104
4.2 MSS-MRU en action 110
4.3 Modèle de réseau et de requête 114

B.1 Caractérisation de la dynamique de l'application 138

Introduction générale

Contexte et Motivation

Depuis quelques années, le monde des applications logicielles, notamment professionnelles, connaît des mutations profondes tirées par l'essor de nouvelles technologies telles que l'Informatique Ubiquitaire, le Cloud Computing, l'Internet des Objets, le Big Data et l'Intelligence Artificielle. Les atouts combinés de ces technologies ouvrent la voie à une nouvelle génération d'applications dites **modernes**. Ces applications sont construites par assemblage de composants en réseau, faiblement couplés, indépendants et implantant des fonctions de plus en plus spécialisées. Elles s'appuient sur des architectures modulaires telles que l'architecture des microservices et utilisent des techniques d'intégration de nouvelle génération, la plus prédominante étant le Cloud. Les éléments caractéristiques des applications modernes, qui reviennent le plus souvent chez les éditeurs tels que [Accenture 2015, Capgemini 2017] ou chez les analystes dont le cabinet de conseil Gartner, à travers son concept de "maillage digital intelligent" [1], sont : facilement accessibles, fluides, connectées et intelligentes :

- **facilement accessibles** signifie que les utilisateurs finaux peuvent exploiter ces applications à tout moment, au moyen de dispositifs divers et indépendamment de leur position géographique.
- **fluides** est synonyme d'une meilleure agilité qui facilite l'évolutivité (l'enrichissement des applications en y intégrant les composants de fournisseurs tiers souvent en mode Software as a Service) et l'extensibilité (possibilité d'instancier dynamiquement de nouveaux composants pour résister à une hausse de charge) des applications.
- **connectées** fait référence au fait que ces applications offrent aux entreprises une interface compatible avec les écosystèmes de leurs partenaires et clients ainsi qu'avec l'Internet des Objets. Elles doivent donc absorber un volume important de données.
- **intelligentes** signifie que ces applications s'adaptent et apprennent constamment des comportements de leurs utilisateurs, grâce à trois capacités critiques : l'automatisation intelligentes, l'analytique intégrée et le pilotage automatique. Cela suppose que les applications modernes sont par essence réparties et que leurs composants s'échangent un flux de trafic assez important lors des opérations de calcul distribué.

Du fait de leur architecture et de leurs caractéristiques, les applications modernes présentent diverses attentes vis-à-vis du réseau :

- elles sont dynamiques dans la mesure où les flux de données évoluent dans le temps, notamment lorsque de nouveaux composants logiciels sont dynamiquement instanciés pour résister à une hausse de charge, et lorsque les besoins en qualité de service (QdS) de ces flux nécessitent d'être adaptés au contexte de l'utilisateur final pour une meilleure qualité d'expérience.
- leurs besoins de communication peuvent être difficiles à exprimer. En effet, il s'avère difficile d'identifier à l'avance l'ensemble des flux et/ou d'exprimer précisément les besoins

1. L'entrelac des interactions entre les individus, leurs périphériques, et leurs contenus et services

de QdS associés. La raison est que les composants qui font partie d'une application moderne, émanent de plusieurs éditeurs qui utilisent le plus souvent des protocoles applicatifs propriétaires non connus du grand public.

– le trafic réseau ne cesse de croître et présente un modèle inédit, différent de celui connu jusqu'ici. Le besoin de communication de ces applications est grandissant, ce qui engendre un volume de données de plus en plus important provenant des objets connectés ainsi que des échanges entre les composants applicatifs et entre les composants et les usagers.

Les réseaux actuels peinent à répondre aux attentes des applications modernes. Celles-ci souffrent de problèmes de performances, [Fielder 2012] lesquelles sont appelées à se dégrader davantage avec la transition vers le Cloud [MRV 2013, Kerravala 2015]. Selon un rapport produit en 2014 par ZK Research (figure 1), la performance des applications est au premier plan des priorités des administrateurs réseau.

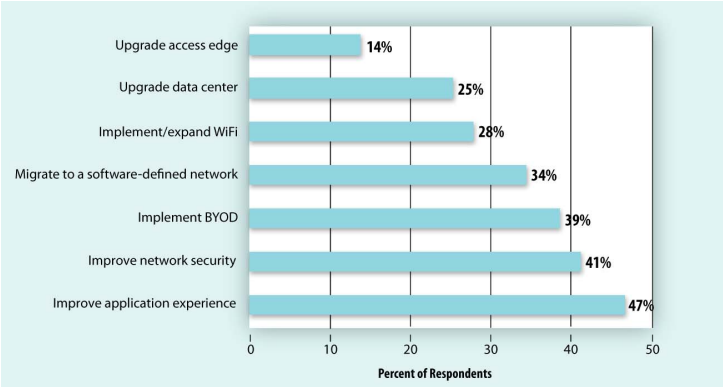


FIGURE 1 – Priorités des administrateurs réseaux (Source : ZK Research 2014)

Les raisons les plus citées sont d'une part la méconnaissance des besoins de QdS des applications, le manque de visibilité non seulement sur les flux de données transitant sur le réseau, mais aussi sur le rendu de performance en temps réels des applications, et d'autre part l'absence de leviers d'action permettant d'envisager un contrôle fin, dynamique et efficace. Il résulte que les applications , lorsqu'elles le peuvent, doivent s'adapter aux performances du réseau, les administrateurs, quant à eux, sont amenés à intervenir en réaction aux retours des utilisateurs, les entreprises ayant les moyens, peuvent opter pour une solution coûteuse en ayant recours au sur-dimensionnement des capacités réseau. Ainsi, fournir un service de communication réseau capable de répondre et de suivre les besoins de ces applications sans gaspiller, par sur-dimensionnement, l'utilisation des ressources réseau, pose plusieurs défis aux réseaux de communication supports, notamment la nécessite d'un haut degré de flexibilité, largement au-delà des possibilités des réseaux de communication actuels.

L'objectif de ce travail de thèse est de développer le concept de réseau guidé par les applications (ADN : Application Driven Networking), réseau capable d'offrir des services de communication personnalisés et flexibles et avec garantie de QdS aux applications. Le qualificatif

personnalisé signifie que le service ADN répond à des besoins de communication applicatifs exprimés avec un niveau de granularité très fin pouvant aller jusqu'aux flux applicatifs élémentaires. Ces besoins peuvent être exprimés explicitement par l'application ou inférés par le réseau par analyse du trafic. L'aspect dynamique signifie que le service ADN est reprogrammé et ajusté pour suivre l'évolution des besoins de l'application dans le temps. Ce service, support des flux de données de l'application, s'exprime sous forme d'un réseau virtuel applicatif constitué d'un ensemble de liens virtuels de bout-en-bout qui peuvent être de type point-à-point ou point-à-multipoint et sont caractérisés par une bande passante et un délai de transfert maximal.

Contributions de la thèse

Les contributions de ce travail de thèse couvrent plusieurs points :

1. Nous avons défini l'architecture générale d'un réseau ADN bâti sur une infrastructure réseau de type SDN (Software Defined Network) en explicitant ses composants fonctionnels ainsi que leurs interactions.

2. Nous avons développé un algorithme d'allocation de ressources réseau qui calcule les chemins de données et les ressources réseau à y réserver pour satisfaire les exigences de bande passante et de délai des services ADN.

3. Nous avons mis au point deux heuristiques de migration des services ADN afin de répartir aux mieux la charge du réseau et d'augmenter l'admissibilité des requêtes des services ADN à venir.

4. Nous avons développé un prototype démonstrateur de réseau ADN qui fournit un ensemble de services ADN à des applications dynamiques basées sur le middleware temps réel DDS (Data Distribution Service).

Structure du manuscrit

La suite de ce manuscrit se présente comme suit :

– Le chapitre 1 **« Notions préliminaires, État de l'art et Positionnement »** présente les principes et concepts des réseaux SDN avec un focus sur le protocole OpenFlow. La problématique générale traitée dans cette thèse est formulée et développée, avant de présenter une synthèse des travaux connexes existants, en utilisant une taxonomie que nous avons définie dans le but de les positionner. Le positionnement et l'approche générale de notre solution y sont également décrits.

– Le chapitre 2 **« Architecture générale d'un réseau ADN »** a pour objectif de présenter l'architecture de la solution réseau ADN développée dans le cadre de cette thèse. Tout d'abord, nous présentons notre vision, nos principes et les lignes directrices qui ont guidé l'élaboration de notre solution, ainsi que les choix de conception de l'architecture

qui ont été effectués. Ensuite, nous présentons les différents composants de l'architecture et leurs exigences fonctionnelles. Pour finir, les différents aspects du système sont décrits suivant le formalisme UML à travers la description des diagrammes des cas d'utilisation, de composants et d'interaction entre ces composants.

– Le chapitre 3 **« Algorithme d'allocation de ressources »** est consacré comme son nom l'indique, à l'allocation dynamique de ressources pour les services ADN. L'objectif est de présenter l'algorithme proposé pour calculer les chemins de données et les ressources à y réserver pour provisionner les réseaux virtuels. Nous présentons le problème d'allocation de ressources avant de positionner notre solution par rapport à celles existantes dans la littérature scientifique. Le modèle d'allocation de ressources proposé est ensuite développé et ses résultats d'évaluation expérimentale sont présentés et analysés.

– Le chapitre 4 **« Mécanismes de défragmentation de ressources »** est dédié à la gestion dynamique des ressources. L'objectif est de présenter les deux algorithmes conçus pour lutter contre la fragmentation des ressources. Comme pour le chapitre 3, le plan adopté dans ce chapitre, est de poser tout d'abord le problème de fragmentation de ressources, avant de positionner nos solutions par rapport à celles existantes dans la littérature scientifique, puis les décrire en détails et finir en présentant les résultats d'évaluations expérimentales.

Pour conclure, le manuscrit expose finalement une vision globale de l'ensemble des travaux réalisés durant cette thèse, ainsi que les perspectives d'évolution et d'amélioration des résultats de recherche pouvant être envisagées.

Chapitre 1

Notions préliminaires, État de l'art & Positionnement

Sommaire

1.1 Software-Defined Networking 5
1.2 Le protocole OpenFlow 6
1.3 L'interface Nord 10
1.4 Problématique 11
1.5 État de l'art des solutions existantes 14
1.6 Notre approche générale 24
1.7 Conclusion 26

1.1 Software-Defined Networking

Dans un réseau classique, les fonctions de contrôle du réseau et d'acheminement des données sont effectuées au sein des équipements réseau (tels que les routeurs). Cette architecture monolithique est considérée comme un frein et des efforts ont été faits pour définir des architectures cherchant à introduire plus de flexibilité et de simplicité dans la gestion et le contrôle du réseau. La notion de "Software-Defined Networking" (SDN) [ONF 2012] regroupe les architectures visant à séparer l'acheminement des données et son contrôle, principalement en retirant l'intelligence des équipements réseaux matériels et en la disposant dans des composants logiciels sur des équipements dédiés.

La figure 1.1 illustre l'architecture SDN et la compare avec l'architecture d'un réseau classique. La séparation du plan de données du plan de contrôle permet une visualisation et une gestion plus globale du réseau. Le déploiement de nouveaux protocoles ou de nouvelles applications est alors facilité, l'intelligence du réseau étant logiquement regroupée dans une entité spécifique, appelée le contrôleur SDN. Comme on peut le voir sur la figure 1.1, le contrôleur devient le point central de l'architecture SDN. La partie supérieure englobe les applications agissant sur le réseau telles que le routage, le contrôle d'accès, la surveillance du réseau... L'interaction avec le contrôleur se fait via des API et les applications obtiennent ainsi une vision simplifiée du réseau sous-jacent. Le plan inférieur est composé d'éléments physiques chargés de l'acheminement des données, typiquement des commutateurs. Le contrôleur situé entre les applications et les équipements physiques peut alors, au travers de différents services, interpréter

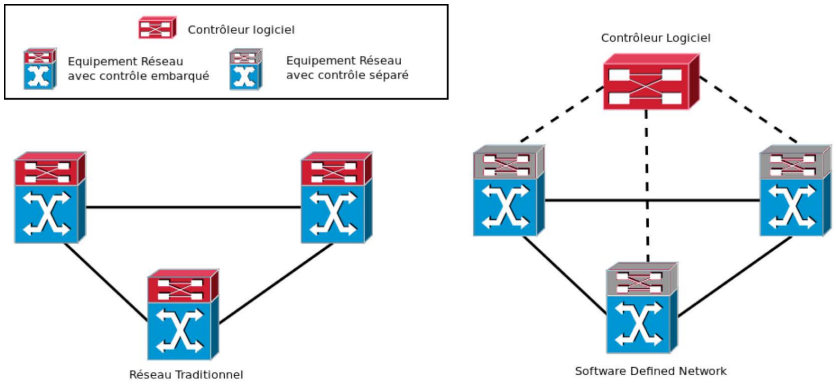


FIGURE 1.1 – Architectures réseau traditionnelle et architecture d'un Software Defined Network (inspiré de [Nunes 2014])

les consignes des applications et les répercuter sur le réseau. Idéalement, les interfaces Nord (Application-Contrôleur) et Sud (Contrôleur-Routeur) proposent des API ouvertes afin de permettre une meilleure interopérabilité entre les systèmes et matériels de différents constructeurs (voir figure 1.2 vue classique et haut-niveau de l'architecture SDN qui présente la vision de l'Open Networking Foundation (ONF), principal promoteur des réseaux SDN).

Pour l'instant, les efforts des chercheurs et des industriels se sont focalisés sur l'interface Sud car celle-ci est indispensable à la programmation du plan de données. Nous verrons par la suite que les approches sont légèrement différentes mais que le principe reste globalement le même. Deux propositions nous semblent les plus abouties : OpenFlow [McKeown 2008] et ForCES [Halpern 2010]. Nous détaillerons dans la suite OpenFlow qui est le protocole standard le plus plébiscité dans ce rôle et maintenu par le consortium ONF.

Si les travaux relatifs à l'interface Nord ne connaissent pas la popularité des protocoles de l'interface Sud (OpenFlow monopolisant les discussions sur les SDN), le succès futur des SDN passera pourtant par l'existence d'API clairement définies au niveau de l'interface Nord. En effet, ceci est indispensable pour le développement d'applications et, donc, une exploitation optimale des architectures SDN. Nous ferons un point sur les mécanismes actuels et sur les objectifs à respecter.

1.2 Le protocole OpenFlow

OpenFlow se revendique comme le premier standard définissant l'interface de communication entre les plans de contrôle et de données d'une architecture SDN. Il définit les éléments du commutateur ainsi que le protocole permettant de gérer celui-ci depuis un contrôleur à distance [ONF 2013a]. Les différents composants (figure 1.3) d'un commutateur OpenFlow sont :

- Les « flow tables » et la « group table » chargées du traitement des paquets,
- Le « OpenFlow Channel », interface permettant la connexion avec le contrôleur.

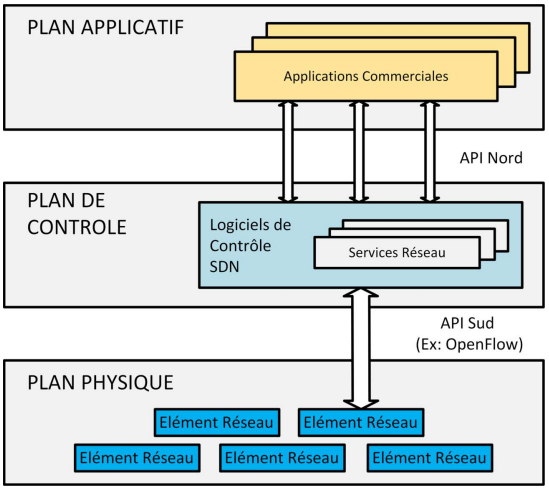


FIGURE 1.2 – Architecture de SDN vue par l'ONF [ONF 2013c], vision classique et haut-niveau de l'architecture SDN.

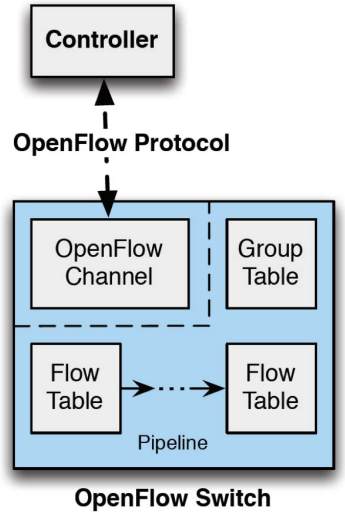


FIGURE 1.3 – Composants d'un commutateur OpenFlow [ONF 2013a]

Le protocole OpenFlow est utilisé par le contrôleur pour ajouter, mettre à jour ou supprimer des entrées dans les tables, de manière réactive (c.à.d. en réponse à un paquet) ou de manière proactive. En définissant une succession de tables, il est possible de réaliser plusieurs traitements successifs sur un paquet avant de l'éjecter ou le diriger vers un port de sortie, ce « pipeline » est illustré par la figure 1.4. Il est intéressant de noter que le standard définit deux catégories de commutateurs compatibles avec OpenFlow : ceux proposant uniquement OpenFlow et les hybrides. Dans le premier cas, tous les paquets doivent suivre le traitement d'OpenFlow par tables. Dans le cas hybride, le commutateur supporte OpenFlow et est capable de traiter les paquets de manière traditionnelle avec par exemple de la commutation Ethernet ou du routage de niveau 3. Ces commutateurs doivent alors posséder un mécanisme de classification qui va diriger les paquets soit vers le traitement par les tables d'OpenFlow soit vers le traitement

Match Fields	Priority	Counters	Instructions	Timeouts	Cookie

TABLE 1.1 – Composants d'une entrée de la table de flux dans un commutateur OpenFlow

Group Identifier	Group Type	Counters	Action Buckets

TABLE 1.2 – Composants d'une entrée de la table de groupe dans un commutateur OpenFlow

classique. En plus de cela, un commutateur hybride peut aussi permettre à un paquet d'aller du traitement OpenFlow vers le traitement classique.

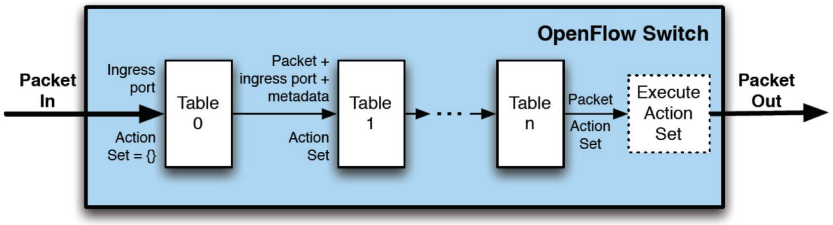


FIGURE 1.4 – Pipeline de traitement d'un paquet dans un commutateur OpenFlow [ONF 2013a]

Afin d'obtenir un traitement le plus fin et efficace possible, les tables constituant le « pipeline » d'OpenFlow peuvent être de différents types, chacune ayant un rôle distinct. Les tables de flux sont les tables de base permettant la transmission des données. Elles contiennent une collection d'entrées constituées notamment d'un champ de correspondance et d'instructions associées (tableau 1.1). Un paquet arrivant dans un commutateur est comparé aux différentes entrées de la première table du « pipeline ». Si une seule entrées correspond, alors l'instruction associée est exécutée. Si plusieurs entrées correspondent, celle avec la plus haute priorité est choisie. Dans le cas où aucune entrée ne correspond, l'entrée par défaut de la table décide par exemple de supprimer le paquet, le remonter au contrôleur ou le faire suivre à la table suivante. Les instructions associées à chaque entrée contiennent soit des actions à exécuter sur le paquet soit la table à parcourir par la suite. Dans ce cas-là, des métadonnées sont associées au paquet afin d'assurer la communication entre les tables. Si l'instruction ne spécifie pas de table suivante, les actions sont exécutées. Ces actions peuvent correspondre, par exemple, à la suppression du paquet, à une opération d'encapsulation/désencapsulation sur le paquet, ou son renvoi vers la file d'attente d'un port de sortie, etc. Il est à noter que OpenFlow peut supporter plusieurs files d'attente par port mais qu'il ne les configure pas, ceci devant être fait par une solution extérieure (CLI, protocole propriétaire, OF-Config, etc.).

La « group table » (tableau 1.2) est une table contenant des entrées de groupes vers laquelle peuvent être dirigés les paquets depuis une table de flux classique. Une suite d'actions est alors exécutée sur le paquet. L'utilisation de ces groupes permet de définir des actions à exécuter sur

Meter Identifier	Meter Bands		Counters

Band Type	Rate	Counters	Type specific arguments

TABLE 1.3 – Composants d'une entrée de la table de mesure (en haute) et bande de mesure qui la compose (en bas)

de multiples flux.

Une instruction introduite par la version 1.3 d'OpenFlow permet de diriger le paquet vers une table de mesure (tableau 1.3). Les métriques qui y figurent peuvent être appelées par les entrées des tables de flux avant que les actions ne soient traitées. Pour une métrique donnée, plusieurs tables de bande sont disponibles et permettent de comparer le débit du flux à une valeur donnée par la table. Si celle-ci est dépassée, le paquet peut être supprimé ou le champ DSCP de l'entête IP peut être modifié. En se servant de ces métriques par flux, OpenFlow est donc capable d'opérations de QdS basiques comme la limitation du débit et cela en amont des files d'attentes qui peuvent être définies sur les ports de sortie.

Dans chaque table d'OpenFlow, un champ « Counters » permet de mémoriser des statistiques sur le/les flux de données qui la traverse et sur la table en elle-même. Par exemple, il est possible de stocker la durée écoulée depuis la définition d'une entrée, le nombre de paquets ayant transité par la table ou pour les ports des informations plus spécifiques comme le taux d'erreurs. Ces statistiques peuvent être récupérées par le contrôleur via l'envoi d'une requête au commutateur et permettent ainsi au contrôleur de se constituer une vision précise des flux circulant sur le réseau.

L'élément « OpenFlow Channel » de la figure 1.3 est l'interface de communication avec le contrôleur. Cette communication se fait via le protocole OpenFlow de manière sécurisée avec Transport Layer Security (TLS) ou en clair avec TCP. Il existe 3 types de messages, chacun comportant des sous-types :

- « Controller-to-switch » : envoyés par le contrôleur pour gérer et vérifier l'état du commutateur ou par exemple acquérir des statistiques,
- « Asynchronous » : envoyés par le commutateur pour informer le contrôleur d'évènements réseaux et de changements d'état,
- « Symmetric » : envoyés par le contrôleur ou le commutateur pour initier la connexion entre les deux éléments et la maintenir active.

Les deux premiers types de messages sont utilisés par exemple pour définir les entrées des différentes tables ou faire remonter/descendre les paquets de nouveaux flux présents dans aucune entrée. Des formats recommandés pour ces messages sont définis dans [ONF 2013a].

1.3 L'interface Nord

Dans la section précédente, deux approches de SDN ont été détaillées. Néanmoins, les définitions faites pour OpenFlow s'intéressent uniquement à l'interface Sud. Nous allons essayer ici de faire un point sur l'état actuel de l'interface Nord du contrôleur dans les architectures SDN basées sur OpenFlow, celui-ci étant le plus répandu. Néanmoins, les approches devraient être les mêmes quel que soit l'architecture SDN et idéalement, les interfaces devraient être compatibles.

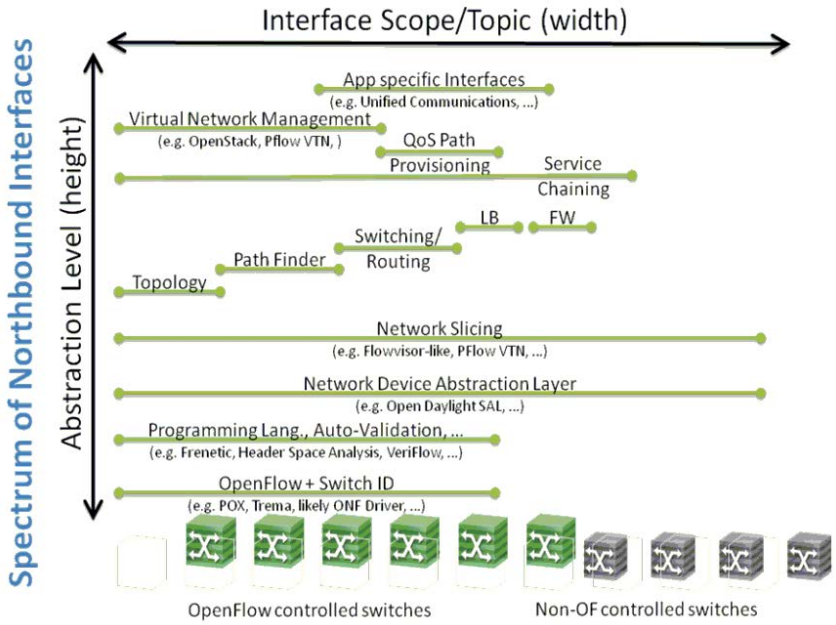


FIGURE 1.5 – Aperçu des interfaces Nord avec leur niveau d'abstraction [ONF 2013b]

Si le contrôleur a la visibilité du réseau entier, les applications qui lui sont connectées sont la réelle intelligence du réseau en proposant des fonctions de gestion avancée, de sécurité et de surveillance du réseau. Les API définies au niveau de l'interface Nord permettent d'offrir une abstraction des couches inférieures et simplifient ainsi la vision des applications. Néanmoins, chaque type d'application nécessite des informations plus ou moins détaillées. Le groupe de travail « North Bound Interfaces » de l'ONF a été lancé en 2013 et tente d'éclaircir les travaux effectués. La figure 1.5 présente leur vision des différents types d'interface (avec une indication sur le niveau d'abstraction dans lesquels ils se placent). Le niveau d'abstraction le plus faible correspond à une programmation quasiment directe du commutateur, le contrôleur ne servant alors que d'intermédiaire. A l'opposé, le plus haut niveau d'abstraction revient à fournir des services purs comme le fait par exemple OpenStack. Pour l'instant, deux approches semblent prendre de l'importance : l'architecture MD-SAL d'OpenDaylight et le langage de haut-niveau Frenetic (ou Pyretic pour la version Python). L'abstraction de la couche service avec une approche modélisation (MD-SAL) d'OpenDaylight se base sur une modélisation des données avec YANG, lui-même basé sur XML. Il est ainsi possible d'obtenir des données accompagnées de leur description ce qui facilite la tâche du contrôleur et des applications. Une architecture REST permet l'accès aux machines distantes. Frenetic [Monsanto 2012] est un langage haut-niveau qui

vise à simplifier la programmation d'un SDN. En partant du constat que la gestion d'un contrôleur et la configuration d'un SDN nécessitent des connaissances poussées sur les flux mais aussi sur le matériel présent, Frenetic a été proposé pour abstraire totalement le protocole utilisé sur l'interface Sud.

Frenetic/Pyretic permettent aux applications de se concentrer uniquement sur leur domaine d'action : la définition globale des différentes politiques du réseau que ce soit en termes de connectivité, de sécurité ou de surveillance.

1.4 Problématique

Afin de répondre au besoin de réactivité des entreprises, chez les éditeurs de logiciels, la tendance est de développer les applications en les rendant facilement accessibles, fluides, connectées et intelligentes. Ces propriétés caractéristiques des applications dites **modernes**, permettent à travers le prisme du réseau, de considérer une application moderne comme un ensemble de flux de données :

- où chaque flux est adressé à un ou plusieurs destinataire(s),
- où les flux variés peuvent correspondre à un trafic Est-Ouest (centre de données $\leftrightarrow$ centre de données) ou Nord-Sud (utilisateur $\leftrightarrow$ centre de données),
- dynamique en ce sens qu'il peut évoluer dans le temps : arrivée de nouveaux flux, ou de nouveaux destinataires d'un flux existant,
- où chaque flux à des exigences en QdS notamment en termes de débit et de latence, qui peuvent évoluer dans le temps,
- où les exigences en QdS des flux sont le plus souvent difficilement exprimables.

A cause de leur architecture qui présente de nombreux avantages, les applications modernes introduisent néanmoins une explosion de flux données ayant un profil inédit avec comme conséquence d'impacter les habitudes du trafic réseau connues jusqu'ici. Ceci combiné au caractère dynamique et les besoins contraignants de ces applications, posent aux réseaux actuels, un défi qui requière des capacités au delà de leurs possibilités. Il résulte que ces applications souffrent de problème de performances, lesquelles sont appelées à s'accentuer.

La problématique traitée peut être formulée comme suit :

- ***Comment fournir un service réseau répondant aux exigences des applications ayant des caractéristiques des applications modernes, en utilisant efficacement les ressources réseau ?***

- ***Comment faciliter la gestion des services réseau et du réseau lui-même dans ce contexte actuel où le réseau est communément qualifié de système de plus en plus complexe ?***

Ces questionnements constituent le fil conducteur de notre travail et prêtent à discussion. Le périmètre de l'espace de discussion mérite alors d'être défini. Nous avons fait le choix de spécifier cette problématique suivant deux angles : sa portée qui fait ici référence au spectre d'environnements réseau qu'elle peut couvrir et les constats relatifs aux difficultés opérationnelles et limites technologiques actuelles.

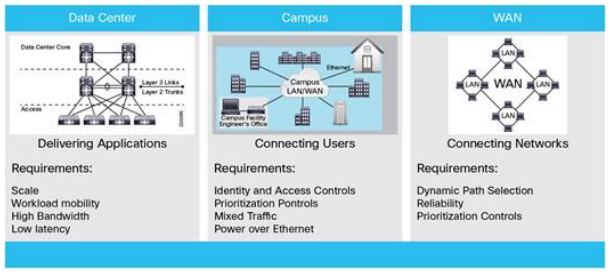
1.4.1 La portée de la problématique

Nous proposons d'identifier les éléments qui ont trait au réseau et relevant du contexte d'exécution et d'utilisation des applications, ainsi que de leur accessibilité selon la localisation des utilisateurs, puisque ces éléments constituent une donnée importante, en ce sens qu'ils peuvent influer sur la formulation de la réponse à cette problématique. Pour cela, rappelons que le réseau consiste, d'un point de vue global, en un ensemble de sous-systèmes de communication indépendants, chacun constituant localement un environnement réseau en soi. Chacun de ces environnements est un élément à prendre en compte dans la problématique à traiter, car il est plausible qu'il contribue d'une certaine façon au problème. Ce faisant, en regardant le réseau de bout en bout, depuis le lieu d'exécution des applications à l'endroit où elles sont exploitées par les utilisateurs finaux, on peut clairement identifier plusieurs segments. La figure 1.6 propose un résumé de ces segments. Indépendamment de la taille des entreprises et de leurs secteurs d'activités, nous avons (sans être exhaustif) les environnements réseau du centre de données, du campus, le réseau Métropolitain et le WAN. On peut également ajouter le segment que représente les différentes connexions à Internet, ou plus généralement les connexions utilisées pour accéder aux applications sur un cloud privé virtuel de l'entreprise ou aux applications en mode SaaS chez un tiers fournisseur de services cloud. Pour cette catégorie d'applications, un nouveau segment entre en jeu, il s'agit de l'environnement réseau qui est sur le cloud et dans lequel s'exécutent ces applications. La figure 1.6b présente une vue d'ensemble de ces segments réseau et de leurs interconnexions. En résumé ces segments peuvent être rangés en deux grandes catégories distinctes :

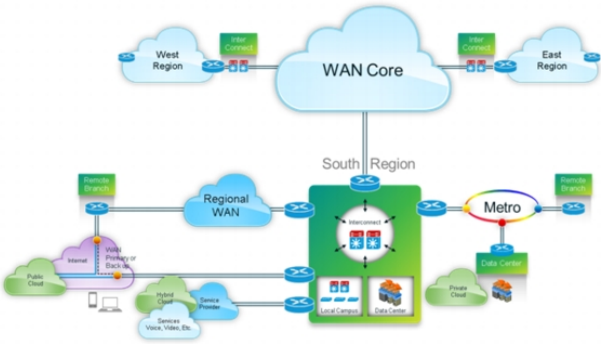
- **Les segments orientés vers les applications** : cette catégorie comprend les environnements réseau du centre de données et du cloud. C'est dans ces environnements que les applications sont déployées et exécutées. Ils servent de support aux échanges (généralement désignés comme le trafic Est-Ouest) de données entre les composants constituant chaque application. C'est notamment le cas des applications multi-tiers ou des applications ayant les composants repartis à travers le réseau.
- **Les segments orientés vers les utilisateurs** : cette catégorie comprend l'environnement réseau du campus, les connections à Internet, WAN et Métropolitain. Ces segments sont destinés à interconnecter les utilisateurs entre eux et avec leurs applications. A ce titre, ils servent de support aux échanges (généralement désignés comme le trafic Nord-Sud) de données entre les utilisateurs et les applications.

Selon le scénario de déploiement (mode on-premise ou SaaS) et d'utilisation (utilisateur mobile ou fixe) des applications, toute ou une partie de l'ensemble de ces segments est mise à contribution. A titre d'exemple, lorsqu'un collaborateur d'une entreprise ayant ses sites géographiquement distribués dans le globe, utilise une application hébergée dans le centre de données abrité dans les bâtiments du siège central, les segments réseau à considérer dans ce scénario sont : le WAN et le réseau du centre de données. Ainsi, la solution globale en réponse à cette problématique sera la juxtaposition des solutions locales à chaque segment. Comme illustré par la figure 1.6a, chacun de ces segments est destiné à un usage bien précis, possède des caractéristiques et exigences spécifiques, et présente les difficultés qui lui sont propres. En raison des différences entre eux, une solution globale universelle, c'est-à-dire celle là qui va répondre à la problématique sans considérer spécifiquement chaque segment, serait sans aucun doute difficile

à réaliser. Il serait alors intéressant de pousser le développement de cette problématique en considérant chaque environnement réseau pour dégager les interrogations qui lui sont spécifiques. Nous avons fait le choix de développer cet aspect dans la section présentant les solutions existantes.



(a) Exigences des segments réseau



(b) Vue d'ensemble des segments réseau et de leurs interconnexions

FIGURE 1.6 – Environnements réseau distincts à considérer (Source : Cisco)

1.4.2 Les constats

Les constats relatifs aux difficultés opérationnelles et limites technologiques actuellement rencontrées par les administrateurs réseaux sont presque toutes les mêmes dans ces environnements. Ils préciseront encore plus la problématique, faciliteront la compréhension des travaux autour de ce sujet et nous seront source d'inspiration. Ils sont listés ci-après :

- **Présence de silos** : les équipes de développement d'application, de gestion de l'infrastructure, de la sécurité, et du réseau, doivent collaborer lors du cycle de vie de déploiement d'une application. La traversée de ces couches opérationnelles constitue un frein au déploiement rapide des applications.
- **Charge opérationnelle lourde** : la nature distribuée du plan de contrôle et des protocoles réseau combinée aux interfaces d'administration propriétaires en ligne de commande, rendent difficiles l'automatisation des opérations de gestion du réseau et de ses services. Les tâches de provisionnement et de configurations effectuées essentiellement manuellement sont laborieuses, chronophages et sujets d'erreurs.
- **Contrôle à granularité épaisse** : le trafic applicatif est géré par les services réseau organisés en classes. Les protocoles de contrôle de routes tels que OSPF qui acheminement les paquets selon l'adresse IP de destination et les mécanismes de gestion de QdS tels que DiffServ qui les traitent selon la classe de service à laquelle ils appartiennent, sont robustes et scalables, en revanche ils ne sont pas assez flexibles, pour supporter un contrôle granulaire (fin à l'échelle du flux applicatif et épais à l'échelle d'un agrégat de flux), lequel aurait permis de supporter plus efficacement les applications critiques qui exigent la différenciation entre leur flux vitaux/essentiels par nature multiservices et ceux à faible priorité.

- **Faible niveau de réactivité** : il est quasiment impossible de réagir en temps-réel à l'évolution du réseau et des besoins des applications. L'intervention humaine rallonge les délais de maintenance du réseau et des services réseau, mais également la gestion desdits services (détection plus rapide de fautes, établissement de diagnostic plus efficace, ajustement et adaptation du service réseau, etc.).
- **Manque de visibilité sur le trafic** : on assiste à une utilisation abusive de ports. C'est le cas des réseaux égal-à-égal qui utilisent les ports non standards pour leurs communications. Par exemple BitTorrent peut s'exécuter sur TCP en utilisant le port 80 si tous les autres ports sont bloqués. Aussi, plusieurs applications critiques notamment les logiciels en mode SaaS, utilisent de plus en plus les techniques de tunneling applicatifs en s'appuyant le plus souvent sur le protocole HTTP qui sert d'enveloppe à leur protocole propriétaire, pour échanger les données. Cette banalisation de l'usage des ports remet en question la classification de trafic qui était basée sur la correspondance entre les applications et leur port standard comme normalisée par l'IANA. Les cartes sont redistribuées, il faudra alors regarder au delà de l'entête IP d'un paquet pour espérer connaître à quelle application il appartient.
- **Méconnaissance des besoins de QdS** : si les applications connaissent naturellement les exigences liées aux conditions d'acheminement de leurs flux de données, elles ont du mal à les communiquer au réseau. Soit parce que l'interface de communication qu'est l'entête des paquets, n'est pas assez évoluée, notamment en termes de pouvoir d'expression et de négociation, soit parce que le réseau ne leur fait pas confiance.
- **Absence de visibilité sur les performances** : les administrateurs s'aperçoivent des problèmes de performances des applications grâce aux retours des utilisateurs. Il n'existe pas de moyens permettant aux applications de signaler au réseau, en temps réels, des informations à ce sujet. Il résulte que, le réseau n'utilise que ses moyens internes, notamment l'analysant de trafic, pour estimer l'état de santé des applications, le niveau de satisfaction des utilisateurs et le cas échéant mettre en avant rapidement d'éventuels problèmes liés aux conditions d'acheminement des flux de données.

1.5 État de l'art des solutions existantes

1.5.1 Taxonomie

Plusieurs solutions émergent dans la littérature. On peut les classer suivant trois grands aspects : le profil de communication des applications visées, le service réseau fourni pour supporter ces applications, l'approche adoptée pour minimiser les coûts opérationnels. La figure 1.7 propose quelques éléments de catégorisation des solutions existantes.

1. Le profil de communication d'une application fait référence aux flux de données qu'elle échange, ainsi que les exigences à satisfaire lors de leur transit sur le réseau. Ces exigences peuvent concerner plusieurs facteurs : Le niveau de sécurité, de disponibilité et de QdS. Ce dernier facteur peut se définir suivant plusieurs paramètres distincts, dont entre autres : la latence, le débit, la gigue et les pertes. Au regard de cet aspect, les solutions peuvent se distinguer sur :

(a) L'approche adoptée pour établir et maintenir le profil de communication : Soit le profil est explicitement communiqué au réseau par l'application au fil de l'eau ou par un opérateur humain comme un manifeste lors du déploiement de l'application (ce cas de figure est commun dans les centres de données et les environnements des fournisseurs de services Cloud) ; soit le réseau l'infère implicitement par exemple en analysant le trafic applicatif qui transite sur l'infrastructure. L'approche peut selon les conditions être simple, notamment lorsqu'il est possible de déduire les caractéristiques de chaque flux à partir de l'entête de ses paquets, ou sophistiquée, lorsque les flux sont méconnus et ne sont pas produits de manière à être directement identifiables sur l'infrastructure.

(b) La nature du profil de communication : Il peut être dynamique ou statique selon que les flux et leurs exigences de QdS évoluent ou pas au cours du temps.

(c) Le niveau d'exigence : il fait référence au caractère contraignant de l'application. Lorsqu'il y a obligation à satisfaire ses exigences, on parle d'exigence stricte, dans le cas contraire on parle d'exigence souple.

2. Le service réseau à fournir pour répondre aux attentes des applications. Au regard de cet aspect, les solutions peuvent se distinguer sur :

   (a) La nature du service réseau : Il peut s'agir d'un overlay applicatif qui s'appuie sur une simple connectivité standard qui peut être de niveau 2, 2.5 ou 3 avec potentiellement de la QdS, ou alors une connectivité plus enrichie qui inclut aussi les services réseau des couches 4 à 7 (pare-feu applicatif, IDS/IPS, etc) comme c'est généralement le cas dans les centres de données.

   (b) La granularité du service réseau : Elle renseigne sur le fait que la QdS est garantie par flux applicatif élémentaire ou par agrégat de flux. L'agrégation peut se faire à échelle d'une application ou d'un type d'application.

   (c) La technique utilisée pour la mise en place du service réseau : Il peut s'agir d'un routage QdS qui cherche à sélectionner la route la plus adaptée aux exigences de chaque flux ; d'un routage avec traitement différentié selon les exigences du flux ; ou alors de la réservation de ressources.

3. L'approche adoptée pour réduire les coûts opérationnels. Il s'agit des moyens proposés pour minimiser la part de l'opérateur dans les tâches de provisionnement, de configuration, d'optimisation et de réparation du réseau de façon à simplifier la gestion du réseau et améliorer ses performances. Au regard de cet aspect, on distingue :

   (a) Une gestion autonome qui est mise en place en mettant en œuvre des mécanismes cognitifs et motrices pour respectivement apprendre de l'expérience passée les politiques de gestion du réseau et les appliquer. Ces mécanismes apprendront en ligne les règles de décision optimales. Cette approche minimise considérablement les interventions de l'opérateur dont les tâches d'administration sont réduites à communiquer au réseau les objectifs métiers. Le réseau se chargera lui-même de les atteindre.

(b) Une gestion adaptative qui est différente de la gestion autonome en ce sens que les politiques de gestion sont explicitement communiquées au réseau par l'opérateur et les actions sont dynamiquement et automatiquement appliquées sans aucune intervention humaine. Cette approche requière plus d'attention de la part de l'opérateur en matière d'intelligence.

(c) Une gestion basique où le provisionnement, la configuration et la gestion du réseau sont faits par l'administrateur. Les compétences humaines sont alors nécessaires pour surveiller, analyser les métriques observées et exécuter les actions en fonction des anomalies détectées.

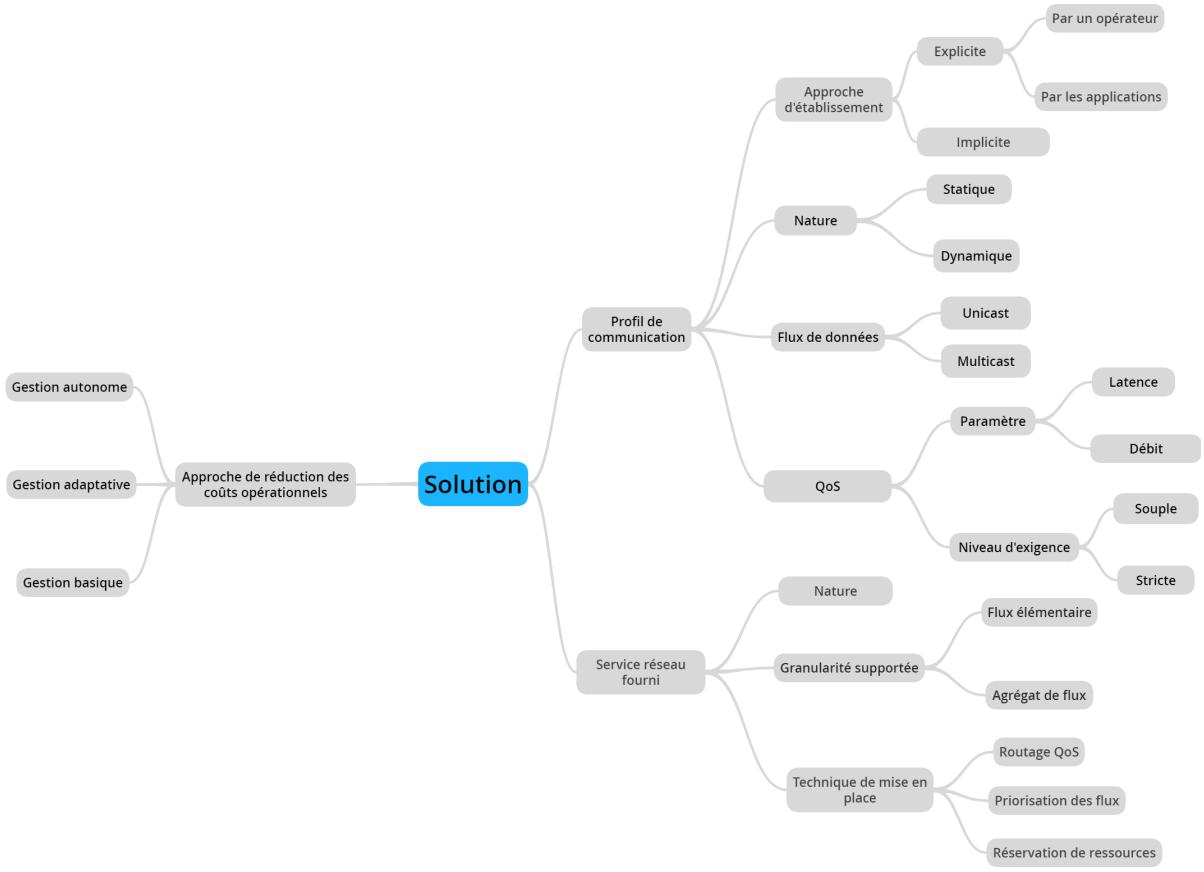


FIGURE 1.7 – Taxonomie

1.5.2 Quelques travaux connexes existants dans la littérature

Force est de noter que, selon les ambitions de chaque proposition, la construction de la solution peut nécessiter la mise en œuvre de plus ou moins de mécanismes, d'algorithmes et de protocoles. Les solutions existantes sont issues des milieux académiques et industriels. Elles peuvent être complètes, c'est généralement le cas des offres industrielles qui sont en revanche

peu détaillées, notamment sur les aspects relevant de l'implémentation, ou partielles, c'est le cas de la majorité des solutions identifiées dans la littérature scientifique, qui se focalisent le plus souvent sur un aspect précis du problème. Nous présentons ci-après quelques travaux connexes. Le résumé produit pour chacun de ces travaux, présente les ambitions fixées, les choix adoptés, les aspects du problème qui y sont abordés et le principe des méthodes développées pour atteindre leurs ambitions. Il s'agit essentiellement des travaux qui s'inspirent du principe des réseaux SDN et qui ont été effectués en parallèle à cette thèse, soit entre les années 2014 et 2017.

– La vision du géant mondial de la virtualisation VMWare est d'appliquer à l'échelle d'un réseau de centre de données, le même modèle opératoire que celui de la virtualisation des serveurs. En virtualisant le réseau et les services réseau, il serait alors possible de créer, ajuster, restaurer et supprimer, un réseau virtuel personnalisé et dédié à chaque application, de manière simple, facile et rapide. Pour ce faire, une couche d'abstraction est introduite entre le matériel réseau et les applications, et vise à découpler les applications afin de leur permettre d'évoluer indépendamment de l'infrastructure et vice versa. L'éditeur propose alors NSX [Vmware 2013], une plate-forme de virtualisation réseau qui permet de créer des réseaux entiers sous forme logicielle et de les intégrer dans la couche hyperviseur, de sorte à les dissocier du matériel physique sous-adjacent et de les gérer depuis un point de contrôle unique et automatisé. De cette façon, les composants réseau peuvent être provisionnés en quelques minutes, sans avoir à modifier l'application, ni l'infrastructure. NSX propose une collection des services très riche allant du switch virtuel aux services de sécurité en passant par des services de routage, de pare-feu, de répartition de charge ou de VPN. La plate-forme permet la mise en œuvre d'architectures multilocataires (multitenant) avec garantie d'isolation entre les locataires. Elle est programmable via un jeu d'API riche et peut être déployée au-dessus de tout type d'infrastructure réseau, mais aussi au-dessus de tout type d'hyperviseur. Ses services seront utilisables par vCloud mais aussi par d'autres gestionnaires de cloud dont OpenStack et CloudStack. En revanche, les réseaux virtuels overlay, bases d'exécution des applications, ne supportent pas naturellement les serveurs *bare-metal*. Les passerelles jouent alors le rôle de pont, pour permettre aux overlays d'interagir avec le monde non virtualisé ou avec les sites externes. Aussi, NSX ne gère que l'environnement virtuel et délègue la gestion de l'environnement physique, imposant ainsi une gestion fragmentée. Il faudra alors avoir recours aux outils supplémentaires pour corréler les deux environnements et lever les frontières, afin d'accéder à une visibilité globale de bout en bout.
Avec NSX, le profil de communication des applications est statique, explicité par l'administrateur et le service réseau fourni aux applications est un réseau virtuel overlay qui offre une garantie de QdS souple à l'échelle d'un agrégat de flux.

– Dans [Cisco 2013], Cisco présente ACI (*Application Centric Infrastructure*), une infrastructure axée sur les applications, pour accélérer leur déploiement dans les centres de données. Selon l'auteur, les centres de données sont construits pour servir de cadre d'exécution pour les applications et, à ce titre et pour les besoins d'agilité, l'infrastructure

doit être à même de supporter efficacement d'une part, les applications indépendamment de leur architecture, leur ajustement à la hausse, à la baisse ainsi que leur retrait, et d'autre part, les machines hébergeant ces applications, qu'elles soient physiques (serveurs *bare-metal*) ou virtuelles (VMs ou containers), même lorsque ces dernières sont appelées à être mobiles. L'architecture de ACI suit les principes des SDN et sa logique s'organise comme suit : découper les organisations en formes de silos qui gouvernent le processus de déploiement des applications, en introduisant un langage (Group-Based Policy) d'abstraction réseau aligné sur celui des applications, pour permettre aux divers acteurs impliqués, de spécifier les besoins en infrastructure des applications ; compiler au moyen d'un contrôleur automatisé, à l'occurrence APIC (Application Policy Infrastructure Controller), ces besoins en langage orienté infrastructure, pour provisionner les ressources adéquates et les services réseau appropriés ; le service réseau fournit à chaque application locataire de l'infrastructure, est un réseau overlay lequel peut être réalisé à l'échelle du centre de données en utilisant le protocole VXLAN géré par APIC. Comme NSX, ACI applique le SDN en bordure du réseau, dans le logiciel, au niveau de l'hyperviseur, avec la particularité de supporter un environnement multihyperviseur. Aussi le réseau overlay peut également être de niveau 2 ou 3, il inclue les services réseau des couches 4 à 7 provenant de différents fournisseurs (par exemple Citrix [Citrix 2015]), a la spécificité de s'intégrer parfaitement avec l'infrastructure sous-adjacente, pour en former un réseau unique. De ce fait, ACI corrèle le fonctionnement du réseau avec celui des applications, fournit une visibilité sur le réseau et les applciations, propose une gestion centralisée, unifiée et automatisée via un seul point d'entrée, son contrôleur APIC.
Si l'équipementier est engagé dans des initiatives ou communautés OpenSource pour renforcer et assurer l'intégration et interopérabilité de sa solution, il ne compte pas délaisser son cœur de métier qui est le matériel. La solution ACI va plus loin que le SDN. En effet, en plus de APIC, l'infrastructure ACI est fournie avec des commutateurs Cisco Nexus 9000 dont le déploiement est aisé, grâce à ses capacités d'auto-configuration : les nouveaux équipements sont détectés et ajoutés automatiquement à l'infrastructure ; pour les mises à jours, l'APIC fournit une solution centralisée permettant de gérer les images système des équipements, ainsi que leur déploiement. L'architecture de commutation de l'ACI s'inspire du modèle résilient X-partite à deux sauts et utilise une topologie spine/leaf où chaque commutateur leaf se connecte à chaque commutateur spine via une liaison Ethernet 40 Gbit/s (pouvant évoluer jusqu'à 100 Gbits/s) pour de meilleures performances et une faible latence. Cette conception assure une évolutivité linéaire, des chemins d'accès multiples et est propice a un équilibrage de la charge.
Avec ACI, le profil de communication des applications est statique, et explicité par l'administrateur. Le service réseau fournit aux applications est un réseau virtuel overlay qui offre une garantie de QdS souple à l'échelle d'un agrégat de flux.

– Dans [Citrix 2015], l'ambition conjointe de Cisco et de Citrix est d'accélérer le déploiement des applications dans les centres de données, en combinant à la fois la mise en place des services de connectivité et la mise en œuvre des services des couches 4 à 7 indispensables à la haute disponibilité, aux performances acceptables, à une sécurité accrue et à

la visibilité des applications. Leur solution *Agile application-driven networks with Cisco ACI and NetScaler* intègre ACI et le contrôleur de mise à disposition des applications NetScaler. NetScaler apporte plusieurs fonctionnalités de couches 4 à 7 et fonctionne « en tant que service ». ACI surmonte les inconvénients de la mise en œuvre de ces services en s'appuyant sur deux techniques étroitement liées : l'insertion de services (diriger de façon dynamique les flux de trafic vers une machine préconfigurée (appliance) physique ou virtuelle fournissant l'un des services des couches 4 à 7) et le chaînage de services (diriger de façon dynamique les flux de trafic via une succession de machines préconfigurées physiques ou virtuelles, fournissant un ensemble de services des couches 4 à 7).
Comme avec ACI, le profil de communication des applications est explicité par l'administrateur et le service réseau fourni est un réseau virtuel overlay qui, en plus d'offrir une garantie de QdS souple à l'échelle d'un agrégat de flux, supporte les services réseau des couches 4 à 7.

– Oracle Solaris et Plurisbus Network ont travaillé de concert pour proposer une solution conjointe appelée *Enabling Application-Driven SDN in the Cloud* qui a pour but de simplifier le déploiement des applications et l'exploitation des infrastructures par ces applications. Comme présenté dans [Solaris 2015], leur offre permet à chaque application évoluant dans un environnement cloud, de piloter son propre réseau virtuel (VLAN) selon ses besoins et ceci dans le cadre d'un accord de niveau de service (SLA) qui se veut flexible : d'une VM à chacun de ces flux de données en passant par chaque application. Cette solution résulte du couplage de « Application-driven SDN » qui est un produit de Oracle Solaris et de l'hyperviseur réseau « Netvisor » de Pluribus Network. Le premier introduit la capacité de construire à travers un ensemble de systèmes complètement distribués, un switch virtuel (Elastic Virtual Switch ou plus court EVS) dans le but de s'affranchir du surcoût qu'induisent les protocoles liés aux réseaux overlay. Il fournit aussi aux applications le contrôle pour prioriser leur propre trafic en appliquant les configurations directement sur la périphérie virtuelle de l'infrastructure réseau. Avec Netvisor, les applications bénéficient d'une part, de la visibilité sur les conditions et les ressources disponibles sur l'infrastructure, laquelle leur est exposée comme un switch virtuel (autour duquel s'organise chaque LAN virtuel) ; et d'autre part, de la propagation d'un bout à l'autre de leur switch virtuel distribué, de la priorité souhaitée. Ceci a été possible en rendant l'infrastructure intermédiaire consciente des applications. Les avantages de cette offre tient à son architecture. En effet, selon les auteurs, les solutions précédentes introduisent une surcouche virtuelle au-dessus des équipements matériels, switchs et routeurs qui amènent en fait de la complexité et une gestion en silos des différents éléments de l'infrastructure. Avec leur solution, la couche virtuelle se voit déléguer par un LAN virtuel le lien avec les applications qui se trouvent souvent sans grande visibilité des couches sous-adjacentes et donc des ressources nécessaires à leur fonctionnement. Le LAN virtuel support des applications, s'appuie alors sur une partie programmable de la fabric qui constitue une grappe de fabrics applicatives, pour être programmé suivant le modèle SDN et selon les besoins de l'application. Cette partie de la fabric correspond aux *top of the rack switchs* fournis par Plurisbus. Cette solution est *hardware dépendante* au même titre que ACI.

Avec cette approche, le profil de communication inféré par le réseau, et le service réseau fourni aux applications est un LAN virtuel interconnectant les machines hébergeant les composants de l'application. Ce service offre une garantie de QdS stricte pouvant aller à l'échelle d'un flux élémentaire.

– Dans [Wang 2016, Zhang Gong 2016], Huawei présente ADN (*Application Driven Networking*) une architecture qui combine les principes de la 5G, du SDN et de NFV afin de construire les réseaux avec la flexibilité nécessaire pour répondre aux diverses exigences de différentes applications incluant celles existantes (par exemple : les communications Internet classiques, les services de communication vocale interactive) et celles utilisant les communications M2M qui incluent les applications ayant des caractères très différents. D'une part on distingue celles qui requièrent beaucoup de bande passante, de celles exigeant une faible latence et de celles dont la fiabilité est un impératif, et d'autre part celles dont le trafic suit la loi de Poisson, de celles dont le modèle de trafic suit une loi inédite dans le domaine des réseaux, appelées à émerger dans un future proche. Les réseaux du futur sont donc destinés à composer avec des applications qui présentent des différences très prononcées, tant au niveau du profil de leurs trafics, qu'au niveau des exigences en QdS associées à ces trafics. Selon le constructeur, ces éléments spécifiques à chaque type d'application devraient permettre de fournir des services réseau personnalisés. Avec l'architecture ADN, un service correspond à un réseau à part entière constituant une tranche (*slice* qui inclue le plan de données et de contrôle) du réseau sous-adjacent. Chaque tranche peut avoir sa propre architecture et exécuter les protocoles et mécanismes taillés pour satisfaire les exigences spécifiques des applications qu'elle supporte. Ces exigences sont inférées par le réseau lui-même, en identifiant automatiquement chaque application et en établissement progressivement son profil de communication à partir de ses flux de trafic, lesquels sont mesurés et analysés en utilisant les techniques d'apprentissage automatique. C'est sur la base de ces informations que le réseau estime les caractéristiques de chaque tranche, et détermine par la suite les ressources physiques à mobiliser pour sa construction. Les tranches sont indépendantes de l'infrastructure sous-adjacente qui peut être bâtie sur n'importe quel medium fixe (fibre optique, cuivre, etc) comme sans fils (LTE, etc).
Avec l'approche ADN de Huawei, le profil de communication des applications est inféré par le réseau, et le service réseau fourni est une tranche du substrat réseau, communément appelée *slice*. Ce service offre une garantie (en termes de débit, de latence et de fiabilité) souple à l'échelle d'un agrégat de flux de données.

Les travaux précédemment présentés adressent le problème de déploiement des applications dans les centres de données, avec en commun l'objectif d'accélérer la mise à disposition des applications. Les solutions proposées s'appuient sur les mêmes piliers : le SDN et la virtualisation en tant que technologies clés pour le provisionnement simple, rapide et efficace des services réseau supports des applications, même si les approches adoptées sont différentes et parfois guidées par les intérêts des auteurs. Dans tous les cas, les services offerts aux applications sont des réseaux virtuels, construits soit suivant le modèle de vir-

tualisation par superposition qui permet une prise en charge de la mobilité des workloads totalement indépendante de l'infrastructure sous-adjacente, soit suivant le modèle infrastructurel qui permet le contrôle et la visibilité des liens virtuels. Les liens virtuels sont destinés à supporter les communications entre les charges, sans se soucier individuellement de chaque flux de données échangé entre les composants tiers des applications. Notons que les auteurs de [Solaris 2015] ont tenté d'atteindre ce niveau de granularité, pour permettre le support des SLA à l'échelle des flux applicatifs dans les environnements de fournisseurs de Cloud. Malheureusement leur solution est couplée à leurs matériels. Les performances des liens virtuels sont tributaires des mécanismes de contrôle de trafic du substrat, lequel, basé sur une architecture IP, offre une connectivité d'un bout à l'autre avec une certaine QdS, généralement garantie à l'échelle d'une classe de trafic. Les caractéristiques des services réseau dérivent du profil de communication des applications, qui est dans tous ces travaux, spécifié par un administrateur lors du déploiement des applications et de ce fait, ne peut pas évoluer au rythme de la logique applicative, comme l'exigent les applications dynamiques, à l'exception de la proposition ADN de Huawei, dans laquelle c'est le réseau qui l'infère de lui même. Cependant les auteurs ne donnent pas de détails sur leur démarche, du moins, pas plus que l'outil utilisé, en occurrence l'*apprentissage automatisé*. De ce fait, ces solutions ne nous semblent pas adaptées aux applications dynamiques que nous ciblons.

Le prochain groupe de travaux que nous allons présenter, suppose implicitement que le déploiement des applications est effectif, et s'appliquent en majorité aux réseaux de campus. La philosophie de ces travaux issus du milieu de l'académique est que le support efficace des applications, passe par la connaissance détaillée de leurs besoins et exigences de communication tout au long de leur exécution. En effet, avec ces informations les applications peuvent recevoir à tout moment, les performances réseau souhaitées, au moyen d'une correspondance optimale de leurs trafics sur une infrastructure programmable. Cependant, pour y parvenir, les auteurs sont unanimes sur le fait qu'on ne peut plus se satisfaire de l'entête des paquets comme interface de communication entre les applications et le réseau. En revanche ils se divisent sur le procédé. Pendant que certains de ces travaux cherchent dans quelle mesure l'application qui, naturellement connaît à tout moment ses besoins de communication, peut directement contrôler le réseau, d'autres investiguent sur la manière avec laquelle le réseau peut établir et maintenir le profil de communication des applications, pour lui-même décider et appliquer les règles de contrôle adéquates.

– Dans [Qazi 2013], Qazi et al. propose ATLAS, un framework qui décrit une façon de fournir aux réseaux SDN, la capacité d'identifier dynamiquement les applications qui communiquent à travers le réseau. ATLAS repose sur une technique d'apprentissage automatique supervisée, dont la phase d'apprentissage inclut un agent qui s'exécute sur les terminaux des utilisateurs, en l'occurrence les employés d'une entreprise. Ce choix est motivé par le fait que plusieurs entreprises s'autorisent de déployer, pour des raisons de gestion, les agents logiciels sur les équipements mis à la disposition de leurs employés. Ces agents collectionnent les informations sur les sockets réseau actives et les applications les utilisant, et les envoient au plan de contrôle réseau où s'exécute un programme d'apprentissage auto-

matisé. A partir de ces informations, le programme va déduire la caractéristique distincte des flux de chaque application, laquelle sera alors corrélée à la politique de contrôle spécifiée par un administrateur, pour définir les règles de correspondances et de traitement de chaque flux au niveau des équipements d'acheminement. Après la phase d'apprentissage, lorsqu'un nouveau flux entrera dans le réseau en provenance d'un utilisateur autre qu'un employé de l'entreprise, il pourra être identifié et traité convenablement.
Avec ATLAS, le réseau infère le profil de communication des applications et applique un traitement différencié à l'échelle d'un agrégat de flux dont les exigences de QdS n'évoluent pas dans le temps.

– Santos et al. [Santos 2017] ont présenté NEAT, un framework pour rendre les environnements réseaux SDN conscients des applications. Le but est d'enrichir les connaissances que possède un réseau au sujet des applications qui s'exécutent sur les terminaux qui lui sont connectés. Les auteurs de NEAT cherchent à renforcer l'intégration des terminaux et du réseau. Le procédé pour y parvenir est résumé comme suit : les terminaux utilisateurs disposent d'un agent NEAT local leur permettant d'interagir avec le réseau via son plan de contrôle à travers une API dédiée. Par le biais de cette interface, les terminaux peuvent alors établir un rapport de confiance avec le réseau, lequel sert de support aux applications pour décliner leur identité et spécifier les exigences de QdS pour chacune de leurs connexions ou pour consulter les conditions courantes du réseau et s'y adapter lorsque possible. Cette démarche permet de contourner le manque de visibilité sur des flux en transit sur le réseau causé par le tunneling du trafic applicatif.
Avec NEAT, les applications explicitent leur profil de communication au réseau qui affecte par la suite leur flux sur des chemins de données pré-calculés et établis à l'avance.

– Montazerolghaem et al. se sont intéressés aux applications utilisant le protocole SIP (comme la VoIP, la réalité virtuelle). SIP est un protocole de signalisation de la couche applicative, normalisé et standardisé par l'IETF. Il a été conçu pour la création, la modification et la terminaison des sessions de communication multimédia. C'est au moyen des négociations SIP que les participants à une session de communication, déclinent l'identité des flux qu'ils comptent émettre et s'accordent sur les codecs à utiliser pour chaque media échangé. Les auteurs présentent OpenSIP [Montazerolghaem 2017], un framework qui cherche à rendre le réseau conscient des applications SIP, en lui révélant la description de chacun des flux média SIP en transit sur l'infrastructure, ainsi que ses besoins en QdS et son état courant. Pour établir et maintenir le profil de communication des applications, les paquets de données SIP sont envoyés depuis les commutateurs OpenFlow vers le contrôleur SDN où fonctionne un programme d'inspection profonde de paquets (DPI). Le DPI traite les paquets pour extraire les messages SIP qui sont par la suite interprétés. OpenSIP n'est pas intrusif vis-à-vis des terminaux utilisateurs, en revanche, il exige que les commutateurs OpenFlow exécutent un DPI pour pouvoir raisonner au delà des couches 2 à 4, afin d'identifier les paquets appartenant à un trafic de signalisation SIP.
Avec OpenSIP, le profil de communication des applications est inféré par le réseau qui établi ensuite sur l'infrastructure les chemins de données pour le

support des flux de l'application.

– Young et al. [Choi 2016] ont traité les applications reparties reposant sur l'intergiciel DDS qui est très plébiscité pour le support des systèmes distribués critiques. Comme les auteurs de OpenSIP, leur approche se base sur l'analyse du trafic de signalisation. Avec DDS, il est possible de caractériser très finement les besoins d'une application : les flux applicatifs et leurs besoins en QdS. En effet, reposant sur un modèle de coopération en « publish/subscribe », une application DDS est une collection de processus (ou programmes) applicatifs qui produisent des données auxquelles d'autres processus applicatifs souscrivent pour les consommer. Ainsi, à partir de la liste des producteurs des différentes données applicatives et de leurs consommateurs respectifs, il est possible de connaître à tout moment les flux de données échangés avec leur source et leur(s) destination(s). Puisque DDS permet aux applications de spécifier (même en milieu d'exécution) les exigences de QdS relatives à la distribution des différentes données applicatives, il est également possible de déterminer, à tout moment, les besoins en QdS associés aux flux de données (exprimées en termes de débit minimal et latence maximale). DDS intègre un service de découverte qui permet, à tout moment, aux producteurs de données de se déclarer sur le réseau et d'y annoncer les données (et la QdS offerte) qu'ils souhaitent produire et similairement aux souscripteurs de se déclarer sur le réseau et d'y annoncer les données (avec la QdS requise) auxquels ils souhaitent souscrire. Ce service permet également d'annoncer sur le réseau tout autre changement dont un désabonnement, un arrêt de production de données ou un changement de QdS. Ces annonces se font au moyen du protocole Simple Discovery Protocol (SDP) qui utilise le format de données RTPS (Real-Time Publish Subscribe). C'est en collectant les paquets de données SDP au niveau du contrôleur et en interprétant les messages RTPS que le réseau est conscient des applications DDS. **Avec cette approche, le profil de communication de l'application est inféré par le réseau qui sélection ensuite le chemin satisfaisant les contraintes de délai avant d'appliquer aux flux de données, le niveau de priorité correspondant.**

– King et al. ont délégué le contrôle du réseau à l'intergiciel DDS. Ils ont développé leur démarche dans [King 2014], où ils présentent MIDAS (MIDdleware Assurance Substrate), et décrivent comment le middleware est capable d'offrir une garantie stricte de la QdS réseau aux applications. La motivation d'un tel choix est que les besoins en QdS des applications ne sont pas à priori connus, et peuvent potentiellement évoluer dans le temps. Par ailleurs, le caractère statique des services réseau actuels, ne leur permet pas de servir efficacement les applications DDS. Avec MIDAS, l'intergiciel, qui connaît à tout moment les flux de données échangés entre les producteurs et les consommateurs, ainsi que leurs besoins en QdS, pourra, au moyen du protocole OpenFlow, directement (sans passer par un contrôleur SDN) programmer les fonctions d'acheminement et de traitement de paquets des nœuds de transfert de manière à satisfaire les applications. Ainsi, en plus de leurs fonctions principales, chaque agent middleware se comporte comme une sorte de contrôleur SDN pour les applications qu'il supporte. A ce titre, chacun des agents devra alors découvrir l'infrastructure réseau pour prendre des décisions de contrôle, les disséminer aux nœuds de transfert et réagir lorsqu'un changement survient sur l'infrastructure.

Ce mode de fonctionnement ne nous semble pas facilité le passage à l'échelle. De plus, en branchant directement le middleware à l'infrastructure, le réseau devient permissif et donc plus sensible aux failles de sécurité et aux erreurs.
Avec MIDAS, le middleware établit le profil de communication des applications et fourni une QdS garantie et potentiellement dynamique, en combinant les mécanismes de contrôle d'accès et de gestion priorité.

– Convaincus qu'une synergie entre les applications d'utilisateurs finaux et le réseau présente de nombreux avantages, Ferguson et al. ont travaillé à concevoir, implémenter et évaluer une API surnommé PANE (*PArticipatory NEtworking*) [Ferguson 2013], pour permettre aux applications de travailler en dialoguant avec le réseau. Au lieu de permettre aux applications de directement contrôler le réseau, elles le feront par l'intermédiaire du contrôleur SDN d'un réseau PANE, en utilisant l'API PANE pour spécifier leurs besoins, qui peuvent évoluer au cours du temps.
Avec PANE, les applications explicitent leur profil de communication et le réseau calcul et réserve les ressources adéquates pour garantir les exigences de QdS potentiellement dynamiques.

1.6 Notre approche générale

1.6.1 Notre vision

Nous défendons la vision selon laquelle, la conception et l'évolution des réseaux devraient être guidées par les applications qui s'exécutent sur le réseau et les utilisateurs qui exploitent ces applications, au lieu de se focaliser en priorité sur les aspects techniques tels que la résilience, le passage à l'échelle, l'optimisation des opérations et l'utilisation efficace des ressources, comme cela a été le cas avec les réseaux traditionnels qui présentent aujourd'hui les limites face à la pression de plus en plus importante des technologies et des usages émergents. Les réseaux de demain sont appelés à être plus intelligents au point de comprendre plus facilement les besoins des applications, prendre les décisions par eux-mêmes et doivent être flexibles pour s'adapter en conséquence. En ce qui concerne les applications, nous pensons qu'elles ne devraient pas être spectatrices du destin de leur performance et encore moins que les utilisateurs doivent en payer les frais.

1.6.2 Définition d'un réseau ADN et positionnement

Notre ambition est de construire un réseau guidé par et pour les applications ou plus court un « réseau ADN (Application Driven Network) ». En d'autres termes, un réseau capable de fournir à *chaque* application un service d'acheminement *flexible* et *personnalisé* qui *répond* et *colle* à ses besoins dynamiques, en réservant des ressources *instantanément* et *automatiquement*. Cela implique que le fonctionnement du réseau (son contrôle) doit, dans une certaine mesure, être guidé par les applications.

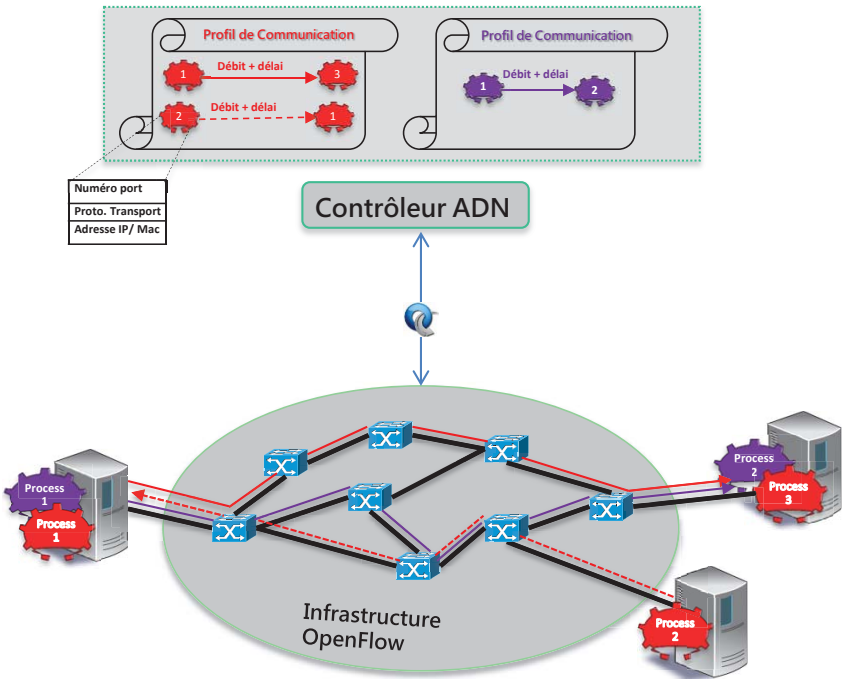


FIGURE 1.8 – Vue haut niveau de l'approche ADN

Profil de communication Les applications visées par ce type de réseau sont **dynamiques** et exigent une certaine QdS qui peut être **plus ou moins stricte**. Le caractère dynamique de ces applications signifie que les flux applicatifs échangés entre les processus de l'application ou entre celle-ci et les utilisateurs, peuvent évoluer dans le temps et que les exigences de QdS (en termes de **débit** et **latence**) pour délivrer ces flux peuvent, à leur tour, changer dans le temps. Chacun de ces flux peut correspondre à un trafic Est-Ouest échangé entre les composants d'une application ou entre les micro-services le constituant, ou alors à un trafic Nord-Sud entre l'application et un ou plusieurs utilisateurs finaux. L'ensemble de ces flux de données ainsi que leurs exigences en QdS constituent le profil de communication de l'application. Ce profil peut être exprimé **explicitement** par l'application ou **inféré** par le réseau.

Service réseau Le service réseau provisionné correspond à un ensemble de **chemins de données** établis sur l'infrastructure, pour supporter les flux applicatifs de manière à garantir la QdS requise par chaque **flux élémentaire**. L' un des intérêts d'un tel choix est de protéger efficacement les flux critiques contre les irrégularités au niveau réseau, lesquelles sont de plus en plus prononcées, difficiles à prévenir et à maîtriser.

Service réseau flexible et personnalisé Signifie que le service réseau change dans le temps pour être adapté aux besoins spécifiques de l'application qui évoluent aussi dans le temps. L'une des motivations de ce choix est que, les applications modernes présentent un profil de communication qui varie parfois selon la logique propre à chaque application. Du fait de cette singularité, ces applications nécessitent d'être traitées distinctement pour une qualité d'expérience usager irréprochable.

Service réseau qui répond et colle Signifie que les performances du service satisfont les exigences de l'application et que les ressources réseau utilisées pour instancier le service sont taillées à la mesure des besoins de l'application.

Ressources réservées instantanément et automatiquement Signifie que la réservation effective des ressources pour le compte de l'application est effectuée "juste à temps" c'est-à-dire quasiment au moment où l'application souhaite les utiliser et par le biais d'un programme automatisé.

Pour offrir un service d'acheminement personnalisé, un réseau ADN repose sur un acheminement basé flux (et non pas basé sur une adresse identifiant un terminal destinataire ou un groupe de terminaux). En d'autres termes, à la réception d'un paquet, un nœud d'un réseau ADN :

- identifie l'application et/ou le flux applicatif auquel appartient le paquet (et non pas le ou les nœuds terminaux destinataires) ;
- déduit les ressources réseau qui lui sont destinées ou qui lui sont nécessaires ;
- mobilise ces ressources pour que le paquet reçoive le traitement attendu.

La figure 1.8 présente une vue haut niveau de l'approche ADN.

1.7 Conclusion

Ce chapitre a pour objectifs principaux d'une part, de dresser un état de l'art sur les travaux traitant tout ou une partie des aspects liés à la problématique générale de notre travail et d'autre part, de présenter l'approche que nous avons adoptée et qui sera développée dans la suite de ce manuscrit.

La première partie a présenté les notions préliminaires requises pour appréhender au mieux les différents aspects que nous allons aborder dans la suite de ce travail. Le concept de réseaux SDN est introduit et précisé. Les principaux éléments d'une architecture réseau SDN ont été listés en se focalisant sur le protocole OpenFlow en tant que Interface Sud la plus aboutie et adoptée. La structure et les composants d'un commutateur OpenFlow, dont les tables d'acheminement de paquets, ont été décrits ainsi que les messages qu'échange un commutateur OpenFlow avec un contrôleur SDN.

Dans la deuxième partie de ce chapitre, la problématique est formulée et développée suivant deux aspects : sa *portée* au regard des environnements réseau à considérer et les *constats* relatifs aux difficultés opérationnelles et limites technologiques actuellement rencontrées par les administrateurs réseau dans ces environnements.

Dans la troisième partie de ce chapitre, une synthèse des travaux traitant directement ou indirectement la problématique générale est fournie, en utilisant une taxonomie que nous avons définie dans le but de positionner chacun de ces travaux. La problématique abordée étant

d'actualité et sujet à un fort intérêt, les travaux qui y sont présentés sont relativement récents et émanent du milieu académique et industriel.

La dernière partie révèle tout d'abord, la vision que nous portons sur les futures applications et les réseaux de demain, lesquels réseaux seront sans aucun doute autant intelligents que ces applications et pourront ainsi mieux les comprendre et donc coexister en étant parfaitement intégrés. Aussi, notre vision du concept ADN y est présentée. Cette présentation inclut la définition d'un réseau ADN ainsi que son positionnement par rapport aux travaux existants.

Chapitre 2

Architecture générale d'un réseau ADN

Sommaire

2.1 Conception d'un réseau ADN 29
2.2 Architecture générale du réseau ADN 33
2.3 Architecture fonctionnelle du réseau ADN 34
2.4 Synthèse 51
2.5 Conclusion 52

Ce chapitre présente les grandes lignes de conception du réseau ADN tel que présenté dans la section 1.6.

Dans ce qui suit, nous rappelons les attentes des applications ciblées vis-à-vis du réseau. Notre démarche de conception est ensuite formulée à travers un certain nombre de principes et de choix de conception. L'architecture générale proposée par la suite est décrite en utilisant une méthode indépendante de toute plate-forme (PIM : Plateform Independent Model) : le formalisme UML (Unified Modeling Language). L'architecture de référence ADN inclut l'ensemble de composants fonctionnels qui doivent-être implantés. Une analyse des exigences fonctionnelles de chacun de ces composants est proposée, leurs interfaces sont précisées, les relations entre eux ainsi que leurs interactions sont illustrées au moyen de diagrammes UML.

2.1 Conception d'un réseau ADN

Cette section présente le système ADN, recense les acteurs interagissant avec le système ainsi que les grandes fonctionnalités qu'il fournit à chacun de ces acteurs. La figure 2.1 décrit graphiquement les cas d'utilisation du système selon le formalisme UML. Il s'agit de la modélisation du système sous l'angle de ses responsabilités vis-à-vis de ses utilisateurs (acteurs).

2.1.1 Les applications et les opérateurs au centre de la conception

Les différentes fonctionnalités du système sont mises à la disposition des acteurs suivants :

– ***Application utilisateur*** : C'est l'utilisateur principal du système. Il s'agit d'une programme informatique constitué de plusieurs composants s'exécutant sur des plate-formes interconnectées à travers le réseau. Plus précisément, il s'agit d'une application dynamique

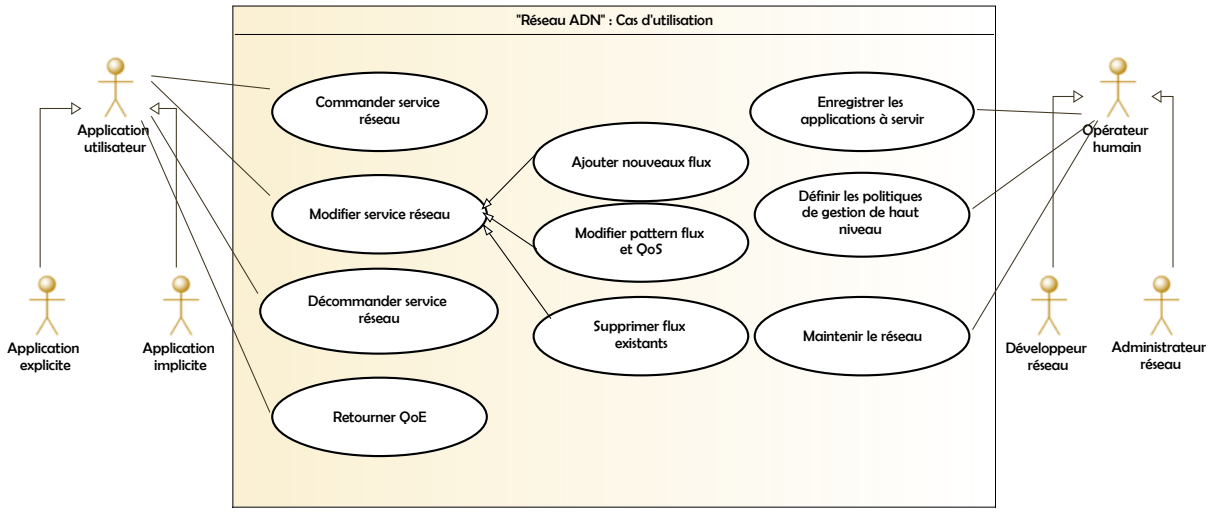


FIGURE 2.1 – Cas d'utilisation du système ADN

qui exige une certaine QdS qui peut être plus ou moins stricte. Le caractère dynamique de cette application signifie que les flux applicatifs échangés entre les processus de l'application peuvent évoluer dans le temps et que les exigences de QdS (en termes de débit et latence) pour délivrer ces flux peuvent, à leur tour changer dans le temps. Suivant le formalisme UML, il s'agit d'un acteur primaire du système, en ce sens qu'il déclenche les cas d'utilisation. Dans notre approche ADN, on distingue deux types d'applications :

- ***Application explicite*** : Il s'agit des applications qui décrivent explicitement leurs besoins au réseau. Une interface unique doit être mise à leur disposition afin de leur permettre de décliner leur identité et d'exploiter les fonctionnalités du système.
- ***Application implicite*** : Il s'agit des applications qui, contrairement à celles explicites, exigent que le réseau infère par lui-même leurs besoins. Ce type d'acteur correspond aux applications incapables de communiquer explicitement leurs besoins. Dans cette catégorie, on peut distinguer les applications "héritées" ou plus généralement, les applications qui n'ont pas été conçues avec à l'esprit la possibilité de communiquer avec le réseau ADN, à l'instar des applications existantes. pour ce type d'applications, c'est le réseau qui infère par lui même leurs besoins en reposant sur des techniques de classification des applications.

Dans les deux cas, les applications doivent être au préalable enregistrées par un opérateur humain pour pouvoir prétendre bénéficier des services ADN. Les fonctionnalités qui leurs sont fournies sont représentées par les cas d'utilisation qu'elles déclenchent. Puisque notre approche présente le réseau aux applications comme une plate-forme de service, l'application peut donc : commander, modifier et décommander un service réseau. Le cas d'utilisation *modifier service réseau* peut être précisé en trois sous-cas, à savoir : *ajouter nouveaux flux*, *modifier pattern flux et QdS* et *supprimer flux existants*. Ces sous-cas d'utilisation sont déclenchés et répétés autant de fois, selon l'évolution des besoins de l'application. Aussi, l'application peut retourner au réseau, en milieu d'exécution, la qualité du service reçue. Cette fonctionnalité représentée par le cas d'utilisation *Retourner*

QoE, permettra au réseau d'optimiser la QdS qu'il offre aux applications. Grâce à cette fonctionnalité, les applications et le réseau pourront coexister de manière plus intégrée.

- ***Opérateur humain*** : C'est une personne physique dotée de compétences lui permettant de gérer le réseau. La gestion du réseau peut être effectuée par deux types d'opérateurs :
  - ***Administrateur réseau*** : Les missions de cet acteur relèvent de l'exploitation du réseau. L'une de ses tâches consiste à déclarer les applications autorisées à exploiter les fonctionnalités du système pour bénéficier des services réseau ADN. Le choix des applications à déclarer peut être guidé par leur importance vis-à-vis des besoins métiers. Il supervise le réseau pour continuellement assurer son bon fonctionnement en intervenant par exemple en cas de pannes matérielles ou lorsqu'une application notifie un problème de performance persistant, que le réseau n'a pas été capable de déceler ou de traiter lui-même. Dans ce dernier cas, l'administrateur doit, flux après flux, remonter le chemin de données emprunté pour localiser la source du problème et appliquer les actions correctrices.
  - ***Développeur réseau*** : Cet acteur agit dans le cadre de la maintenance évolutive et correctrice du système. Il est défini comme une personne ayant la capacité de corriger des bugs logiciels, et de développer et introduire de nouveaux modules dans le système pour le faire évoluer. A titre d'exemple, pour inférer les besoins des applications utilisant le protocole SIP, le réseau requiert une connaissance sur ce protocole pour pouvoir comprendre les négociations entre les composants applicatifs. Le développeur intervient alors pour enrichir le réseau de cette connaissance. Il intervient aussi dans la phase opérationnelle du système pour lui dicter les objectifs qui seront automatiquement traduits en directives réseau à appliquer au niveau des équipements. Un exemple d'objectif peut concerner les critères à optimiser pour une utilisation efficace des ressources. Le réseau tentera en permanence d'atteindre cet objectif. Comme nous cherchons à réduire davantage l'intervention humaine, le réseau peut lui-même se donner des objectifs selon l'évolution du contexte et sur la base des informations dont-il dispose, lesquelles peuvent être au besoin, enrichies par le développeur réseau.

2.1.2 Principes de conception

L'idée centrale d'un réseau ADN est de fournir des services (i.e. des chemins de données) personnalisés aux applications sur la base d'une description fine des flux applicatifs échangés et de leurs besoins en terme de QdS (que l'on appellera dans la suite : le profil de communication de l'application). Les motivations d'un tel choix sont qu'une connaissance précise des besoins applicatifs est une condition nécessaire pour déployer le « bon » service (celui qui respecte avec précision les besoins) avec une utilisation optimale des ressources. En comparaison aux travaux antérieurs visant la fourniture de services avec un certain niveau de QdS, c'est l'une des caractéristiques clef de l'approche que nous proposons. Il est à noter que cette connaissance fine du profil de communication d'une application n'est pas une condition sine qua non pour l'utilisation des services ADN, comme expliqué ci-avant.

Le profil de communication d'une application peut être établi par le réseau ADN à partir d'une description explicite des flux à échanger et de leurs exigences de QdS en termes de débit et de latence. Au besoin, le réseau ADN peut compléter ce profil par le déploiement de mécanismes d'estimation du trafic applicatif. Ces profils peuvent être soumis explicitement par les applications elles-mêmes ou par un agent applicatif externe avec potentiellement un administrateur dans la boucle ou alors implicitement (c'est à dire à travers des dialogues effectués sur le réseau au moyen de la signalisation niveau applicatif) communiqué au réseau ADN. Dans ce dernier cas, des mécanismes d'inspection profonde de paquets (que l'on notera DPI pour Deep Packet Inspection) alimentés avec les signatures applicatives pertinentes sont déployés aux extrémités du réseau pour identifier et intercepter le trafic de signalisation des applications supposées utiliser le service ADN. Les paquets appartenant à ce trafic seront alors analysés, pour écouter les négociations entre les composants applicatifs, dans le but d'inférer les flux qu'ils souhaitent échanger ainsi que la QdS associée. Pour les applications qui ne communiquent pas (ni explicitement, ni implicitement) leurs besoins, en s'appuyant sur les protocoles tels que NetFlow et sFlow, les techniques d'estimation statistique du trafic peuvent être utilisées pour les évaluer en temps réels, moyennant une surcharge supplémentaire. Il est clair que, dans ce cas de figure, la précision et l'exactitude du profil de communication sont impactées. En s'inspirant du système de contrôle en boucle fermée, on peut compenser cette possibilité d'inexactitude en optimisant en continu cette estimation, sur la base du *feedback* des applications lorsque celles-ci renseignent, au cours de leur exécution, sur leurs performances courantes liées à l'utilisation du réseau.

Un autre principe est que le service réseau fourni aux applications est exprimé sous la forme d'un réseau virtuel (que l'on notera VNET pour Virtual NETwork) composé d'un ensemble de liens virtuels de bout-en-bout. Chaque lien virtuel est soit de type point-à-point, soit, point-multipoints, et est caractérisé par une capacité de transmission et un délai de transfert maximal à respecter. Il y a une adéquation entre les besoins de chaque flux et les caractéristiques du lien virtuel qu'emprunteront ses paquets. Contrairement à une approche de virtualisation par superposition, les VNETs sont définis au moyen de stratégies faisant correspondre les flux au réseau virtuel approprié à l'aide des champs protocolaires des couches 2 à 4 de l'entête des paquets, lorsque celles-ci permettent de définir exclusivement un flux donné. Dans le cas contraire, soit les paquets de chaque flux seront alors marqués à l'entrée du réseau, en utilisant l'identité du lien virtuel auquel correspond le flux, soit, au niveau de chaque nœud de transfert traversé, chaque paquet sera sujet d'une analyse profonde pour identifier à quel flux il appartient, afin qu'il reçoive le traitement approprié. Cette approche de mise en œuvre de la virtualisation est souvent désignée sous le terme de virtualisation infrastructurelle (« fabric-based »). Grâce aux propriétés de la virtualisation, à savoir, le partage, l'isolation et l'indépendance, il est plus facile d'assurer une colocation efficace des services réseau distincts destinés à servir les applications différentes, sur la même infrastructure.

Étant donné que les flux de données (et leurs besoins) sont susceptibles de varier dans le temps, le VNET à provisionner est à son tour dynamique aussi bien au niveau de sa topologie que de la capacité de ses liens. Nous reposons sur une infrastructure réseau de type SDN qui, par

sa possibilité d'être programmée en temps-réel, permet le support efficace de services réseau dynamiques, lesquels peuvent être construits et déployés automatiquement sur l'infrastructure, et modifiés à la volée. Cette infrastructure peut être entièrement ou partiellement dédiée au réseau ADN. Dans ce dernier cas, une technique de virtualisation réseau est au préalable appliquée à l'infrastructure physique pour affecter exclusivement des partitions (ou slices) des éléments du réseau à l'infrastructure SDN/OpenFlow virtuelle sur laquelle reposera le réseau ADN. Avec le concept SDN, les applications peuvent facilement communiquer avec le réseau via l'interface Nord. On saisira cette opportunité pour renforcer le dialogue entre application/réseau, en aménageant cette interface, afin de mettre davantage les applications à contribution, notamment d'une part, pour décliner leur identité et exprimer leurs besoins, d'autre part pour notifier leur problème de performances au réseau, tout ceci dans un rapport de confiance. Le support de l'acheminement basé flux (flow-based forwarding) imposé aux nœuds par le modèle SDN de l'ONF que nous utilisons ici, permettra d'envisager un contrôle de flux plus fin que ce que propose les équipements traditionnels.

Il est clair que le fait de viser les applications critiques ayant des besoins de QdS évolutifs, en offrant une garantie de QdS fine à l'échelle d'un flux applicatif élémentaire, au moyen des services réseau dynamiques, requiert beaucoup d'attention. L'attention du plan de contrôle réseau pour établir, maintenir le profil de communication des applications et surveiller le réseau et les services réseau déployés ; l'attention des équipements de transfert pour maintenir les règles d'acheminement correspondant à chacun des flux critiques qui les traversent ; l'attention des administrateurs pour communiquer au réseau les règles d'affaires, une connaissance sur les applications pouvant consommer les services ADN, et pour réagir en cas de panne matérielle ou logicielle. Il est donc crucial qu'un réseau ADN puisse composer avec ces divers facteurs pouvant constituer une résistance à l'échelle, en optimisant en permanence les états maintenus, les ressources utilisées et les coûts opérationnels. Le dernier principe de conception est donc de construire un réseau ADN avec des capacités autonomiques. Nous ciblons les propriétés d'auto-configuration, d'auto-protection et d'auto-optimisation. Le support des fonctionnalités d'autogestion rendra le réseau encore plus intelligent. Cette intelligence lui ouvre la voie pour être prêt à faire face aux défis à venir.

2.2 Architecture générale du réseau ADN

La figure 2.2 présente l'architecture générale ADN. Cette dernière met en évidence l'application de contrôle réseau (ou fonction réseau) « ADN service provisioning » dont le rôle est de provisionner les réseaux virtuels pour des applications sur une infrastructure de type SDN/OpenFlow. Cette application s'interface d'un côté avec les applications des utilisateurs finaux afin d'établir leur profil de communication, et de l'autre côté, avec le contrôleur SDN pour déployer les services associés.

La figure 2.3 présente le choix que nous avons envisagés pour développer la fonction de provisionnement. D'autres alternatives sont envisageables : une première possibilité serait de reposer sur une interface nord du contrôleur de haut niveau : Soit, en utilisant des langages de programmation réseau de haut-niveau tels que par exemple Pyretic [Monsanto 2013] ou Procera

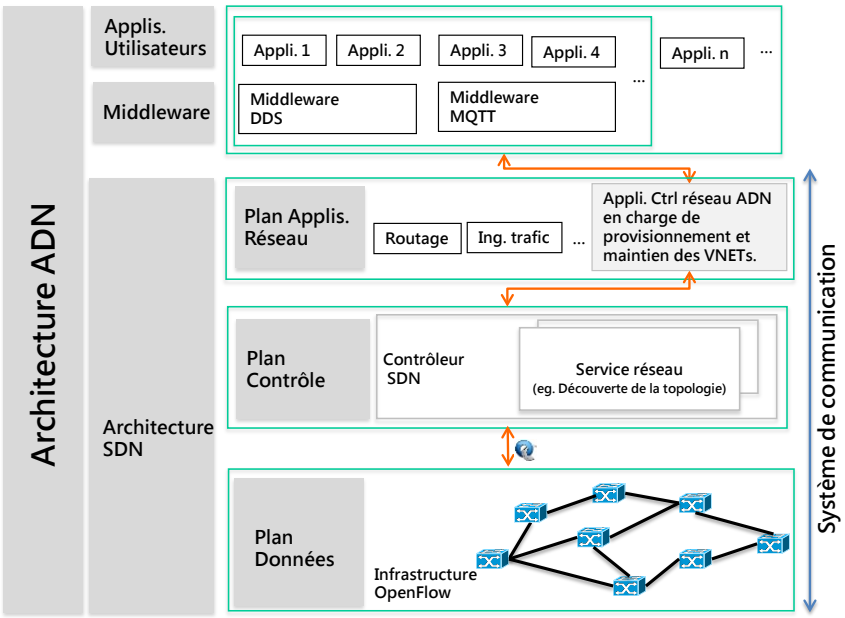


FIGURE 2.2 – Architecture générale ADN

[Voellmy 2012], soit, en reposant sur des services réseau offerts par certains contrôleurs SDN (certains contrôleurs offrent des services capables de provisionner des réseaux virtuels sur une infrastructure SDN).

Ces précédentes alternatives présentent néanmoins des limites par rapport à nos attentes et notamment, l'impossibilité de provisionner des liens virtuels point-multipoints et de spécifier des exigences qui combinent le délai et le débit. De plus, en l'état actuel des choses, ils reposent sur des algorithmes d'allocation de ressources simples (qui seront certainement amenés à évoluer) et dont certains n'ont pas comme objectif principal l'utilisation optimale des ressources réseau. En conséquence, le choix effectué dans ce travail est de reposer sur une interface nord de bas niveau où l'on spécifie au contrôleur les règles OpenFlow à installer au niveau de chaque commutateur de l'infrastructure SDN (ce choix est représenté sur la figure 2.3 par la flèche pleine orange). De ce fait, nous faisons le choix de ne pas utiliser un hyperviseur réseau qui aurait pu prendre en charge le déploiement du réseau virtuel et la ségrégation entre réseaux virtuels. Dans ce travail, ce déploiement est réalisé par les règles OpenFlow produites par notre fonction de provisionnement ; la ségrégation des réseaux virtuels est assurée par la cohérence entre les règles OpenFlow générées pour les établir sur l'infrastructure.

2.3 Architecture fonctionnelle du réseau ADN

La construction du réseau ADN sur la base des principes énoncés à la section 2.1.2, nécessite la mise en œuvre d'un certain nombre de composants fonctionnels. La Figure 2.4 décrit l'architecture fonctionnelle de l'application de contrôle réseau « ADN service provisioning » qui implémente l'approche ADN que nous proposons. Le diagramme de la figure 2.5 décrit plus ex-

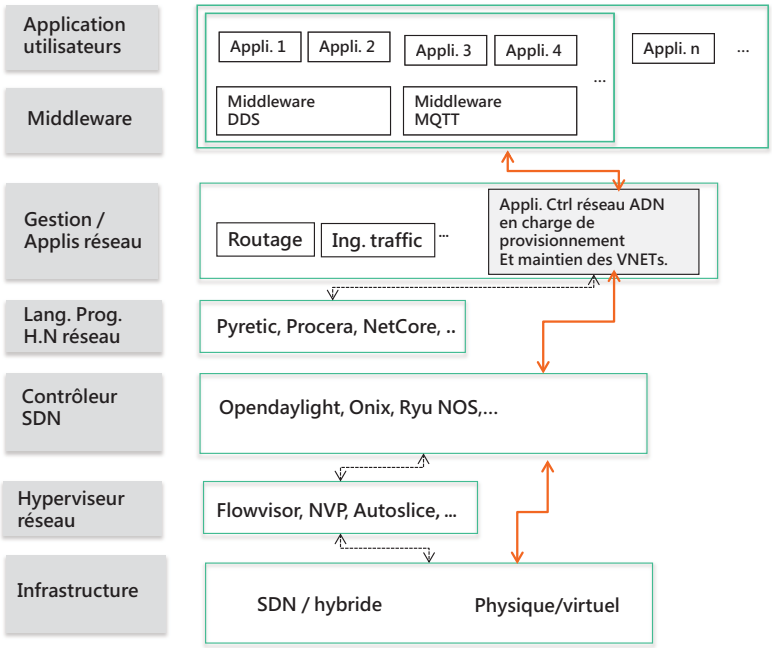


FIGURE 2.3 – Choix de conception relatifs à l'application de provisionnement

plicitement les composants identifiés en mettant en avant les ports de l'application de contrôle réseau ainsi que les interfaces fournies et requises par chacun de ses composants. L'analyse fonctionnelle des différents composants est présentée ci-après.

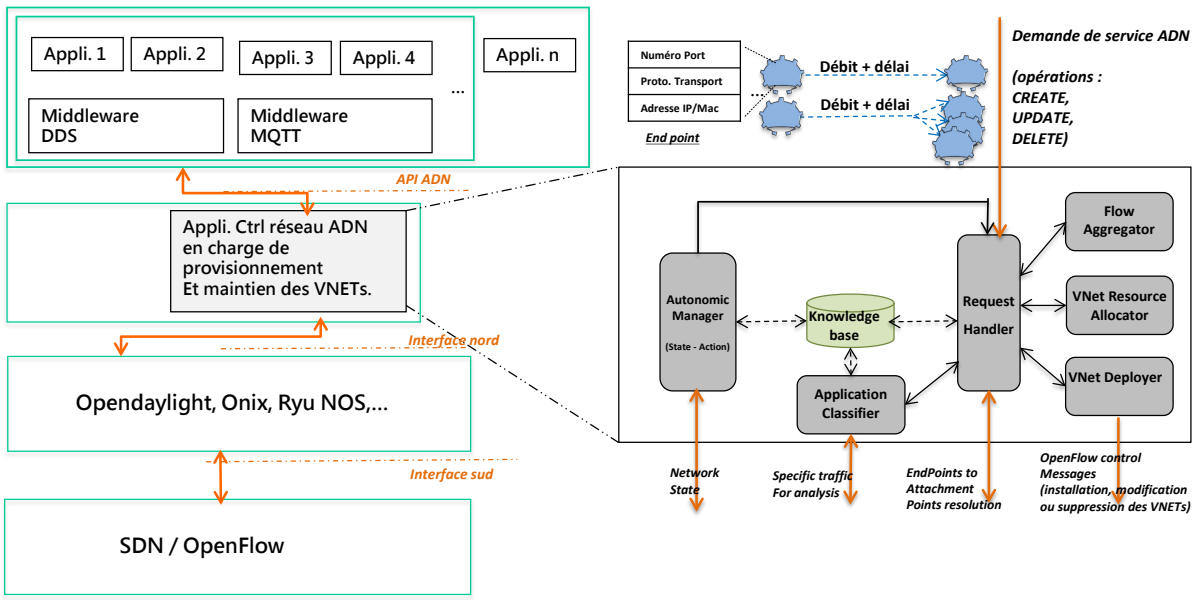


FIGURE 2.4 – Architecture fonctionnelle ADN

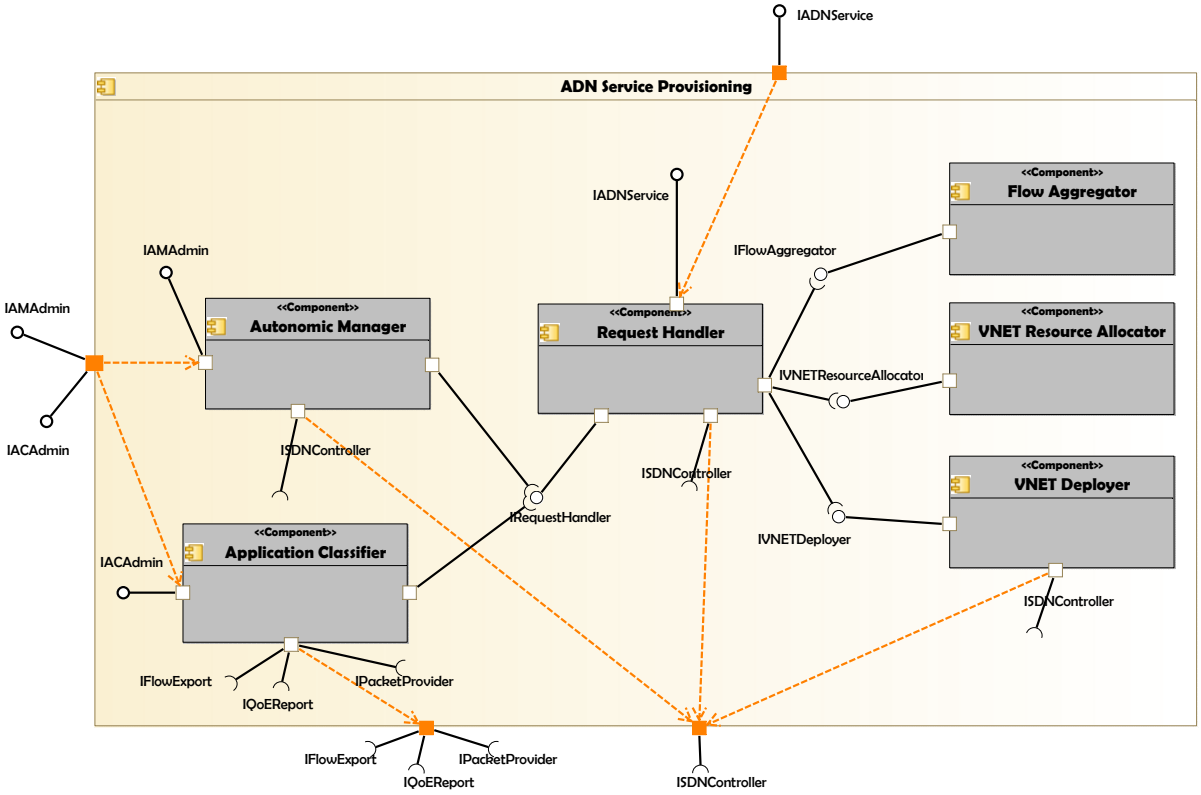
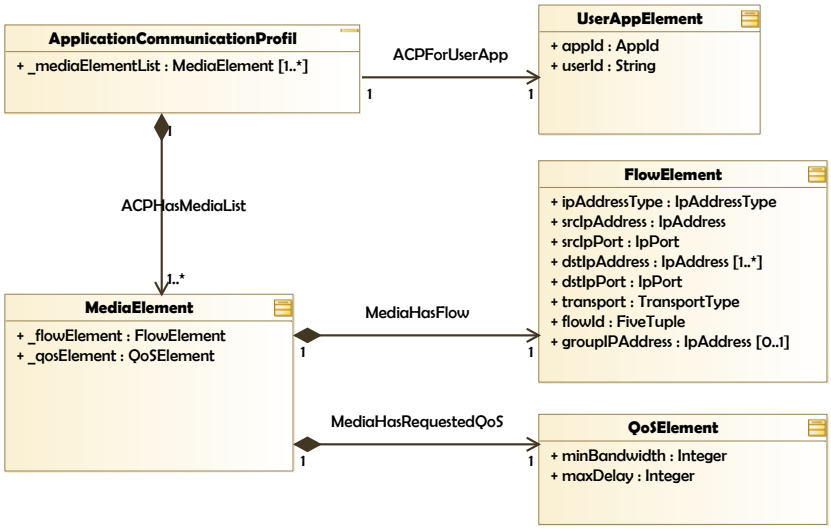


FIGURE 2.5 – Diagramme de composants de l' *ADN Service Provisioning*

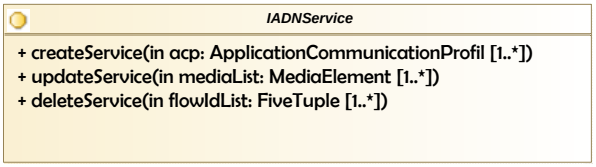
2.3.1 Request Handler

Ce composant joue le rôle de « front-end » de l'application de contrôle réseau et orchestre l'exécution des différents composants impliqués dans le traitement d'une demande de création, de modification ou de suppression d'un service.

Une bonne pratique de conception est d'essayer de limiter non seulement le couplage entre les composants internes du système, mais aussi entre le système et ses utilisateurs. C'est ce que propose le motif de conception *façade* que nous avons utilisé pour organiser les composants de l'application de contrôle réseau ADN. Ce faisant, les applications n'auront qu'à explicitement exprimer leurs demandes au système en les adressant à une entité appelée *façade*, sans se soucier de la manière avec laquelle celles-ci seront traitées. Les demandes sont effectuées via l'interface *IADNService*. Une demande contient, selon le type d'opération (création, modification et suppression), toute ou une partie du profil de communication de l'application. Les figures 2.6a et 2.6b décrivent respectivement le profil de communication d'une application et les opérations pouvant être effectuées sur un service.
Ainsi, les applications sont découplées des services qui leur sont offerts avec l'avantage de pouvoir évoluer indépendamment du système et réciproquement. Seule une modification de l'interface de service (les données ou les opérations) aura une répercussion sur les applications. Le composant « Request Handler » joue le rôle de *façade*, et de ce fait, il implémente l'interface ADN exposée



(a) Profil de communication d'une application

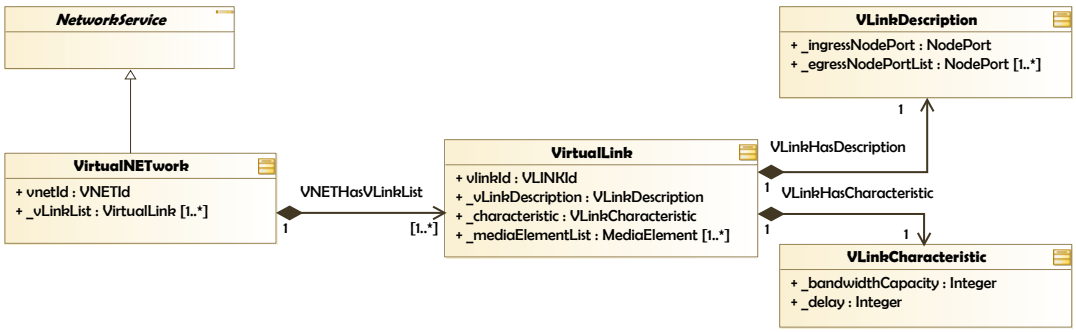


(b) L'interface de service

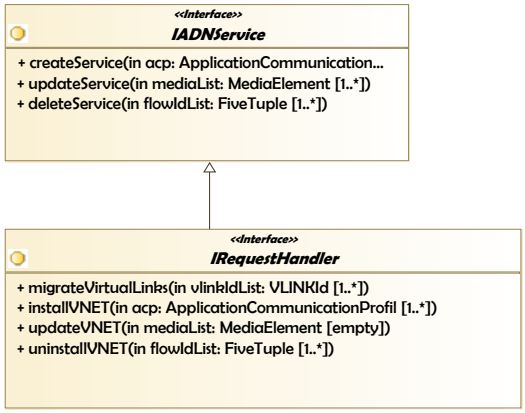
FIGURE 2.6 – L'API Nord du réseau ADN

aux applications, traduit chaque opération en requête *interne* sur le VNET correspondant et met en œuvre les composants nécessaires pour le traitement de la requête.
En implémentant cette interface, il masque aux applications la complexité sous-adjacente et les détails internes au système. Le réseau est alors présenté comme une plate-forme de services, lesquels services peuvent être commandés, décommandés et modifiés par les applications, selon leur besoin, au moyen d'opérations mises à leur disposition. Pour rappel, un service réseau est exprimé sous forme de réseau virtuel applicatif ou VNET dont les caractéristiques de ses liens virtuels (VLinks) sont en parfaite adéquation avec le profil de communication de l'application ayant commandé le service. Le diagramme de la figure 2.7a modélise un VNET. Si les applications sont capables d'apprécier la prestation du service rendu par le réseau, cependant elles ne connaissent pas la nature du dit service. Autrement dit, la notion de VNET leur est méconnue. Chacune des opérations qu'elles effectuent via l'interface de service ADN est alors traduit par le « Request Handler » en une requête de rajout, de mise à jour ou de retrait d'un VNET de l'infrastructure. La figure 2.7b présente, en plus de l'interface de service ADN que ce composant implémente, les requêtes supportées.

L'orchestration des composants impliqués dans le traitement d'une requête, se fait suivant un *workflow* qui dépend du type de requête. A la réception d'une requête de rajout de VNET, le « Request Handler » lance l'exécution du composant « Flow Aggregator » pour vérifier s'il



(a) Modèle d'un VNET



(b) L'interface de service du *Request Handler*

FIGURE 2.7 – Modèle de service ADN et interface de service du *Request Handler*

est possible d'agréger ou de regrouper certains flux du VNET. Ensuite, il lance le composant « VNET Resource Allocator » pour calculer les chemins de données supportant le VNET avec les ressources à réserver le long de ces chemins. Enfin, il lance le « VNET Deployer » pour installer les chemins de données qui supportent le VNET sur l'infrastructure ADN. Ce fonctionnement du « Request Handler » est illustré par la figure 2.8 qui met également en avant une première phase dont l'objectif est de déterminer sur quel commutateur OpenFlow est rattachée chaque extrémité de lien virtuel. Ces informations sont obtenues des contrôleurs OpenFlow via une interface, en l'occurrence l'interface *ISDNController*.

2.3.2 Flow Aggregator

Par la possibilité offerte de pouvoir décrire finement les besoins applicatifs, l'approche ADN a comme avantage de pouvoir fournir les garanties de QdS souhaitées par les applications tout en utilisant efficacement les ressources. En revanche, il est clair que l'approche pose des problèmes d'échelle. Un aspect important de ce problème d'échelle est le nombre des entrées qui sont installées dans la table des flux pour supporter le service.

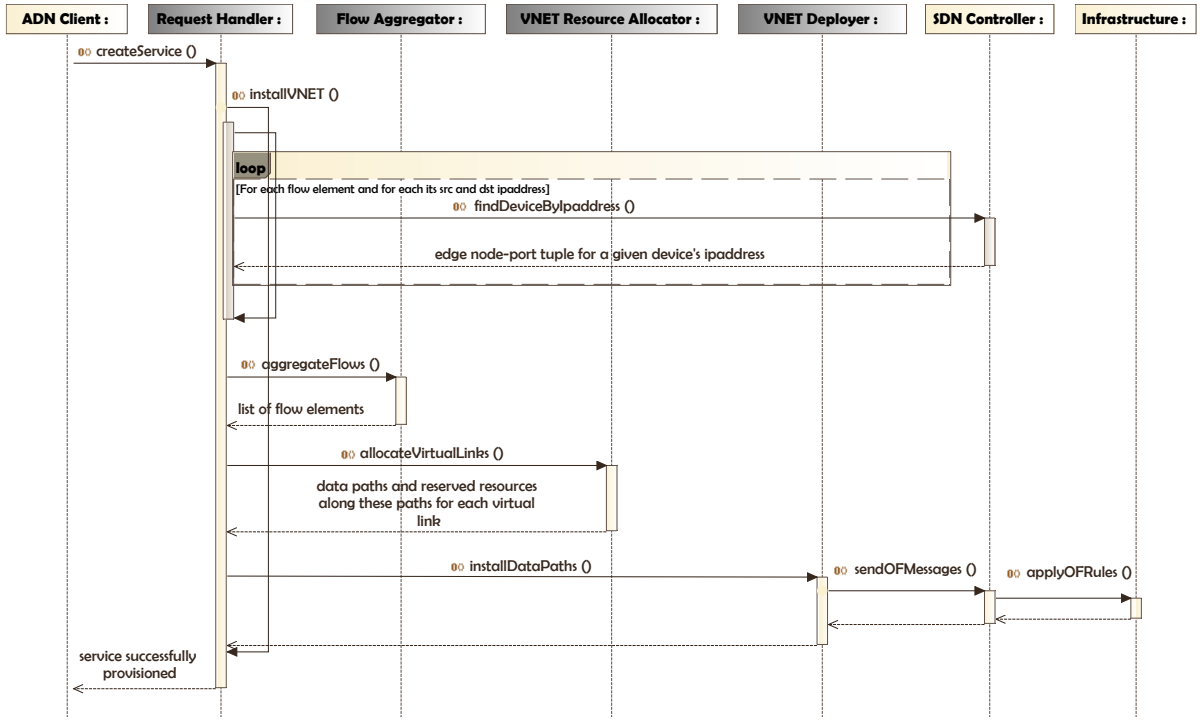
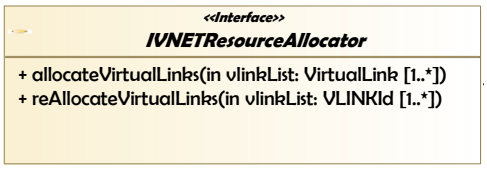


FIGURE 2.8 – Fonctionnement du système lors d'une requête de « création de service »

Même si l'algorithme d'allocation prend en compte cette contrainte et cherche à minimiser le nombre d'entrées consommées au niveau des tables des commutateurs SDN et à répartir leur charge, il serait conjointement intéressant d'agir en amont, c'est-à-dire lors de la construction du VNET à provisionner. C'est exactement le rôle du composant « Flow Aggregator ». Il a pour tâche d'établir le VNET final à provisionner et pour lequel le « VNET Resource Allocator » aura à déterminer les ressources à allouer, en regroupant autant que possible les liens virtuels. Ces regroupements ont un coût en termes de consommation de ressources. C'est l'algorithme de ce composant (sur la base de la politique de gestion du réseau ADN) qui optimisera ce compromis entre ressources réseau consommées et nombre d'entrées installées dans les nœuds SDN.

Algorithme d'agrégation de flux La version actuelle de cet algorithme est assez basique. L'algorithme regroupe les flux ayant les nœuds sources identiques et les nœuds destinataires identiques : la bande passante équivalente du flux agrégé est la somme des bandes de ses flux élémentaires et son délai de transfert maximal est équivalent au délai exigé par son flux élémentaire le plus contraignant. Ici, le nœud peut être soit la machine sur laquelle s'exécutent les processus sources et destinataires ou alors le commutateur de bordure sur lequel cette machine est attachée. L'algorithme effectue le choix selon la situation (charge du réseau, type de requêtes etc...).



(a) L'interface fournie par le *VNET Resource Allocator*



(b) Modèle d'un chemin de données

FIGURE 2.9 – Interface fournie par le *VNET Resource Allocator* et modèle d'un chemin de données

2.3.3 Virtual NETwork Resource Allocator

A la réception d'une demande de rajout d'un VNET, ce composant a pour tâche de calculer l'ensemble optimal des chemins de données (avec les ressources réseau associées) à utiliser pour supporter les liens virtuels qui composent le VNET. Plusieurs critères d'optimisation et donc plusieurs algorithmes sont possibles. Les critères considérés sont la minimisation de la quantité de ressources allouées à chaque réseau virtuel et l'équilibrage de la charge entre les différents nœuds et liens de l'infrastructure physique. Avec ces critères, l'admissibilité des prochaines demandes d'allocation de ressources est favorisée. Pour les mêmes fins, deux mécanismes sont envisageables et peuvent être activés à la demande pour certaines requêtes, lors du calcul des chemins de données. Il s'agit du *path-splitting* et de la *migration*. Le premier mécanisme permet qu'un lien virtuel soit supporté par plusieurs chemins physiques et donc agit pour augmenter l'admissibilité d'une requête d'établissement de VNET. Le second quant à lui, permet, lorsque les ressources disponibles ne sont pas assez suffisantes pour provisionner un nouveau réseau virtuel, la réallocation des réseaux virtuels déjà installés sur l'infrastructure sans les pénaliser. Deux types de ressources réseau sont prises en compte dans le calcul : classiquement, la bande passante des liens mais également les ressources de commutation des nœuds qui sont représentées par la table des *flux*, de la table des *groupes* OpenFlow et la table des *meters*. Enfin, il est à noter que ce composant est mis en œuvre lors des demandes de mise à jour d'un VNET. Les demandes de retrait d'un VNET sont gérées entièrement par le composant « Request Handler ».

Lorsque les caractéristiques d'un VNET changent suite à l'évolution des flux ou de leurs besoins en QdS, le VNET doit être mis à jour. Dans ce cas, ce composant recalcule les chemins de données (avec les nouvelles ressources réseau associées) des liens virtuels concernés dans le but d'ajuster le service réseau en conséquence. La figure 2.9 présente l'interface de services qu'offre ce composant, ainsi que le modèle d'un chemin de données.

2.3.4 Virtual NETwork Deployer

Ce composant indispensable, participe au traitement de n'importe quelle requête (de rajout, de mise à jour et de retrait) concernant un VNET et intervient aussi lors de l'exécution des tâches intrinsèques au système, notamment celles liées à la gestion de ressources et à la maintenance des VNETs. Le rôle de ce composant est d'assurer les fonctions d'installation, de modification et de désinstallation effectives des chemins de données sur l'infrastructure. La figure 2.10 présente l'interface fournie par ce composant.

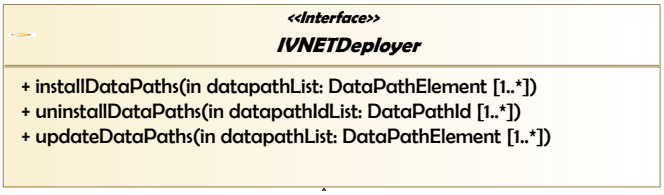


FIGURE 2.10 – Interface fournie par le *VNETDeployer*

Rappelons que nous avons fait le choix de reposer sur une interface nord de bas niveau, correspondant à celle exposée par un contrôleur SDN. Si ce choix nous donne l'avantage de bénéficier de la liberté de développer l'application de contrôle réseau ADN de bout en bout, en respectant le plus fidèlement possible les principes de conception énoncés, il impose en revanche que, l'application de contrôle réseau programme l'infrastructure réseau suivant un modèle de programmation impératif. Autrement dit, il doit programmer l'infrastructure en connaissant toute la suite des actions à réaliser pour atteindre un objectif donné. Ceci en raison du fait que toute l'intelligence du réseau ADN est déléguée à son application de contrôle. Cela revient pour le « Virtual NETwork Deployer », qui a vocation à programmer l'infrastructure, à devoir produire et injecter sur l'ensemble des équipements tous les états nécessaires lors de l'exercice de ses fonctions.

Les algorithmes qu'implante ce composant pour assurer ses trois fonctions ont une mécanique similaire qui s'articule autour de deux grandes étapes successives, à savoir :

1. générer les messages états contenant les commandes à appliquer au niveau des commutateurs : il peut s'agir des commandes pour l'installation de nouvelles règles, pour la modification des règles existantes ou pour leur suppression ;

2. soumettre ces messages au contrôleur SDN via son interface nord, en lui demandant de les livrer aux commutateurs, où les commandes seront exécutées.

Nous nous limitons ici à la présentation de l'algorithme qu'exécute la fonction d'installation.

Algorithme 1 : Algorithme de déploiement

Input :
V : is the set of nodes (OpenFlow Switches). Each node i processes and forwards data to another node j or to rest of the network via a port noted p_{ij} ;
E : is the set of physical links. The link between i and j is noted (i, j) ;
K : is the set of virtual links. Each virtual link $k \in K$ is characterised by : a source node s_k element of V, a bandwidth b_k, and a set of destination nodes T_k part of V except s_k ;
f : is a 4-dimensional dictionary of integer noted $f_k^t(i, j)$, representing bandwidth allocated at link (i, j) to the packets of virtual link k, that are flowing from the source node s_k to the destination node t ;
F : is a 3-dimensional dictionary of integer noted $f_k(i, j)$, representing bandwidth allocated at link (i, j) to the virtual link k ;
V' : is the set of nodes that crossed by at least one virtual link ;
Global variables :
match : *ofp_match* ; group_mod : *ofp_group_mod* ; nextHops a set of nodes ; flowModMessageList : *ofp_flow_mod*[] ; groupModMessageList : *ofp_group_mod*[] ; meterModMessageList : *ofp_meter_mod*[] ; groupID, meterID, bundleID : *integer* ;

```
1  begin
2      foreach i ∈ V' do
3          foreach k ∈ K do
4              if (i = s_k) then
5                  meterID ⟵ get_meter_id(i, k)
6                  insert { id ⟵ meterID, bands[0] ⟵ { rate ⟵ b_k, type⟵ drop } } into
                     meterModMessageList
7              nextHops ⟵ computeNextHops(i, k, V', E, f)
8              if (|nextHops| = 1) then
9                  foreach j ∈ nextHops do
10                     if (i = s_k) then
11                         insert { match ⟵ create_match(), instructions ⟵ { meter ⟵ meterID,
                             write-actions ⟵ { output.port p_ij } } } into flowModMessageList
12                     else
13                         insert { match ⟵ create_match(), instructions ⟵ { write-actions ⟵ {
                             output.port p_ij } } } into flowModMessageList
14             else
15                 groupID ⟵ get_group_id(i, k)
16                 group_mod.id ⟵ groupID ; group_mod.type ⟵ select
17                 foreach j ∈ nextHops do
18                     group_mod.buckets[j] ⟵ { weight ⟵ f_k(i, j), actions ⟵ {ouput.port ⟵ p_ij } }
19                 insert group_mod into groupModList
20                 if (i = s_k) then
21                     insert { match ⟵ create_match(), instructions ⟵ { meter ⟵ meterID, write-actions
                         ⟵ { group.id ⟵ groupID } } } into flowModMessageList
22                 else
23                     insert { match ⟵ create_match(), instructions ⟵ { write-actions ⟵ { group.id ⟵
                         groupID } } } into flowModMessageList
24         bundleID ⟵ get_bundle_id(i)
25         OFPBCT_OPEN_REQUEST { id ⟵ bundleID, flags ⟵ ordered }
26         foreach msg ∈ meterModMessageList do
27             OFPT_BUNDLE_ADD_MESSAGE { id ⟵ bundleID, message ⟵ msg }
28         foreach msg ∈ groupModMessageList do
29             OFPT_BUNDLE_ADD_MESSAGE { id ⟵ bundleID, message ⟵ msg }
30         foreach msg ∈ flowModMessageList do
31             OFPT_BUNDLE_ADD_MESSAGE { id ⟵ bundleID, message ⟵ msg }
32         reset (meterModMessageList) ; reset(groupModMessageList) ; reset (flowModMessageList)
33     foreach i ∈ V' do
34         OFPBCT_CLOSE_REQUEST { id ⟵ get_bundle_id(i) }
35         OFPBCT_COMMIT_REQUEST { id ⟵ get_bundle_id(i) }
```

2.3.4.1 Algorithme de déploiement de réseau virtuel

Cet algorithme repose sur un ensemble de pré-requis sur les éléments d'un commutateur OpenFlow ainsi que sur le protocole OpenFlow. Ils sont, dans un premier temps, rappelés dans la section suivante. La section d'après décrit l'algorithme proposé.

2.3.4.1.1 Quelques rappels et compléments sur OpenFlow

Un commutateur Openflow est constitué d'une ou plusieurs tables de flux (*flow tables*), une table de groupes (*group table*) et une table de meters (*meter table*). Le premier type de table est utilisé pour exécuter des actions d'acheminement basiques et de modification des paquets, le second pour des actions d'acheminement plus complexes (diffusion, partage de charge, etc.) et le troisième pour exécuter des actions de régulation ou de mise en forme de trafic. Le contrôleur peut alimenter, mettre à jour et enlever des entrées de chacune de ces tables d'un switch en lui transmettant les messages du protocole OpenFlow suivants :

- OpenFlow Flow Modification Message (noté *ofp_flow_mod*) qui est utilisé par le contrôleur pour installer, enlever et mettre à jour les entrées de la table de flux d'un commutateur. Chaque entrée est composée d'une règle de correspondance ("match rule") qui identifie les paquets concernés par cette entrée et un ensemble d'instructions (appelé *instruction-set*) qui sont exécutées lors d'une correspondance d'un paquet. Ces instructions permettent de modifier le contenu du paquet et d'alimenter l'ensemble des actions cumulées (appelé *action-set*) qui seront appliquées au paquet lorsqu'il sera commuté vers l'interface de sortie. Plusieurs instructions sont définies dans le protocole OpenFlow dont :
  - *write-actions* qui permet de rajouter des actions dans l'ensemble *action-set* du paquet. Parmi les actions figurent les moyens de modifier certains champs des paquets ou empiler/dépiler des champs, la redirection du paquet vers une interface de sortie ou vers un groupe de la table *group table*.
  - *meter* qui permet d'aiguiller un paquet vers une interface de la *meter table* pour des fins de régulation de trafic.
- Openflow Group Modification Message (noté *ofp_group_mod*) qui est utilisé par le contrôleur pour modifier le contenu de la *group table*. Une entrée de cette table contient un identificateur *group id* qui est utilisé par une entrée de la table de flux pour la désigner ainsi qu'une liste de action *buckets* qui permettent d'entrevoir différentes manières d'acheminer un paquet. Plusieurs types de groupes ont été définis par OpenFlow dont :
  - **Select** dont l'objectif est de permettre le partage de charge entre plusieurs chemins. A chaque bucket est associé un poids et une liste d'actions incluant une interface de sortie. Lorsqu'un paquet est dirigé vers un groupe de ce type, sur la base des poids de chaque bucket et du trafic déjà relayé par le groupe, un bucket est choisi pour prendre en charge l'acheminement du paquet.
  - **All** qui est utilisé pour réaliser du multicast ou du broadcast. Lorsqu'un paquet est dirigé vers un groupe de ce type, une copie de ce paquet est livrée à chaque bucket pour traitement.
- OpenFlow Meter Modification Message (noté *ofp_meter_mod*) qui permet au contrôleur de modifier le contenu de la *meter table*. Une entrée de cette table possède une identi-

ficateur (noté *meter-id*) et définit un régulateur de trafic potentiellement constitué de plusieurs régulateurs basiques appelés *meter bands*. A un *meter band* est associé un débit cible qu'il va devoir appliquer. Dans le cas où les paquets dirigés vers ce meter ne respectent pas ce débit, selon le type du meter band (*drop* ou *dscp-remark*), les paquets non conformes sont éliminés ou simplement remarqués avec un niveau de priorité faible.

Les entrées de la table des flux (*flow tables*) peuvent faire référence à des entrées de la *group table* ou de la *meter table*. Il est par conséquent nécessaire que ces dernières soient installées dans leur table respective avant toute référence depuis la table des flux. Ainsi, un contrôleur SDN doit s'assurer que les messages OpenFlow de modification de la table des meters précèdent ceux relatifs à la modification de la table des groupes qui, à leur tour, précédent les messages de modification des tables de flux.

2.3.4.1.2 Algorithme de déploiement

L'algorithme a pour objet de construire l'ensemble des messages de modification des tables OpenFlow que le contrôleur aura à transmettre aux différents commutateurs de l'infrastructure qui supportent le réseau virtuel demandé. Ainsi, pour chacun de ces commutateurs, trois listes de messages seront construites, à savoir la liste des messages *ofp_meter_mod*, la liste des messages *ofp_group_mod* et la liste des messages *ofp_flow_mod*, puis transmises vers le commutateur dans cet ordre. Le mécanisme de bundle introduit dans la version 1.4 d'OpenFlow est utilisé pour permettre le stockage et la pré-validation de ces messages au niveau de chaque commutateur avant une confirmation globale sur tous les commutateurs concernés par le réseau virtuel à déployer. Ce mécanisme permet de synchroniser les changements à appliquer sur plusieurs commutateurs et en cas de problème sur un commutateur donné, invalide les demandes de modification sur les autres commutateurs.

L'algorithme de déploiement est présenté dans 1 et décrit ci-après. Très grossièrement :

1. Pour tout commutateur i concerné par le réseau virtuel [ligne 2] et pour tout lien virtuel k de ce réseau virtuel [ligne 3] :
   - **[ligne 4 - ligne 6]** : si le commutateur i est le nœud source du lien virtuel k, insérer une demande de création d'un meter dans la liste des messages *meterModMessageList* de i pour s'assurer qu'il ne dépasse pas le débit qui lui est garanti ;
   - **[ligne 14- ligne 23]** : si du "*path-splitting*" intervient au niveau du nœud i, insérer une demande de création de groupe dans la liste des messages *groupModMessageList* avant de demander la création de l'entrée de la table des flux des paquets de k (à insérer dans la liste de messages *flowModMessageList*) avec comme action à rajouter dans l'action-set la redirection des paquets vers le groupe nouvellement créé avec préalablement le passage par le meter suscité dans le cas où i est le nœud source ;
   - **[ligne 8 - ligne 13]** : si du "*path-splitting*" n'intervient pas au niveau du nœud i, demander la création de l'entrée de la table des flux des paquets de k avec comme principale instruction de rediriger le paquet vers son interface de sortie avec un éventuel passage préalable par le meter si i est le nœud source de k ;

2. [**ligne 24 - FIN**] : Les listes relatives à chaque nœud étant constituées, les bundles sont créés au niveau de chaque commutateur pour récupérer dans l'ordre les messages relatifs à la table des meters puis ceux de la table des groupes puis ceux des tables de flux.

2.3.5 Application Classifier

Comme indiqué ci-avant, le réseau ADN supporte aussi bien les applications qui expriment explicitement leurs besoins que celles qui ne le font pas. Le composant *Application Classifier* a pour objectif d'identifier en temps-réel les applications qui ont le droit d'utiliser les services ADN, d'établir et de maintenir leur profil de communication puis de soumettre la requête d'établissement ou de mise à jour sur le VNET correspondant. Grâce au SDN, les applications sont conscientes du réseau ADN. Réciproquement, grâce à ce composant, le réseau ADN devient conscient des application en ce sens qu'il le rend capable d'inférer par lui-même leurs besoins. L'implémentation de ce composant peut être plus ou moins complexe selon les situations :

1. *Le profil de communication peut être complètement établi en analysant le trafic de signalisation.*
   Cette situation correspond au scénario dans lequel les applications communiquent implicitement leurs besoins. En d'autres termes, il s'agit des applications utilisant un protocole de signalisation pour négocier, à travers le réseau, les flux médias qu'elles souhaitent échanger. C'est notamment le cas des applications utilisant le protocole de signalisation SIP et des applications reposant sur l'intergiciel DDS. Dans cette situation, des mécanismes d'inspection profonde de paquets (DPIs) alimentés avec les signatures applicatives pertinentes lesquelles sont administrées par un opérateur via l'interface *IACAdmin*, sont déployés aux extrémités du réseau pour identifier et analyser le trafic de signalisation des applications supposées utiliser le service ADN. Au lieu d'identifier les paquets et de les classer comme c'est typiquement le cas, les DPIs doivent plutôt rediriger une copie de chaque paquet appartenant à ce trafic, vers ce composant via l'interface *IPacketProvider*. L'« Application Classifier » doit alors décoder et analyser ces paquets à la volée, pour écouter et comprendre en direct les négociations entre les composants applicatifs, dans le but d'inférer les flux qu'ils souhaitent échanger ainsi que les exigences de QdS associée. Notons que la visibilité du SDN va de la couche 2 à 4 et celle du DPI s'étend de la couche 4 à 7. En combinant ces deux vues, le réseau ADN profite d'une vue panoramique sur toutes les couches de 2 à 7. Ainsi, comme recommandé par la spécification [Y.2770 2012], il est possible de connaître l'identité des applications (Youtube, Facebook, Skype, etc) qui s'exécutent sur le réseau, leur méta-data extraits (User ID, SIP Caller, Video Codec, etc) ainsi que les méta-data calculés tels que, le délai, la gigue, le débit de l'application. C'est sur la base de ces informations que le profil de communications des applications est maintenu.

   Comme l'inspection profonde des paquets peut être coûteuse en temps, car elle s'effectue au niveau logiciel, lorsque possible, l'usage des DPIs peut être substitué par les règles SDN, lorsque la visibilité du trafic le permet. C'est à dire lorsque la visibilité est de niveau 2 à 4. Notons par ailleurs que les paramètres de QdS généralement utilisés lors des négociations sont du niveau applicatif ; ce composant devra alors assurer leur correspondance

en métriques de QdS niveau réseau, soit alors en bande passante et délai. Dans ce rôle, il masque les détails réseau aux applications.

2. *Le profil de communication peut-être partiellement établi en analysant le trafic média.* Cette approche s'applique aux applications ne communiquant, ni explicitement, ni implicitement leurs besoins. Dans cette situation deux scénarios sont possibles :

   (a) *le protocole de transport du flux media est connu* : Dans ce cas, on pourra refaire usage des DPIs, toujours aux extrémités du réseau, mais cette fois juste pour identifier les nœuds d'entrée et de sorties de chacun des flux applicatifs. Autrement dit, l'usage du DPIs servira à déterminer le pattern des flux de données supposés utiliser le service ADN.

   (b) *le protocole de transport du flux media est inconnu* : Il s'agit généralement des applications qui utilisent de plus en plus les techniques de tunneling applicatifs en s'appuyant le plus souvent sur le protocole HTTP qui sert d'enveloppe à leur protocole propriétaire, pour échanger les données. Ces protocoles propriétaires méconnus du grand public, rendent difficile l'utilisation des DPIs. Comme principale alternative, on pourra utiliser les algorithmes d'apprentissage automatisé (ML pour Machine Learning en anglais) [Li 2014]. Ils seront également déployés aux extrémités du réseau, aussi pour identifier les nœuds d'entrée et de sortis de chacun des flux applicatifs. Il convient également de noter que les approches basées sur les DPIs sont précises mais sont relativement lentes et consomment beaucoup de ressources de calcul par rapport aux approches basées sur l'apprentissage automatique, lesquelles en revanche sont moins précises et requièrent une phase d'apprentissage [Qazi 2013]. Il s'agit d'algorithmes de ML supervisés tels que les réseaux bayésiens [Heckerman 2013] et les arbres de décisions [Abbes 2004]. Ces algorithmes utilisent un trafic réseau déjà connu comme donnée d'entraînement, pour déduire les caractéristiques des flux de l'application. L'entraînement peut se faire *en ligne* ou *hors ligne*. Lorsqu'un nouveau flux entre dans le réseau, l'algorithme compare les caractéristiques du flux à celles déduites lors de la phase d'apprentissage pour déterminer à quelle application il appartient.

   Dans les deux scénarios, il reste à déterminer les besoins en QdS des flux identifiés. En s'appuyant sur les protocoles tels que NetFlow [Claise 2004] et sFlow [Panchen 2001] dont les agents tournent également au niveau des nœuds de bordure, ce composant collectionne par le biais de l'interface *IFlowExport*, les statistiques du trafic de chaque flux et utilise les techniques d'estimation du trafic pour évaluer en temps réels leur besoin en bande passante. Outre le sur-débit induit, il est clair que, dans ce cas de figure, la précision et l'exactitude du profil de communication sont impactées. On peut lorsque possible compenser cette inexactitude en optimisant en continu cette estimation, sur la base du *feedback* des applications lorsque celles-ci informent (par exemple au moyen de logs ou de protocoles de reportage [Fallon 2011]) au cours de leur exécution, sur leurs performances liées à la prestation courante du réseau. Plusieurs applications supportent cette fonctionnalité. C'est le cas des applications SIP, dont le standard spécifie, à travers la rfc [1] 6035 [Sinnreich 2010],

1. de l'anglais request for comments

un ensemble d'information qui renseignent sur la qualité d'un appel SIP. La spécification de l'intergiciel DDS [OMG 2007] fait de même avec l'introduction de la notion de *Built-in Topics*. Aussi l'IETF (Internet Engineerign Task Force) a étendu le protocole RTPC (Real-Time Transport Control Protocol) pour qu'il supporte plus finement la remontée d'information, et a proposé pour cela RTCP XR (RTPC Extended Reports) [Singh 2018]. Tous ces systèmes de notifications permettent d'optimiser dynamiquement et automatiquement le service fourni aux applications en se basant sur la qualité d'expérience de l'utilisateur (QdE), sans avoir recours à une intervention humaine.

On peut conclure que ce composant assure la gestion du profil de communication des applications autorisées à consommer les services ADN, à partir duquel il effectue les demandes de service pour le compte de ces applications. Techniquement, il capture les besoins des applications avant de les traduire en demande de service réseau ADN. Une fois envoyée via l'interface *IRequestHandler*, la demande sera par la suite traitée par le « Request Handler » de la même façon qu'une demande explicitement soumise par une application.

2.3.6 Autonomic Manager

Il est crucial pour un système tel le réseau ADN de pouvoir s'adapter dynamiquement afin de faire face efficacement aux pannes logicielles et matérielles, aux attaques, à l'évolution du contexte et des besoins des applications. En considérant la complexité des adaptations induites, par souci de réactivité, de réduction de coûts et de baisse de la probabilité d'erreur, ces adaptations ne peuvent pas raisonnablement être conduites par des opérateurs humains. L'informatique autonome se propose de déléguer ce pilotage aux systèmes eux-mêmes en les rendant capables d'auto-gestion : d'où l'idée de ce composant. L'objectif du composant « Autonomic Manager » est d'ajouter certaines propriétés autonomiques à l'application de contrôle réseau « ADN service provisioning ». Il gère les composants décrits ci-avant en implantant la boucle autonomique MAPE-K (Monitoring, Analysis, Planning, Execution and Knowledge) proposée par IBM.

Sans être exhaustif, les scénarios identifiés auxquels l'« Autonomic Manager » est programmé à réagir sont décrits ci-après :

- A la suite d'un changement de topologie réseau, il décide si une réallocation de ressources doit être enclenchée (propriété d'auto-réparation)
- En fonction des ressources réseau disponibles et de la requête d'ajout ou de mise à jour d'un VNET, il paramètre l'algorithme d'allocation de ressources (par l'activation ou l'inhibition du *path-splitting*) et/ou de l'agrégation de flux ;
- Similairement, après des demandes de libérations de VNETs, il décide du moment où il est pertinent de re-calculer les ressources allouées aux VNETs présents pour mieux distribuer la charge des éléments réseau (auto-optimisation) ;
- Par ses actions de monitoring du trafic réseau, il détecte que le débit courant d'un lien virtuel a évolué et décide d'activer un processus d'estimation du trafic relatif à l'application (auto-configuration) pour ensuite soumettre une requête de mise à jour du VNET auquel le lien appartient.

La figure 2.11 présente l'interface fournie par ce composant.

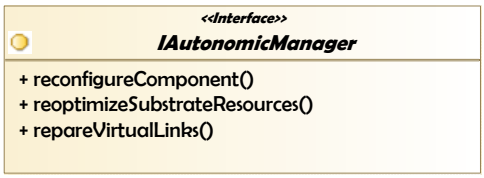


FIGURE 2.11 – Interface fournie par l'*Autonomic Manager*

2.3.6.1 La gestion autonome

2.3.6.1.1 Les propriétés auto-*

L'auto-gestion du système est atteinte au travers de l'implantation de quatre propriétés fondamentales [Kephart 2003] :

1. L'*auto-configuration* permet à un système de s'installer et de se paramétrer, lui-même et chacun de ses composants, selon ses objectifs ou pour répondre à des besoins clients. Il ne s'agit pas simplement de gestion du déploiement initial, mais également de reconfiguration et mise à jour du paramétrage suite à l'évolution de l'environnement (e.g., re-allocation de bande passante) et à l'ajout ou la suppression de composants (effectués par le système ou par des humains). Des composants peuvent ainsi intégrer ou disparaître du système fluidement. Ce dernier comprends les implications de ces ajouts/suppressions et peut s'adapter en cas de besoin.
2. L'*auto-guérison* décrit la capacité d'un système à détecter, diagnostiquer et surmonter les problèmes pouvant survenir dans son domaine. Elle s'applique en particulier à la gestion des bugs et pannes tant logicielles que matérielles. Cette activité peut être pro-active, anticipant et évitant les dysfonctionnements.
3. L'*auto-protection* permet à un système d'assurer son intégrité et sa survie. Pour ce faire, il résout de manière réactive ou pro-active, les problèmes à large échelle provenant en particulier d'erreurs en cascade ou d'attaques externes. Ces problèmes sont plus précisément spécifiés comme ceux n'étant pas corrigés par des mesures d'auto-guérison.
4. L'*auto-optimisation* réfère à l'utilisation efficace de ressources. Il ne s'agit pas nécessairement d'effectivement atteindre un optimal, mais au moins de continuellement chercher à s'en approcher. Cet optimal et cette notion d'adéquation de l'utilisation des ressources est liée à une évaluation du système à travers différentes métriques reflétant son efficacité en terme de performance et de coût.

2.3.6.1.2 La boucle de contrôle MAPE-K

IBM a suggéré un modèle de référence [Kephart 2003] pour la définition de gestionnaires autonomes, des modules fournissant un comportement autonome aux composants du système. Il se base, comme le suggérait Horn [Horn 2001], sur une boucle de contrôle particulière nommée boucle de contrôle autonome ou boucle MAPE-K. La figure 2.12 présente la version réactualisée [Computing 2006] par IBM.

Un gestionnaire autonome est un module logiciel typiquement paramétré initialement par des opérateurs humains qui lui fournissent des objectifs et politiques de haut-niveau lui permettant d'assurer la gestion du système. Il est à l'écoute du système et de ses composants grâce à

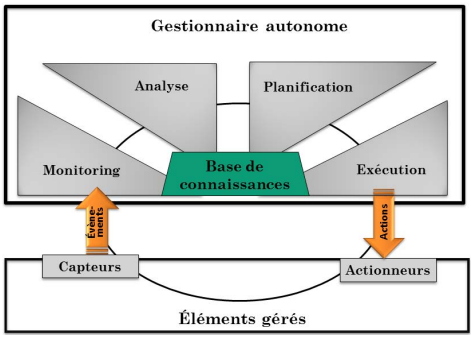


FIGURE 2.12 – Boucle de contrôle autonomique MAPE-K (IBM 2006)

une collection de capteurs ou senseurs lui fournissant tout type d'informations pertinentes à la gestion du système. Grâce à sa connaissance du système, il pourra analyser ces informations et planifier des adaptations en concordance avec les politiques haut-niveau qu'il doit mettre en œuvre. Finalement, il va veiller à l'exécution de ces adaptations, agissant sur le système à l'aide d'un ensemble d'actionneurs.

Par la suite, nous introduisons chacune des étapes de la boucle MAPE-K. Afin de les illustrer, nous considérons un exemple qui utilise un scénario mettant en avant la propriété d'auto-optimisation. Supposons que la politique administrée par l'opérateur humain concernant l'utilisation des ressources est d'équilibrer autant que possible la charge des éléments (nœuds d'acheminement et liens de transmission) réseau. Notons que le niveau d'équilibre est donné par une valeur réelle appelée *indice de disparité* qui renseigne sur la répartition de la charge des éléments réseau. Cet indice vaut 1 lorsque la charge des éléments est identique et lorsque certains éléments sont plus chargés que d'autres il est supérieur à 1.

- ***Observation***
  La tâche d'observation est dédiée à la surveillance du système, de ses propriétés et de son environnement à travers la collecte de toute sorte d'information. Les informations pertinentes sont ensuite formatées et transmises à la tâche suivante sous la forme de symptômes.
  Bien que l'observation s'appuie lourdement sur des sondes, elle ne se contente pas de simplement transmettre les données qu'elles envoient. Afin de ne propager que les informations pertinentes, cette tâche peut inclure des mécanismes d'agrégation et de filtrage, voire même d'analyse légère. Par exemple, l'absence de remontée d'information peut présager de la défaillance d'un senseur et doit donc être traitée comme une information en tant que telle.
  Dans notre exemple, le module d'observation va détecter un changement concernant les ressources suite à une demande de libération de ressources allouées à un VNET. Cette demande peut être provoquée entre autre par l'arrêt d'une application. Si l'indice de disparité est supérieur à un seuil donné, ce module va produire et transmettre le symptôme *présence de disparité* au module suivant en charge de l'analyser.

- ***Analyse***

La tâche d'analyse traite les informations contenues dans les symptômes reçus en fonction de politiques et stratégies de gestion de haut niveau. Elle repère des écarts entre fonctionnement actuel et fonctionnement souhaité, et génère des requêtes de changement transmises au module suivant (Planification).
Le module d'analyse doit donc sélectionner parmi les liens virtuels présents sur l'infrastructure, ceux dont les ressources doivent être recalculées pour atteindre ou se rapprocher de l'état optimal. L'analyse génère donc une requête de changement exigeant la migration de ces liens virtuels.

- ***Planification***
  La tâche de planification élabore des plans d'actions décrivant la manière dont les changements requis vont être mis en œuvre. Ces plans décrivent chacune des actions élémentaires à effectuer. Ils comportent également une composante temporelle et établissent ainsi un ordre ou des priorités au sein de ces actions.
  Dans l'exemple, la planification, défait les allocations actuelles des liens virtuels précédemment sélectionnés, pour en recalculer de nouvelles en adéquation avec l'objectif visé. Ce calcul est assuré par l'un des algorithmes d'allocation de ressources exécuté par le composant « VNET Resource Allocator ». Cet algorithme, précédemment configuré à cet effet, déterminera les chemins de données ainsi que les ressources à allouer le long de ces chemins pour le compte de chacun des liens virtuels candidats à la migration. La planification décrit comment les nouvelles allocations doivent être effectuées sur chaque nœud et lien physique. La décision dans le processus d'adaptation apparaît dans les phases d'analyse et de planification.

- ***Exécution***
  Le module d'exécution est l'interface finale entre la boucle de contrôle et le système réel. Elle se charge de mettre en œuvre les plans prévus en exécutant les actions appropriées sur les éléments gérés par l'intermédiaire des actionneurs à sa disposition.
  Finalement, la tâche d'exécution consiste à désinstaller les chemins supportant les liens virtuels réalloués, pour les remplacer par de nouveaux chemins plus adaptés. L'exécution met en œuvre le « VNET Deployer » pour générer les commandes OpenFlow contenant les actions à appliquer au niveau de chaque nœud d'acheminement, avant de les leur livrer via le contrôleur SDN.

- ***Base de connaissance - « Knowledge Base »***
  La base de connaissance est l'épine dorsale de la boucle MAPE-K. Elle contient toute la connaissance pertinente à la gestion du système : sa représentation, les politiques haut niveau à suivre, les métriques d'évaluation, les règles d'inférences propre à chaque tâche, l'historique de fonctionnement, les sondes et actionneurs disponibles...

Chacune des tâches peut y piocher des informations ou contribuer à cette base, dans le cadre de la constitution d'un historique. Dans l'exemple illustratif que nous avons proposé,

- outre les informations sur les éléments réseau constituant l'infrastructure, leurs capacités et ressources disponibles nécessaires pour évaluer leur charge, l'observation y consulte la

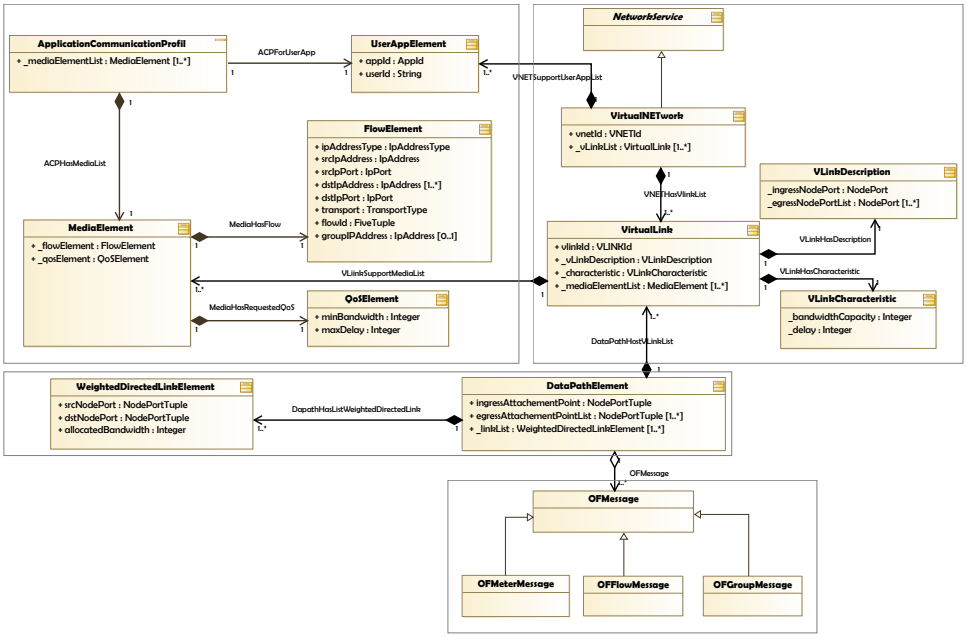


FIGURE 2.13 – Relation entre les entités manipulées par le système

valeur du seuil de disparité.
- l'analyse s'y informe sur les liens virtuels actifs pour sélectionner ceux à réallouer sur la base de la politique qui y est spécifiée.
- la planification récupère dans la base de connaissance l'état courant du réseau et y enregistre les nouvelles allocations.
- l'exécution y trouve les informations nécessaires à la prise de contact avec un nœud d'acheminement, comme son adresse IP et ses numéros de port de contrôle et de gestion.

Cette liste souligne l'aspect central de la base de connaissance dans la boucle MAPE-K.

2.4 Synthèse

La figure 2.13 résume les relations entre les entités manipulées par le système. Elle met en avant les états maintenus par les composants du système, et donc souligne les problèmes d'échelle qui peuvent potentiellement se poser. En résumé, une application est définie par son profil de communication sur la base duquel le système déploie le service réseau qui répond et colle aux exigences de ses flux de données. Le service réseau provisionné correspond à un VNET lui même constitué d'un ensemble de VLinks. Un VLink supporte un groupe de flux déterminé par l'agrégateur de flux qui calcule en même temps la description et les caractéristiques du VLink. Chaque VLink est hébergé par un chemin de données où une quantité de bande passante et d'entrées de tables OpenFlow lui sont réservées respectivement sur les liens et nœuds physiques appartenant à ce chemin de données. Finalement pour établir chaque VLink sur l'infrastructure, une liste de messages OpenFlow de types OFFlowMessage, OFGroupMessage et OFMeterMessage est adressée aux commutateurs de l'infrastructure appartenant au chemin de données.

2.5 Conclusion

L'objectif principal de ce chapitre a été de décrire l'architecture générale d'un réseau ADN et de présenter une vision d'ensemble sur son fonctionnement global.

Nous avons présenté les principes clés et les choix de conception qui ont guidés l'élaboration de cette architecture. En choisissant de bâtir le réseau ADN sur une infrastructure de type SDN, nous avons été amenés à discuter sur le positionnement de l'application de contrôle réseau ADN. A l'issue de cette discussion, nous avons fait le choix de reposer sur une interface nord de bas niveau, avec comme conséquence de développer cette application de contrôle réseau en tirant exclusivement parti des services de base offerts par un contrôleur SDN standard. Les composants logiciels constituant cette application ont été identifiés et une analyse des exigences fonctionnelles de chacun d'eux a été présentée.

Nous avons également présenté les algorithmes qu'implantent certains de ces composants, notamment, l'algorithme qu'exécute la fonction d'installation des réseaux virtuels qui incombe au composant « VNET Deployer », pour établir les chemins de données sur l'infrastructure, ainsi qu'une version basique de l'algorithme qu'exécute la fonction d'agrégation de flux du composant « Flow Aggregator ».

Dans les prochains chapitres, nous nous focaliserons sur les composants « VNET Resource Allocator » et « Autonomic Manager » avec comme objectif de décrire les algorithmes et les mécanismes qu'ils implantent.

Ces composants ont été développés sur une plateforme réelle et intégrés pour constituer un prototype preuve de concept. L'annexe B décrit le démonstrateur ADN qui a été mis en place pour illustrer l'application de l'approche ADN à un cas d'étude qui considère une application basée sur le middleware DDS. Il s'agit d'une application de simulation distribuée impliquant plusieurs simulateurs de véhicule dirigés par des apprenants qui participent à un exercice ou une formation à la conduite en groupe. A travers un scénario applicatif simple, il illustre la détection, par le réseau ADN, de nouveaux besoins applicatifs puis le déploiement ou la mise à jour à la volée du service réseau correspondant. Il illustre également que le réseau ADN est capable de s'accommoder avec un niveau de granularité des besoins applicatifs très fin.

Chapitre 3

Algorithme d'allocation de ressources

Sommaire

3.1 Problèmes d'allocation de ressources dans un contexte de virtualisation réseau 54
3.2 État de l'art de la littérature existante 58
3.3 Positionnement de notre problème 61
3.4 Modélisation 63
3.5 Modèle linéaire 67
3.6 Évaluation expérimentale 70
3.7 Conclusion 94

Pour pouvoir supporter au même moment plusieurs applications, le réseau ADN doit être capable de partager les ressources de l'infrastructure entre les réseaux virtuels dédiés aux applications. La capacité limitée des ressources exige de les allouer de manière efficace afin de servir un nombre important d'applications tout en leur garantissant la qualité de service souhaitée. La considération de cet aspect constitue une spécificité de l'approche que nous proposons et est motivée par l'idée de fournir des services réseau qui collent (à l'opposition d'être sur-dimensionnés) et répondent aux besoins des applications. Ceci soulève le problème d'allocation de ressources lequel est complexe et dont la résolution se heurte à une difficulté plus ou moins élevée selon la prise en compte ou non de certains aspects du problème.

Ce chapitre est consacré au composant VNET Resource Allocation et se focalise sur l'algorithme qu'il implante pour calculer les chemins physiques ainsi que les ressources à allouer le long de ces chemins, pour le provisionnement des liens virtuels, en réponse à une requête de **création de service réseau**. C'est sur la base des informations sur l'infrastructure en l'occurrence celles liées à la topologie du réseau, les ressources actuellement disponibles et la demande courante, que l'algorithme détermine une solution satisfaisante qui optimise un objectif donné sans pénaliser les allocations existantes.

Après la définition du problème d'allocation de ressources dans le contexte de la virtualisation des réseaux en général, nous présentons une revue des solutions existantes dans la littérature scientifique. La section d'après positionne le problème dans le cadre de cette thèse, avant de décrire dans la section suivante la solution que nous proposons. La prochaine section présente

les expérimentations qui ont été réalisées ainsi que l'analyse des résultats d'évaluation obtenus. Enfin, la dernière section conclut ce chapitre.

3.1 Problèmes d'allocation de ressources dans un contexte de virtualisation réseau

3.1.1 L'allocation de ressources pour des réseaux virtuels

L'allocation de ressources pour des réseaux virtuels est un problème NP-Difficile [Lischka 2009] qui consiste à allouer un réseau virtuel R_V sur un réseau physique R_P. Cette allocation consiste à trouver un placement M des routeurs et des liens virtuels, tel que :

- Chaque routeur (nœud) virtuel n^v soit alloué sur un routeur (nœud) physique $n^p = M(n^v)$.
- Chaque lien virtuel l^v reliant deux nœuds virtuels n_1^v et n_2^v soit alloué sur un chemin $M(l^v) = (l_1^p, l_2^p, \ldots, l_k^p)$ reliant $M(n_1^v)$ et $M(n_2^v)$.

Comme expliqué précédemment, le placement M doit en général respecter un ensemble de contraintes en termes de QdS, performances et sécurité. Les différentes contraintes pouvant être concernées par l'allocation sont définies dans la section 3.1.4.

Ainsi, l'allocation de ressources pour virtualiser des réseaux met en jeu deux problèmes d'allocation interdépendants : allocation de nœuds virtuels à des nœuds physiques et allocation de liens virtuels à des chemins physiques.

Un nœud virtuel ne peut être distribué sur plusieurs nœuds physiques, et pour une requête de virtualisation donnée, un nœud physique ne peut héberger qu'un seul nœud virtuel parmi ceux appartenant à un même réseau virtuel.

Un lien virtuel peut avoir un ensemble de nœuds destination $n_{d_1}^p, n_{d_2}^p, \ldots, n_{d_k}^p$ (liens de type multicast), ou un seul (lien de type unicast). De plus, il peut être alloué sur différents chemins physiques (utilisation du *path-splitting*).

Dans la suite de ce travail, nous nous sommes focalisé sur l'allocation de liens virtuels.

3.1.2 Cas particulier de l'allocation de ressources pour des liens virtuels

L'allocation de liens virtuels trouve sont intérêt auprès de fournisseurs de services proposant la création de « tunnels » entre différentes destinations, avec des garanties en termes de bande passante, QdS, sécurité et performances.

L'allocation de ressources pour des liens virtuels est un sous-problème de l'allocation de ressources pour des réseaux virtuels dans lequel les nœuds virtuels sont déjà placés sur le réseau physique. On cherche donc à trouver un ou plusieurs chemins entre plusieurs routeurs physiques (une source n_s^p et des destinations $n_{d_1}^p, n_{d_2}^p, \ldots, n_{d_k}^p$), sous différentes contraintes (cf. section 3.1.4).

3.1.3 Les requêtes

Dans un problème d'allocation de ressources pour des réseaux virtuels, les requêtes sont des demandes d'allocation d'un ou plusieurs réseaux ou liens virtuels. Elles peuvent être très variées et être plus ou moins contraintes (voir 3.1.4).

L'allocation de ressources peut intervenir :

1. A chaque arrivée de requête : On cherche alors à déterminer puis affecter les ressources nécessaires pour répondre à celle-ci sans remettre en cause les ressources déjà allouées sur le réseau physique pour les requêtes précédentes. Ce cas de figure (que nous considérerons ici) trouve son application dans le cadre d'un fournisseur de réseaux virtuels à la demande avec un déploiement de quelques minutes.
2. En connaissant préalablement toutes les requêtes de réseaux virtuels à satisfaire : On peut alors tirer profit de cette connaissance globale pour obtenir une meilleure allocation. Ce cas de figure intervient dans des phases de dimensionnement et d'optimisation du réseau. Ce modèle de fourniture de réseaux virtuels est beaucoup moins flexible que le précédent.

3.1.4 Les différents aspects des problèmes d'allocation de ressources pour réseaux virtuels

Les problèmes d'allocation de ressources pour les réseaux virtuels peuvent prendre plusieurs formes ou s'exprimer de différentes manières selon les contraintes et objectifs visés.

3.1.4.1 La topologie du réseau

Dans les problèmes de virtualisation, l'une des données les plus importantes est la topologie des réseaux virtuels et physiques : les routeurs et les liens. En général, on représente une topologie réseau sous forme d'un graphe orienté $G = (V, E)$ où V représente l'ensemble des routeurs (nœuds) et $E \subseteq V \times V$ l'ensemble des liens.

3.1.4.2 La bande passante exigée

La bande passante est certainement l'aspect le plus important à prendre en compte lors des problèmes d'allocation de ressources pour réseaux virtuels et liens virtuels. Il est nécessaire de trouver un chemin physique reliant deux nœuds avec une capacité suffisante par rapport aux demandes, c'est à dire :

- Respectant les contraintes de capacité au niveau des liens physiques formant le chemin.
- Respectant les contraintes de conservation du flux au niveau des routeurs composant le chemin physique.

En plus de ces deux contraintes fondamentales sur la bande passante, d'autres caractéristiques peuvent être prises en compte, notamment :

- Le **Path Splitting** - Distribuer le trafic correspondant à un lien virtuel (une source et une destination) sur plusieurs chemins physiques. Cette caractéristique requiert des protocoles spécifiques au niveau du réseau (le besoin d'étiqueter certains paquets, entre autres).

– Le **Multicast** - Envoyer un message d'une source vers m destinations, sans avoir besoin de créer m chemin disjoints.

Lors de la définition des requêtes de virtualisation, il est possible de considérer soit le débit maximal, soit le débit demandé par la requête. Dans le premier cas, le réseau virtuel sera alors surdimensionné, dans le deuxième cas il y a des risques de congestion au niveau des routeurs.

3.1.4.3 Les routeurs

– **Coûts liés à la virtualisation des routeurs** : Dans un contexte d'allocation de réseaux virtuels, un des aspects importants à prendre en compte est la charge des routeurs physiques sur lesquels sont alloués des routeurs virtuels : un routeur physique ne peut supporter qu'un nombre fini de routeurs virtuels dépendant de sa capacité et des besoins des composants virtuels. La charge d'un routeur liée à la virtualisation peut apparaître sous différentes formes : Charge processeur, RAM, etc. En réalité, il est aujourd'hui difficile de quantifier les ressources utilisées par la virtualisation d'un routeur [Rathore 2010].
– **Coûts liés à la commutation des paquets** : À la fois dans un contexte d'allocation de réseaux et de liens virtuels, la charge des routeurs liée à la commutation des paquets est un élément non négligeable à prendre en compte pour la résolution du problème de virtualisation. Dans un contexte SDN / OpenFlow, la charge de commutation d'un routeur est étroitement liée au nombre de règles traitées par le routeur (i.e. la taille de la table de flux). Les scénarios de routage étant très variés (multicast, path-splitting, etc.), il est actuellement difficile de quantifier de façon générique le nombre de règles nécessaire à la réalisation du routage d'un lien virtuel.

3.1.4.4 La localisation

Dans certains problèmes, il peut être nécessaire d'ajouter des contraintes de localisation sur les nœuds virtuels. Elles peuvent être de différents types :

– Des contraintes sur la distance entre les positions physiques des éléments (exemple : deux routeurs physiques sur lesquels sont alloués deux routeurs virtuels reliés par un lien virtuel ne peuvent être distants de plus de 200 mètres) - Cette contrainte peut être utile lors de l'allocation sur des réseaux composés de plusieurs ensembles de nœuds (bâtiments / pièces), afin de forcer l'emplacement de plusieurs routeurs virtuels dans des endroits peu distants (pour des questions logistiques, par exemple).
– Des contraintes de placement pour les nœuds virtuels : un nœud virtuel ne peut être placé que dans une zone prédéfinie (ou un ensemble spécifique de nœuds physiques).
– Des contraintes d'allocation : certains nœuds virtuels sont déjà placés sur le réseau physique.

3.1.4.5 Délais de transfert des paquets

Le délai définit le temps nécessaire pour qu'un paquet circule d'un routeur physique n_1 vers un routeur physique n_2. La contrainte en général exprimée est celle du « délai admissible », i.e le délai maximal qu'un paquet ne doit pas dépasser. C'est une contrainte importante en terme

de QdS, et donc un aspect très important dans les problèmes de virtualisation. Le délai peut être pris en compte à différents endroits :

- Sur les liens, via le temps de transmission d'un message (correspondant au temps d'émission du message par un routeur plus le temps de propagation du message sur le lien).
- Sur les routeurs, en fonction de la congestion de ceux-ci (cf. flux moyen / max) et du temps de traitement des paquets (dépendant de la charge des routeurs).

3.1.4.6 La migration des allocations

La migration est un aspect mis de plus en plus en avant dans les réseaux reconfigurables autorisant la réallocation de requêtes de virtualisation déjà placées afin de permettre le traitement d'une nouvelle requête. En général, la migration consiste à identifier les éléments critiques du réseau avant d'essayer de les soulager sans remettre en cause les besoins et contraintes des requêtes déjà traitées.

La migration possède en général un coût non négligeable, à la fois au niveau du traitement des requêtes (algorithme plus lourd car en plus de calculer les allocations des liens virtuels constituant la requête, l'algorithme recalcule en même temps celles des réseaux virtuels à migrer) et au niveau des réaffectations (temps de migration des ressources d'un élément vers un autre) alors que le réseau virtuel est en phase opérationnelle [Melo 2013].

3.1.4.7 Objectifs

3.1.4.7.1 Équilibrage des charges

Lors de la résolution du problème d'allocation de réseaux ou de liens virtuels, un des objectifs fréquemment considéré consiste à rechercher un équilibrage de l'utilisation du réseau physique. Cette équilibrage consiste à préférer distribuer la bande passante sur plus de liens physiques, plutôt que surcharger un nombre restreint de liens. On peut également vouloir équilibrer la charge des routeurs intermédiaires, afin d'éviter toute congestion.

3.1.4.7.2 Coûts et revenus financiers

Suivant le contexte, l'aspect financier lié à l'allocation de réseaux et de liens virtuels peut être intégré au problème. Il peut apparaître sous 2 formats : Le coût et le revenu.

Le coût est principalement lié à l'utilisation des composants physiques du réseau (lien, routeur, etc.).

Le revenu est dépendant de la requête. Dans un concept de fournisseur, une requête avec un revenu élevée peut être vue comme prioritaire par rapport aux autres. Dans ce même concept, la satisfaction d'un maximum de requête (le *taux d'acceptation*) est un enjeu important.

3.2 État de l'art de la littérature existante

La plupart des travaux existants de la littérature actuelle peuvent être séparés en deux familles : ceux traitant de l'**Allocation de réseaux virtuels** (*Virtual Network Mapping / Embedding*) dans lesquels on considère de manière simultanée les deux problèmes d'allocation de nœuds et de liens, et ceux traitant de l'**Allocation de liens virtuels** dans lesquels on ne s'intéresse qu'à l'allocation des liens (les extrémités de ces liens étant déjà placés sur le réseau physique).

3.2.1 L'allocation de ressources pour des réseaux virtuels

L'allocation de réseaux virtuels a récemment suscité un engouement dans la communauté scientifique qui a mené à l'apparition de différentes méthodes de résolution.

Dans [Nogueira 2011], Nogueira et al. proposent un **algorithme polynomial** pour l'allocation de réseaux virtuels basé sur une **heuristique** et utilisant la notion de stress. Le stress est une valeur propre à chaque élément physique dépendant des ressources disponibles lors de la requête (Bande passante utilisée et disponible pour les liens, **CPU / RAM** pour les routeurs).

Dans [Lischka 2009] Lischka et Karl utilisent un algorithme de **recherche de sous-graphe isomorphisme** pour trouver une allocation possible d'un réseau virtuel sur un réseau physique. Leur algorithme prend en compte la charge **des routeurs et des liens** ainsi qu'un **délai** calculé par rapport au nombre de sauts sur un chemin physique (nombre de liens physiques utilisés pour créer un lien virtuel).

Hsu et al. proposent dans [Hsu 2012] différents algorithmes pour l'allocation de routeurs et liens virtuels avec possibilité de **path-splitting** et **migration**. L'algorithme de path-splitting proposé est récursif et n'utilise le path-splitting que si nécessaire : pour un besoin B en bande passante, l'algorithme va d'abord chercher un chemin de capacité au moins égale à B, s'il ne trouve pas, il tente de trouver deux chemins de capacité au moins égale à $B/2$ (avec des arcs en commun de capacité au moins égale à B), et ainsi de suite. Si l'algorithme ne parvient pas à trouver un chemin après plusieurs itérations, il va alors chercher à migrer des chemins existants. Les résultats de Hsu et al. montrent l'intérêt du path-splitting et de la migration dans les problèmes d'allocation de réseaux virtuels. Cette approche utilisant le path-splitting et la migration est également abordée par Yu et al. dans [Yu 2008] qui propose une méthode de type « Try and Error » : Allocation des nœuds, tentative d'allocation des liens sans path-splitting, tentative avec path-splitting si les premières ont échoué (basée sur la résolution d'un problème de *Multi Commodity Flow*), migration des nœuds si nécessaires et enfin migrations des liens (changement de ratio dans les splits ou ré-allocation totale) si aucun des essais précédents n'a abouti. L'approche de Yu et al. diffère en revanche de celle de Hsu et al. dans la gestion de l'arrivée des requêtes : Alors que Hsu et al. traitent les requêtes une par une, Yu et al. attendent d'avoir un certains nombre de requêtes avant de les allouer dans le but de commencer par celles offrant le meilleur revenu.

Les deux articles [Chowdhury 2009] (réédité en 2012, voir [Chowdhury 2012]) et [Melo 2013] utilisent une formulation de **Programmation Linéaire en Nombres Entiers** pour tenter de résoudre le problème. Dans [Melo 2013], Melo et al. proposent une résolution du problème sans relaxation, avec prise en compte de plusieurs contraintes (**Bande passante**, **délai**, **posi-**

tion des routeurs, **charge des routeurs** (**CPU / RAM**)), et des résultats avec différentes fonctions objectives. Chowdhury et al. [Chowdhury 2009] proposent une résolution utilisant une relaxation du problème : la relaxation permet de trouver rapidement un placement des routeurs. Deux versions sont proposées : la première alloue chaque routeur virtuel n_v sur le routeur physique n_p possédant le meilleur score retourné par la résolution du problème relaxé, la deuxième ajoute de l'aléatoire dans le choix des routeurs. Les chemins sont ensuite alloués via un algorithme de *Multi Commodity Flow*.

3.2.2 L'allocation de ressources pour des liens virtuels

L'article [Frei 2005] propose une méthode de **résolution par contraintes** (CSP) ne prenant en compte que les besoins en **bande passante** des requêtes. Les routeurs sont regroupés pour former des « Blocking Islands », permettant ainsi l'utilisation d'heuristiques efficaces pour le choix des variables et des valeurs à affecter durant la résolution du CSP. Frei et Faltings montrent que leur méthode permet d'éviter l'allocation de liens critiques lorsque ce n'est pas nécessaire, contrairement à des algorithmes de plus court chemin classiques.

Le problème de l'allocation de ressources pour liens virtuels peut facilement s'assimiler à un problème de *Multi Commodity Flow*. Masri et al. [Masri 2011] proposent un algorithme basé sur une méta-heuristique **Colonie de Fourmis** pour résoudre le problème du routage de plusieurs flots multisource et multicast avec prise en compte de la **bande passante** et minimisation du **délai** et du **coût**. Wei et al. proposent dans [Wei 2010] un algorithme basé sur la prédiction du trafic dans le réseau et utilisant les algorithmes de Djikstra et de résolution d'un *Multi Commodity Flow* : L'algorithme tente d'abord de résoudre le problème en utilisant l'algorithme de Djikstra, puis un algorithme de résolution de *Multi Commodity Flow* si le premier n'a pas abouti.

Xi et Yeh proposent dans [Xi 2010] un algorithme distribué pour le routage de flots multicast. L'algorithme trouve d'abord un chemin de la source vers chaque destination t, utilisant sur chaque lien $(i; j)$ une quantité de bande passante $b_t(i; j)$. La quantité de bande passante consommée $b(i; j)$ par le lien multicast peut alors être calculée via $b(i; j) = max b_t(i; j)$. Kucharzak et Walkowiak proposent dans [Kucharzak 2012] différentes heuristiques pour l'allocation d'arbres **multicast** prenant en compte uniquement des contraintes au niveau de la **bande passante**.

3.2.3 Synthèse

Le tableau 3.1 fournit un récapitulatif des différents aspects pris en compte dans chaque article cité dans la section précédente.

Les caractéristiques et contraintes considérées par les travaux de la littérature actuelle sont variés. Alors que tous prennent en compte les besoins en bande passante, seulement quelques un prennent en compte le délai, ou la charge des routeurs. La plupart des algorithmes proposés ne font pas usage du multicast, du path-splitting ou de la migration pour obtenir des meilleurs résultats. Finalement, aucun ne prend en compte la charge des routeurs liée à la commutation de paquets.

	Allocation	Contraintes	Type de liens	Techniques
[Melo 2013]	Réseaux	Bande Passante Délai Localisation des routeurs Charge routeurs (CPU / RAM)	Unicast	
[Hsu 2012]	Réseaux	Bande Passante Localisation des routeurs	Unicast	Path-Splitting Migration
[Nogueira 2011]	Réseaux	Bande Passante Charge routeurs (CPU / RAM)	Unicast	
[Wei 2010]	Réseaux	Bande Passante	Unicast	
[Chowdhury 2009]	Réseaux	Bande Passante Localisation des routeurs Charge routeurs (CPU / RAM)	Unicast	
[Lischka 2009]	Réseaux	Bande Passante Délai Charge routeurs (CPU / RAM)	Unicast	
[Hsu 2012]	Liens	Bande Passante	Unicast	Path-Splitting Migration
[Kucharzak 2012]	Liens	Bande Passante	Multi-sources	
[Masri 2011]	Liens	Bande Passante Délai	Multi-sources Multicast	
[Xi 2010]	Liens	Bande Passante	Multicast	
[Frei 2005]	Liens	Bande Passante	Multicast	

TABLE 3.1 – Récapitulatif des différents types d'allocation et contraintes étudiés par chaque article

3.3 Positionnement de notre problème

3.3.1 Définition

3.3.1.1 Le problème

Dans la suite, nous nous intéressons aux problèmes d'**allocation de liens virtuels**, avec prise en compte des contraintes de **bande passante** (avec possibilité de path-splitting), de **délai** et de **charge des routeurs liée à la commutation de paquets**.

Bien que la plupart de ces aspects aient déjà été étudiés de manières isolés dans de nombreux travaux, il n'existe pas à notre connaissance de méthode pour l'allocation de réseaux ou liens virtuels les incluant tous. De plus, d'après notre étude, aucun article ne prend en compte à ce jour la charge des routeurs liées à la commutation dans les problèmes d'allocation de réseaux virtuels.

3.3.1.2 Les requêtes

On considère des requêtes qui arrivent les unes après les autres, sans remise en cause de l'allocation des requêtes précédentes. Chaque requête sera caractérisée par un temps d'arrivée et une durée de vie.

Les requêtes d'allocation traitées seront composées d'une ou de plusieurs *sessions*. Chaque *session* représente un lien virtuel possédant une source unique (un routeur physique) et un ensemble de destinations (routeurs physiques), permettant ainsi des liens point à point et point à multipoint. Une session est caractérisée par une demande identique en termes de bande passante et de délai à respecter.

De plus, afin de limiter une trop grande fragmentation liée à l'utilisation du path-splitting, nous ajoutons à chaque session une valeur minimale de bande passante. En effet, il peut être préférable, pour une session demandant une bande passante de 1 Gbps, de ne pas allouer des quantités de bande passante inférieures à 200 Mbps sur un lien physique.

3.3.1.3 Les allocations possibles

En fonction de ses caractéristiques, un lien virtuel pourra être alloué sur le réseau physique en utilisant :

- Un seul chemin : lien virtuel point à point (unicast), sans « path-splitting ».
- Un arbre : lien virtuel point à multipoint (multicast), sans « path-splitting ».
- Un ensemble de chemins : lien virtuel point à point (unicast) ou point à multipoint (multicast), avec « path-splitting ».

3.3.2 Hypothèses

Dans la suite, on se placera dans l'hypothèse d'un réseau entièrement virtualisé où l'utilisation d'une ressource physique est bornée par sa capacité, i.e. elle ne peut pas être allouée à une requête si celle-ci a un besoin supérieur à la capacité restante.

3.3.2.1 Perte de paquets et délai lié à la congestion

Sous cette hypothèse, on peut considérer qu'il n'y a pas de congestion au niveau des routeurs (l'utilisation des liens entrants ne peut être supérieure à celle des liens sortants), et donc qu'il n'y a pas de perte de paquets ou de délai liés à la congestion des routeurs.

3.3.2.2 Charge des routeurs

Comme indiqué dans nos principes de conception, nous supposons une infrastructure filaire de type SDN/OpenFlow. La spécification du protocole OpenFlow définit un modèle d'acheminement de paquets au niveau d'un commutateur conforme au standard, en utilisant les tables comme abstraction pour masquer les implémentations sous-adjacentes. On distingue entre autres un ou plusieurs tables de *flux* (ou flow tables) et une table de *groupe* (ou group table). Ces tables sont utilisées pour définir un lien virtuel au niveau d'un nœud du chemin physique affecté à ce lien virtuel. En d'autres termes, les entrées de ces tables sont utilisées pour stocker les règles de correspondance des paquets appartenant aux flux de données supportés par le lien virtuel ainsi que les actions à appliquer sur ces paquets. Pendant que les entrées des tables de flux sont destinées à maintenir les règles de correspondance et les actions simples telles que *faire sortir le flux via une interface*, les actions plus complexes sont quant-à elles définies dans la table de groupe. Cette dernière table est nécessaire lorsque plusieurs ou un groupe d'interfaces sont utilisées pour acheminer un flux. C'est typiquement le cas lorsque le flux est soit divisé (splitté) pour être reparti entre plusieurs liens de sortie, et/ou multiplié (dupliqué) sur plusieurs liens notamment pour le support des liens virtuels point-à-multipoints.

La capacité des tables est actuellement limitée à quelques milliers d'entrées [Nakagawa 2013]. Pour cette raison, il est indispensable de prendre en compte le nombre d'entrées disponibles au niveau des tables des nœuds d'acheminement, faute de quoi l'établissement des liens virtuels sur l'infrastructure peut être sujet d'erreurs ou alors peut réussir sans toutefois garantir le niveau de service souhaité [Bays 2016]. Dans la suite, on utilisera le terme ressources de commutation pour faire référence aux entrées des tables de flux et de groupe. Aussi, le nombre d'entrées requises dans les tables de flux et la table de groupe pour définir un lien virtuel, constitue la charge de commutation du routeur vis-à-vis de ce lien virtuel.

Nous faisons l'hypothèse suivante : 1 entrée est requise dans les tables de flux si et seulement si les flux de données supportés par le lien virtuel traversent le nœud, quel que soit leur nombre, indépendamment de leur volume et de la complexité des règles de correspondance qui seront utilisées pour définir le lien virtuel. Aussi, 1 entrée est requise dans la table de groupe si et seulement si les flux de données supportés par le lien virtuel traversent le nœud et sortent via deux de ses liens physiques adjacents.

3.3.2.3 Full-Duplex

On considère que l'intégralité des liens du réseau physique fonctionnent en mode full-duplex, c'est à dire permettant les communications dans les deux sens **simultanément**. Ainsi un lien (i, j) de capacité 100 Gbps possède des ressources pouvant soutenir un lien virtuel de capacité

100 Gbps de i vers j ET un lien virtuel de même capacité de j vers i. Les ressources utilisées dans un sens n'entrent donc pas en compte dans le calcul des ressources utilisées dans l'autre sens.

3.4 Modélisation

Après avoir défini le problème de virtualisation considéré dans le chapitre précédent, ce chapitre propose une modélisation non linéaire en utilisant le formalisme de la programmation mathématique.

3.4.1 Données en entrée

3.4.1.1 Le réseau physique

La topologie du réseau physique est représentée par un graphe $G = (V, E)$ où V ($|V| = P$) est l'ensemble des nœuds physiques et E ($|E| = Q, E \subseteq V \times V$) l'ensemble des liens physiques reliant deux nœuds.

On note sans distinction (i, j) ou e un lien physique, en fonction du contexte, et on considère que tous les liens sont bidirectionnels (full-duplex).

Chaque nœud possède une seule table de flux et est défini par la capacité L_i de sa table de flux et la capacité M_i de sa table de groupe. Chaque lien est défini par sa capacité B_{ij} et son temps de propagation D_{ij}.

$$\forall (i, j) \in E : D_{ij} = D_{ji} \; et \; B_{ij} = B_{ji} \tag{3.1}$$

On définit la fonction $Ng : V \to 2^V$ qui associe à un nœud i donné sa liste de voisins, i.e. :

$$Ng(i) = \{j | j \in V, (i, j) \in E \; ou \; (j, i) \in E\}$$

3.4.1.2 Les requêtes

On suppose en entrée un ensemble de K liens virtuels point à point ou point à multipoint $k_1, \ldots, k_K$, chaque lien virtuel k étant défini par :

- Une source $s_k \in V$.
- Un ensemble de destination(s) $T_k \subseteq V - \{s_k\}$. Dans le cas d'un lien unicast, $|T_k| = 1$.
- Un besoin en bande passante $b_k \in \mathbb{N}^*$ et un délai acceptable $d_k \in \mathbb{N}^*$.
- La taille maximale des paquets p_k transmis par la session.
- La bande passante minimale b_k^{min} utilisable sur un lien (pour la gestion du « Path-Splitting » afin d'éviter un trop gros découpage dans les flux).

Remarque : Si $b_k^{min} = b_k$, le flux associé à la session ne peut pas être divisé et réparti sur différents chemins. En pratique, on définira b_k^{min} comme un pourcentage (ratio) de b_k, $b_k^{min} = PS_{ratio} * b_k$ avec $PS_{ratio} \in [0, 1]$ (Il y a possibilité de path-splitting si $PS_{ratio} \leq 0.5$, pour éviter des calculs inutiles on fixera $PS_{ratio} = 1.0$ lorsque que le path-splitting n'est pas autorisé).

3.4.1.3 Évolution temporelle

En pratique, on considère des requêtes arrivant les unes à la suite des autres, chacune étant traitée indépendamment des autres (i.e. chaque arrivée de requête donne lieu à une nouvelle résolution du modèle, puisque le modèle est construit pour l'allocation d'une seule requête).

Lors de l'arrivée d'une requête au temps t, afin de prendre en compte les précédentes allocations, il est nécessaire de connaître les ressources du réseau utilisées au temps t, soit :

- La charge actuelle d'un routeur i : $\overline{L_i}$ et $\overline{M_i}$
- La charge actuelle d'un lien e : $\overline{B_e}$

3.4.2 Variables

3.4.2.1 Flux

Pour chaque lien virtuel $k \in K$ et chaque destination $t \in T_k$, on définit une fonction $f_k^t : E \to \mathbb{N}$ telle que $f_k^t(i,j)$ définit la quantité de bande passante consommée sur (i,j) pour faire transiter des données de s_k vers t.

La quantité réelle de bande passante utilisée pour un lien virtuel $k \in K$ sur un lien physique $(i,j) \in E$, $f_k(i,j)$ est alors définie par :

$$f_k(i,j) = \max_{t \in T_k} f_k^t(i,j) \tag{3.2}$$

3.4.2.2 Délai

A nouveau, pour chaque lien virtuel $k \in K$ et chaque destination $t \in T_k$, on définit une fonction $d_k^t : V \to \mathbb{N}$ tel que $d_k^t(i)$ représente le délai maximum pour qu'un message à destination du nœud destination $t \in T_k$ atteigne le nœud i en partant de la source s_k. Le calcul des $d_k^t(i)$ utilise la valeur de bande passante consommée sur les liens et est défini par 3.3.

$$\forall k \in K, \forall t \in T_k, \forall i \in V : d_k^t(i) = \begin{cases} 0 & si\ i = s_k \\ \max\limits_{\substack{j \in Ng(i) \\ f_k^t(j,i) > 0}} (d_k^t(j) + \frac{p_k}{f_k^t ij} + d_{ji}) & si\ i \neq s_k \end{cases} \tag{3.3}$$

Le délai $d_k^t(i)$ prend en compte l'ensemble des voisins du nœud i ($j \in Ng(i)$) tels que le lien (j,i) soit utilisé par le lien virtuel k pour la cible t ($f_k^t(j,i) > 0$).

Remarque : Le délai calculé est une borne supérieure du délai réel, ce qui, dans certains cas, pourra entraîner le refus biaisé d'une requête et donc empêchera la résolution optimale du problème.

3.4.2.3 Charge des routeurs

Pour chaque nœud $i \in V$ et chaque lien virtuel $k \in K$, on associe deux variables, $l_k(i) \in \{0,1\}$ et $m_k(i) \in \{0,1\}$ définissant les ressources de commutation utilisées par le lien virtuel k sur le nœud i. Il s'agit respectivement du nombre d'entrées utilisées dans la table de flux et dans la table de groupe pour définir le lien virtuel k. Le calcul de la charge, sous l'hypothèse définie en 3.3.2.2 est donné par l'équation suivante :

$$\forall k \in K, \forall i \in V : l_k(i) = \begin{cases} 1 & si \ |\{j | j \in Ng(i), f_k(j,i)\}| \geq 1 \\ 0 & sinon \end{cases} \tag{3.4}$$

$$\forall k \in K, \forall i \in V : m_k(i) = \begin{cases} 1 & si \ |\{j | j \in Ng(i), f_k(i,j)\}| \geq 2 \\ 0 & sinon \end{cases} \tag{3.5}$$

3.4.3 Contraintes

3.4.3.1 Conservation du flux et respect de la demande en bande passante

Comme dans un problème de *Multi-Commodity Flow* standard, on impose une contrainte sur le flux dans chaque nœud du réseau physique, spécifiant que la quantité de données entrantes est égale à la quantité de données sortantes (à l'exception des nœuds source et destinations).

$$\forall k \in K, \forall t \in T_k, \forall i \in V : \sum_{j \in Ng(i)} (f_k^t(i,j) - f_k^t(j,i)) = \begin{cases} b_k & si \ i = s_k \\ -b_k & si \ i = t \\ 0 & sinon \end{cases} \tag{3.6}$$

3.4.3.2 Respect de la contrainte de capacité des liens physiques

Pour s'assurer que la capacité en bande passante de chaque lien physique est respectée, on pose la contrainte suivante :

$$\forall (i,j) \in E : \sum_{k \in K} f_k(i,j) \leq B_{ij} - \overline{B_{ij}} \tag{3.7}$$

Remarque : Cette contrainte n'est plus valable si certains liens du réseau physique ne sont plus en mode full-duplex.

3.4.3.3 Respect de la contrainte de bande passante minimale

Pour s'assurer que si une partie du flux f_k^t, associé à la cible t d'un lien virtuel k est alloué sur le lien e, alors la quantité allouée est au moins b_k^{min}, on ajoute la contrainte suivante :

$$\forall k \in K, \forall t \in T_k, \forall e \in E : f_k^t(e) \neq 0 \Rightarrow f_k^t(e) \geq b_k^{min} \tag{3.8}$$

3.4.3.4 Respect de la contrainte de délai

Le respect de la contrainte de délai pour un lien virtuel k est définie par 3.9.

$$\forall k \in K, \forall t \in T_k : d_k^t(t) \leq d_k \tag{3.9}$$

3.4.4 Fonction objective

Lors de la résolution du problème de virtualisation considéré, nous allons chercher à la fois à équilibrer la charge du réseau physique mais aussi à limiter l'utilisation des ressources du réseau physique.

3.4.4.1 Équilibre des charges

La fonction objective considérée cherche à équilibrer la charge du réseau physique exprimée en termes de charge au niveau des routeurs et des liens. Pour exprimer cet objectif d'équilibrage de charge, deux versions de cette fonction objective sont proposées : une basée sur la minimisation de la somme des carrés (3.10), l'autre sur la minimisation du maximum (3.11). Chacune de ces versions est décomposée en deux parties : la première représente la charge des liens (pondérée par le coefficient α) et la seconde représente la charge des routeurs (pondérée par le coefficient β).

$$\text{minimiser } \alpha * \sum_{e \in E} \left(\frac{1}{B_e} * \left(\overline{B_e} + \sum_{k \in K} f_k(e) \right)^2 \right) + \beta * \sum_{i \in V} \left(\frac{1}{L_i} * \left(\overline{L_i} + \sum_{k \in K} l_k(i) \right)^2 \right) \quad (3.10)$$

$$\text{minimiser } \alpha * \max_{e \in E} \left\{ \frac{1}{B_e} * \left(\overline{B_e} + \sum_{k \in K} f_k(e) \right) \right\} + \beta * \max_{i \in V} \left\{ \frac{1}{L_i} * \left(\overline{L_i} * \sum_{k \in K} l_k(i) \right) \right\} \quad (3.11)$$

La deuxième version proposée (3.11) a l'avantage d'être facilement linéarisable, mais perd très vite de son intérêt lorsqu'un des composants du réseau physique devient surchargé (le max vaut alors 1 et il n'y a plus de recherche d'équilibre).

3.4.5 Variation sur le calcul de la fonction objective

Les valeurs principales utilisées par la fonction objective sont les $f_k(i)$ et $l_k(i)$.

Pour chacun des 2 types de valeurs, on peut considérer 3 variations au niveau des fonctions objectives :

1. Ne prendre en compte que les ressources utilisées par la requête actuelle pour chaque élément.
2. Prendre en compte les ressources utilisées par la requête ainsi que les ressources déjà consommées sur chaque élément, mais ne pas compter les éléments sur lesquels aucune ressource n'est consommée par la requête actuelle.
3. Prendre en compte tous les éléments, ainsi que les ressources déjà utilisées, même si aucune ressource n'est consommée par la requête actuelle sur l'élément.

Dans le cas 1, il suffit de poser $\overline{B_e} = 0$ et $\overline{L_i} = 0$ pour tout $e \in E$ et $i \in V$. Le cas 3 est celui présenté dans ce document. Le cas 2 s'obtient en remplaçant les ensembles d'itérations des sommes et maximums de la façon suivante :

$$E' = \left\{ e \in E, \sum_{k \in K} f_k(e) > 0 \right\} \text{ et } V' = \left\{ i \in V, \sum_{k \in K} l_k(i) > 0 \right\}$$

3.4.6 Apport du modèle par rapport à l'existant

Le modèle présenté ici permet l'allocation de ressources pour des liens virtuels sur un réseau physique. Bien qu'il existe déjà plusieurs modèles pour ce problème (voir section 3.2.2), la particularité de celui-ci réside dans :

- La possibilité d'allouer des liens de type point à point (unicast) ou point à multipoint (multicast).
- La prise en compte de contraintes QoS de type bande passante et délai au niveau des liens virtuels.
- La prise en compte au niveau des routeurs de la charge des liens virtuels.

3.5 Modèle linéaire

Le modèle précédemment proposé n'est pas linéaire, son efficacité pratique est ainsi limitée. Dans cette section, nous allons présenter une linéarisation possible.

La linéarisation consiste à ramener chaque contrainte c du problème (ainsi que la fonction objective) sous une forme linéaire du type $a_{c,1} * x_1 + a_{c,2} * x_2 + \ldots + a_{c,n} * x_n \leq b_c$, avec $x_1 \geq 0, \ldots, x_n \geq 0$ les variables du problème (f_k^t et l_k^t par exemple dans notre modèle). Le modèle peut alors être écrit sous la forme :

$$\begin{cases} \text{minimiser } C^\top \mathrm{X} \\ AX \leq B \end{cases}$$

Avec $X = (x_0, \ldots, x_n)$, C un vecteur de taille n (nombre de variables), B un vecteur de taille m (nombre de contraintes) et A une matrice de taille $m \times n$.

Afin de linéariser le modèle, plusieurs variables et contraintes doivent être ajoutées, et la fonction objective doit être reformulée.

3.5.1 Variables

Afin de linéariser la contrainte impliquant un max (3.2), la quantité réelle de bande passante allouée à un lien virtuel $k \in K$ sur un lien physique $(i, j) \in E$, $f_k(i, j)$, est maintenant définie par un ensemble de contraintes (cf. 3.14).

On utilisera pour cela deux nouveaux ensembles de variables, $g_k^t(i, j) \in \mathbb{B}$ et $g_k(i, j) \in \mathbb{B}$, indiquant la présence d'un flux sur le lien (i, j) pour le lien virtuel k, à destination de t pour les premières. On ajoute la relation suivante entre $g_k^t(i, j)$ et $f_k^t(i, j)$ (cf. équation 3.15 pour la contrainte linéaire) :

$$\forall k \in K, \forall (i, j) \in E : g_t^k(i, j) = 0 \Leftrightarrow f_t^k(i, j) = 0 \tag{3.12}$$

$$\forall k \in K, \forall (i,j) \in E : g^k(i,j) = 0 \Leftrightarrow f^k(i,j) = 0 \tag{3.13}$$

On définit également deux variables f_{max} et c_{max} correspondant respectivement à la charge maximale des liens et des nœuds du réseaux physiques, pour l'ensemble des liens virtuels $k \in K$, et qui seront utilisées dans les fonctions objectives (toujours afin de linéariser les max).

3.5.2 Contraintes

3.5.2.1 Flux et bande passante

Pour linéariser le max défini en 3.2, on ajoute la contrainte suivante :

$$\forall k \in K, \forall e \in E, \forall t \in T_k : f_k^t(e) \leq f_k(e) \tag{3.14}$$

Afin de satisfaire la contrainte définie en 3.12, on ajoute la contrainte linéaire suivante :

$$\forall k \in K, \forall e \in E : g_k(e) \leq f_k(e) \; et \; f^k(e) \leq b_k * g_k(e) \tag{3.15}$$

$$\forall k \in K, \forall t \in T_k, \forall e \in E : g_k^t(e) \leq f_k^t(e) \; et \; f_k^t(e) \leq b_k * g_k^t(e) \tag{3.16}$$

Afin de prendre en compte la contrainte définie en 3.8, il suffit de contraindre la partie gauche de l'équation 3.16, qui devient alors :

$$\forall k \in K, \forall e \in E : b_k^{min} * g_k^t(e) \leq f_k^t(e) \; et \; f_k^t(e) \leq b_k * g_k^t(e) \tag{3.17}$$

3.5.2.2 Délai

On ajoute les contraintes suivantes concernant le délai de transmission des messages appartenant à une session k :

$$\forall k \in K, \forall t \in T_k : d_k^t(s_k) = 0 \tag{3.18}$$

$$\forall k \in K, \forall t \in T_k, \forall (i,j) \in E : d_k^t(i) + \frac{p_k}{b_k^{min}} + d_{ij} - (1 - g_k^t(i,j)) * \Psi \leq d_k^t(j) \tag{3.19}$$

Avec Ψ un nombre assez grand pour qu'un nœud sur lequel ne transite aucune donnée à destination de t pour la session k ne soit pris en compte dans l'équation précédente.

On notera ici que $\forall (i,j) \in E,\ b_k^{min} \leq f_k^t(i,j)$, et donc $\frac{p_k}{f_k^t(i,j)} < \frac{p_k}{b_k^{min}}$. Le délai obtenu après linéarisation est donc une majoration du délai obtenu par la modélisation normale (lui même une majoration du délai réel), la contrainte de délai réel est donc toujours respectée.

$$\forall k \in K, \forall t \in T_k : d_k^t(t) \leq d_k \tag{3.20}$$

La contrainte 3.20 est juste une copie de 3.9, alors que 3.18 et 3.19 remplacent la contrainte 3.3.

3.5.2.3 Charge des nœuds

La contrainte 3.4 est remplacée en utilisant les $g_k(i,j)$ par la contrainte 3.21.

$$\forall k \in K, \forall i \in V : l_k(i) \leq \sum_{\substack{j \in V \\ (j,i) \in E}} g_k(j,i) \leq P * l_k(i) \tag{3.21}$$

La contrainte 3.5 est remplacée en utilisant les $g_k(i,j)$ par la contrainte 3.22.

$$\forall k \in K, \forall i \in V : 2 * m_k(i) \leq \sum_{\substack{j \in V \\ (j,i) \in E}} g_k(j,i) \leq 1 + Q * m_k(i) \tag{3.22}$$

3.5.2.4 Charges maximales

Toujours pour linéariser les *max* utilisés dans la fonction objective 3.11, on ajoute les deux contraintes suivantes :

$$\forall e \in E : \frac{1}{B_e} * \left(\overline{B_e} + \sum_{k \in K} f_k(e) \right) \leq f_{max} \tag{3.23}$$

$$\forall i \in V : \frac{1}{L_i} * \left(\overline{L_i} + \sum_{k \in K} l_k(i) \right) \leq c_{max} \tag{3.24}$$

3.5.3 Fonction objective

La fonction objective définie 3.10 est quadratique. Bien qu'elle puisse être implémentée directement dans la plupart des solveurs, elle augmente de façon exponentielle le temps de résolution du problème.

À la place de cette fonction quadratique, la fonction suivante est utilisée :

$$\text{minimiser } \alpha_1 * f_{max} + \alpha_2 * S_E + \beta_1 * c_{max} + \beta_2 * S_V + \gamma * S_D \tag{3.25}$$

Où S_E, S_V et S_D représentent respectivement la somme des flux alloués, des capacités utilisés, des délais aux cibles et sont définis par :

$$S_E = \frac{\sum_{e \in E} \left(\frac{1}{B_e} * \left(\overline{B_e} + \sum_{k \in K} f_k(e) \right) \right)}{|E|} \tag{3.26}$$

$$S_V = \frac{\sum_{i \in V} \left(\frac{1}{L_i} * \left(\overline{L_i} + \sum_{k \in K} l_k(i) \right) \right)}{|V|} \tag{3.27}$$

$$S_D = \sum_{k \in K} \sum_{t \in T_k} d_k^t(t) \tag{3.28}$$

3.5.4 Complexité

La complexité du modèle linéaire se retrouve dans le nombre de variables et contraintes nécessaires à sa résolution. Pour un réseau composé de P routeurs et Q liens, l'allocation d'une requête composée de K liens virtuels, avec un maximum de T destinations par lien virtuel nécessite :

- $O(KT * (2Q + 2P))$ variables.
- $O(KT * (4Q + 2P))$ contraintes.

3.6 Évaluation expérimentale

Nous présentons dans cette section les résultats de l'analyse de performances de cet algorithme sur la topologie réseau GÉANT ainsi que sur une topologie réseau de Campus. Les objectifs de ces différentes analyses sont :

1. La mise en évidence de l'apport (taux d'acceptation) du modèle par rapport à des méthodes heuristiques.
2. La caractérisation des temps de calcul en fonction du taux d'arrivée des requêtes, de leurs types (unicast/multicast) et de l'utilisation du path-splitting.
3. L'influence des paramètres sur les résultats du modèle (à nouveau en terme de taux d'acceptation).
4. La visualisation de l'équilibre des charges dans le réseau via l'utilisation d'une fonction de répartition sur le taux d'utilisation des liens et des nœuds.

3.6.1 Modèle de simulation considéré

3.6.1.1 Modèle de réseau

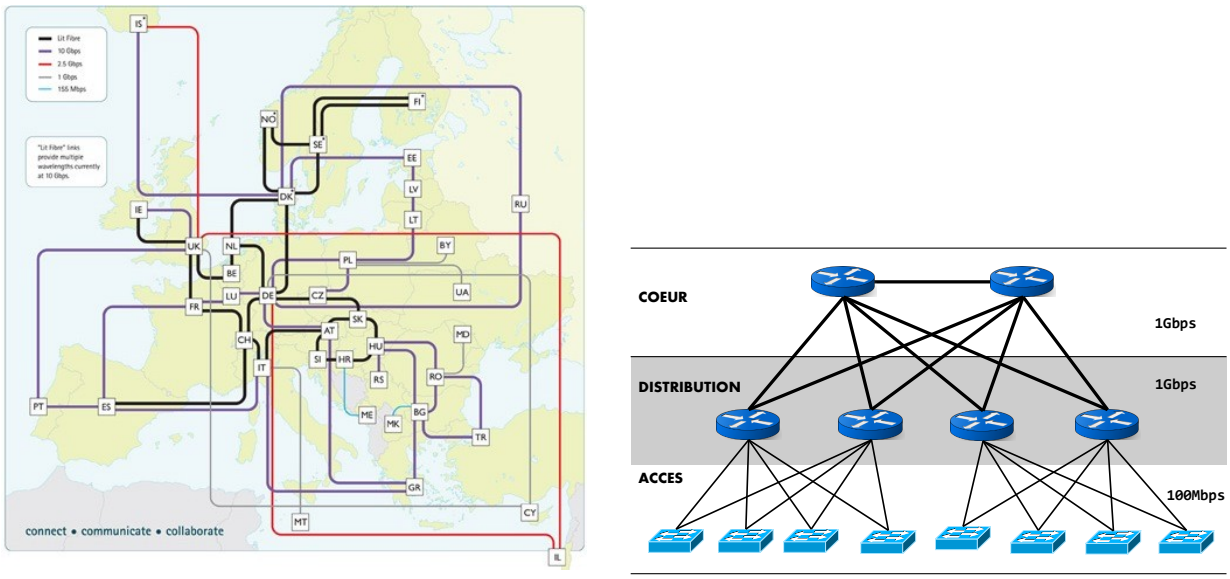
Contrairement aux travaux existants qui expérimentent sur des réseaux générés aléatoirement, le choix a été fait d'utiliser une topologie réseau réelle. Nous utilisons deux modèles de réseau pour les expérimentations :

- le réseau européen de recherche, GÉANT, qui possède des liens reliant la plupart des capitales et villes importantes européennes. La figure 3.1a présente la carte du réseau GÉANT. La version considérée du réseau contient 41 routeurs (ou nœuds) et 60 liens.
- un réseau de Campus constitué de 14 nœuds et 25 liens (voir figure 3.1b).

3.6.1.1.1 Capacité des liens

Les capacités des liens du réseau GÉANT sont fournies par la carte. Pour les liens n'ayant pas une capacité précise, les règles suivantes sont utilisées :

- pour les liens dont la capacité est comprise entre 1 Gbps et 10 Gbps : la capacité de ces liens est fixée à 5 Gbps
- pour les liens dont la capacité est supérieure à 100 Gbps : leur capacité est ramenée à 100 Gbps.



(a) Carte du réseau GÉANT - version Octobre 2015

(b) Topologie réseau de Campus

FIGURE 3.1 – Modèles de réseau considérés

Pour le réseau de Campus, les liens d'extrémité sont fixés à 100 Mbps contre 1 Gbps pour l'épine dorsale du réseau.

3.6.1.1.2 Délais de transmission des liens

La carte du réseau GÉANT ne fournit pas directement les délais de transmissions des liens, mais ceux-ci sont calculables en utilisant la localisation des routeurs. Les liens sont considérés en ligne droite d'un routeur à l'autre, et la vitesse de transmission v_{tr} est fixée à 3.10^8 m/s. La distance terrestre $D(a, b)$ entre deux routeurs est calculée à partir de leurs longitudes et latitudes, le délai de propagation du routeur a vers le routeur b, $tr(a, b)$, est alors obtenu simplement par :

$$tr(a, b) = \frac{D(a, b)}{v_{tr}}$$

Pour le réseau de campus, les délais de propagation sont considérés nuls.

3.6.1.1.3 La capacité des routeurs

La capacité de commutation des routeurs, exprimées en nombre d'entrées disponibles dans la table des flux, est fixée à 200 entrées. On considère ici qu'une partie des entrées de la table des flux des nœuds est utilisée pour le provisionnement des VNETs, le reste étant utilisé pour d'autres fonctions réseau. La capacité de la table de groupe OpenFlow est fixée à 100 entrées.

3.6.1.2 Modèle de charge

3.6.1.2.1 Le taux d'arrivée des requêtes

Les requêtes arrivent selon un processus de Poisson de paramètre λ, avec λ variant de 4 à 10 requêtes toutes les 100 unités de temps par pas de 1 : $\lambda \in \{0.04, 0.05, 0.06, 0.07, 0.08, 0.09, 0.1\}$.

3.6.1.2.2 La durée de vie des requêtes

Les requêtes possèdent une durée de vie moyenne de 1000 unités de temps (UT [1]) suivant une loi exponentielle de paramètre $\mu = 1/1000$.

3.6.1.2.3 Les différentes types de requêtes

Deux types de requêtes ont été considérés pour les expérimentations : Unicast et Multicast.

Les requêtes Unicast : Les requêtes unicast contiennent entre 6 et 8 liens virtuels point à point. Les paramètres des liens virtuels sont générés aléatoirement selon des lois uniformes, avec les contraintes suivantes :

- Bande passante Entre 600 et 800 Mbps (resp. 1 à 3 Mbps) pour les expérimentations relatives à GÉANT (resp. Réseau de Campus).
- Délai pas pris en compte.

Les requêtes Multicast : Les requêtes multicast contiennent plusieurs liens virtuels point à multipoint : Entre 6 et 8 liens virtuels, chacun possédant entre 2 et 4 destinations. Les paramètres de chaque lien virtuel sont générés aléatoirement selon des lois uniformes, avec les contraintes suivantes :

- Bande passante Entre 600 et 800 Mbps (resp. 1 à 3 Mbps) pour les expérimentations relatives à GÉANT (resp. réseau de campus).
- Délai pas pris en compte.

Remarque : Pour qu'une requête VNET contenant plusieurs liens virtuels soit acceptée par le modèle, il faut que tous ses liens puissent être alloués sur le réseau, i.e. s'il n'existe pas de chemin de capacité suffisante reliant la source et les destinations d'un lien, la requête entière est rejetée.

3.6.1.2.4 Le path-splitting

Les expérimentations ont été effectuées avec l'utilisation du path-splitting ($PSratio \in \{10\%, 33\%, 45\%\}$) et sans ($PSratio = N/A$). Un lien virtuel avec un $PSratio$ de 10% (resp. 33% et 45%) ne pourra être allouée sur plus de 10 (resp. 3 et 2) chemins différents, chaque chemin étant utilisé pour 10% (resp. 33% et 45%) de la bande passante. Avec un $PSratio \geq 50\%$ (N/A dans notre cas), le lien doit être alloué sur un seul chemin physique.

1. 1UT correspond à la durée moyenne de traitement d'une requête

3.6.1.2.5 Le choix des sources et destinations

Les nœuds sources et destinations des liens virtuels sont choisis suivant une probabilité proportionnelle à la somme des capacités des liens auxquels ils sont connectés, soit :

$$p_i \propto \sum_{(i,j)\in E} B_{i,j} \tag{3.29}$$

3.6.1.3 Les paramètres d'expérimentation

3.6.1.3.1 La durée d'expérimentation

Les expérimentations ont toutes été effectuées sur une durée de 10000 unités de temps. Les courbes 3.4 et 3.11 montrent que cette durée est suffisante pour atteindre un état stationnaire du système.

3.6.1.3.2 La fonction objective

La fonction objective utilisée est celle définie en 3.25, avec la variation numéro 3 (variation utilisée dans la présentation du modèle).

Les paramètres α, β et γ : Les coefficients α_1 et α_2 ont été fusionnés : $\alpha_1 = \alpha_2 = \alpha$. Les coefficients β_1 et β_2 ont également été fusionnés : $\beta_1 = \beta_2 = \beta$. Pour les expérimentations utilisant le réseau GÉANT, α a été fixé égal à 1 et 4 valeurs différentes de β ont été utilisées : $\beta \in \{1, 50, 100, 150\}$. De manière similaire, pour les expérimentations utilisant le réseau de Campus, β a été fixé égal à 1 et 4 valeurs différentes de α ont été utilisées : $\alpha \in \{1, 50, 100, 150\}$. La minimisation des délais n'étant pas recherchée dans les expérimentations, le coefficient γ a été fixé à la valeur 0.

3.6.1.3.3 Fonctionnement des expérimentations

On considérera dans la suite des instances de test : une instance est un ensemble de requêtes, générées sur une durée fixe (10000 unités de temps dans notre cas) avec un taux d'arrivée et des paramètres définis.

En considérant le modèle réseau GÉANT (resp. de Campus), pour chaque type de requête (unicast et multicast), et chaque taux d'arrivée (de 4 à 10 requêtes toutes les 100 unités de temps), une instance est générée avec les paramètres β et PS_{ratio} (resp. α et PS_{ratio}) fixés.

Chaque instance générée a ensuite été rejouée en faisant varier les paramètres $\beta \in \{1, 50, 100, 150\}$ et $PSratio \in \{10\%, 33\%, 45\%, N/A\}$ (resp. $\alpha \in \{1, 50, 100, 150\}$ et $PSratio \in \{10\%, 33\%, 45\%, N/A\}$), portant le total d'instance à $2 * 7 * 4 * 4 = 224$. Chaque instance peut alors être caractérisée ainsi : $(Type, \lambda, \beta, PS_{ratio})$ (resp. $(Type, \lambda, \alpha, PS_{ratio})$).

3.6.2 Métriques de performance

Les performances de l'algorithme ont été mesurées à l'aide de plusieurs métriques : le taux d'acceptation des requêtes, le taux d'utilisation des liens et des nœuds, et la charge globale des liens et des nœuds.

Les métriques ont été mesurées :

- À la fin de la simulation (à $T = 10000$ unités de temps), les graphes sont alors fournis en fonction du taux d'arrivée des requêtes λ.
- Sur toute la période de simulation d'une expérience liée à une instance $(Type, \lambda, \beta, PS_{ratio})$ ou $(Type, \lambda, \alpha, PS_{ratio})$, les graphes sont alors fournis en fonction du temps de simulation T (en unité de temps).

3.6.2.1 Le taux d'acceptation des requêtes

Le taux d'acceptation des requêtes correspond simplement au nombre de requêtes qui ont été allouées sur le réseau, par rapport au nombre total de requêtes générées pendant la durée d'une simulation (ou une partie de cette durée dans le cas d'étude sur la période de simulation).

$$\tau_{acceptation} = \frac{\text{Nombre de requêtes allouées}}{\text{Nombre de requêtes reçues}}$$

Exemple : Si une instance contient 400 requêtes, et que 260 requêtes ont été allouées avec succès, alors le taux d'acceptation est de $\tau = \frac{260}{400} = 65\%$.

3.6.2.2 La charge globale du réseau

3.6.2.2.1 La charge globale des liens du réseau

La charge globale des liens du réseau correspond à la quantité totale de ressources consommées sur les liens (à un instant donné) par rapport à la capacité totale des liens, soit :

$$C_{liens} = \frac{\sum_{e \in E} \overline{B_e}}{\sum_{e \in E} B_e}$$

Avec B_e la capacité du lien e et $\overline{B_e}$ les ressources consommées sur le lien e à l'instant considéré.

Remarque : Comme on considère des liens full-duplex, il est important de noter que l'ensemble E des liens contient, pour chaque lien de GÉANT ou Campus, deux liens : Un de i vers j et un de j vers i. Lors du calcul de la charge globale des liens du réseau, chaque lien du réseau GÉANT ou du réseau de Campus est donc considéré deux fois, i.e. pour que la charge soit de 100%, il faudrait que chaque lien utilise la totalité de ses ressources dans les deux sens.

3.6.2.2.2 La charge globale des nœuds du réseau

La charge globale des nœuds du réseau correspond à la quantité totale d'entrées de la table de flux sur les nœuds (à un instant donné) par rapport à la capacité totale de la table de flux des nœuds, soit :

$$C_{n\not{s}uds} = \frac{\sum_{i \in V} \overline{L_i}}{\sum_{i \in V} L_i}$$

Avec L_i la capacité de la table de flux du nœud i et $\overline{L_i}$ la quantité totales d'entrées consommées dans la table de flux du nœud i à l'instant considéré.

3.6.2.3 Le taux d'utilisation du réseau

3.6.2.3.1 Le taux d'utilisation des liens

Le taux d'utilisation des liens correspond à la moyenne du taux d'utilisation de chaque lien (à un instant donné), c'est à dire les ressources consommées sur le lien par rapport à la capacité du lien, soit :

$$\tau_{liens} = \frac{1}{|E|} * \sum_{e \in E} \frac{\overline{B_e}}{B_e}$$

Avec B_e la capacité du lien e et $\overline{B_e}$ les ressources consommées sur le lien e à l'instant considéré.

Remarque : Tout comme pour la charge globale des liens du réseau, il est important de noter que pour atteindre un taux d'utilisation de 100%, il faudrait que chaque lien utilise 100% de ses ressources dans les deux sens. Tous les liens n'ont pas la même capacité donc $\tau_{liens} \neq C_{liens}$. Contrairement à la charge du réseau, le taux d'utilisation des liens ne fait pas la différence entre un lien de capacité 50 Gbps chargé à 80% et un lien de capacité 4 Gbps chargé à 80%.

3.6.2.3.2 Le taux d'utilisation des nœuds

Le taux d'utilisation des nœuds correspond à la moyenne du taux d'utilisation de chaque nœud (à un instant donné), c'est à dire les ressources consommées sur le nœud par rapport à la capacité de la table de flux du nœud, soit :

$$\tau_{n\not{s}uds} = \frac{1}{|V|} * \sum_{i \in V} \frac{\overline{L_i}}{L_i}$$

Avec L_i la capacité de la table de flux du nœud i et $\overline{L_i}$ la quantité totale d'entrée consommées dans la table de flux du nœud i à l'instant considéré.

3.6.3 Méthodes heuristiques existantes

Notre algorithme a été comparé à différentes heuristiques de Plus Court Chemin (PCC). En effet, les algorithmes PCC sont couramment utilisés pour le routage de paquets dans les réseaux actuels, et peuvent donc s'adapter facilement pour de l'allocation de liens virtuels.

Pour une requête unicast, l'algorithme de PCC utilisé est standard (avec les coûts définis ci-dessous). Pour une requête multicast, la procédure adoptée est la suivante :

1. L'allocation des liens virtuels se fait les uns à la suite des autres. La requête est rejetée si, pour un des liens, le chemin trouvé ne possède pas une capacité suffisante.
2. Pour les liens multicast, chaque chemin (*source ; destination*) est calculé par l'algorithme PCC de façon isolée. L'allocation finale du lien est ensuite calculée par le *max* sur chaque lien physique des capacités allouées aux liens logiques (procédure semblable à celle utilisée par notre modèle).

Trois versions du calcul du coût d'un lien ont été considérées :

Coût $= 1$ Chaque lien possède un coût de 1, cela revient à chercher le chemin passant par le moins de routeurs possibles.

Coût $= f(B_e)$ Le coût d'un chemin est inversement proportionnel à sa capacité (Coût $= 1/B_e$). C'est le calcul de coût utilisé dans le protocole OSPF.

Coût $= g(B_e, \overline{B_e})$ Le coût d'un chemin est inversement proportionnel aux ressources disponibles sur le lien (Coût $= \frac{1}{\overline{B_e} - B_e}$).

3.6.4 Résultats expérimentaux sur le réseau GÉANT

3.6.4.1 Comparaison avec des méthodes heuristiques

Les graphiques de la figure 3.2 présentent les résultats de comparaison de notre méthode avec les algorithmes précédemment décrits. Les paramètres $\beta = 150$ et $PS_{ratio} = N/A$ ont été choisis suite aux résultats comparatifs présentés dans la partie 3.6.4.4.

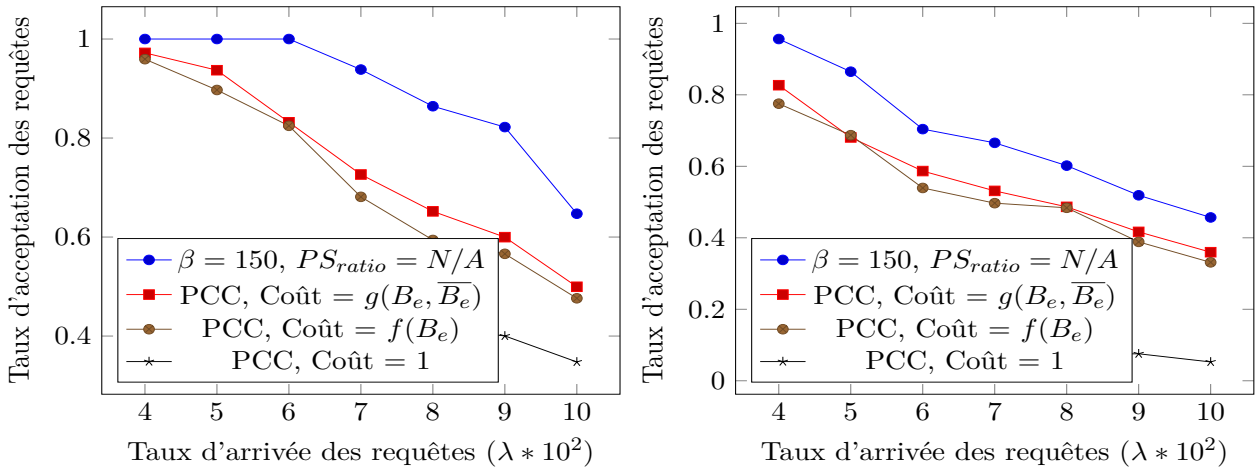
Les courbes montrent que quel que soit le type d'instance considérée, notre algorithme présente un apport notable en terme de taux d'acceptation. L'augmentation est plus importante sur des instances de type multicast (5% pour des taux d'arrivée élevés, jusqu'à 15% pour un taux d'arrivée de 0.04) que sur des instances de type unicast (entre 2 et 10%).

3.6.4.2 Caractérisation des temps de calcul

Les courbe 3.3 indiquent le temps moyen et maximal d'allocation des requêtes sur le réseau. Les valeurs fournies sont calculées sur l'ensemble des requêtes acceptées en utilisant les temps CPUs retournés par le solveur CPLEX [2].

Sur les instances de type unicast, les temps de calculs sont en moyenne faibles (inférieurs à 600 millisecondes sans path-splitting, et à 1 seconde avec path-splitting) et ne dépassent jamais 4 secondes sans path-splitting et 7 secondes avec path-splitting (la limite de 20 secondes fixées pour le solveur n'est jamais atteinte). Pour les instances de type multicast, les temps de calcul sont plus élevés (entre 2 et 4 secondes en moyenne) et la limite du solveur n'est jamais atteinte. Cette différence s'explique facilement lorsque l'on compare le nombres de variables et contraintes nécessaires pour la modélisation des requêtes de type multicast et unicast.

2. http : ://www.01.ibm.com/software/commerce/optimization/cplex-optimizer/

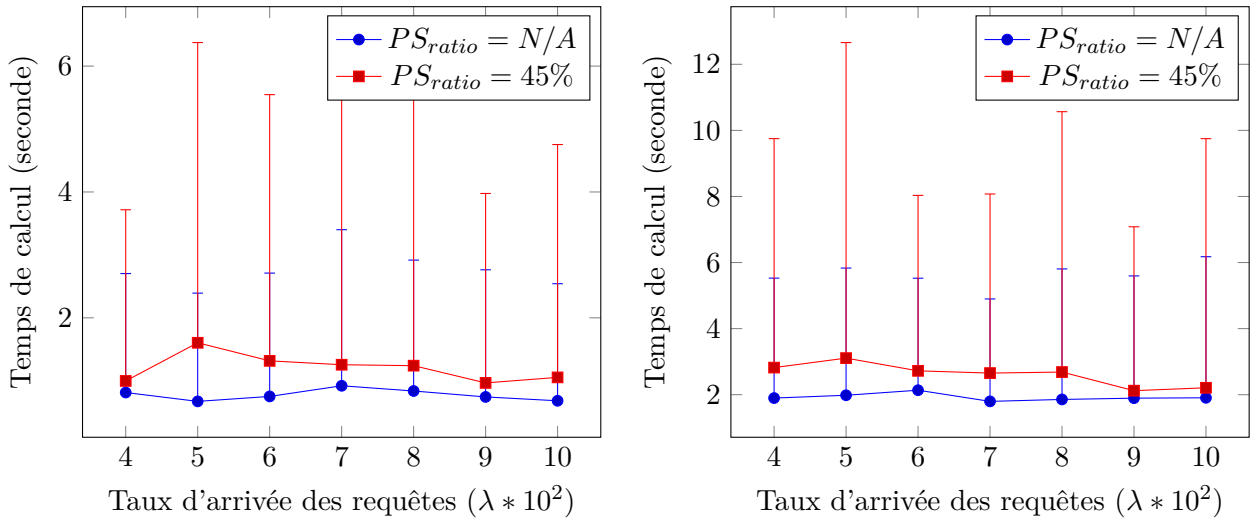


(a) Requêtes de type Unicast

(b) Requêtes de type Multicast

FIGURE 3.2 – Comparaison de différents algorithmes d'allocation

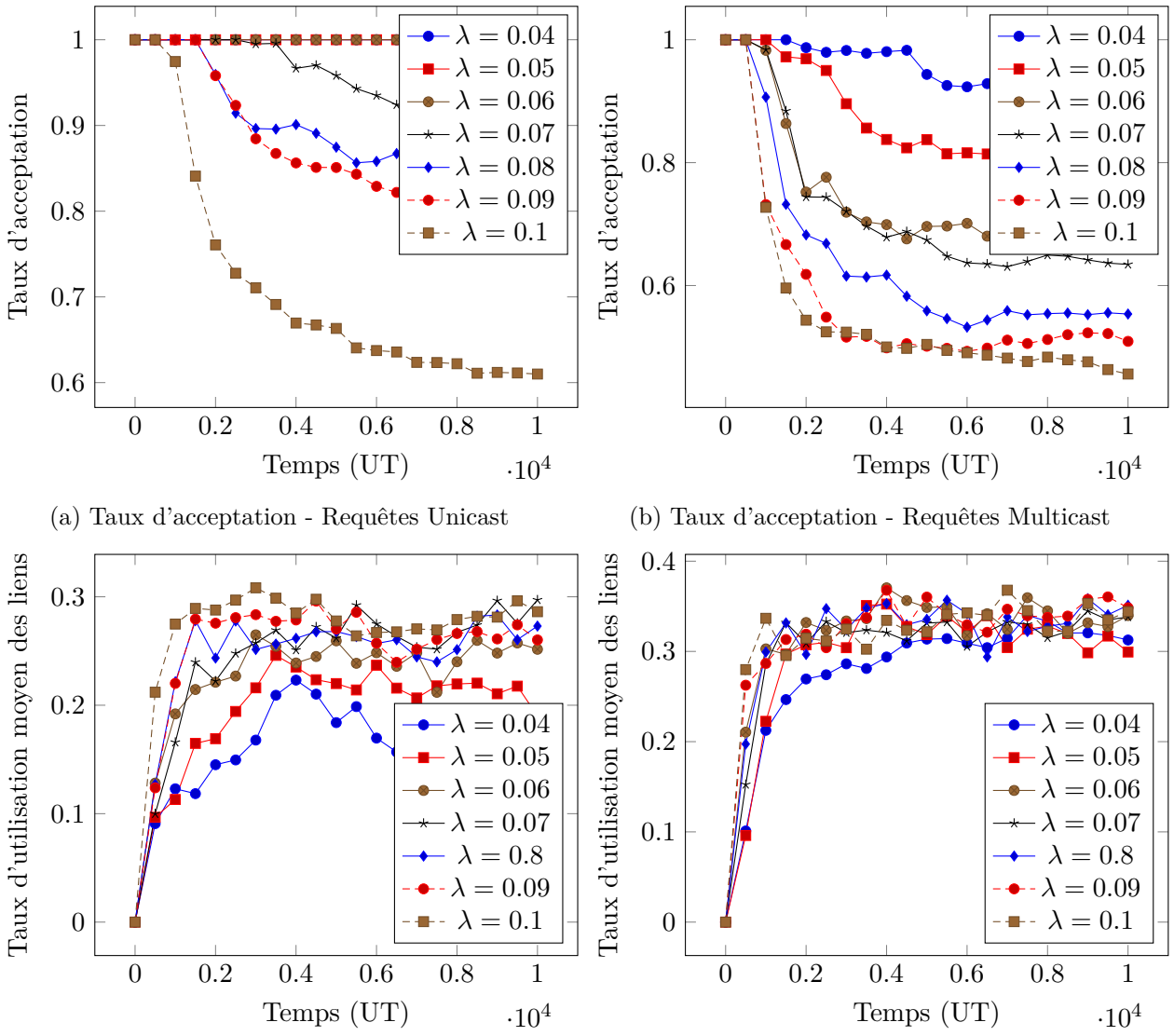
Quel que soit le type d'instance considéré, l'utilisation du path-splitting augmente de façon notable le temps de calcul, bien que celui-ci reste largement acceptable dans un contexte de fournisseur de réseaux virtuels (les temps de calcul restent négligeables en comparaison des temps de déploiement qui peuvent atteindre plusieurs minutes). L'influence du path-splitting sur le temps de calcul est directement liée à l'espace de recherche des variables de flux f_k^t - Dans un scénario sans path-splitting, elles sont limitées à deux valeurs possibles (0 ou b_k), ce qui n'est plus le cas à partir du moment où on utilise un PS_{ratio} inférieur à 50%.



(a) Temps de convergence - Requêtes Unicast - $\beta = 150$

(b) Temps de convergence - Requêtes Multicast - $\beta = 150$

FIGURE 3.3 – Influence du path splitting et du type de requête sur le temps de calcul



(a) Taux d'acceptation - Requêtes Unicast

(b) Taux d'acceptation - Requêtes Multicast

(c) Taux d'utilisation des liens - Requêtes Unicast

(d) Taux d'utilisation des liens - Requêtes Multicast

FIGURE 3.4 – Taux d'acceptation et d'utilisation des liens en fonction du temps pour des requêtes Multicast et Unicast -$\beta = 150$, $PS_{ratio} = 0.45$

3.6.4.3 Validation de la durée de simulation

Les graphiques de la figure 3.4 montrent l'évolution du taux d'acceptation et du taux d'utilisation des liens du réseau au cours de la simulation. La fluctuation des courbes s'explique par l'allocation et la libération en permanence des ressources due au fait que les requêtes ont une durée de vie limitée. En effet, il est très peu probable qu'une requête entrante puisse remplacer fidèlement une requête ayant précédemment quitté l'infrastructure. La différence des allocations correspondantes à ces deux requêtes provoque une variation dans le temps de l'état des ressources réseau, laquelle variation est à l'origine des fluctuations observées. Ces fluctuations

ont tendance à avoir une amplitude plus élevée à faible charge.

À la fin de la simulation (T = 10000 unités de temps), le système a atteint un régime stationnaire. Ce résultat nous permet de cibler les analyses suivantes en regardant uniquement les valeurs en fin de simulation.

Sur les courbes 3.4a et 3.4b, on remarque que les instances composées de requêtes de type multicast se stabilisent beaucoup plus vite (vers T = 2000 unités de temps) que les requêtes de type unicast. La chute brutale du taux d'acceptation observée sur ces courbes s'explique par l'augmentation brutale et la stabilisation du taux d'utilisation des liens, visible sur les graphiques 3.4c et 3.4d.

Contrairement aux instances composées de requêtes de type unicast (graphique 3.4c), les instances composées de requêtes de type multicast (graphique 3.4d) atteignent toutes le même taux d'utilisation des liens, quel que soit leur taux d'arrivée. Même pour des taux d'arrivée faibles ($\lambda = 0.04$), le réseau semble saturer rapidement, ainsi le nombre maximal de requêtes pouvant être allouées en même temps sur le réseau est très vite atteint.

On peut dire que, pour les instances dont les paramètres β et PS_{ratio} sont respectivement fixés à 150 et 0.45, quel que soit le type de requêtes et leur taux d'arrivée, le taux d'acceptation reste stable lorsque le taux d'utilisation des liens atteint la valeur 0.3.

3.6.4.4 Influence des paramètres du modèle

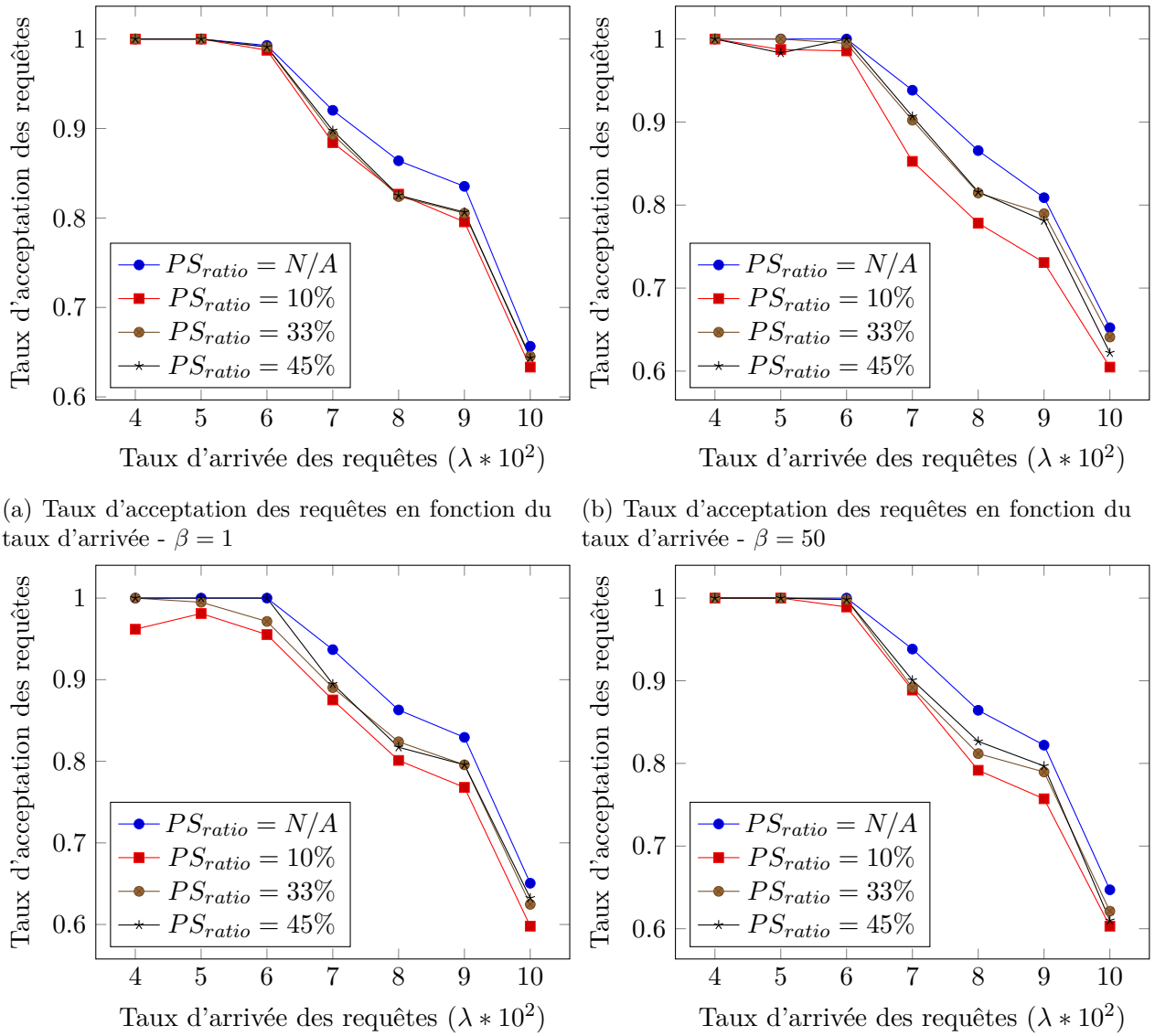
Dans cette section, nous allons étudier l'influence respective des paramètres PS_{ratio} et β de notre modèle.

3.6.4.4.1 Taux d'acception des requêtes

Les courbes des graphiques 3.5a à 3.5d et 3.6a à 3.6d présentent les taux d'acceptation des requêtes pour différentes instances de type unicast et multicast.

Requêtes de type unicast Sur les graphiques 3.5a à 3.5d, le path-splitting ne montre pas d'intérêt quelles que soient les instances considérées. Quel que soit β, les instances n'utilisant pas de path-splitting ($PS_{ratio} = N/A$) présentent toujours de meilleurs résultats que les autres. Elles surclassent les autres instances avec un gain allant de 1.5% à 9%. En revanche, sur ces mêmes graphiques, le paramètre β ne semble avoir quasiment aucun impact sur les résultats, puisque toutes les courbes de la figure 3.5 indiquent des taux d'acceptation semblables lorsque β est l'unique paramètre qui varie.

Requêtes de type multicast Les courbes de la figure 3.6 montrent que lorsque le taux d'arrivée des requêtes augmente, le taux d'acception des requêtes multicast diminue plus brutalement que celui des requêtes unicast, quelles que soient les instances. Pareil qu'avec les instances composées de requêtes de type unicast, le path-splitting ne montre également pas un intérêt



(a) Taux d'acceptation des requêtes en fonction du taux d'arrivée - $\beta = 1$

(b) Taux d'acceptation des requêtes en fonction du taux d'arrivée - $\beta = 50$

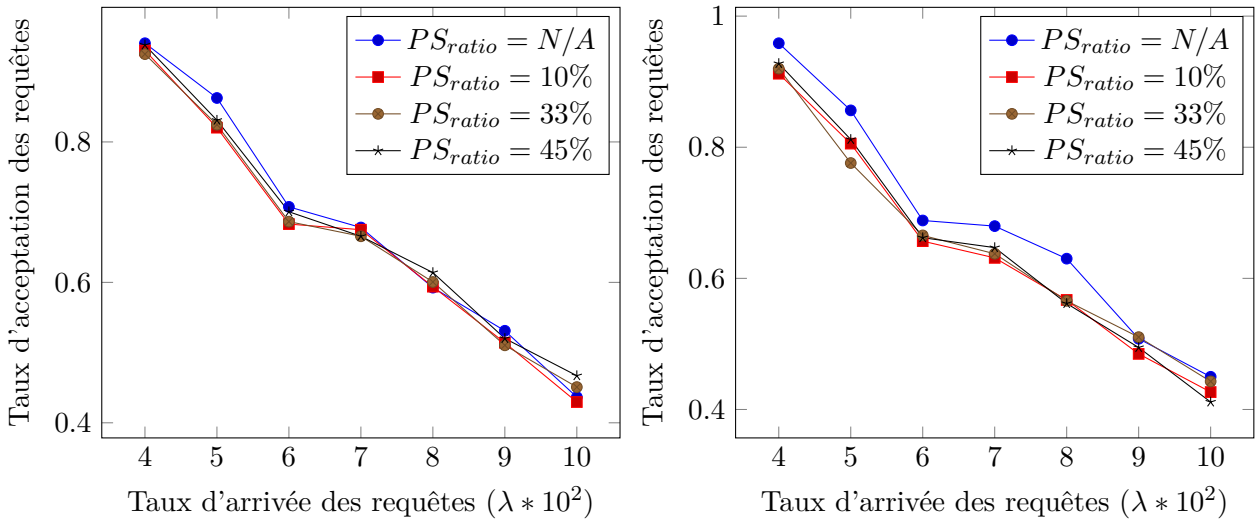
(c) Taux d'acceptation des requêtes en fonction du taux d'arrivée - $\beta = 100$

(d) Taux d'acceptation des requêtes en fonction du taux d'arrivée - $\beta = 150$

FIGURE 3.5 – Taux d'acceptation des requêtes de type **unicast** en fonction de leur taux d'arrivée et des paramètres de simulation

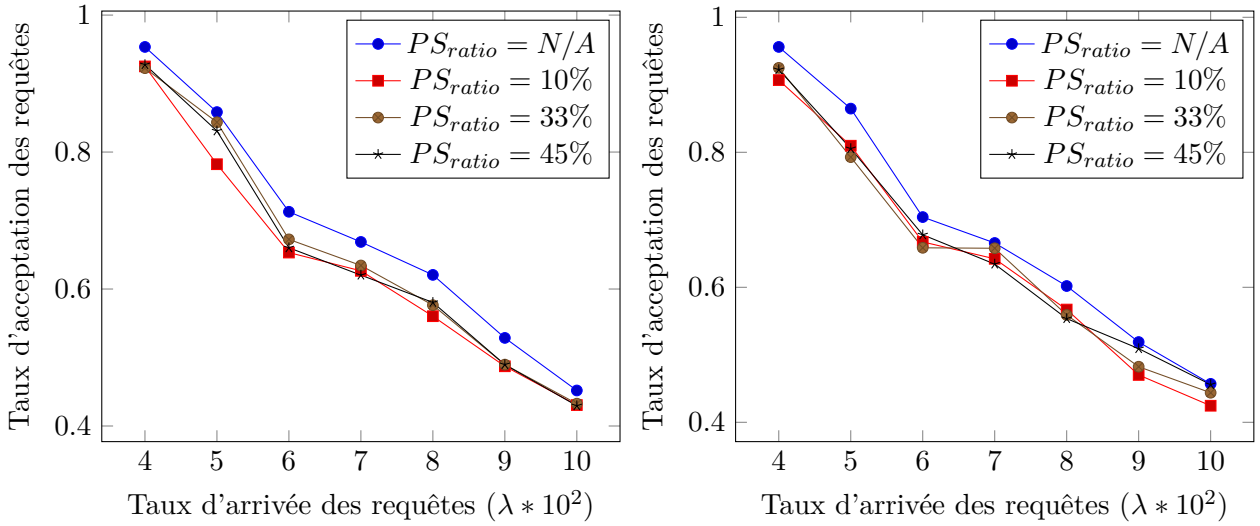
pour les instances composées de requêtes de type multicast. Ne pas utiliser le path-splitting semble la meilleure solution, quel que soit le taux d'arrivée des requêtes. Tout comme pour les instances de type unicast, le paramètre β ne semble pas avoir une influence notable sur les résultats des simulations.

Les courbes 3.7d et 3.7b révèlent que la charge globale des nœuds et la charge globale des liens du réseau sont toutes deux plus faibles pour les instances composées de requêtes multicast et



(a) Taux d'acceptation des requêtes en fonction du taux d'arrivée - $\beta = 1$

(b) Taux d'acceptation des requêtes en fonction du taux d'arrivée - $\beta = 50$



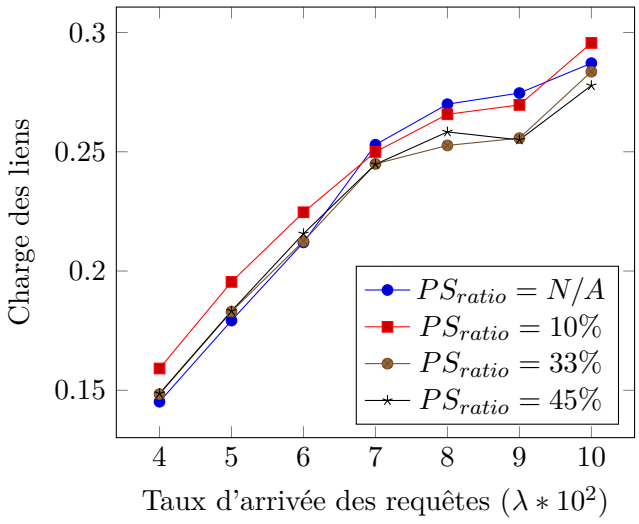
(c) Taux d'acceptation des requêtes en fonction du taux d'arrivée - $\beta = 100$

(d) Taux d'acceptation des requêtes en fonction du taux d'arrivée - $\beta = 150$

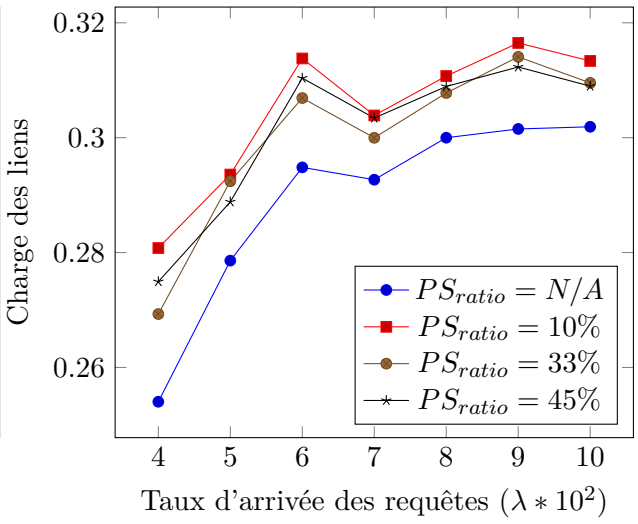
FIGURE 3.6 – Taux d'acceptation des requêtes de type **multicast** en fonction de leur taux d'arrivée et des paramètres de simulation

n'utilisant pas le path-splitting alors que celles-ci ont un meilleur taux d'acceptation par rapport aux instances qui en font usage. Ce qui signifie que, en voulant trop répartir les allocations entre les nœuds et les liens du réseau, les instances utilisant le path-splitting sur-utilisent les ressources. Cette utilisation abusive peut être entre à l'origine des rejets. Il serait possible d'atténuer cet effet du path-splitting en donnant plus d'importance à l'utilisation des ressources qu'à l'équilibre de la charge des nœuds et des liens du réseau. Ceci peut s'appliquer en fixant les paramètres de la fonction objection de telle sorte que $\alpha_1 < \alpha 2$ et $\beta_1 < \beta_2$. Cette prescription

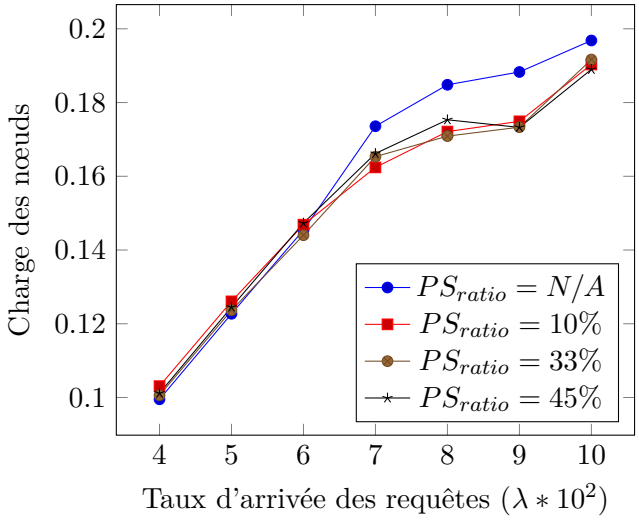
n'est pas toujours efficace dans l'absolue, en revanche elle peut potentiellement l'être pour le modèle de simulation précédemment défini et qui considère le réseau GÉANT.



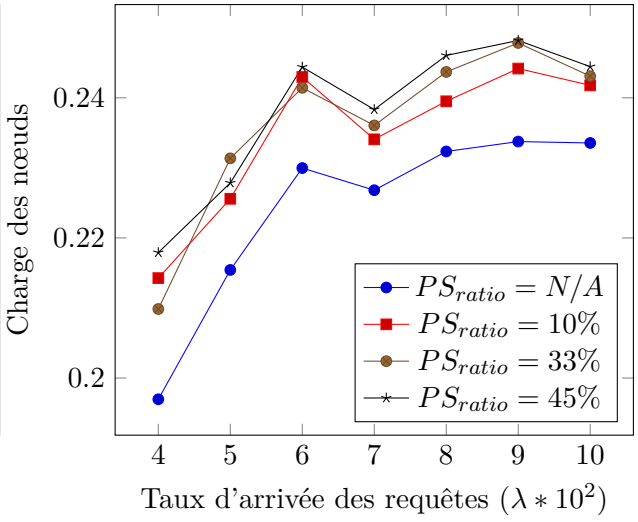
(a) Cas des liens - Requêtes de type unicast



(b) Cas des liens - Requêtes de type multicast



(c) Cas des nœuds - Requêtes de type unicast



(d) Cas des nœuds - Requêtes de type multicast

FIGURE 3.7 – Charge du réseau en fonction du taux d'arrivée pour des réquêtes de type unicast et multicast - $\beta = 150$

3.6.4.4.2 Charge du réseau

On ne considère ici que des instances avec $\beta = 150$, les analyses précédentes ayant montré la faible influence de ce paramètre sur les résultats du modèle.

Les courbes de la figure 3.7 montrent la charge globale des liens et des nœuds du réseau. Pour les instances de type unicast (graphiques 3.7a et 3.7c), à faible taux d'arrivée ($\lambda \leq 0.06$) la charge du réseau dépend très peu des paramètres d'expérimentation, ce qui signifie qu'un meilleur taux d'acceptation des requêtes n'implique pas une surcharge notable du réseau. C'est notamment le cas avec l'instance n'utilisant pas de path-splitting. Elle présente un meilleur taux d'acceptation comme le montre la figure 3.5d, alors que, comme l'indique les courbes des graphiques 3.7a et 3.7c, elle présente une charge au plus égale à celle des autres instances. En revanche, à des taux d'arrivée plus élevés, elle a tendance à consommer plus de ressources notamment parce qu'elle accepte beaucoup plus de requêtes que les autres instances de même type. On a environ un gain moyen de 1.5% pour les faibles taux d'arrivée contre 9% pour les taux d'arrivée plus élevée. (cf section 3.6.4.4.1).

Les courbes des instances de type multicast 3.7b et 3.7d sont beaucoup moins proches que celles des instances de type unicast. Contrairement au cas des requêtes de type unicast, avec les requêtes de type multicast, la charge du réseau dépend des paramètres d'expérimentation. La différence de charge est plus prononcée entre l'instance utilisant le path-splliting et les autres instances. En revanche, entre les instances utilisant le path-splitting, la différence de charge n'est pas sensible.

Les différences entre les courbes des instances de type unicast et multicast, principalement au niveau de leur amplitude, peuvent s'expliquer par une dépendance aux requêtes (nombre de liens, source, destination(s), etc.) beaucoup plus forte du côté des instances multicast.

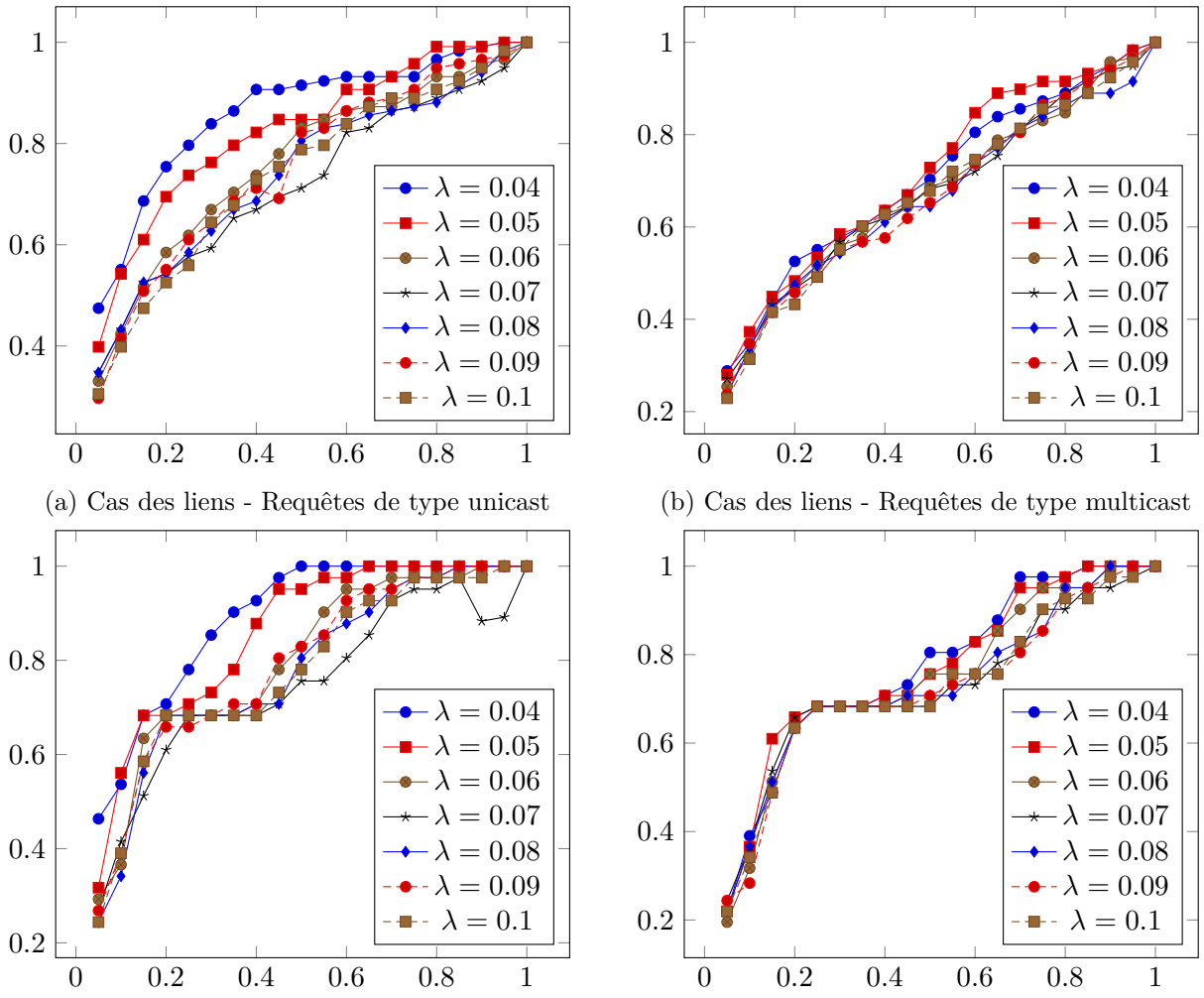
D'une manière générale, pour un type de requête donné, deux scénarios possibles expliquent le fait qu'une instance de ce type présente une charge du réseau plus élevée :

1. soit parce qu'elle admet plus de requêtes et donc présente un meilleur taux d'acception.
2. ou alors parce qu'elle sur-utilise les ressources (c'est-à-dire les ressources utilisées sont strictement supérieures à la quantité nécessaire et suffisante pour allouer les liens virtuels des requêtes admises).

3.6.4.5 Visualisation de l'équilibre de l'utilisation du réseau

La figure 3.8 présente les fonctions de répartition du taux d'utilisation des liens et des nœuds du réseau (à la fin de la simulation, soit pour $T = 10000$ unités de temps) pour les instances de type unicast (3.8a et 3.8c) et multicast (3.8b et 3.8d) pour différents taux d'arrivée.

Comme sur le graphique 3.4d, on remarque que les fonctions de répartition du taux d'utilisation pour des instances de type multicast varient peu en fonction du taux d'arrivée des requêtes (les courbes sont très proches sur les graphiques 3.8b et 3.8d). Quel que soit le taux d'arrivée des requêtes, environ 45% des liens et des nœuds sont utilisés à plus de 50% de leur capacité. Par ailleurs, on peut remarquer dans le graphique 3.8d, qu'on trouve une fraction identique de nœuds et de liens chargés à plus de 80%, soit environ 20%. Cela signifie que les entrées dans la table de flux des nœuds peuvent autant impacter sur l'admissibilité des requêtes que la bande



(a) Cas des liens - Requêtes de type unicast

(b) Cas des liens - Requêtes de type multicast

(c) Cas des nœuds - Requêtes de type unicast

(d) Cas des nœuds - Requêtes de type multicast

FIGURE 3.8 – Fonction de répartition du taux d'utilisation des liens et des nœuds pour des réquêtes de type unicast et multicast - $\beta = 150, PS_{ratio} = 45\%$

passante des liens. Cependant, force est de noter que, en face des 20% de nœuds utilisés à plus de 80%, au moins 50% des nœuds sont utilisés à moins de 20% de leur capacité. Autrement dit, 20% de nœuds manquent de ressources de commutation, en même temps, plus de 50% en disposent beaucoup plus. Une telle disparité de charge entre les nœuds du réseau n'est pas favorable à l'admissibilité des requêtes.

Les graphiques 3.8a et 3.8c confirment les résultats des courbes 3.5d : le taux d'utilisation des liens et des nœuds du réseau augmente avec l'augmentation du taux d'arrivée des requêtes. Pour de faibles taux d'arrivée ($\lambda \leq 0.05$), moins de 20% des liens et des nœuds sont utilisés à plus de 50%. Ce pourcentage atteint 30% pour un taux d'arrivée de 10 requêtes toutes les 100 unités de temps.

3.6.4.6 Analyse du taux d'acceptation des instances de type unicast et multicast

Il est important de noter que les seules requêtes de type unicast refusées par le modèle sont celles reliant deux points critiques du réseau (liens surchargés ou de faibles capacités). En supposant que le modèle essaie toutes les solutions possibles (ce qui semble être le cas puisque les temps limites de calcul fournis au solveur ne sont jamais atteints, voir 3.3), le refus d'une requête de type unicast reliant un routeur i à un routeur j implique qu'il n'existe aucun lien ou nœud de capacité suffisante entre i et j.

En revanche, une requête contenant plusieurs liens de type multicast peut être refusée à partir du moment où un seul de ses liens ne peut être alloué sur le réseau (une seule destination non atteignable). Dans les simulations effectuées, une requête de type multicast contient en moyenne 7 liens virtuels, chacun contenant en moyenne 3 destinations distinctes (ainsi qu'une source), soit 21 destinations pour un total de 28 routeurs. Si un seul de ces routeurs ne peut être « relié », alors la requête sera rejetée.

Le réseau GÉANT contient 41 routeurs, 16 d'entre eux étant reliés au cœur du réseau par des liens de capacités inférieures à 10Gbps. Bien que le choix des sources et destinations ne soit pas totalement aléatoire (voir 3.6.1.2.5), et qu'un routeur puisse être utilisé comme destination (ou source) de plusieurs liens au sein d'une même requête, la probabilité qu'un de ces routeurs isolés soit choisi n'est pas négligeable.

En supposant que tous les routeurs isolés sont connectés à un seul lien de 5 Gbps et que tous les autres routeurs soient reliés à deux liens de capacité 100 Gbps (cas pessimiste), la probabilité de choisir un routeur isolé serait 50 fois inférieure à celle de choisir un routeur « normal ». De façon simplifiée, on pourrait alors considérer que sur 50 routeurs tirés aléatoirement, un soit isolé, ce qui nous amènerait à avoir un routeur isolé toutes les deux requêtes.

En sachant que la capacité moyenne d'un lien virtuel multicast est de 700 Mbps, et en supposant le cas idéal où les routeurs isolés soient connectés par des liens de capacité 10 Gbps, sept requêtes seraient suffisantes pour saturer un de ces routeurs. En considérant que les routeurs isolés sont tirés les uns à la suite des autres (un toutes les deux requêtes), il faudrait $15 * 16 * 2 = 480$ **requêtes pour saturer l'ensemble des routeurs isolés** (sous l'hypothèse que les sessions restent actives suffisamment longtemps).

Ce chiffre de 480 requêtes peut être vu comme une borne supérieure du nombre réel de requêtes nécessaires à la saturation de tous les routeurs isolés, ce qui donne un temps de saturation d'environ 4800 unités de temps pour un taux d'arrivée $\lambda = 0.1$ et 12000 unités de temps pour un taux de 0.04. Une fois cette saturation atteinte, au minimum une requête sur deux sera refusée car contenant un de ces routeurs isolés.

3.6.5 Résultats expérimentaux sur le réseau de Campus

Les résultats obtenus avec le réseau de Campus sont assez similaires à ceux obtenus avec le réseau GÉANT. Nous présentons dans ce qui suit une synthèse.

Remarque : Force est de noter que, les courbes des instances utilisant le path-splitting, lesquelles étaient déjà assez proches avec les expérimentations portant sur le réseau de GEANT, sont presque identiques, voire visuellement non distinctes, avec le réseau de Campus (voir section 3.6.5.4). En effet, les simulations correspondantes à ces instances ont produit des résultats numériques très proches et la plupart du temps identiques. La raison est que, pour le réseau de Campus, la bande passante des liens virtuels est un nombre entier très petit, en l'occurrence choisi entre 1 et 3 Mbps (voir section 3.6.1.2.3) : ces grandeurs sont proches de la réalité liée à ce type d'environnement réseau et des besoins des applications y opérant. En faisant ce choix, les variables entières f_k^t du problème se trouveront entre 0 et 3. Le PS_{ratio} qui a pour effet d'élargir ou de restreindre l'espace de recherche de ces variables, n'a pratiquement pas d'effet. Plus concrètement, lorsque $b_k = 1$, quelle que soit la valeur de PS_{ratio} les variables seront cherchées entre 0 et 1.

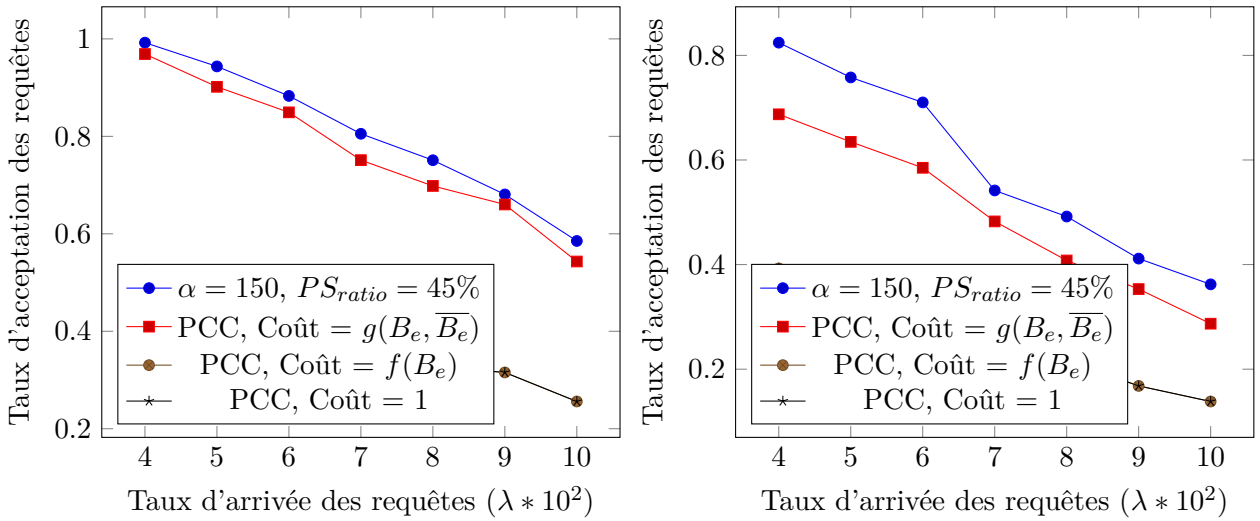
Il aurait été intéressant de convertir le modèle de simulation, du Mbps vers le Kbps pour peut-être espérer obtenir une différence notable entre les instances utilisant le path-splitting. Cependant, nous pensons que les résultats qui auraient été obtenus présenteraient les tendances semblables à celles présentées ci-après.

3.6.5.1 Comparaison avec des méthodes heuristiques

Les graphiques de la figure 3.9 présentent les résultats de comparaison de notre méthode avec les heuristiques considérées dans ce travail. Les paramètres $\alpha = 150$ et $PS_{ratio} = 45\%$ ont été choisis suite aux résultats comparatifs présentés dans la partie 3.6.5.4.

Comme pour le cas du réseau GÉANT, notre algorithme présente de meilleurs taux d'acceptation que les heuristiques, quel que soit le type d'instance considéré et le taux d'arrivée des requêtes. Le gain est plus important sur des instances de type multicast. Par rapport à l'heuristique de plus court chemin la plus performante, i.e. PCC $Co\hat{u}t = g(c_e, \overline{c_e})$, pour les instances de type multicast (resp. unicast) le gain s'élève à 5% (resp. 4%) pour des taux d'arrivée élevé, et atteint 15% (resp. 4%) pour un taux d'arrivée de 0.04. Par ailleurs, ces gains sont beaucoup plus importants par rapport aux deux autres heuristiques de PCC.

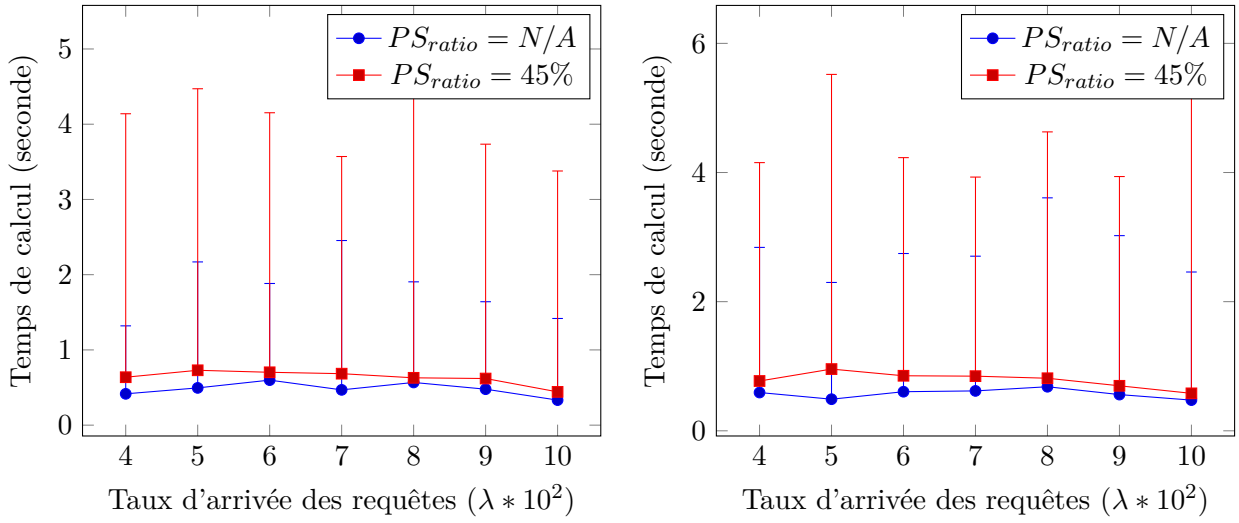
Notons que les gains produits sont inférieurs à ceux obtenus avec le réseau GÉANT. L'une des raisons est que le réseau de Campus offre une diversité de chemins moins importante que le réseau GÉANT. Même si notre algorithme dispose des moyens (recherche parmi toutes les solutions possibles, path-splitting, etc ...), dans le cas idéal, le modèle réseau doit lui offrir la possibilité d'en tirer parti. Dans le cas contraire, il serait intéressant d'utiliser une heuristique, en l'occurrence celle de PCC la plus performante.



(a) Requêtes de type Unicast

(b) Requêtes de type Multicast

FIGURE 3.9 – Comparaison de différents algorithmes d'allocation



(a) Temps de convergence - Requêtes Unicast - $\alpha = 150$

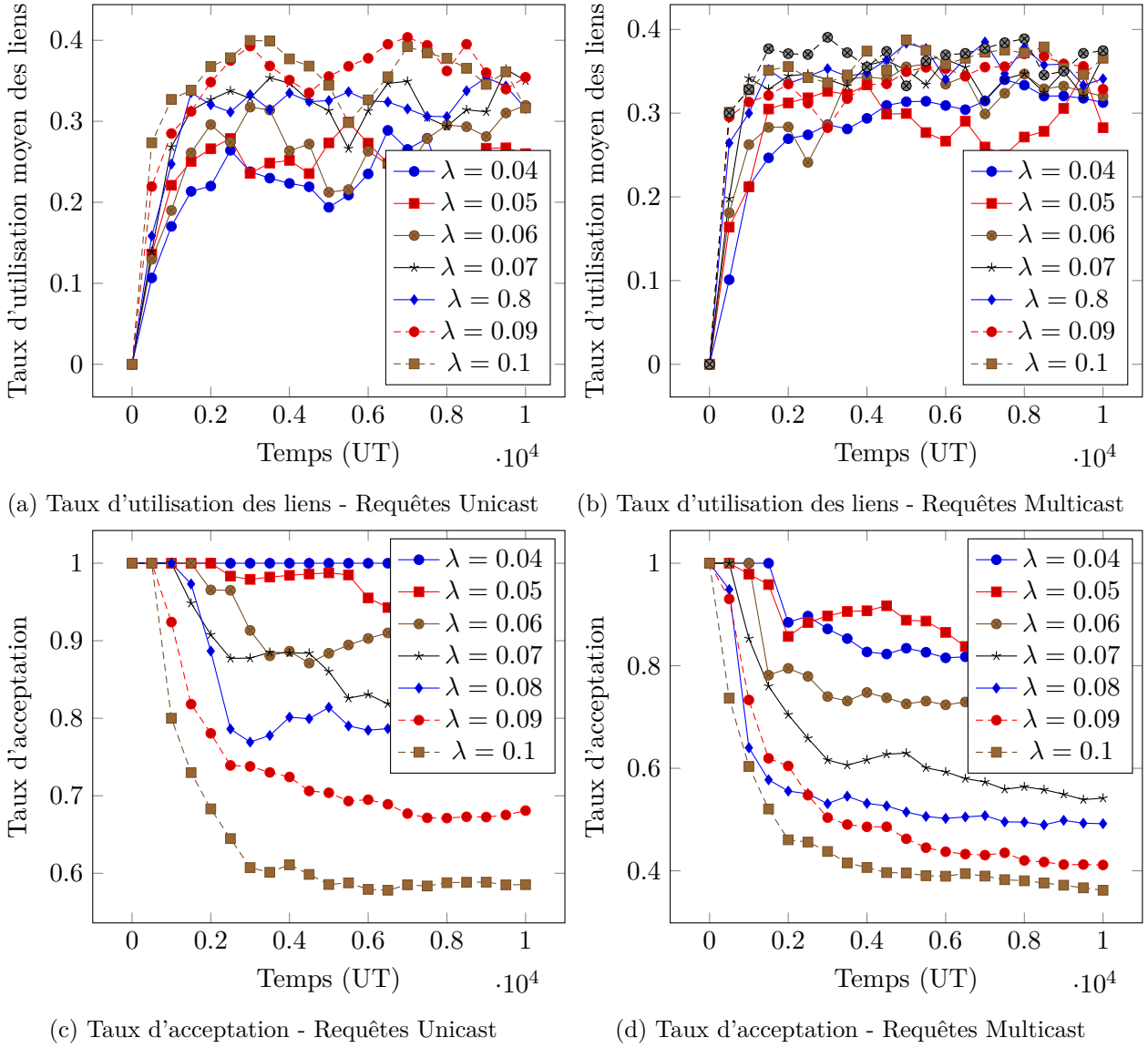
(b) Temps de convergence - Requêtes Multicast - $\alpha = 150$

FIGURE 3.10 – Influence du path splitting et du type de requête sur le temps de calcul

3.6.5.2 Caractérisation des temps de calcul

La figure 3.10 présente les temps de convergence moyen et maximal de notre algorithme, en variant le taux d'arrivée des requêtes et ce, pour les requêtes de type unicast et multicast. Ces temps demeurent inférieurs à 600 millisecondes (resp. 1 seconde) sans path-splitting, et à 1 seconde (resp. 1.5 secondes) avec path-splitting pour les requêtes de type unicast (resp. mutlicast). Ces temps de calcul sont inférieurs à ceux obtenus en utilisant le réseau GÉANT.

Les différences sont plus prononcées pour les requêtes de type multicast.



(a) Taux d'utilisation des liens - Requêtes Unicast

(b) Taux d'utilisation des liens - Requêtes Multicast

(c) Taux d'acceptation - Requêtes Unicast

(d) Taux d'acceptation - Requêtes Multicast

FIGURE 3.11 – Taux d'acceptation et d'utilisation des liens en fonction du temps pour des requêtes Multicast et Unicast -$\alpha = 150$, $PS_{ratio} = 0.45$

3.6.5.3 Validation de la durée de simulation

Les graphiques de la figure 3.11 montrent l'évolution du taux d'acceptation et du taux d'utilisation des liens du réseau au cours de la simulation. La fluctuation des courbes s'explique par l'allocation et la libération en permanence des ressources due au fait que les requêtes ont une durée de vie limitée.

À la fin de la simulation (T = 10000 unités de temps), le système a atteint un régime stationnaire. Comme pour le cas du réseau GÉANT, ce résultat nous permet de cibler les analyses suivantes en regardant uniquement les valeurs en fin de simulation.

Sur les courbes 3.11c et 3.11d, on remarque que les instances composées de requêtes de type multicast se stabilisent beaucoup plus vite (vers T = 2000 unités de temps) que les requêtes de type unicast. C'était également le cas avec le réseau GÉANT : les raisons qui y étaient évoquées sont également valables ici. À partir des courbes 3.11a et 3.11b, on peut remarquer que, le taux d'utilisation moyen des liens atteint 40%. Les expérimentations montrent que celui des nœuds est aussi élevé et avoisine également 40%. Cela montre de nouveau l'efficacité de notre méthode, quel que soit le type de requêtes et leur taux d'arrivée.

3.6.5.4 Influence des paramètres du modèle

Dans cette section, nous allons étudier l'influence respective des paramètres PS_{ratio} et α de notre modèle.

3.6.5.4.1 Taux d'acception des requêtes

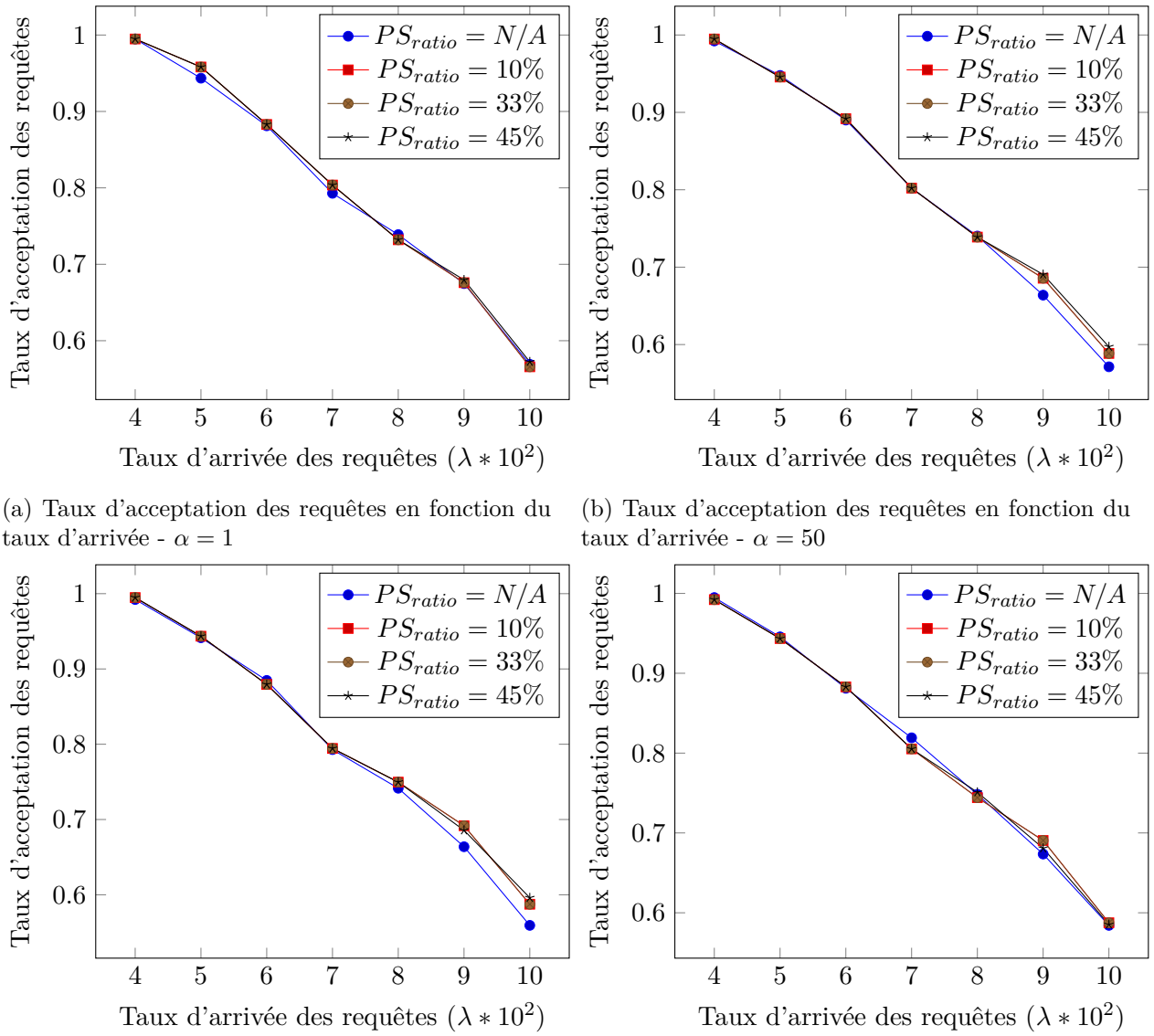
Les courbes des graphiques 3.12a à 3.12d et 3.13a à 3.13d présentent les taux d'acceptation des requêtes pour différentes instances de type unicast et multicast.

Requêtes de type unicast Sur les graphiques 3.12a à 3.12d, le path-splitting montre quelque peu un intérêt, ce qui n'était pas le cas avec le réseau GÉANT où nos études ont porté sur le paramètre β. Quel que soit α, les instances utilisant le path-splitting présentent toujours de meilleurs résultats. Elles surclassent l'instance qui n'en fait pas usage, avec un gain cependant assez faible, proche de 2% en moyenne. Par ailleurs, sur ces mêmes graphiques, le paramètre α ne semble avoir quasiment aucun impact sur les résultats, puisque toutes les courbes de la figure 3.12 indiquent des taux d'acceptation semblables, i.e lorsque α est l'unique paramètre qui varie.

Requêtes de type multicast Les courbes de la figure 3.13 montrent que lorsque le taux d'arrivée des requêtes augmente, le taux d'acception des requêtes multicast diminue plus vite que celui des requêtes unicast (voir figure 3.12), quelles que soient les instances. Contrairement aux instances de type unicast, avec les instances composées de requêtes de type multicast, l'intérêt du path-splitting est mitigé. Il est difficile de trancher sur son utilisation on non. En revanche, l'utilisation du path-splitting semble globalement la meilleure solution pour de faibles taux d'arrivée des requêtes ($\lambda < 0.06$). Tout comme pour les instances de type unicast, le paramètre α ne semble pas avoir une influence notable sur les résultats des simulations.

3.6.5.4.2 Charge du réseau

On ne considère ici que des instances avec $\alpha = 150$, les analyses précédentes ayant montré la faible influence de ce paramètre sur les résultats du modèle.



(a) Taux d'acceptation des requêtes en fonction du taux d'arrivée - $\alpha = 1$

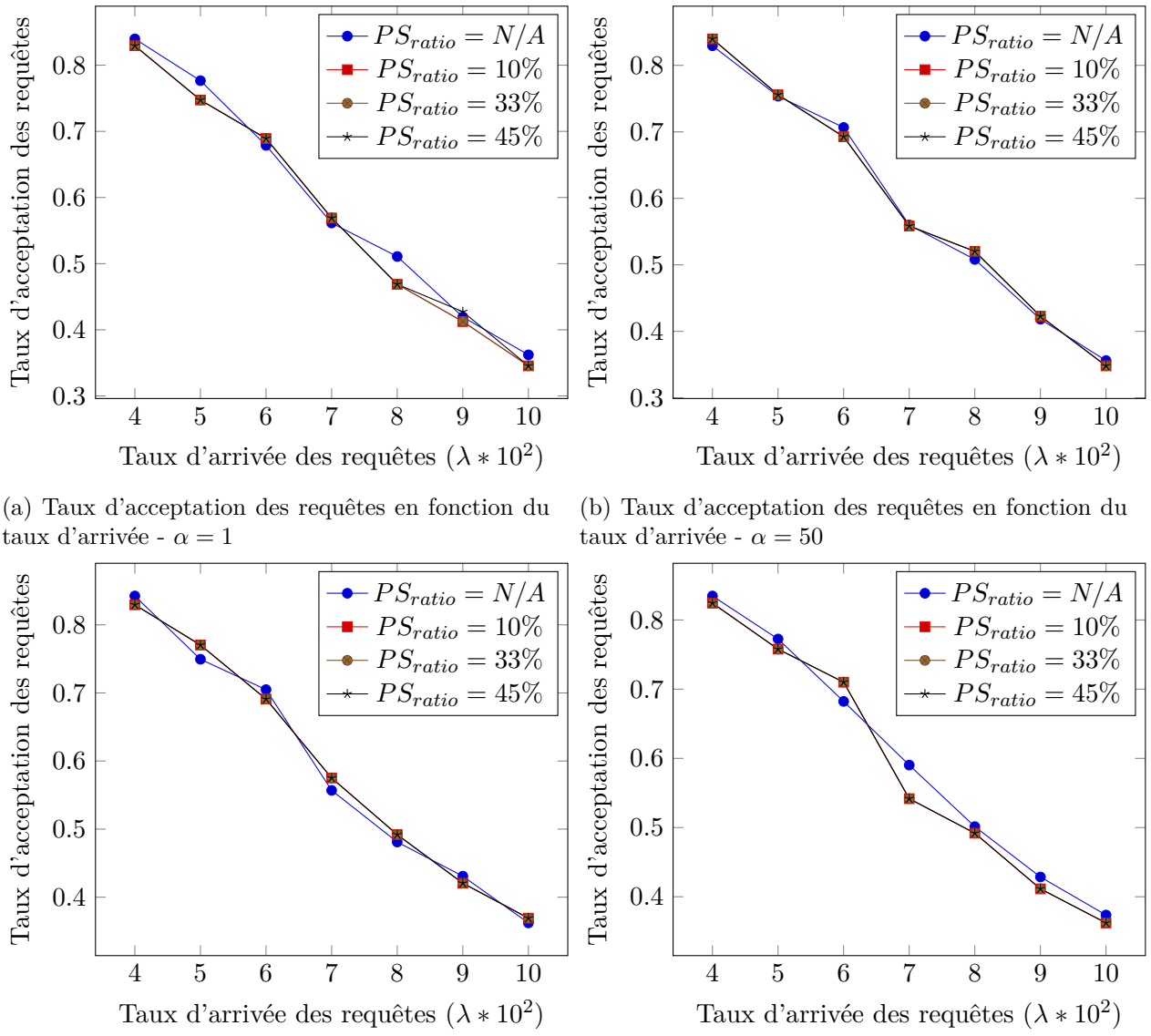
(b) Taux d'acceptation des requêtes en fonction du taux d'arrivée - $\alpha = 50$

(c) Taux d'acceptation des requêtes en fonction du taux d'arrivée - $\alpha = 100$

(d) Taux d'acceptation des requêtes en fonction du taux d'arrivée - $\alpha = 150$

FIGURE 3.12 – Taux d'acceptation des requêtes de type **unicast** en fonction de leur taux d'arrivée et des paramètres de simulation

La figure 3.14 présente la charge globale des liens et des nœuds du réseau en fonction du taux d'arrivée des requêtes, pour les instances de type unicast et multicast. Dans chaque graphique de cette figure, les courbes des instances utilisant le path-splitting sont quasiment toujours identiques. Autrement dit, la charge du réseau varie très peu avec le PS_{ratio} quel que soit le type d'instance. Ceci explique le très faible écart observé, dans les précédentes figures dont celles montrant les taux d'acceptation 3.9, entre les courbes des instances utilisant le path-splitting. Les raisons déjà évoquées dans la section 3.6.5 expliquent cette forte ressemblance. Cependant,



(a) Taux d'acceptation des requêtes en fonction du taux d'arrivée - $\alpha = 1$

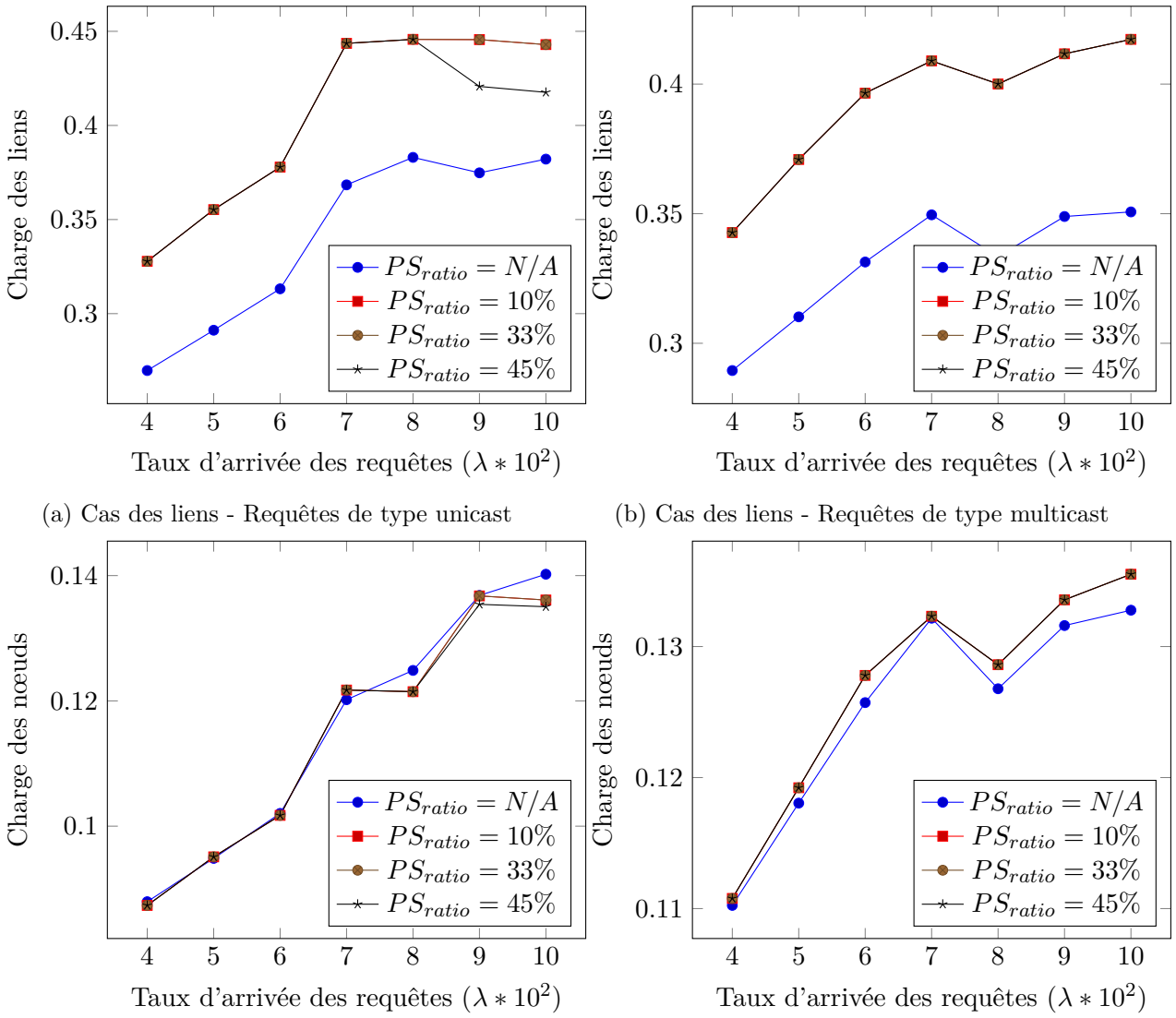
(b) Taux d'acceptation des requêtes en fonction du taux d'arrivée - $\alpha = 50$

(c) Taux d'acceptation des requêtes en fonction du taux d'arrivée - $\alpha = 100$

(d) Taux d'acceptation des requêtes en fonction du taux d'arrivée - $\alpha = 150$

FIGURE 3.13 – Taux d'acceptation des requêtes de type **multicast** en fonction de leur taux d'arrivée et des paramètres de simulation

parmi ces instances utilisant le path-splitting, celles dont la valeur du PS_{ratio} vaut 45% se distinguent en consommant parfois peu de ressources, alors qu'elles acceptent autant, voire plus de requêtes que les autres instances de même type, pour le même taux d'arrivée. Cela justifie pourquoi les paramètres $\alpha = 150$ et $PS_{ratio} = 45\%$ ont été choisis pour comparer notre méthode aux heuristiques de plus court chemin (voir figure 3.9).



(a) Cas des liens - Requêtes de type unicast

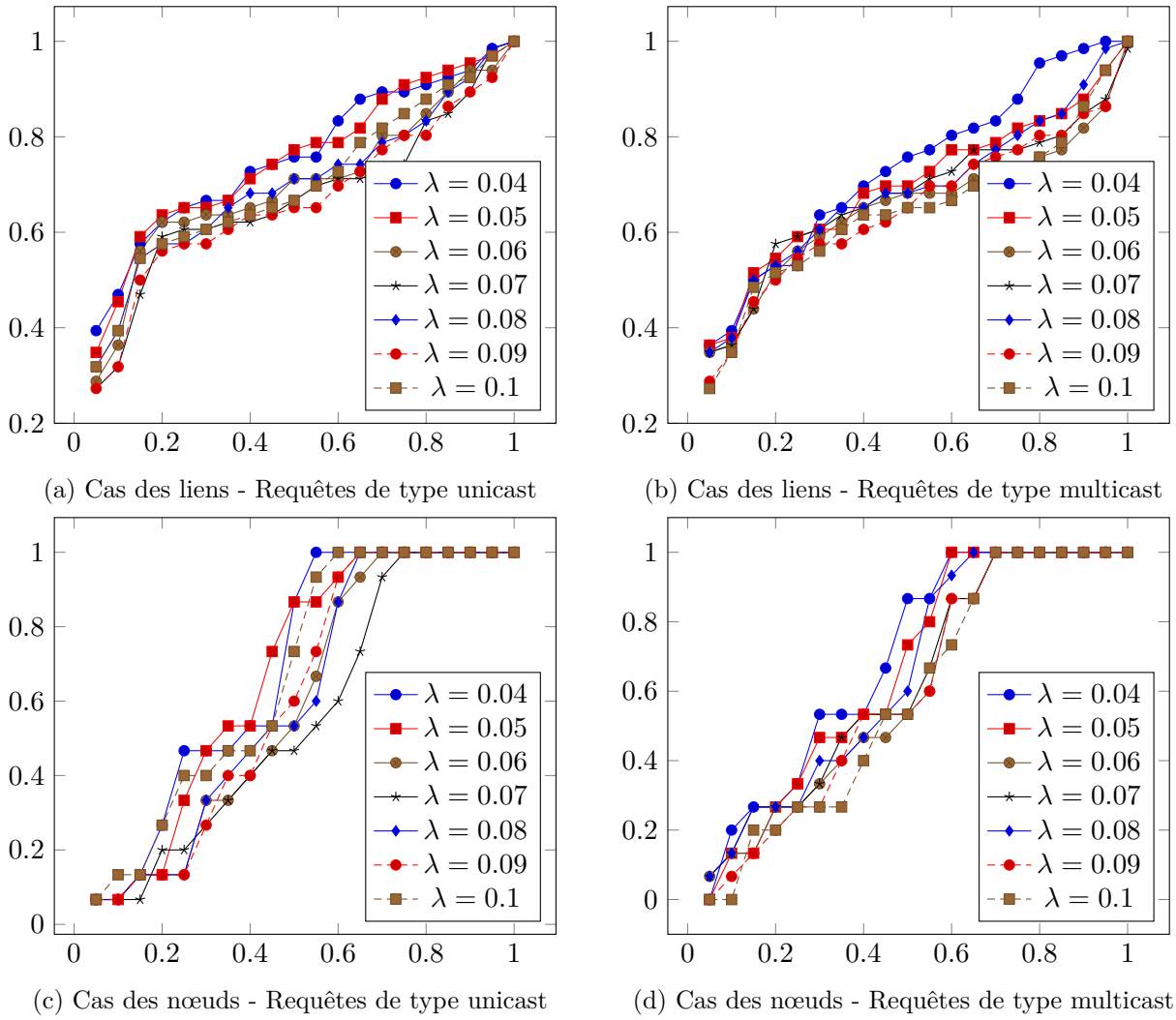
(b) Cas des liens - Requêtes de type multicast

(c) Cas des nœuds - Requêtes de type unicast

(d) Cas des nœuds - Requêtes de type multicast

FIGURE 3.14 – Charge du réseau en fonction du taux d'arrivée pour des réquêtes de type unicast et multicast - $\alpha = 150$

Contrairement à ce qui a été observé dans les figures 3.5, 3.6, et 3.7 avec le réseau GÉANT, où les instances n'utilisant pas de path-splitting présentaient des **meilleurs taux d'acceptation** alors qu'elles consommaient **peu de ressources**, avec le réseau de Campus, les instances les **plus performantes**, en l'occurrence celles utilisant le path-splitting, consomment toujours **plus de ressources** 3.14. Ceci sans aucun doute parce qu'elles acceptent beaucoup plus de requêtes comme le montrent les figures 3.12 et 3.13.



(a) Cas des liens - Requêtes de type unicast

(b) Cas des liens - Requêtes de type multicast

(c) Cas des nœuds - Requêtes de type unicast

(d) Cas des nœuds - Requêtes de type multicast

FIGURE 3.15 – Fonction de répartition du taux d'utilisation des liens et nœuds pour des réquêtes de type unicast et multicast - $\alpha = 150, PS_{ratio} = 45\%$

3.6.5.5 Visualisation de l'équilibre de l'utilisation du réseau

La figure 3.15 présente les fonctions de répartition du taux d'utilisation des liens et des nœuds du réseau (à la fin de la simulation, soit pour $T = 10000$ unités de temps) pour les instances de type unicast (3.15a et 3.15c) et multicast (3.15b et 3.15d) pour différents taux d'arrivée.

Ce qui frappe le plus dans la figure 3.15 est que, les courbes correspondantes aux fonctions de répartition du taux d'utilisation des liens (resp. des nœuds) ont la même allure quel que soit le type de requêtes et leur taux d'arrivée. Cependant, la fonction croit plus vite pour les taux d'arrivée moins élevés. À faible taux d'arrivée de requêtes, la charge du réseau est moins élevée (voir figure 3.14), ce qui explique la présence d'une fraction élevée de liens (resp. de nœuds) peu

utilisés. Cette observation est plus prononcée avec les instances de type unicast que avec celles de type multicast.

Quel que soit le taux d'arrivée des requêtes, on note la présence des liens utilisés à plus de 98% de leur capacité, ce qui explique les taux d'acceptation n'atteignant pas les 100% comme l'indique la figure 3.9 : le meilleur taux d'acceptation est environ 99.24% .

Malgré la présence des liens et des nœuds utilisés à moins de 20% de leur capacité, quel que soit le taux d'arrivée des requêtes, au moins 40% des liens (resp. 60% des nœuds) sont utilisés à plus de 40% (resp. 40%) de leur capacité. Ceci montre de nouveau l'efficacité de notre méthode. La différence de taux d'utilisation entre les liens (resp. les nœuds) du réseau est inévitable dans un contexte dynamique où les réseaux virtuels arrivent et quittent l'infrastructure au cours du temps. Toute fois une telle disparité n'est pas favorable à l'admissibilité des requêtes.

3.7 Conclusion

Nous avons présenté dans ce chapitre, l'algorithme d'allocation de ressources que nous proposons pour le provisionnement des réseaux virtuels applicatifs sur une infrastructure SDN/OpenFlow.

Cet algorithme présente plusieurs spécificités par rapport aux travaux de la littérature, dont : la prise en compte (1) des liens point-à-multipoints, (2) de plusieurs critères de QoS, et (3) des ressources de commutation dans le processus d'allocation. Il s'agit d'une méthode exacte formulée en utilisant la programmation linéaire en nombres entiers. Les allocations sont actuellement calculées sans prendre en compte la possibilité de migrer les liens virtuels déjà alloués, ce qui aurait favorisé l'admissibilité des requêtes. Ce point sera abordé à l'intérieur du prochain chapitre.

Les expérimentations réalisées ont porté sur la topologie réseau GÉANT ainsi que sur une topologie réseau de Campus. Les évaluations se sont focalisées sur des conditions aux limites (avec une forte surcharge), pessimistes par rapport à la réalité. Elles ont permis de vérifier la supériorité de notre algorithme par rapport à certaines heuristiques. Elles ont également permis de vérifier que les temps de convergence étaient acceptables, sauf pour quelques situations plutôt rares. Il serait utile de mieux caractériser ces situations pour appliquer d'autres algorithmes basés sur des heuristiques que l'on développerait en prenant en compte les contraintes liées aux infrastructures réseau de type SDN/OpenFlow. Elles ont aussi permis d'étudier l'influence des paramètres de l'algorithme en quantifiant leur apport respectif.

Chapitre 4

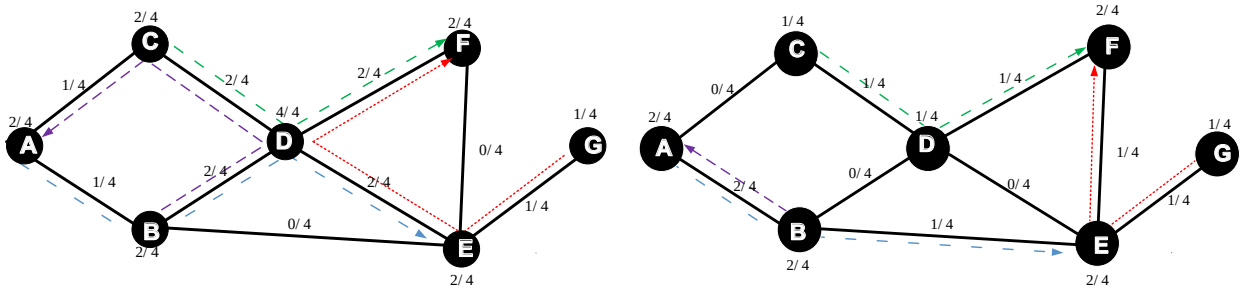
Mécanismes de défragmentation de ressources

Sommaire

4.1 Description contextuelle du problème 96
4.2 Problème de fragmentation de ressources dans un contexte de virtualisation réseau 97
4.3 État de l'art de la littérature existante 100
4.4 Définition du problème 105
4.5 Mécanisme de défragmentation proactif 106
4.6 Mécanisme de défragmentation réactif 111
4.7 Évaluation expérimentale 113
4.8 Conclusion 123

L'une des spécificités du système de communication ADN est de fournir à chaque application un service réseau sur mesure. De ce fait, les ressources réseaux mobilisées pour instancier chaque service réseau seront taillées aux exigences courantes de l'application au moment de la demande d'établissement du service. Malgré l'intérêt de cette mesure, le caractère dynamique des applications a une influence sur l'état des ressources disponibles. En effet, l'allocation et la libération des ressources, dues respectivement à l'arrivée et au départ des applications ou de certains de leurs flux de données, peuvent conduire l'infrastructure réseau dans un état non optimal, dans la mesure où certains liens et nœuds physiques sont surchargés tandis que d'autres sont sous-utilisés. Cette disparité des charges, autrement vue comme une fragmentation des ressources disponibles, impacte l'admissibilité de nouveaux flux et la satisfaction de nouvelles exigences plus contraignantes des flux en cours de transit sur le réseau, quand bien même ces ressources disponibles sont globalement suffisantes. Il est clair que, migrer les services réseau en cours d'exploitation, en recalculant les chemins de données ainsi que les ressources à réserver le long de ces chemins supports des flux applicatifs, permettra de mieux équilibrer la charge des nœuds et des liens de l'infrastructure. C'est entre autres là où le composant *gestionnaire autonomique* entre en jeu : en l'occurrence, il doit surveiller le réseau pour déceler une situation de fragmentation, l'analyser, prendre des décisions et les appliquer pour parer au problème. C'est la propriété d'auto-optimisation des ressources, laquelle permet au système ADN de s'adapter de manière autonome afin d'accroître sa capacité à satisfaire les applications.

L'objectif de ce chapitre est de présenter les deux solutions que nous proposons pour adresser ce problème. La suite du chapitre est structurée comme suit : la première section contextualise le



(a) État original (b) État adapté

FIGURE 4.1 – Illustration

problème, la section d'après présente une définition plus générale du problème dans un contexte de virtualisation de réseau et la section suivante aborde l'état de l'art des travaux traitant ce problème et existant dans la littérature scientifique. Nos contributions sont développées dans les deux sections suivantes et évaluées dans la section 4.7. La dernière section clôture ce chapitre par une conclusion.

4.1 Description contextuelle du problème

Pour rappel, le service réseau ADN correspond à un réseau virtuel constitué d'un ensemble de liens virtuels de bout-en-bout, qui peuvent être de type point-à-point ou point-à-multipoint et chacun caractérisé par un débit et un délai de transfert maximal. Ce service est provisionnée pour le compte d' une nouvelle application à son arrivée sur le réseau. A l'arrivée d'une application, une requête de création d'un réseau virtuel dont les caractéristiques de ses liens virtuels correspondent aux exigences des flux de données échangés entre les composants de cette application, est émise. L'algorithme d'allocation de ressources est mis en œuvre lors du traitement de cette requête pour calculer les chemins de données et les ressources physiques (bande passante des liens et les entrées des règles de la table de flux des nœuds d'acheminement) à allouer le long de ces chemins de données pour chaque lien virtuel inclus dans la requête. Les requêtes ne sont pas connues (ni leur date d'arrivée prévue, ni leur durée de vie sur le réseau ainsi que les caractéristiques de leurs liens virtuels) à l'avance et sont traitées de manière séquentielle (l'une après l'autre) : il s'agit d'un scénario *à la demande*[1] qui s'approche le mieux du contexte que nous ciblons, où les applications peuvent arriver et quitter le réseau à tout le moment. Ainsi, même si l'algorithme d'allocation retourne une solution optimale, elle l'est temporairement, puisque le départ d'une application remettra en question l'optimalité des allocations ayant été effectuées durant son existence. La raison est que la libération des ressources, suite au départ de l'application, modifiera les conditions réseau par rapport à celles qui avaient été considérées au moment du calcul de ces allocations. La sous-optimalité de certaines allocations se manifeste donc par la présence de la fragmentation.

1. en opposition aux scénarios où toutes les requêtes sont connues à l'avance

Pour illustrer ce problème, considérons la figure 4.1 qui présente deux graphiques décrivant l'état d'une infrastructure réseau avant (graphique 4.1a désigné comme *état original*) et après (graphique 4.1b désigné comme *état adapté*) élimination de la fragmentation. Dans ces deux graphiques, sur chaque lien et nœud de l'infrastructure réseau considérée, est indiqué leur taux d'utilisation : c'est-à-dire la quantité de ressource utilisée par rapport à sa capacité. Les valeurs indiquées sont obtenues sous l'hypothèse que la capacité maximale des nœuds et des liens est égale à 4. Aussi, on suppose ici qu'un lien virtuel (représenté en trait pointillé) consomme 1 unité de bande passante et 1 entrée dans la table de flux des nœuds qu'il traverse. Sur le graphique 4.1a, on peut remarquer que, pendant que 2 liens $\{(B,E)$ et $(E,F)\}$ sont non utilisés, 4 $\{(B,D),(C,D),(D,F),(D,E))\}$ sont utilisés à 50% de leur capacité, saturant en même tant le nœud D. Après application de la défragmentation, l'infrastructure passe à l'état adapté où seul un lien est utilisé à 50% de sa capacité et la charge maximale des nœuds est réduite à 50%. Aussi la charge globale du réseau a considérablement diminué : de $\frac{15}{28} \approx 53.57\%$ à $\frac{11}{28} \approx 39.28\%$ pour les nœuds et de $\frac{11}{36} \approx 30.55\%$ à $\frac{7}{36} \approx 19.44\%$ pour les liens. Une requête de création d'un lien virtuel exigeant 3 unités de bande passante entre les nœuds A et D peut dorénavant être admise.

4.2 Problème de fragmentation de ressources dans un contexte de virtualisation réseau

La situation de fragmentation de ressources d'une infrastructure réseau est fréquemment observée dans les environnements où le substrat est virtualisé pour partager ses ressources physiques entre plusieurs réseaux virtuels, dans un cadre généralement contractuel. C'est le cas chez les fournisseurs de Cloud [Schaffrath 2012] où les ressources sont consommées à la demande des utilisateurs via un portail numérique leur permettant de commander des machines virtuelles et de les interconnecter à travers un réseau virtuel personnalisé qu'ils administrent, opèrent et exploitent en toute liberté. De même, dans les environnements de fournisseurs d'infrastructure réseau, tels que ceux des fournisseurs d'accès à internet, les ressources destinées entre autres aux fournisseurs de service (notamment des fournisseurs de contenus), sont également exposées à la fragmentation [Mijumbi 2014]. Notons que, en pratique, dans ces environnements, une nouvelle catégorie de produits est très prisée pour partitionner dynamiquement et automatiquement les infrastructures bâties selon l'approche SDN : il s'agit des plate-formes de virtualisation réseau ou hyperviseurs réseau [Blenk 2016]. Dans ce contexte de virtualisation réseau, pendant que, les algorithmes d'allocation de ressources calculent les ressources physiques à allouer aux réseaux virtuels en identifiant le placement sur le réseau physique de leurs entités virtuelles pour les héberger de manière optimale, les mécanismes de défragmentation quant à eux, ont pour but de réoptimiser les ressources. Ces derniers doivent déterminer le moment opportun pour délocaliser/migrer les réseaux virtuels hébergés pour une utilisation efficace des ressources. Leur finalité principale est de maintenir le réseau dans un état favorable à l'admissibilité de nouveaux réseaux virtuels en adaptant ceux en phase opérationnelle aux conditions réseau qui changent au cours du temps.

Le défi se décompose en deux points :

- **à quel moment le processus de migration doit-il être enclenché ?**
- **la migration (sélection et réallocation et reconfiguration)** concernera quels réseaux virtuels (sélection), quel nouveau placement sera affecté à chacun d'eux et quelle quantité de ressource leur sera allouée en tenant compte de leurs exigences (réallocation) ? Tout ceci sur une durée raisonnable et surtout, il faudra tenir compte du fait que la reconfiguration des réseaux virtuels réalloués, induit un sur-débit et peut provoquer une interruption des services qu'ils supportent.

Au delà de la question (qui sera traitée dans la section suivante) sur le moment auquel le processus de migration doit être enclenché, comme vu précédemment, au cœur de tout mécanisme de défragmentation, se trouve le problème de migration qui se décompose lui-même en trois sous-problèmes : la sélection et la réallocation des réseaux virtuels, suivies par leur reconfiguration effective. Pour des raisons de modularité, cette dernière préoccupation est prise en charge par le composant *VNET Deployer* de l'application de contrôle réseau ADN. Elle ne sera donc pas abordée dans ce chapitre. En revanche, le sur-débit induit par la reconfiguration sera considéré dans la formulation des deux autres sous-problèmes, sur lesquels la suite de cette section sera focalisée. Il est à noter que ces deux sous-problèmes se groupent en un problème d'optimisation combinatoire qui consiste à trouver parmi les réseaux virtuels hébergés au moment du déclenchement du processus de migration, ceux à déplacer et leurs nouvelles allocations permettant de conduire l'infrastructure dans un état favorable à l'admissibilité des réseaux virtuels à venirs.

4.2.1 Sous-problème de sélection et réallocation des réseaux virtuels

Connaissant chaque réseau virtuel en cours d'hébergement et de son placement courant ou emplacement d'origine, ainsi que l'infrastructure réseau sous-adjacente, on cherche une fonction de déplacement ou de placement adapté notée M_a qui consiste à déplacer (entièrement ou partiellement) chaque réseau virtuel $R_V = (N^V, L^V)$ de son emplacement d'origine noté $M_o(R_V)$, vers un emplacement adapté noté $M_a(R_V)$ sur une infrastructure $R_P = (N^P, L^P)$ tel que :

- chaque routeur (nœud) virtuel $n^v \in N^V$ qui était à l'origine placé sur $M_o(n^v) = n_o^p \in N^P$, soit réalloué sur un routeur physique adapté $M_a(n^v) = n_a^p \in N^P$. Notons qu'un nœud virtuel ne peut être distribué sur plusieurs nœuds physiques. Aussi, un nœud physique ne peut héberger qu'un seul nœud virtuel parmi ceux appartenant à un même réseau virtuel.
- chaque lien virtuel $l^v \in L^V$ reliant deux nœuds virtuels n_1^v, $n_2^v \in N^V$, qui était à l'origine placé sur un chemin physique $M_o(l^v) = (l_{o_1}^p, l_{o_2}^p, ..., l_{o_k}^p)$ reliant $M_o(n_1^v)$ et $M_o(n_2^v)$, soit déplacé sur un chemin physique adapté $M_a(l^v) = (l_{a_1}^p, l_{a_2}^p, ..., l_{a_i}^p)$ avec $l_{a_i}^p$, $l_{o_k}^p \in L^P$. Un lien virtuel peut avoir un seul (lien de type unicast) ou plusieurs (lien de type multicast) nœuds destination, et peut être déplacé sur plusieurs chemins physiques (utilisation du path-splitting).

Dans ce sous-problème, la fonction de placement d'origine M_o est réalisée par l'algorithme d'allocation de ressources, tandis que la fonction de placement adapté M_a est à la charge du mécanisme de défragmentation lequel est hébergé, dans notre cas, au sein du

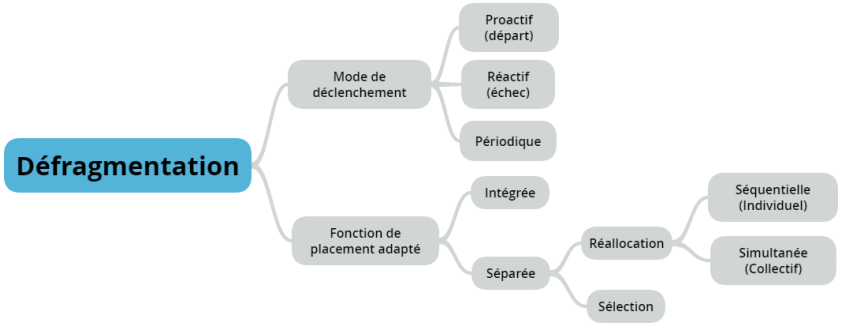


FIGURE 4.2 – Taxonomie des approches de défragmentation de ressources

gestionnaire autonomique. Si le résultat retourné par la fonction de placement adapté est différent (totalement ou partiellement) du placement d'origine, le réseau virtuel sera alors sujet à la reconfiguration.

4.2.2 Les différents aspects du problème de fragmentation de ressources

L'élaboration d'une fonction de placement adapté démarre sur la base d'un certain nombre de facteurs à considérer au regard des aspects liés à ce problème. Parmi les principaux aspects on distingue : l'infrastructure réseau sous-adjacente, les requêtes de réseaux virtuels et l'objectif de la fonction de placement adapté. Les facteurs liés à l'infrastructure réseau et aux requêtes sont les mêmes que ceux présentés dans le chapitre 3. L'objectif de la fonction de placement adapté est présenté ci-après.

4.2.2.1 L'objectif de la fonction de placement adapté

La finalité d'un mécanisme de défragmentation qui est de conduire l'infrastructure réseau dans un état favorable à l'admissibilité des requêtes, cependant la manière de définir la notion d'*état favorable* est très discutée. Cette définition est déterminante, car elle est communiquée comme objectif à la fonction de placement adapté M_a, lui donnant ainsi le but qu'elle cherchera à atteindre. Les travaux traitant ce problème se distinguent les uns des autres, aussi par le choix effectué sur l'objectif. Certains travaux proposent de réduire le nombre d'entités (liens et nœuds) physiques congestionnées [Fajjari 2011], d'autres optent pour un objectif assez similaire visant à minimiser le taux d'utilisation maximal des entités physiques [Yu 2008]. Une autre catégorie de travaux [Mijumbi 2014][Wanis 2013] propose de réduire la congestion tout en consommant les ressources de manière modérée, car ces deux critères favorisent autant l'admissibilité des requêtes. Si la traduction formelle de la notion d'*état favorable* est très contrastée, force est de constater que les auteurs se rejoignent sur la nécessité de minimiser le coût induit par les reconfigurations qui vont suivre. En effet, les reconfigurations induisent un sur-débit lorsqu'elles sont automatisées ou un sur-coût opérationnel lorsqu'elles requièrent l'intervention d'un opérateur humain et peuvent aussi provoquer l'interruption de services applicatifs supportés par les réseaux virtuels reconfigurés. Cette interruption de service est dans certains contextes sujet à pénalités.

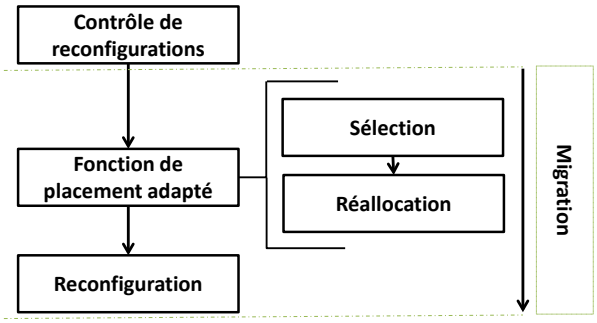


FIGURE 4.3 – Structure générale d'un mécanisme de défragmentation de ressources

4.3 État de l'art de la littérature existante

4.3.1 Taxonomie des approches de défragmentation de ressource

Les approches de défragmentation de ressources peuvent être classées en trois grandes familles selon le moment jugé opportun pour déclencher l'exécution du mécanisme. La figure 4.2 présente une taxonomie des approches de défragmentation de ressources. On distingue les familles : ***proactive***, ***réactive*** et ***périodique***. Les approches de la famille proactive déclenchent le mécanisme lorsqu' un réseau virtuel quitte l'infrastructure, libérant ainsi les ressources qui lui étaient allouées. L'idée est d'anticiper sur une potentielle fragmentation et d'agir le cas échéant de manière sereine. Les approches appartenant à la famille réactive lancent l'opération de défragmentation suite à l'échec du traitement d'une demande d'allocation de ressources pour un réseau virtuel. L'idée de réagir juste à temps, permet sur la base du revenu qu'engendrera le réseau virtuel en échec, de décider d'appliquer (si un gain est réalisé) ou non les reconfigurations en fonction du coût (pénalités, etc ...) qu'elles pourront induire. La dernière famille quant à elle déclenche le mécanisme périodiquement.

Une fois le mécanisme déclenché, la ***fonction de placement adapté*** est mise en œuvre pour sélectionner et réallouer les réseaux virtuels déjà embarqués. Elle devra alors résoudre un problème d'allocation de ressources plus libre en ce sens où tous les réseaux virtuels actifs peuvent être réalloués. Bien que cette liberté est avantageuse pour une adaptation optimale des réseaux virtuels, elle rend la recherche plus coûteuse en terme de complexité temporelle. Il faudra alors envisager les solutions approximatives pour couvrir les usages réalistes, lesquels requièrent un coût de calcul modéré. Pour cela, deux choix de conception s'imposent dans la littérature scientifique et se distinguent selon le rapport entre les processus de sélection et réallocation : l'un où la fonction de placement adapté est construite selon une démarche *intégrée* et l'autre démarche dite *séparée*.

1. ***Intégrée*** : Dans ce cas, les processus de sélection et de réallocation sont coordonnés. Autrement dit, la fonction de placement adaptée cherche en même temps les réseaux virtuels à déplacer et les ressources à leur attribuer pour approcher l'objectif visé. Pour la rendre réaliste face aux instances de ce type de problème rencontrées dans la pratique, une étape de présélection de toute ou une partie de la topologie des réseaux virtuels

(que l'on désignera dans la suite de cette section par *composant virtuel*) à inclure dans le processus est prévue en amont. Cette présélection restreint l'espace de recherche des solutions réalisables et contribue par conséquent à accélérer la résolution. La résolution est d'autant plus rapide que le nombre et la structure des composants virtuels à migrer est respectivement peu élevé et moins complexe. En revanche la qualité de la solution est d'autant plus dégradée. Ce choix de conception est généralement effectué par les approches réactives, lesquelles cherchent à embarquer le réseau virtuel en échec en considérant la possibilité de réallouer les composants virtuels présélectionnés ou candidats à la migration, avec comme objectif de maximiser le gain à réaliser et de réoptimiser les ressources. Cette mécanique est similaire à celle des algorithmes d'allocation de ressource dits *dynamiques* [Fischer 2013].

2. ***Séparée*** : Dans ce cas, les processus de sélection et de réallocation sont séparés et donc exécutés l'un après l'autre respectivement. En séparant la sélection et la réallocation en deux problèmes indépendants, les solutions de défragmentation ayant effectuées ce choix de conception, peuvent tirer parti des algorithmes d'allocation de ressource dits *statiques* ayant fait l'objet de plusieurs travaux antérieurs, pour réaliser la réallocation. Il ne reste plus qu'à se concentrer sur le problème de sélection. Toutefois, il est à noter que certains travaux proposent leur propre algorithme de réallocation, lorsqu'aucun algorithme ne couvre en totalité leurs exigences. On distingue les approches qui cherchent à réallouer le collectif de composants virtuels candidats à la migration de manière simultanée, de ceux qui optent pour une réallocation individuelle de manière séquentielle.
   (a) Réallocation ***Simultanée*** : Dans ce cas, les composants virtuels sont réalloués simultanément pour une meilleure utilisation des ressources. En revanche, la recherche des nouvelles allocations est plus lourde, rendant ce choix plus adapté aux approches proactives et périodiques.
   (b) Réallocation ***Séquentielle*** : Dans ce cas, les composants virtuels sont réalloués les uns après les autres. Ce choix est généralement effectué par les approches réactives qui cherchent à rapidement embarquer le réseau virtuel en échec. Ces approches réactives sont généralement élaborées suivant une méthode itérative, où à chaque itération, un composant virtuel est réalloué, suivie d'une tentative d'allocation du réseau virtuel.

4.3.2 Synthèse des travaux existants

Les mécanismes proposés dans les travaux existants peuvent être structurés en quatre étages successifs, chacun jouant un rôle prépondérant dans l'efficacité et les performances de la solution. On distingue l'étape de contrôle des reconfigurations, l'étape de sélection des composants virtuels candidats à la migration, l'étape de réallocation des composants virtuels sélectionnés et l'étape de reconfiguration. Comme mentionné dans la section 4.2, l'étape de reconfiguration ne sera pas abordée dans ce chapitre. La figure 4.3 résume la structure générale de ces mécanismes.

4.3.2.1 Contrôle des reconfigurations

La finalité de cet étage est d'atténuer à long terme la quantité de reconfigurations et d'éviter une éventuelle instabilité du réseau. Le coût des reconfigurations inclut le temps lié aux opéra-

tions de reconfiguration (suppression des configurations existantes, génération et installation de nouvelles) lesquelles sont fastidieuses et sujettes à erreurs. De plus, il est clair que la reconfiguration des réseaux virtuels pendant qu'ils sont en phase opérationnelle peut impacter sur leur niveau de disponibilité. Afin de réduire les coûts opérationnels, il a été communément admis qu'il faudra minimiser à long terme la quantité de reconfiguration.

Avec l'avènement des réseaux programmables de type SDN, les opérations de reconfiguration (la génération et l'installation des règles SDN au niveau des commutateurs) sont déléguées à des programmes automatisés (Dans le cadre de notre travail, il s'agit du *VNETDeployer*) qui les appliquent de manière automatique et dynamique, pour une mise à jour plus fiable et rapide. Cette programmabilité introduite par SDN, permet de réduire significativement les coûts opérationnels. De plus, des travaux récents [Liu 2013] [Jin 2014] ont été menés dans le contexte du SDN, pour le support des mises à jour transparentes, permettant ainsi d'éviter les interruptions de service lors du processus de reconfiguration des commutateurs. Dans ce contexte, on peut donc soustraire au coût des reconfigurations, les coûts opérationnels et les pénalités liées à l'interruption de service. En revanche, il faudra dorénavant prendre en compte le sur-débit induit par les règles SDN envoyées depuis le contrôleur aux commutateurs.

A cet étage, le défi est de déterminer, une fois le mécanisme de défragmentation déclenché, à quelles conditions il est pertinent d'enclencher le processus de migration afin de minimiser la quantité de reconfigurations. Pendant que certains travaux ont fait le choix d'enclencher systématiquement la migration [Tran 2013], [Tran 2012], [Estrada 2010] [Fajjari 2011] [Mijumbi 2014], d'autres [Wanis 2013] [Farooq Butt 2010] [Zhou 2010] [Zhu 2006] proposent de l'enclencher lorsqu'au moins une entité réseau présente un taux d'utilisation supérieur à un seuil de congestion prédéfini.

4.3.2.2 Sélection des composants virtuels candidats

En connaissant la structure des composants virtuels candidats à la migration ainsi que leur nombre, à cet étage, l'objectif est de (pré)sélectionner ceux à inclure dans le processus de (sélection et de) réallocation. Il est à noter que plus la structure est complexe (exemple la topologie complète des réseaux virtuels) et plus le nombre de composants virtuels est élevé, meilleure sera la qualité du résultat de l'adaptation au regard de l'utilisation des ressources. En revanche le coût de migration sera important. Il est donc indispensable de bien traiter ce compromis selon les cas d'application. Les auteurs des travaux [Zhou 2010] [He 2008] [Zhu 2006] ont choisi de migrer la topologie complète des réseaux virtuels à adapter. D'autres ont opté pour la migration d'une partie de la topologie des réseaux virtuels. Il s'agit en l'occurrence des nœuds virtuels et de leurs liens adjacents [Wanis 2013] [Fajjari 2011] ou simplement des liens virtuels, les nœuds étant considérés dans ce cas fixes [Mijumbi 2014] [Yu 2008].

Au delà du choix de la structure des composants virtuels et de leur nombre, le principal défi à cet étage est d'identifier les composants virtuels dont le placement courant est moins adapté à l'objectif visé. Autrement, dit il s'agit de déterminer les composants virtuels à l'origine de la fragmentation, d'établir leur niveau de responsabilité afin de sélectionner les plus critiques.

Pour cela, une étape importante consiste à analyser l'état des liens et des nœuds physiques de l'infrastructure sous-jacente en se focalisant sur les ressources fragmentées ciblées par l'opération de défragmentation (Il peut s'agir par exemple de la bande passante des liens ou du CPU des nœuds). Il faudra par la suite établir la correspondance entre chacune de ces ressources et les entités (liens et nœuds) virtuelles consommant ce type ressource (par exemple la bande passante est allouée aux liens virtuels).

4.3.2.3 Réallocation des composants virtuels sélectionnés

La performance de toute solution au problème de fragmentation dépend en grande partie de l'algorithme de réallocation. A cette étape, les composants virtuels candidats à la migration sont réalloués. Leur réallocation est soit simultanée [Tran 2012] [Yu 2008], ou séquentielle [Mijumbi 2014] [Wanis 2013] [Zhu 2006]. Comme mentionné à la section 4.3.1, la réallocation et la sélection sont exécutées soit de manière coordonnée ou alors de manière indépendante. Dans le premier cas de figure, la fonction de placement adapté est construite selon une démarche *intégrée*. Dans le second cas de figure, la démarche de construction de la fonction de placement adapté est dite *séparée*.

Certains travaux proposent à cet étage un algorithme ad hoc pour répondre à leur besoin spécifique et d'autres choisissent de prendre un algorithme générique. Parmi ceux proposant un algorithme ad hoc, on a identifié certains qui requièrent des informations spécifiques de la part de l'algorithme d'allocation initiale de ressources. Il s'agit des informations qui ne sont pas communément fournies par les algorithmes d'allocation de ressources. Tout mécanisme de cette catégorie est qualifié comme *couplé* à l'algorithme d'allocation de ressource. Il s'agit généralement des mécanismes basés sur une approche réactive, à l'exemple de celui proposé par Farooq et al. dans [Farooq Butt 2010] qui requiert des informations sur les nœuds et les liens physiques à l'origine de l'échec du traitement de la requête d'allocation d'un réseau virtuel. Tout mécanisme n'appartenant pas à cette catégorie est qualifié comme *portable* en ce sens qu'il peut s'interfacer avec n'importe quel algorithme classique d'allocation de ressource.

4.3.2.4 Synthèse

Le tableau 4.1 fournit un récapitulatif des différents aspects de chaque solution de défragmentation proposée dans les travaux précédemment cités. Au meilleur de notre connaissance, seule l'approche proactive adoptée par Mujimbi et al. dans [Mijumbi 2014] considère, en plus de la bande passante, les entrées des tables de flux comme ressources pouvant être potentiellement fragmentées dû à l'arrivée et au départ des liens virtuels. Les autres travaux nécessitent des évolutions pour être adaptés aux infrastructures réseaux de type SDN. Cependant, la solution proposée par ces auteurs nous semble présenter quelques limites :

1. Elle est permissive aux reconfigurations, car l'étape de migration est systématiquement déclenchée. Nous pensons que, même s'il est judicieux d'appliquer systématiquement les reconfigurations afin d'éliminer en permanence la fragmentation et de maintenir continuellement l'infrastructure réseau dans un état favorable à l'admissibilité des requêtes, il est clair que les reconfigurations peuvent dans certaines situations être sans intérêt,

	Famille	Condition de migration	Composant virtuel à migrer	Ressource fragmentée	Réallocation + Sélection
[Mijumbi 2014]	Proactive	N/A	Liens	Entrées table de flux Bande passante	Séparée Portable Séquentielle
[Wanis 2013]	Proactive	Congestion des liens	Nœuds et ses liens adjacents	Bande passante	Séparée Portable Séquentielle
[Tran 2013]	Réactive	N/A	Nœuds et liens	Bande passante	Intégrée Portable Simultanée
[Fajjari 2011]	Réactive	N/A	Nœuds et ses liens adjacents	Bande passante	Séparée Portable Séquentielle
[Zhou 2010]	Périodique	Congestion des nœuds Congestion des liens	Topologie entière	Bande Passante CPU	Séparée Portable Séquentielle
[He 2008]	Périodique	Congestion des nœuds Congestion des liens	Topologie entière	Bande Passante CPU	Séparée Portable Séquentielle
[Yu 2008]	Périodique	N/A	Liens	Bande Passante	Intégrée Portable Simultanée
[Zhu 2006]	Périodique	Congestion des nœuds Congestion des liens	Topologie entière	Bande Passante	Séparée Portable Séquentielle
Notre solution [Tegueu 2017b]	Proactive	Congestion des liens Congestion des nœuds Niveau de fragmentation	Liens	Entrées table de flux Bande Passante	Séparée Portable Simultanée
Notre solution [Tegueu 2017a]	Réactive	N/A	Liens	Entrées table de flux Bande Passante	Intégrée Portable Simultanée

Légende : N/A = Non Appliqué, CPU = Unité Centrale de Traitement

TABLE 4.1 – Classification des approches de défragmentation de ressource

notamment lorsque le réseau est peu chargé. Il serait alors intéressant de proposer une approche plus rigoureuse au regard des reconfigurations, en les appliquant que dans les situations pertinentes qu'il faudra déterminer.

2. La méthode utilisée pour sélectionner les liens virtuels candidats à la migration, bien qu'étant intuitive, nous semble néanmoins être perfectible. Le principe est de défragmenter les ressources de bande passante et les entrées des tables de flux en déplaçant les liens virtuels consommant le plus de ressources sur les chemins physiques à faible capacité résiduelle. Nous rejoignons les auteurs sur le principe, mais nous émettons quelques réserves sur la démarche adoptée. L'inconvénient de cette démarche réside principalement sur le fait que, dans certains scénarios, les liens virtuels avec une forte demande de bande passante domineront la sélection. C'est notamment le cas dans les scénarios où les demandes en bande passante des liens virtuels présentent une forte disparité. Une conséquence est que les nœuds et les liens physiques exclusivement congestionnés par les liens virtuels ayant une demande relativement faible, ne verront pas leur charge baissée par l'effet de la défragmentation.
3. La réallocation des liens virtuels candidats de manière séquentielle, comme c'est le cas dans cette solution, est moins efficace qu'une réallocation simultanée, laquelle en plus d'être adaptée aux approches proactives, proposerait certainement meilleur arrangement de ressources.

4.4 Définition du problème

Les données du problème sont quasi-similaires à celles du problème d'allocation de ressources traité dans le chapitre précédent.

4.4.1 Le réseau physique

La topologie du réseau physique est représentée par un graphe $G = (V, E)$ où V est l'ensemble des nœuds physiques et E ($|E|, E \subseteq V \times V$) l'ensemble des liens physiques reliant deux nœuds. On notera sans distinction (i, j) ou e un lien physique, en fonction du contexte, et on considérera que tous les liens sont bidirectionnels (full-duplex). Chaque nœud possède une seule table de flux et est défini par la capacité L_i de sa table de flux et la capacité M_i de sa table de groupe. Chaque lien est défini par sa capacité B_{ij} et son temps de propagation D_{ij}.

$$\forall (i, j) \in E : D_{ij} = D_{ji} \; et \; B_{ij} = B_{ji}$$

4.4.2 Les allocations d'un lien virtuel actif

A chaque lien virtuel k, l'algorithme d'allocation de ressources affecte un chemin de données que nous noterons F_k , consistant en un ensemble de liens et de nœuds physiques, chacun allouant des ressources au lien virtuel k. Sur chaque lien physique $(i, j) \in F_k$ la quantité de bande passante allouée au lien virtuel k est notée $f_k(i, j)$. De la même manière, le nombre d'entrées de la table de flux allouées au lien virtuel k sur un nœud physique $i \in F_k$ est notée $l_k(i)$.

Algorithme 2 : Proactive resource defragmentation mechanism

```
   Input  : G(V,E); K set of active Virtual Links ; F_k ∀k ∈ K ; θ ; N_max
1  begin
2      Initialize ℵ_θ ⟵ set of substrate links (i,j) : s(i,j) ≥ θ ∪ set of substrate nodes i :
         s(i) ≥ θ ; I_V ; I_E ; Success ⟵ false ; ζ ⟵ ∅
3      if (ℵ_θ ≠ ∅) and (I_V ≥ τ or I_E ≥ τ) then
4          ζ ⟵ Select Virtual Links(K, F_k, θ, N_max)
5          Revert currently assigned allocations to ζ from G(N,L)
6          Success ⟵ Reallocate(ζ, G(V,E))
7          if (Success) then
8              Commit new assigned allocations to ζ into infrastructure
9          else Rollback currently assigned allocations to ζ into G(V,E)
10
```

4.4.3 Objectif de la défragmentation

L'objectif est de :

- Migrer les liens virtuels dont la réallocation minimisera le taux d'utilisation maximal des nœuds et des liens de l'infrastructure, afin d'éviter ou de réduire la congestion, tout ceci en économisant au mieux les ressources.
- Minimiser le nombre de reconfigurations, exprimé en nombre total de liens virtuels effectivement reconfigurés.

Le problème de fragmentation ainsi énoncé, est un problème d'optimisation multicritères. Il est assimilé à un problème d'optimisation combinatoire, connu comme étant NP-difficile. Pour adresser ce problème, nous proposons deux heuristiques : la première, présentée dans la section suivante, est basée sur une approche proactive et la seconde, basée sur une approche réactive, est développée dans la section d'après.

4.5 Mécanisme de défragmentation proactif

Le tableau 4.1 positionne cette solution ([Tegueu 2017a]) par rapport à celles existantes dans la littérature. Il s'agit d'un mécanisme proactif déclenché au départ d'un réseau virtuel de l'infrastructure, et qui cherche à défragmenter, non seulement la bande passante des liens mais aussi les entrées des tables de flux des nœuds d'acheminement. L'algorithme 2 résume les étapes du mécanisme de défragmentation.

4.5.1 Contrôle des reconfigurations

Pour le contrôle des reconfigurations (Algorithme 2 : Ligne 3), nous proposons d'enclencher la migration si le réseau est θ**-congestionné** et τ**-fragmenté**. Deux métriques ont été définies pour déterminer ces deux conditions réseau : le **stress** noté s et l'**indice de fragmentation**

noté I. Ces métriques sont formulées comme suit :

- **Stress** : Le réseau est dit **θ-congestionné** si au moins une de ses entités est θ-congestionnée. Un lien (i, j) ou un nœud i est θ-congestionné si son stress est supérieur ou égal à θ, formellement $s(i, j) \geq \theta$ et $s(i) \geq \theta$ respectivement. Avec :

$$s(i,j) = \frac{w(i,j) * \sum_k (f_k(i,j))}{B_{ij}} \; et \; s(i) = \frac{w(i) * \sum_k (l_k(i))}{U_i} \tag{4.1}$$

  Où $s(i, j)$ et $s(i)$ représentent respectivement le stress du nœud i et du lien (i, j). $w(i, j)$ et $w(i)$ représentant respectivement l'importance du lien (i, j) et du nœud i. L'importance est un poids normalisé, calculé hors ligne en fonction de la centralité, communicabilité [Estrada 2010]. Ce poids permet de prendre en compte la sensibilité à la congestion des entités liée à leur position dans la topologie du réseau. Plus une entité est importante, plus elle doit être décongestionnée, car il est plus probable qu'elle soit sollicitée par l'algorithme d'allocation de ressources pour satisfaire les futures demandes.

- **Indice de fragmentation** : Le réseau sera dit **τ-fragmenté** si l'indice de fragmentation des nœuds ou des liens est supérieur à τ. L'indice de fragmentation des nœuds et des liens sont respectivement formulés comme suit :

$$I_V = \frac{Max(s(i),\ \forall i \in V)}{\frac{\sum_{i \in V}(s(i))}{|V|}} \; et \; I_E = \frac{Max(s(i,j),\ \forall (i,j) \in E)}{\frac{\sum_{(i,j) \in E}(s(i,j))}{|E|}} \tag{4.2}$$

L'indice de fragmentation est supérieur ou égal à 1 et inférieur ou égal à $|V|$ pour le cas des nœuds et inférieur à $|E|$ pour les liens. Si $I_V = I_E = 1$, alors le stress est uniformément reparti entre les liens et les nœuds du réseau, ce dernier est dit en équilibre.

La migration (Algorithme 2 : Lignes 4 à 8) ne sera enclenchée que si le réseau est θ–congestionné et τ-fragmenté. La prochaine étape du mécanisme consiste à sélectionner les liens virtuels à migrer. Elle est abordée ci-après.

4.5.2 Sélection des liens virtuels candidats à la migration

Dans cette solution, les processus de sélection et de réallocation sont séparés. L'algorithme 3 proposé pour la sélection des liens virtuels à migrer, détermine un nombre maximum donné de liens virtuels, noté N_{max}, parmi ceux actifs et dont la réallocation serait en adéquation avec l'objectif visé (cf section 4.4.3).

4.5.2.1 Principe de l'algorithme de sélection (Algorithme 3)

L'idée derrière cet algorithme est de déterminer l'ensemble ζ de liens virtuels consommant le plus de ressources sur les entités à faible capacité résiduelle, telle que leur réallocation réduira le taux d'utilisation de toutes les entités surchargées ou d'autant d'entités surchargées possibles. Cette démarche adaptée au scénario *en ligne*, garantit la détection et un désengorgement efficace de toutes les entités congestionnées à condition que le nombre de N_{max} soit suffisamment élevé.

Algorithme 3 : Select virtual links

Input : $G(V, E)$; X set of active Virtual Links ; $F_k \, \forall k \in X$; θ; N_{max}
Output : ζ set of selected virtual links

```
1  begin
2      Initialize ζ ⟵ ∅; Γ ⟵ ∅; a priority queue PQ ⟵ ∅
3      repeat
4          Γ ⟵ MSS-MRU(G(V,E), X\ζ, F_k, θ, N_max − ζ.length)
5          ζ ⟵ ζ ∪ Γ
6      until (Γ ≠ ∅)
7      if (ζ.length < N_max) then
8          PQ ⟵ ∅
9          foreach k ∈ X\ζ do
10             Calculate R_k
11             PQ.enqueue(k) using R_k as priority
12         while (ζ.length < N_max) do
13             ζ ⟵ ζ ∪ {PQ.dequeue()}
```

L'algorithme s'organise en deux grandes étapes successives :

1. La première étape (Algorithme 3 : Lignes 3 à 6) détermine à chaque itération, un ensemble de liens virtuels noté Γ, parmi ceux alloués sur les entités θ-congestionnées et qui n'ont pas encore été sélectionnés, c'est-à-dire appartenant à $K\backslash\zeta$, où K désigne l'ensemble des liens virtuels actifs sur le réseau. L'ensemble $\Gamma \subset K\backslash\zeta$ est calculé au moyen d'une primitive qui, à chaque fois qu'elle est exécutée, cherche à construire le plus petit ensemble de liens virtuels consommant le plus de ressources, tel que, toute entité θ-congestionnée a une part de ses ressources allouées au moins à l'un d'entre eux : cet ensemble est nommé **ensemble couvrant minimum utilisant le plus de ressources** ou en anglais **Minimum Spanning Set with Maximum Resources Utilisation** en abrégé **MSS-MRU**. La méthode de construction d'un MSS-MRU est présentée dans l'algorithme 4.

2. La seconde étape (Algorithme 3 : Lignes 7 à 13), quant-à elle, consiste en complément aux liens virtuels précédemment déterminés, à sélectionner les liens virtuels jusqu'à ce que le nombre total de liens virtuels atteigne N_{max}. Ces liens complémentaires sont sélectionnés prioritairement selon leur quantité de ressources consommées. Les liens sélectionnés aux deux étapes formeront l'ensemble de liens virtuels candidats à la migration, noté ζ.

4.5.2.2 Le comparateur

Pour construire l'ensemble MSS-MRU, nous avons introduit un comparateur. Il définit la façon avec laquelle les liens virtuels seront automatiquement ordonnés lors de leur insertion dans la file d'attente prioritaire. C'est grâce à lui que l'ensemble est *minimum* et constitué de liens *consommant le plus de ressources*. Ces deux aspects sont satisfaits par le biais de deux

Algorithme 4 : Minimum Spanning Set with Maximum Resources Utilisation (MSS-MRU)

Input : $G(V, E)$; X set of active Virtual Links ; $F_k \, \forall k \in X; \theta; N_{max}$
Output : Γ set of spanning virtual links

```
1  begin
2      Initialize ℵ̄_θ ⟵ ∅; Γ ⟵ ∅; ℵ_θ ⟵ set of substrate links (i,j) : s(i,j) > θ ∪ set of
         substrate nodes i : s(i) > θ ; a priority queue PQ ⟵ ∅
3      while (Γ.length < N_max) and (ℵ̄_θ.length < ℵ_θ.length) do
4          PQ ⟵ ∅
5          foreach k ∈ X\Γ do
6              Calculate A_k and Ā_k
7              PQ.enqueue(k) using Comparator
8          k ⟵ PQ.dequeue()
9          ℵ̄_θ ⟵ ℵ̄_θ ∪ (ℵ_θ ∩ F_k)
10         Γ ⟵ Γ ∪ {k}
```

critères : *l'impact d'un lien virtuel* et ses *ressources consommées* définis comme suit :

– L'**impact d'un lien virtuel** k sur un ensemble d'entités est le nombre d'entités de cet ensemble hébergeant le chemin de données F_k attribué à ce lien, à l'exception de ses nœuds source et destinations (car la reconfiguration du lien virtuel ne changera pas son impact sur ces nœuds). On distingue l'impact d'un lien k sur l'ensemble des entités θ-congestionnées noté A_k et l'impact du lien sur l'ensemble des entités θ-congestionnées non encore couvertes ($\aleph_\theta \backslash \overline{\aleph}_\theta$), noté $\overline{A_k}$. Formellement :

$$A_k = |(\aleph_\theta \backslash \{\{s_k\} \cup T_k\}) \cap F_k| \text{ and } \overline{A_k} = \left|((\aleph_\theta \backslash \overline{\aleph_\theta}) \backslash \{\{s_k\} \cap T_k\}) \cap F_k\right| \tag{4.3}$$

Nous estimons que, plus l'impact d'un lien virtuel est important, sa reconfiguration produira un meilleur gain en terme de réduction de surcharge des entités réseau.

– Les **ressources consommées** par un lien virtuel : L'une des spécificités de notre solution est la précision avec laquelle est formulée la quantité de ressources R_k, consommée par un lien virtuel k. Elle considère non seulement, classiquement la bande passante consommée par le lien virtuel, mais aussi les entrées consommées dans la table des flux des nœuds traversés. Formellement :

$$R_k = \eta * \sum_{(i,j) \in E} (f_k(i,j)) + \varepsilon * \sum_{i \in F_k} l_k(i) \tag{4.4}$$

Avec η et ε des constantes permettant d'élever les deux facteurs à la même échelle, ou de donner à un type de ressource plus d'importance par rapport à l'autre. Nous estimons que plus les ressources consommées par un lien sont élevées, sa reconfiguration produira un meilleur gain en termes de ressources.

Itération	Γ	$\aleph_\theta$	$\overline{\aleph_\theta}$	X
Initialisation	$\emptyset$	$(B,D),(D,E)$ $(C,D),(D,F)$ C,D,B,E	$\emptyset$	$k_1(4,4,7),k_2(4,4,7)$ $k_3(3,3,5),k_4(4,4,7)$
1	k_1	$\overline{(B,D)},\overline{(D,E)}$ $(C,D),(D,F)$ $C,\overline{D},\overline{B},E$	$(B,D),(D,E)$ D,B	$\overline{k_1(4,4,7)},k_2(2,4,7)$ $k_3(2,3,5),k_4(2,4,7)$
2	k_1,k_2	~~(B,D)~~, ~~(D,E)~~ $\overline{(C,D)},(D,F)$ $\overline{C}$,~~D~~,~~B~~, E	$(B,D),(D,E)$ (C,D) D,B,C	~~$k_1(4,4,7)$~~, $\overline{k_2(2,4,7)}$ $k_3(1,3,5),k_4(2,4,7)$
3	k_1,k_2,k_4	~~(B,D)~~, ~~(D,E)~~ ~~(C,D)~~, $\overline{(D,F)}$ ~~C~~,~~D~~,~~B~~,$\overline{E}$	$(B,D),(D,E)$ (C,D) D,B,C,E	~~$k_1(4,4,7)$~~, ~~$k_2(2,4,7)$~~ $k_3(1,3,5),\overline{k_4(2,4,7)}$

Légende : $(A \longrightarrow E) = k_1; (B \longrightarrow A) = k_2; (C \longrightarrow F) = k_3; (G \longrightarrow F) = k_4$
$k(x,y,z) = k(\overline{A}_k, A_k, R)$

TABLE 4.2 – MSS-MRU en action

Finalement, un lien virtuel est plus prioritaire par rapport à un autre, s'il a un impact plus élevé sur l'ensemble des entités θ-congestionnées non encore couvertes, sinon, à impact égal, celui ayant l'impact le plus élevé sur l'ensemble des entités θ-congestionnées est alors prioritaire, sinon celui consommant plus de ressources est prioritaire.

4.5.2.3 MSS-MRU en action

Reprenons le problème de fragmentation de ressources tel qu'il a été illustré dans la figure 4.1 et présenté dans la section 4.1. Les hypothèses sont les mêmes : la capacité des nœuds et des liens est égale à 4, un lien virtuel (représenté en trait pointillé) consomme 1 unité de bande passante et 1 entrée dans la table de flux des nœuds qu'il traverse. A partir de cette exemple illustratif, nous appliquons l'algorithme MSS-MRU en supposant $\theta = 0.50, \eta = \varepsilon = 1$. La table 4.2 présente, pas-à-pas, le comporte de l'algorithme qui à la fin, sélectionne, comme on s'y attendait, les liens virtuels $(A \longrightarrow E), (B \longrightarrow A)$ et $(G \longrightarrow F)$. Ces liens forment l'ensemble couvrant minimum et consommant le plus de ressources.

4.5.3 Réallocation des liens virtuels sélectionnés

C'est à cette étape (Algorithme 2 : Ligne 6) que les liens virtuels précédemment sélectionnés sont réalloués. En règle générale, l'efficacité de toute solution dépend de la qualité d'adaptation des nouvelles allocations assignées aux liens virtuels, vis-à-vis de l'objectif visé concernant les ressources réseau ciblées par la défragmentation. Dans notre cas, comme indiqué dans la section 4.4.3, à chaque opération de défragmentation, l'objectif est de réduire le niveau d'utilisation

Algorithme 5 : Reactive resource defragmentation mechanism

Input : $G(V, E)$; K set of incoming VLs request ; X set of active VLs ; $F_k \, \forall k \in X$; θ ; N_{max}

begin
  Initialize $Success \longleftarrow false; \zeta \longleftarrow \emptyset$
  $\zeta \longleftarrow$ `Pre-Select VLs`$(G(V, E), X, F_k, \theta, N_{max})$
  Revert currently assigned allocations to ζ
  $Success \longleftarrow$ `Selection + Reallocate`$(G(V, E), K, \zeta)$
  if *(Success)* **then**
    Commit new assigned allocations to ζ and to K ;
  else Rollback currently assigned allocations to ζ

maximal des liens et des nœuds de l'infrastructure physique tout en économisant au mieux les ressources : **bande passante** des liens et **entrées de la table de flux** des nœuds. L'algorithme de réallocation doit donc optimiser ces critères pour être en adéquation avec l'objectif visé. D'autres critères liés à la réallocation et permettant de distinguer cette solution de celles existantes dans la littérature sont :

1. l'algorithme de réallocation n'exige aucune information spécifique de la part de l'algorithme d'allocation initiale. L'intérêt est de rendre la solution **portable** donc capable d'opérer avec n'importe quel algorithme d'allocation de ressource.
2. les liens virtuels sont réalloués de manière **simultanée** pour une meilleure adaptation des nouvelles allocations.

Nous avons fait le choix d'utiliser notre algorithme d'allocation de ressources présenté au chapitre 3 en l'adaptant à ce nouveau rôle. En effet, la spécificité de cet algorithme lié au contexte SDN, outre sa capacité à répartir les allocations entres les liens et les nœuds du réseau tout en économisant les ressources réseaux, est d'offrir des leviers permettant d'adapter son comportement, en l'occurrence en donnant plus d'importance au taux d'utilisation maximal des entités, ceci en fixant les paramètres β_1 et α_1 à une valeur élevée. Une autre particularité exploitée ici est la souplesse que cet algorithme offre par la possibilité de réallouer simultanément les liens virtuels sélectionnés.

4.6 Mécanisme de défragmentation réactif

Le tableau 4.1 positionne cette solution ([Tegueu 2017b]) par rapport à celles existantes dans la littérature. Il s'agit d'un mécanisme réactif déclenché lorsque le traitement d'une requête d'allocation de ressources échoue, et qui cherche à défragmenter, non seulement la bande passante des liens mais aussi les entrées des tables de flux des nœuds d'acheminement. L'algorithme 5 résume les étapes de ce mécanisme de défragmentation. Les processus de sélection et de réallocation sont intégrés et exécutés de manière coordonnée, tout en cherchant à allouer en même temps le réseau virtuel dont l'échec d'allocation a déclenché la défragmentation.

4.6.1 Pré-sélection des liens virtuels candidats à la migration

Comme c'est généralement le cas dans les mécanismes de défragmentation réactifs, pour résister au passage à l'échelle, l'étape de réallocation est précédée par une étape de présélection (Algorithme 5 : Ligne 3) dont l'objectif est de sélectionner un sous-ensemble de liens virtuels à réallouer. L'intérêt est de considérer juste une poignée de liens virtuels actifs à inclure dans le processus de réallocation afin que le temps de traitement de la requête soit raisonnable. Ici, la pré-sélection est réalisée par l'algorithme MSS-MRU présenté dans la section 4.5.

4.6.2 Sélection et Réallocation

A cette étape (Algorithme 5 : Ligne 5), le réseau virtuel K est alloué en migrant toute ou une partie des liens virtuels pré-sélectionnés. Comme il a été indiqué dans la section présentant la solution proactive, l'efficacité de toute solution dépend de la qualité d'adaptation des nouvelles allocations assignées aux liens virtuels, vis-à-vis de l'objectif visé concernant les ressources réseau ciblées par la défragmentation. Aussi, dans notre cas, à chaque opération de défragmentation, l'objectif est de réduire le niveau d'utilisation maximal des liens et des nœuds de l'infrastructure physique tout en économisant au mieux les ressources : **bande passante** des liens et **entrées de la table de flux** des nœuds. L'algorithme de réallocation doit donc optimiser ces critères pour être en adéquation avec l'objectif visé. D'autres critères liés à la réallocation et permettant de distinguer cette solution de celles existantes dans la littérature sont :

1. l'algorithme de réallocation n'exige aucune information spécifique de la part de l'algorithme d'allocation initiale. L'intérêt est de rendre la solution **portable** donc capable d'opérer avec n'importe quel algorithme d'allocation de ressource.
2. les liens virtuels sont réalloués de manière **simultanée** pour proposer des allocations mieux adaptées à l'objectif visé.

Pour les mêmes raisons que celles évoquées à la section 4.5.3, nous avons fait le choix d'utiliser notre algorithme d'allocation de ressources présenté au chapitre 3, en l'adaptant afin qu'il puisse supporter la migration. Cette adaptation consiste à inclure de nouvelles contraintes liées à la réallocation des liens virtuels actifs et à modifier la fonction objective en y ajoutant le coût des reconfigurations.

4.6.2.1 Contraintes de réallocation

$\forall k \in \zeta$ et $\forall (i,j) \in E, G_k(i,j)$ représente est une constante binaire indiquant si le lien actif k est à l'origine alloué ou non sur le lien physique (i,j). $G_k(i,j)$ vaut 1 si le virtuel k est alloué sur le lien physique (i,j) et 0 dans le cas contraire. Nous introduisons z_k une nouvelle variable binaire qui après résolution du modèle, prendra la valeur 1 si le lien virtuel $k \in \zeta$ est effectivement réalloué et valeur 0 sinon. Rappelons que la variable binaire $g_k(i,j) \forall k \in \zeta, (i,j) \in E$ indique si le lien virtuel k est alloué ou non sur le lien physique (i,j). L'allocation d'un lien virtuel $k \in \zeta$ change au niveau d'un lien physique (i,j) s'il était à l'origine alloué sur ce lien ($G_k(i,j) = 1$) et ce n'est plus le cas après l'opération de réallocation ($g_k(i,j) = 0$) ou s'il est nouvellement alloué sur ce lien physique ($G_k(i,j) = 0$ et $g_k(i,j) = 1$). Ces deux changements peuvent être détectés par cette expression logique : $y_k(i,j) = G_k(i,j) * \overline{g_k(i,j)} + \overline{G_k(i,j)} * g_k(i,j)$ dont la version

linéaire est $y_k(i,j) = G_k(i,j) * (1 - g_k(i,j)) + (1 - G_k(i,j)) * g_k(i,j)$. Ainsi, le nombre total de changements noté y_k, entre l'allocation courant du lien virtuel k et sa nouvelle allocation est :

$$y_k = \sum_{(i,j) \in E} y_k(i,j)$$

La variable z_k dérive de y_k comme ceci :

$$\forall k \in \zeta : z_k = \begin{cases} 0, & if\ y_k = 0 \\ 1, & if\ y_k \geq 1 \end{cases}$$

Cette équation peut-être linéarisée comme suit :

$$\frac{y_k}{2 * |E|} \leq z_k \leq y_k \tag{4.5}$$

4.6.2.2 Fonction objective

La fonction objective reprend celle (définie à la section 3.25) de l'algorithme d'allocation de ressources, à laquelle est ajouté le coût des reconfigurations qu'il faudra aussi minimiser. Elle est formellement définie comme suit :

$$\begin{aligned} &\text{minimise} \\ &\alpha_1 * f_{max} + \\ &\alpha_2 * \frac{1}{|E|} * \sum_{(i,j) \in E} \left(\frac{1}{B_{ij}} * \left(\overline{B_{ij}} + \sum_{k \in K \cup \zeta} f_k(i,j) \right) \right) + \\ &\beta_2 * \frac{1}{|V|} * \sum_{i \in V} \left(\frac{1}{L_i} * \left(\overline{L_i} + \sum_{k \in K \cup \zeta} l_k(i) \right) \right) + \\ &\beta_1 * c_{max} + \\ &\gamma * \sum_{k \in K \cup \zeta} \sum_{t \in T_k} d_k^t(t) + \\ &\rho * \sum_{k \in \zeta} z_k \end{aligned} \tag{4.6}$$

Le dernier terme représente le coût des reconfigurations où le coefficient ρ est utilisé pour donner plus ou moins d'importance à ce critère.

4.7 Évaluation expérimentale

Nous présentons dans cette section les analyses des performances des deux algorithmes de défragmentation proposés. Les expériences ont été réalisées en utilisant la topologie du réseau GÉANT et une topologie de type réseau de Campus. Les objectifs de ces différentes analyses sont :

Réseau				Requête		
Modèle	Topologie	Capacité des liens	Capacité des tables	Type	Description	Caractéristiques des LVs
GÉANT	41 nœuds 60 liens	entre [5000 - 100000]	flux 200 groupe 100	Unicast	entre [6 - 8] LVs entre [1 - 1] dest. par LV	BP entre [600 - 800] délai = N/A
				Multicast	entre [6 - 8] LVs entre [2 - 4] dest. par LV	BP entre [600 - 800] délai = N/A
Campus	14 nœuds 25 liens	entre [100 - 1000]	flux 200 groupe 100	Unicast	entre [6 - 8] LVs entre [1 - 1] dest. par LV	BP entre [1 - 3] délai = N/A
				Multicast	entre [6 - 8] LVs entre [2 - 4] dest. par LV	BP entre [1 - 3] délai = N/A

Légende : LV = Lien Virtuel, N/A = Non Appliqué, dest. = destinataire, BP = Bande Passante

TABLE 4.3 – Modèle de réseau et de requête

1. La mise en évidence de l'apport (taux d'acceptation) et du coût (nombre total de reconfigurations) de nos solutions par rapport à des mécanismes existants.
2. L'influence des paramètres sur les résultats (à nouveau en terme de taux d'acceptation et de nombre total de reconfigurations).
3. La validation des objectifs visés par nos mécanismes de défragmentation, à savoir : réduire le taux d'utilisation maximal des nœuds et des liens tout en économisant au mieux les ressources.

4.7.1 Modèle de simulation considéré

Les différents aspects à définir pour les simulations sont les mêmes que ceux pris en compte pour l'évaluation de notre algorithme d'allocation de ressources, à savoir : le modèle de réseau, le modèle de charge et les paramètres d'expérimentation. Chacun de ces aspects est instancié ici de la même manière que défini dans à la section 3.6.1. La table 4.3 rappelle les paramètres utilisés pour générer les requêtes pour chacun des deux modèles réseau considérés. Les caractéristiques de chacun de ces modèles réseau y sont également présentées.

4.7.1.1 Modèle de réseau

De manière similaire, nous utilisons deux modèles de réseau pour les expérimentations :
- le réseau européen de recherche, GÉANT, qui possèdent des liens reliant la plupart des capitales et villes importantes européennes. La figure 3.1a présente la carte du réseau GÉANT.
- un réseau de Campus constitué de 14 nœuds et 25 liens (voir figure 3.1b).

4.7.1.2 Modèle de charge

Nous utilisons le même modèle de charge que celui défini dans la section 3.6.1. Rappelons que les requêtes arrivent selon un processus de Poisson de paramètre λ, avec λ variant de 4 à 10 requêtes toutes les 100 unités de temps par pas de 1 : $\lambda \in \{0.04, 0.05, 0.06, 0.07, 0.08, 0.09, 0.1\}$. Nous considérons également les deux types de requêtes : Unicast et Multicast. Notons que, pour chaque type de requête et pour chaque taux d'arrivée, nous avons utilisé ici les **mêmes** instances de requêtes que ceux générées pour l'évaluation de notre algorithme d'allocation de ressources. Pour rappel, une instance est un ensemble de requêtes, générées sur une durée fixe (10000 unités de temps dans notre cas) avec un taux d'arrivée et des paramètres définis. Nous vous invitons à consulter la table 4.3 pour plus de détails sur les caractéristiques des requêtes.

4.7.1.3 Les paramètres d'expérimentation

4.7.1.3.1 La durée d'expérimentation

Les expérimentations ont aussi toutes été effectuées sur une durée de 10000 unités de temps. Autrement dit, les instances sont rejouées jusqu'à l'instant 10000 unités de temps : durée suffisante pour atteindre un état stationnaire du système.

4.7.1.3.2 L'algorithme d'allocation de ressources

L'algorithme d'allocation de ressources est configuré en utilisant les paramètres pour lesquels il a produit les meilleurs performances à l'issue des résultats d'évaluation qui avaient été obtenus. Pour les expérimentations utilisant le réseau GÉANT, $\beta_1 = \beta_2 = \beta = 150$, $\alpha_1 = \alpha_2 = \alpha = 1$ et $PS_{Ratio} = N/A$. Pour les expérimentations utilisant le réseau de Campus, $\beta_1 = \beta_2 = \beta = 1$, $\alpha_1 = \alpha_2 = \alpha = 150$ et $PS_{Ratio} = 45\%$. La minimisation des délais n'étant pas recherchée dans les expérimentations, le coefficient γ a été fixé à la valeur 0.

4.7.1.3.3 Le mécanisme de défragmentation proactif

Le contrôle des reconfigurations : Le paramètre τ est fixé à la valeur 0 et 2 valeurs de θ ont été utilisées : $\theta \in \{0.90, 0.95\}$.

La sélection : 2 valeurs de N_{max} ont été utilisées : $N_{max} \in \{20, 30\}$. Pour les expérimentations utilisant le réseau GÉANT, les paramètres η et ε sont respectivement fixés à 1 et 600. Pour les expérimentations utilisant le réseau de Campus, les paramètres η et ε sont respectivement fixés à 1 et 2. Ces paramètres sont fixés de manière à donner autant d'importance aux entrées de la table de flux qu'à la bande passante.

La réallocation : la réallocation est effectuée par une instance de notre algorithme d'allocation de ressources qui utilise les mêmes configurations que celles présentées dans la section 4.7.1.3.2. Pour minimiser le taux d'utilisation maximal des nœuds et des liens, les paramètres

α_1 et β_1 sont dynamiquement ajustés comme suit : $\alpha_1 = 1500$ (resp. $\beta_1 = 1500$) si les liens (resp. si les nœuds) sont θ-congestionnés.

4.7.1.3.4 Le mécanisme de défragmentation réactif

La pré-sélection : 2 valeurs de N_{max} ont été utilisées : $N_{max} \in \{20, 30\}$. Pour les expérimentations utilisant le réseau GÉANT, les paramètres η et ε sont respectivement fixés à 1 et 600. Pour les expérimentations utilisant le réseau de Campus, les paramètres η et ε sont respectivement fixés à 1 et 2. Ces paramètres sont fixés de manière à donner autant d'importance aux entrées de la table de flux qu'à la bande passante.

La sélection et réallocation : l'algorithme utilise les mêmes configurations que celles présentées dans la section 4.7.1.3.2. Les paramètres α_1 et β_1 sont aussi dynamiquement ajustés comme suit : $\alpha_1 = 1500$ (resp. $\beta_1 = 1500$) si les liens (resp. si les nœuds) sont θ-congestionnés. Le paramètre ρ est fixé à la valeur 1.

4.7.1.3.5 Fonctionnement des expérimentations

En considérant le modèle réseau GÉANT (resp. de Campus), pour chaque type de requête (unicast et multicast), et chaque taux d'arrivée (de 4 à 10 requêtes toutes les 100 unités de temps), chacune des instances qui avaient été générée, est rejouée en faisant varier les paramètres $\theta \in \{0.95, 0.90\}$ et $N_{max} \in \{20, 30\}$, de chacun de nos deux mécanismes (proactif et réactif) portant le total de simulation à $2 * 7 * 2 * 2 * 2 = 112$. Soit 112 * 2 = 224, si on considère les deux modèles réseau GÉANT et Campus.

4.7.2 Métriques de performance

Les performances de nos mécanismes ont été mesurées à l'aide de plusieurs métriques : le taux d'acceptation des requêtes, le taux d'utilisation maximal des liens et des nœuds, et la charge globale des liens et des nœuds, le nombre total de reconfigurations.

La valeur de chaque métrique a été prélevée :

- À la fin de la simulation (à $T = 10000$ unités de temps), les graphes sont alors fournis en fonction du taux d'arrivée des requêtes λ.
- Sur toute la période de simulation d'une expérience liée à une instance, les graphes sont alors fournis en fonction du temps de simulation T (en unité de temps).

4.7.2.1 Le taux d'acceptation des requêtes

Le taux d'acceptation des requêtes correspond simplement au nombre de requêtes qui ont été allouées sur le réseau, par rapport au nombre total de requêtes générées pendant la durée d'une simulation (ou une partie de cette durée dans le cas d'étude sur la période de simulation).

$$\tau_{acceptation} = \frac{\text{Nombre de requêtes allouées}}{\text{Nombre de requêtes reçues}}$$

4.7.2.2 La charge globale du réseau

4.7.2.2.1 La charge globale des liens du réseau

La charge globale des liens du réseau correspond à la quantité totale de ressources consommées sur les liens (à un instant donné) par rapport à la capacité totale des liens, soit :

$$C_{liens} = \frac{\sum_{e \in E} \overline{B_e}}{\sum_{e \in E} B_e}$$

Avec B_e la capacité du lien e et $\overline{B_e}$ les ressources consommées sur le lien e à l'instant considéré.

4.7.2.2.2 La charge globale des nœuds du réseau

La charge globale des nœuds du réseau correspond à la quantité totale d'entrées de la table de flux sur les nœuds (à un instant donné) par rapport à la capacité totale de la table de flux des nœuds, soit :

$$C_{n \not\leq uds} = \frac{\sum_{i \in V} \overline{L_i}}{\sum_{i \in V} L_i}$$

Avec L_i la capacité de la table de flux du nœud i et $\overline{L_i}$ la quantité totales d'entrées consommées dans la table de flux du nœud i à l'instant considéré.

4.7.2.3 Le taux d'utilisation maximal du réseau

4.7.2.3.1 Le taux d'utilisation maximal des liens

Le taux d'utilisation maximal des liens correspond au plus grand taux d'utilisation des liens (à un instant donné), c'est à dire les ressources consommées sur le lien par rapport à la capacité du lien, soit :

$$\tau_{liens}^{max} = \max_{\forall e \in E} \frac{\overline{B_e}}{B_e}$$

Avec B_e la capacité du lien e et $\overline{B_e}$ les ressources consommées sur le lien e à l'instant considéré.

4.7.2.3.2 Le taux d'utilisation maximal des nœuds

Le taux d'utilisation maximal des nœuds correspond au plus grand taux d'utilisation des nœuds (à un instant donné), c'est à dire les ressources consommées sur le nœud par rapport à la capacité de la table de flux du nœud, soit :

$$\tau_{n \not\leq uds}^{max} = \max_{\forall i \in V} \frac{\overline{L_i}}{L_i}$$

Avec L_i la capacité de la table de flux du nœud i et $\overline{L_i}$ la quantité totale d'entrées consommées dans la table de flux du nœud i à l'instant considéré.

4.7.2.4 Le nombre total de reconfigurations

Le nombre total de reconfigurations correspond à la somme totale du nombre de liens virtuels qui sont effectivement migrés chaque fois que la défragmentation est déclenchée.

4.7.3 Méthodes existantes

Nos solutions ont été comparées à SDN-VN, mécanisme proactif proposé par Mijumbi et al. dans [Mijumbi 2014]. A notre connaissance, c'est le seul mécanisme de défragmentation qui cible, en plus de la bande passante, les entrées de la table de flux.

Dans la suite de ce chapitre, on utilisera GA-Pro (resp. GA-React) pour faire référence à notre solution proactive (resp. réactive), où GA signifie Gestionnaire Autonomique.

4.7.4 Résultats expérimentaux sur le réseau GÉANT

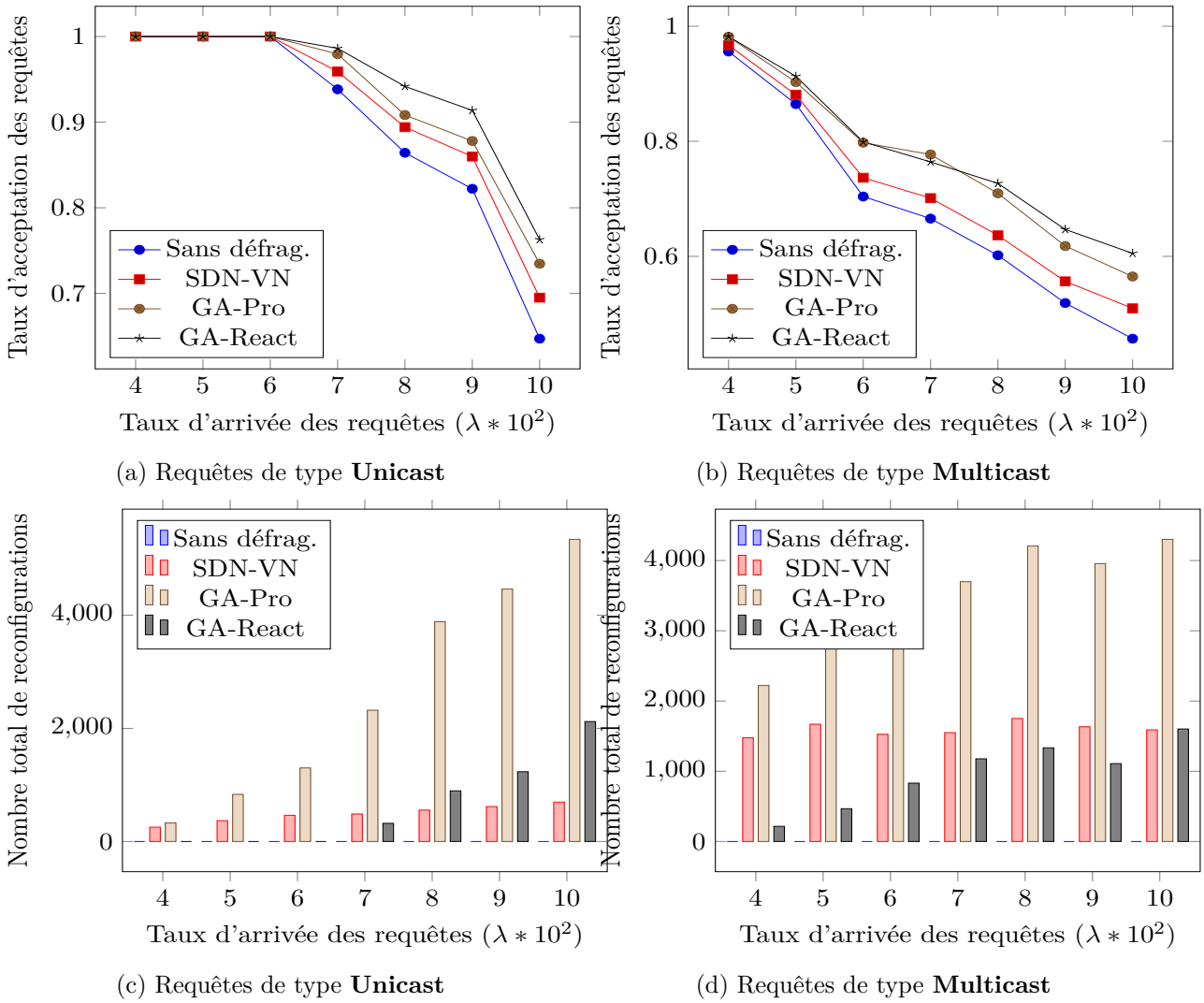
4.7.4.1 Comparaison avec les méthodes existantes

Les graphiques de la figure 4.4 présentent les résultats de comparaison de nos méthodes avec la méthode SDN-VN. Les paramètres $\theta = 0.95$ et $N_{max} = 20$ ont été choisis suite aux résultats comparatifs présentés dans la partie 4.7.4.2.

Les courbes montrent que, quel que soit le type d'instance considérée, la défragmentation apporte un gain en termes de taux d'acception des requêtes. Ce gain augmente avec le taux d'arrivée des requêtes et atteint environ 10% lorsque le taux d'arrivée vaut 0.1. Nos mécanismes présentent un apport plus notable comparé à la méthode SDN-VN, notamment l'approche réactive qui a tendance à produire de meilleures performances, quel que soit le type d'instance considéré.

Les courbes des figures 4.5 et 4.6 expliquent comment nos mécanismes, en l'occurrence la solution GA-Pro, favorisent l'admissibilité des requêtes.

Les figures 4.5 présentent l'évolution du taux d'utilisation maximal des liens et des nœuds au cours du temps, pour les instances de type unicast et multicast lorsque $\lambda = 0.04$. On peut remarquer que, pour chaque type de requête, le taux d'utilisation maximal des liens (de même que celui des nœuds) est identique avec et sans défragmentation depuis le début de la simulation jusqu'à l'instant 1453 UT. A cet instant, la défragmentation est appliquée pour la première fois suite au départ d'un réseau virtuel. Les courbes de cette figure montrent deux choses : le caractère adaptatif de notre solution et sa capacité à réduire le taux d'utilisation maximal. En effet, les courbes 4.5a et 4.5b indiquent que les liens ont tendance à être plus congestionnés (taux d'utilisation dépassant parfois $\theta = 0.95$) que les nœuds. GA-Pro réduit, lorsque possible, le taux d'utilisation maximal des liens, quitte à augmenter celui des nœuds comme le montrent les figures 4.5c et 4.5d.



(a) Requêtes de type **Unicast**

(b) Requêtes de type **Multicast**

(c) Requêtes de type **Unicast**

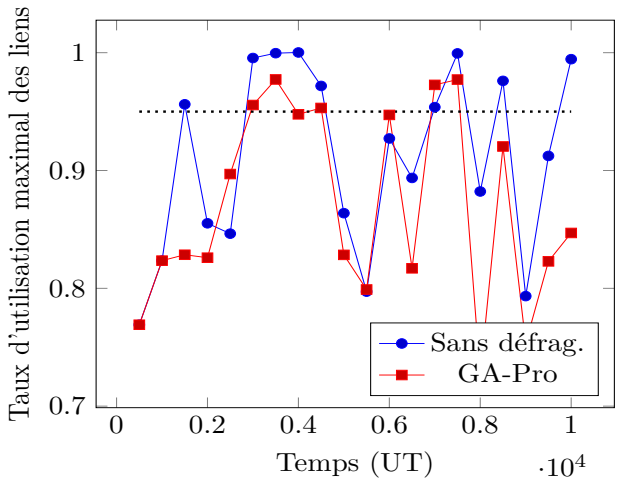
(d) Requêtes de type **Multicast**

FIGURE 4.4 – Comparaison de différentes méthodes de défragmentation

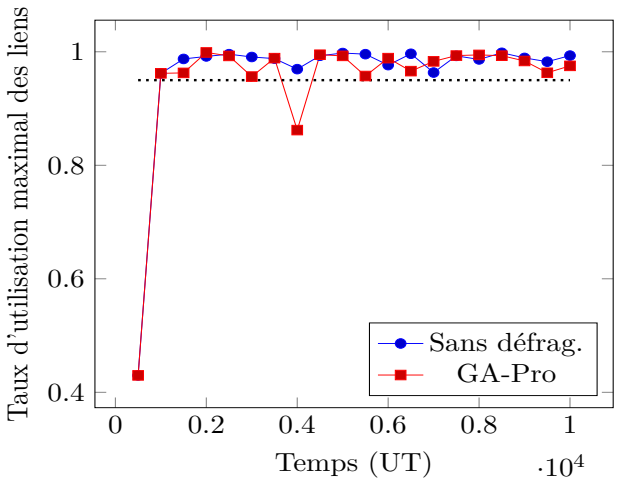
Les figures 4.6 présentent l'évolution de la charge des liens et des nœuds au cours du temps, pour les instances de type unicast et multicast lorsque $\lambda = 0.04$. Les courbes de ces figures montrent clairement que, quel que soit le type d'instance, la charge des nœuds et des liens du réseau diminue lorsque la défragmentation est utilisée. Ceci s'explique par la capacité de GA-Pro à réduire la quantité de ressources consommées.

Notons que les analyses précédentes sont valables quel que soit le taux d'arrivée des requêtes. Elles permettent par la même occasion de valider l'adéquation entre le fonctionnement de nos mécanismes et l'objectif visé concernant les ressources.

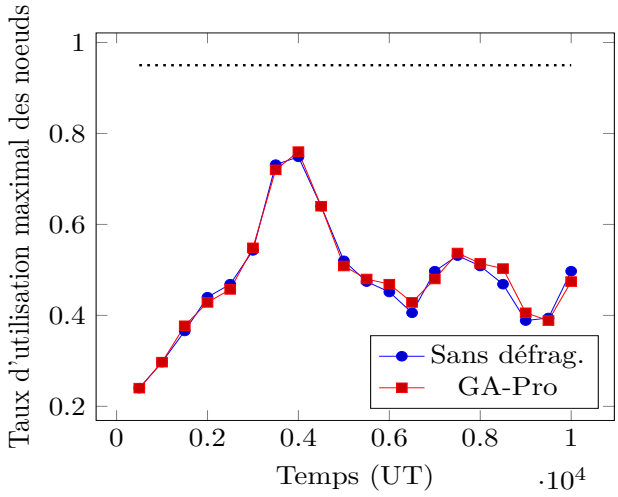
En ce qui concerne le coût de migration, les figures 4.4c et 4.4d montrent que GA-Pro produit plus de reconfigurations par rapport à GA-React et à SDN-VN. La raison est que, GA-Pro est



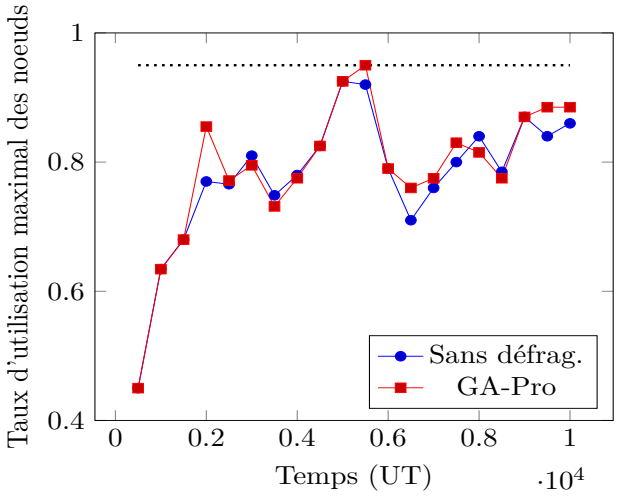
(a) Taux d'utilisation maximal des **liens - Requêtes Unicast**



(b) Taux d'utilisation maximal des **liens - Requêtes Multicast**



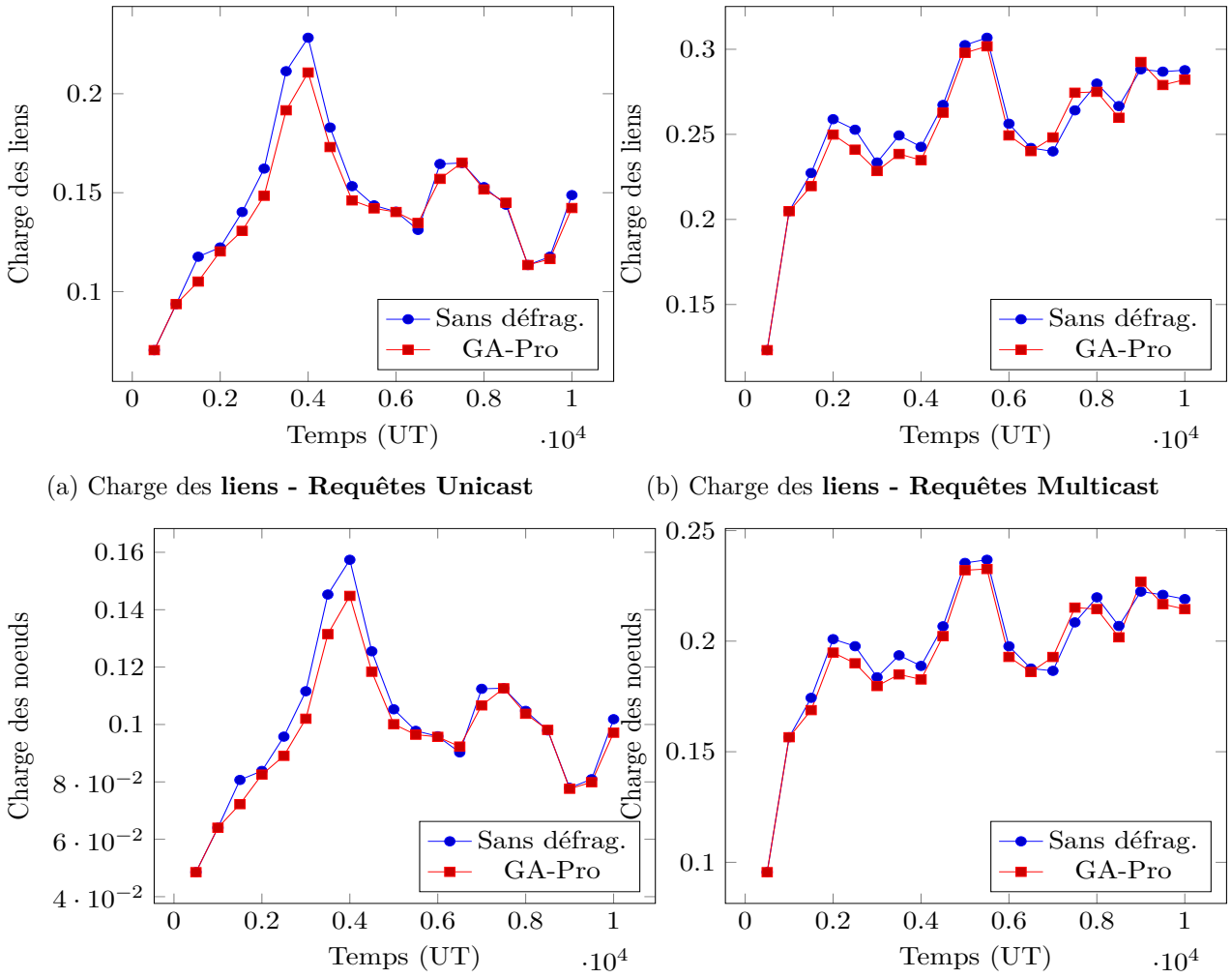
(c) Taux d'utilisation maximal des **noeuds - Requêtes Unicast**



(d) Taux d'utilisation maximal des **noeuds - Requêtes Multicast**

FIGURE 4.5 – Taux utilisation maximal en fonction du temps et du type de requêtes - $\lambda = 0.04$

déclenché autant de fois que de requêtes sont acceptées, justifiant par la même occasion l'augmentation du nombre de reconfigurations avec le taux d'arrivée des requêtes. En plus, à chaque fois que l'étape de migration est enclenchée, en moyenne 80% de liens virtuels sélectionnés sont effectivement reconfigurés, contrairement à SDN-VN où juste en moyenne 20% sont sujets de reconfiguration. Le score de 80% de GA-Pro nous semble intéressant et peut signifier que les méthodes de sélection et de réallocation sont en adéquation et visent bien le même objectif. Le score de 20% de SDN-VN peut être dû au fait que les liens virtuels sont réalloués séquentiellement comme l'impose la mécanique de cette solution. Une seconde hypothèse est que le modèle de réallocation considéré (dans ce cas notre algorithme d'allocation de ressources) n'est pas adapté à la méthode de sélection de SDN-VN ou alors il n'est pas convenablement configuré.



(a) Charge des **liens - Requêtes Unicast**

(b) Charge des **liens - Requêtes Multicast**

(c) Charge des **noeuds - Requêtes Unicast**

(d) Charge des **noeuds - Requêtes Multicast**

FIGURE 4.6 – Charge du réseau en fonction du temps et du type de requêtes - $\lambda = 0.04$

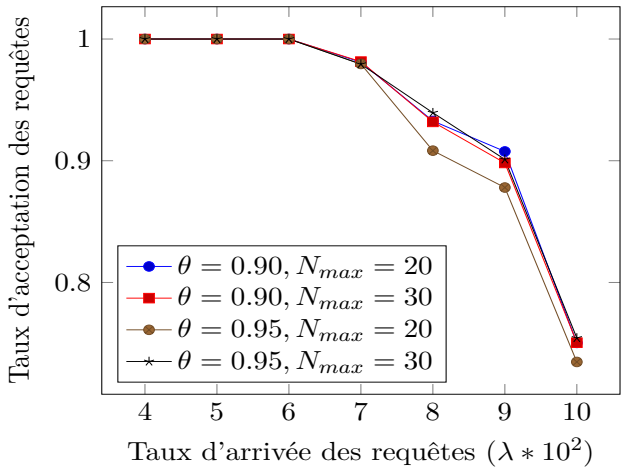
Cependant, les auteurs de SDN-VN ne donnent pas de détail à ce sujet.

4.7.4.2 Influence des paramètres du modèle

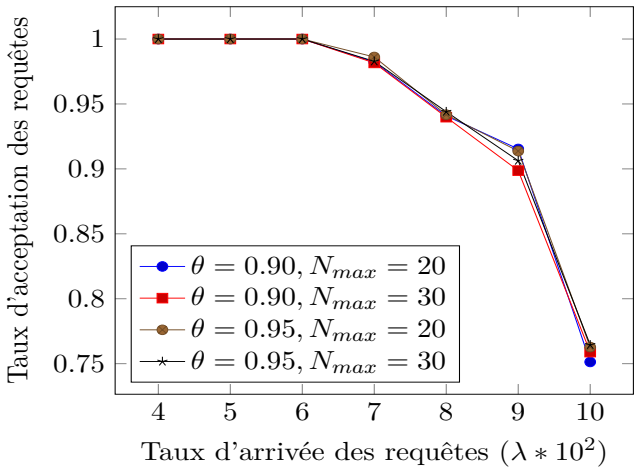
Dans cette section, nous allons étudier l'influence respective des paramètres θ et N_{max} de nos solutions. Rappelons que θ et N_{max} représente respectivement le seuil de congestion au delà duquel GA-Pro est déclenché et le nombre maximum de liens virtuels candidats à la migration.

4.7.4.2.1 Taux d'acception des requêtes

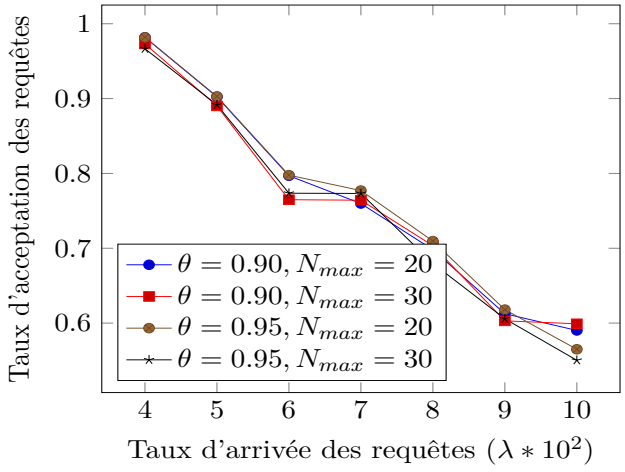
Les graphiques de la figure 4.7 présentent les taux d'acceptation des requêtes pour différentes instances de type unicast et multicast.



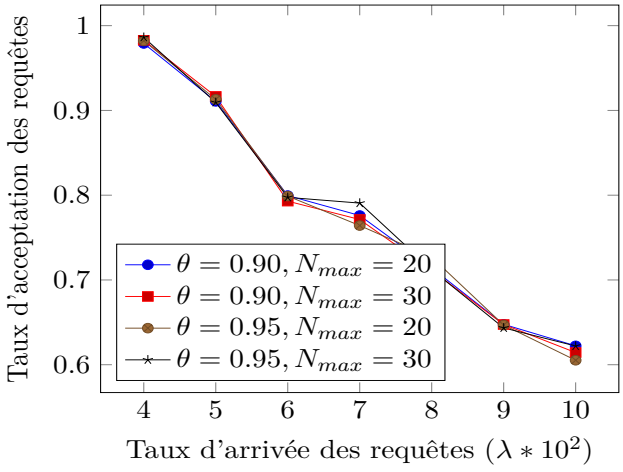
(a) Taux d'acceptation des requêtes de type **unicast** en fonction du taux d'arrivée - **GA-Pro**.



(b) Taux d'acceptation des requêtes de type **unicast** en fonction du taux d'arrivée - **GA-React**.



(c) Taux d'acceptation des requêtes de type **multicast** en fonction du taux d'arrivée - **GA-Pro**.



(d) Taux d'acceptation des requêtes de type **multicast** en fonction du taux d'arrivée - **GA-React**.

FIGURE 4.7 – Taux d'acceptation des requêtes en fonction de leur taux d'arrivée et des paramètres de simulation.

Requêtes de type unicast Dans les figures 4.7a et 4.7b, les courbes très proches montrent que les performances en terme de taux d'acceptation varient peu en fonction de θ et N_{max}, quel que soit le mécanisme considéré. Cependant, les performances de GA-Pro semblent invariantes lorsque θ et N_{max} varient dans le même sens. Autrement dit, si θ croît (resp. décroît), par exemple de 0.90 à 0.95 (resp. de 0.95 à 0.90) et N_{max} de 20 à 30 (resp. de 30 à 20), les taux d'acceptation restent approximativement identiques. Dans tous les cas, la défragmentation est meilleure lorsque N_{max} est grand. Par ailleurs, comme le montre la figure 4.7b, GA-React est plus performant lorsque θ et N_{max} sont élevés. Ceci en raison du fait que les entités congestionnées à l'origine de l'échec du traitement d'une requête, peuvent être détectées lorsque θ est élevé et avec N_{max} grand, il est plus possible de réduire fortement leur taux de charge afin que ces entités

puissent supporter la nouvelle requête. Cependant, il faut faire attention aux temps de calcul des allocations qui augmentent rapidement lorsque N_{max} croît : c'est l'inconvénient majeur des approches réactives qui réagissent dans l'urgence (lors du rejet d'une requête).

Requêtes de type multicast Les courbes de la figure 4.7c et 4.7d montrent que lorsque le taux d'arrivée des requêtes augmente, le taux d'acception des requêtes multicast diminue plus vite que celui des requêtes unicast indépendamment des valeurs de θ et N_{max} . Contrairement aux instances de type unicast, avec GA-Pro et les instances composées de requêtes de type multicast, il est difficile de mettre en avant l'intérêt ou l'influence de chaque paramètre. En revanche, l'analyse précédente portant sur GA-React et les instances de type unicast, est également valable sur les instances de type multicast. Il faudra dans ce cas faire davantage attention au temps de calcul des allocations, d'autant plus que la durée de traitement des requêtes de type multicast est plus élevée.

4.7.4.2.2 Nombre total de reconfigurations

Les figures 4.8 présentent le nombre total de reconfigurations en fonction du taux d'arrivée des requêtes, pour les instances de type unicast et multicast pour les simulations utilisant GA-Pro et GA-React. On observe clairement que le nombre total de reconfigurations a tendance à augmenter avec le taux d'arrivée des requêtes. Un autre aspect frappant est que, GA-React produit environ trois fois moins de reconfigurations que GA-Pro, quel que soit le type d'instance considéré. Aussi, le nombre total de reconfigurations augmente lorsque N_{max} croît.

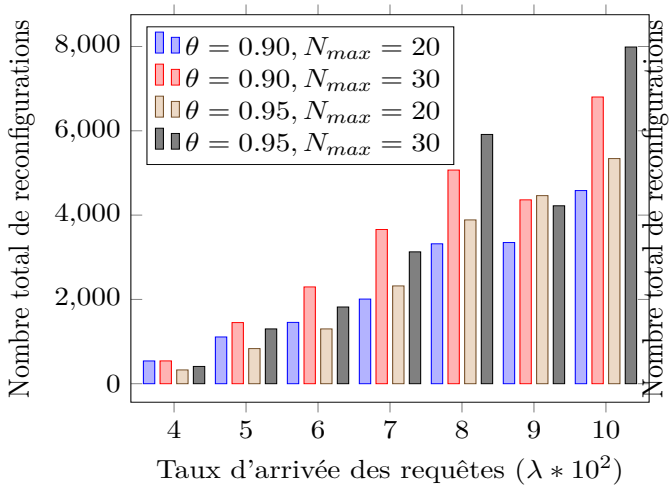
4.7.5 Résultats expérimentaux sur le réseau de Campus

Les courbes décrivant les résultats d'évaluation obtenus à l'issue des expérimentation réalisées sont disponibles en annexe A. Les analyses portant sur ces résultats étant similaires à celles effectuées avec le réseau GÉANT, elles ne seront donc pas reprises dans cette section.

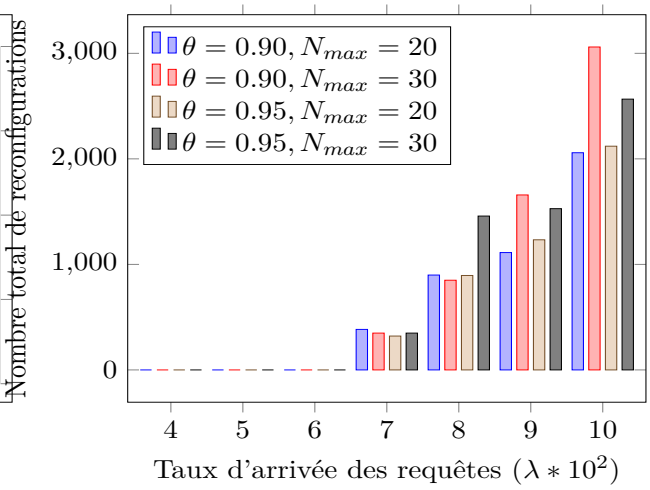
4.8 Conclusion

Nous avons présenté dans ce chapitre, les solutions que nous proposons pour lutter contre le problème de fragmentation de ressources. Il s'agit de deux mécanismes de défragmentation de ressources : l'un proactif déclenché au départ d'un réseau virtuel de l'infrastructure et l'autre déclenché suite à l'échec de traitement d'une demande d'allocation de ressources pour un réseau virtuel.

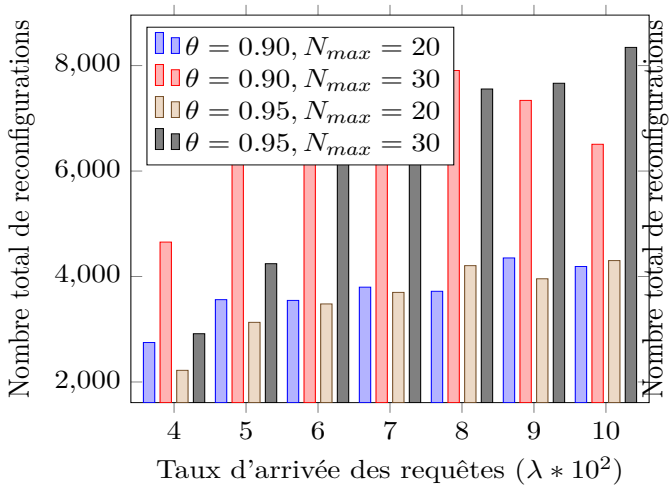
Les mécanismes proposés présentent plusieurs spécificités par rapport aux travaux de la littérature, dont : (1) la défragmentation cible en plus de la bande passante des liens, les ressources de commutation des nœuds ; (2) l'introduction d'une nouvelle métrique appelée indice de fragmentation laquelle est utilisée, soit conjointement avec l'indice de congestion pour affiner l'étape de contrôle de configurations, soit pour détecter le type de ressources concerné par la fragmentation pour s'y adapter par la suite ; (3) l'objectif visé est de réduire le taux d'utilisation maximal tout en économisant les ressources avec la propriété originale de réallouer les liens virtuels dont



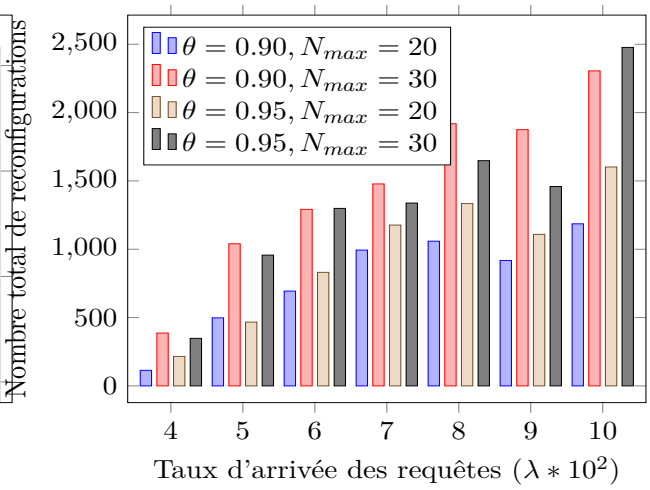
(a) Nombre total de reconfigurations en fonction du taux d'arrivée des requêtes de type **unicast** - **GA-Pro**.



(b) Nombre total de reconfigurations en fonction du taux d'arrivée des requêtes de type **unicast** - **GA-React**.



(c) Nombre total de reconfigurations en fonction du taux d'arrivée des requêtes de type **multicast** - **GA-Pro**.



(d) Nombre total de reconfigurations en fonction du taux d'arrivée des requêtes de type **multicast** - **GA-React**.

FIGURE 4.8 – Nombre total de reconfigurations en fonction du taux d'arrivée des requêtes et des paramètres de simulation.

la migration profitera à toutes les entités les plus chargées, (4) la charge de chaque entité est évaluée en prenant en compte sa sensibilité à la congestion.

Les expérimentations réalisées ont porté sur la topologie réseau GÉANT ainsi que sur une topologie réseau de Campus. Les évaluations se sont focalisées sur des conditions aux limites (avec une forte surcharge), pessimistes par rapport à la réalité. Elles ont permis de vérifier la supériorité de nos mécanismes par rapport à certaines méthodes existantes. Elles ont également

permis de quantifier le gain apporté par nos mécanismes en terme de taux d'acceptation de requêtes ainsi que le nombre de reconfigurations qu'ils induisent. Elles ont aussi permis d'étudier l'influence de certains paramètres, en quantifiant leurs apports respectifs.

Conclusion générale

Problématique et approche générale

Dans ce travail de thèse, nous nous sommes intéressés à la problématique de fourniture des services réseau dynamiques avec garantie de qualité de service (QdS), ciblant les applications ayant des besoins de communication dynamiques en termes, d'une part, des flux de données échangés entre les composants applicatifs (rajout/suppression de flux ou le changement du nombre de destinataires d'un flux), et d'autre part, des exigences en QdS (débit et latence maximale qui changent dans le temps) requises par ces flux.

Afin de répondre à la diversité des besoins de communication des applications ciblées, nous avons fait le choix de décrire le service réseau, support des flux applicatifs, sous forme de réseau overlay applicatif constitué d'un ensemble de liens virtuels de bout en bout pouvant être de type point-à-point et point-à-multipoint, chacun caractérisé par une bande passante et un délai de transfert maximal. La topologie du réseau overlay, les caractéristiques des liens virtuels (bande passante, délai, ou le nombre de points terminaux) peuvent évoluer dans le temps, ce qui confère au service réseau l'aspect dynamique.

Nous avons également considéré le cas où les besoins de communication sont explicitement communiqués au réseau par l'application ou par un agent applicatif, ainsi que le cas où ces besoins sont inférés par le réseau pour pouvoir répondre à la situation où il s'avère difficile à l'application de l'exprimer.

Nous avons dans le cadre de cette thèse développé le concept de réseau ADN, un réseau capable d'offrir des services ADN tels que décrits ci-avant, reposant sur une infrastructure réseau de type SDN (Software Defined Network) ou sur une instance virtuelle d'une telle infrastructure ayant une partie de ses ressources dédiées aux services ADN. En s'appuyant sur un tel type d'infrastructure, un réseau ADN tire partie de sa capacité à être programmée en temps-réels pour rapidement et automatiquement instancier les services réseau et les ajuster de manière à suivre l'évolution des besoins des applications. En partant du fait que les approches courantes ne sont pas adaptées pour répondre à certaines applications notamment les applications d'affaire [2], nous optons pour une approche basée sur une description assez précise des flux applicatifs. L'intérêt d'un tel choix est qu'il permet de fournir aux applications un service réseau qui garantie la QdS à l'échelle d'un flux applicatif élémentaire et en allouant efficacement les ressources réseau.

Il est clair qu'un tel choix pose des problèmes d'échelle et c'est pour cette raison que notre solution ne cible que certains types d'application pour lesquels l'approche se justifie.

Pour y parvenir, une interface est mise à la disposition des applications pour qu'elles puissent explicitement exprimer directement au réseau leurs besoins de communications en indiquant dynamiquement les flux applicatifs qu'elles échangent et en précisant également les exigences

2. populairement connue sous l'appellation "application business"

en QdS requises par ces flux. Pour les applications n'intégrant pas cette capacité, ces flux sont déterminés par le réseau ADN.

Bilan des contributions

Cette partie résume les quatre principales contributions apportées dans cette thèse.

Nous avons proposé **l'architecture d'un réseau ADN** et identifié ses composants fondamentaux. Ces composants ont été décrits à travers une analyse fonctionnelle. Ils sont listés et brièvement décrits ci-après :

- Request Handler : orchestre les autres composants lors du traitement d'une requête de création/modification/suppression d'un service ADN.
- Flow Aggregator : regroupe les flux applicatifs autant que possible pour former les liens virtuels constituant le service ADN à provisionner.
- VNET Resource Allocator : calcule les chemins de données et les ressources à y réserver pour provisionner les services réseau ADN.
- VNET Deployer : assure l'installation/la reconfiguration/la désinstallation effective des service ADN de l'infrastructure.
- Application Classifier : détermine les besoins de communication des applications.
- Autonomic Manager : assure certaines fonctions de gestion pouvant être automatisées telles que l'optimisation des ressources, l'optimisation des paramètres des algorithmes et mécanismes qu'implantent les autres composants.

Le reste des contributions porte sur les algorithmes et les mécanismes qu'implantent certains de ces composants :

VNET Resource Allocator Nous avons proposé un **algorithme d'allocation de ressources** en charge de calculer les chemins de données et les ressources à y réserver pour provisionner les services réseau ADN. Cet algorithme présente plusieurs spécificités par rapport aux travaux de la littérature, dont : la prise en compte (1) des liens point-à-multipoints, (2) de plusieurs critères de QoS, et (3) des ressources de commutation dans le processus d'allocation. Les ressources de commutations sont exprimés en termes d'entrées de la table de flux et de la table de groupe des commutateurs OpenFlow. La prise en compte de ce type de ressources est primordiale puisque avec les technologies actuelles, notamment les mémoires TCAMs, la capacité de ces tables est limitée, car elles coûtent chers, consomment beaucoup d'énergie et dissipent trop de chaleur. L'algorithme proposé est une méthode exacte formulée en utilisant la programmation linéaire en nombres entiers.

Autonomique Manager Nous avons proposé deux heuristiques pour lutter contre le problème de fragmentation de ressources : l'une proactive, déclenchée au départ d'une application de l'infrastructure et l'autre réactive, déclenchée suite à l'échec de traitement d'une demande d'allocation de ressources pour un service ADN. Ces **mécanismes de défragmentation de ressources** présentent plusieurs spécificités par rapport aux travaux de la littérature, dont : (1)

la défragmentation cible en plus de la bande passante des liens, les ressources de commutation des nœuds ; (2) l'introduction d'une nouvelle métrique appelée indice de fragmentation laquelle est utilisée, soit conjointement avec l'indice de congestion pour affiner l'étape de contrôle de configurations, soit pour détecter le type de ressources concerné par la fragmentation pour s'y adapter par la suite ; (3) l'objectif visé est de réduire le taux d'utilisation maximal tout en économisant les ressources avec la propriété originale de réallouer les liens virtuels dont la migration profitera à toutes les entités les plus chargées, (4) la charge de chaque entité est évaluée en prenant en compte sa sensibilité à la congestion.

Perspectives

En résumé, ces travaux de thèse ont permis de poser les bases d'un réseau ADN en définissant son architecture, en proposant des algorithmes efficaces pour ces principaux composants et en développant un démonstrateur preuve de concept pour des applications basées sur le middleware DDS. Plusieurs perspectives se présentent à nous.

Proposer un algorithme efficace d'agrégation de flux afin de permettre au réseau ADN de mieux appréhender les problèmes d'échelle que posent le nombre significatif de flux applicatifs sur les informations d'état à maintenir et surtout sur la taille des tables de flux des commutateurs SDN. La présence d'exigences de QoS associées aux flux complique les possibilités d'agrégation et implique un gaspillage des ressources réseau à allouer au flux agrégé. L'algorithme se devra de calculer des agrégations dont les gains apportés par l'agrégation dépasse le gaspillage de ressources induit et devra certainement adopter des stratégies différentes en fonction de l'état de charge du réseau. L'une des pistes que nous privilégions est d'étendre les techniques de compression des tables de flux pour y inclure les contraintes de qualité de service et la prise en compte des liens point-à-multipoints.

Certaines fonctions réseau ADN ont la possibilité d'exécuter plusieurs algorithmes, de telle sorte que selon la situation (état de charge du réseau, type de requête, etc.), l'algorithme le plus adapté soit utilisé. Pour l'algorithme d'allocation de ressources, nous avons proposé une méthode basée sur de la Programmation Linéaire en Nombres Entiers (PLNE). Cette méthode a montré quelques limites par rapport au temps de calcul lorsque le réseau est faiblement chargé et la requête est constituée d'un nombre important de liens virtuels point-à-multipoints. En complément à cet algorithme, nous finalisons une heuristique basée sur les algorithmes génétiques pour pouvoir répondre efficacement à la situation suscitée avec des temps de calcul inférieurs à la seconde. L'analyse de performances de cet algorithme et l'étude comparative à l'algorithme PLNE pour identifier les meilleurs conditions d'utilisation de chaque algorithme sont une perspective à court terme de ce travail de thèse.

La sélection de l'algorithme à utiliser et l'ajustement des valeurs des paramètres de chaque algorithme sont à la charge du composant « Autonomic Manager » qui est supposé, pour une situation identifiée, procéder à la sélection des algorithmes appropriés et de leurs paramètres. La définition des politiques permettant au « manager autonomique » de déduire ou inférer ses

choix n'a pas été explorée par les travaux de cette thèse. Elle constitue donc une perspective importante à ce travail.

Le réseau ADN se veut capable d'inférer et de suivre les besoins en communication des applications, si ces dernières ne sont pas en mesure de les signaler au réseau. Le composant « Application Classifier » repose sur des sondes d'inspection profonde de paquets pour réaliser cette fonction. Deux types d'applications ont été considérés dans ces travaux comme cas d'étude : les applications bâtis sur l'intergiciel DDS et les applications exploitant le protocole SIP. Dans les deux cas de figure, les flux et les exigences de QdS des applications peuvent être directement déduits de certains paquets de contrôle (DDS et SIP) qu'il suffit de capturer et inspecter. Une perspective est d'adresser le cas où les besoins en communication ne sont pas exprimés et donc ne peuvent être extraits d'un message quelqu'il soit. Le but est d'explorer et d'évaluer la précision des techniques d'analyse comportementale du trafic et des analyses statistiques de l'inspection profonde des paquets pour classer les application et inférer leurs exigences de QdS.

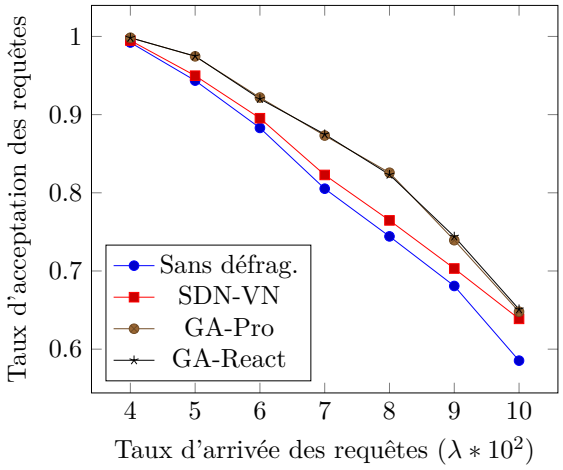
Une dernière perspective est d'étendre le modèle de service ADN pour lui permettre de supporter, en complément aux liens virtuels, des chaînages de fonctions réseau virtuelles qu'emprunteront certains flux de paquets.

Annexe A

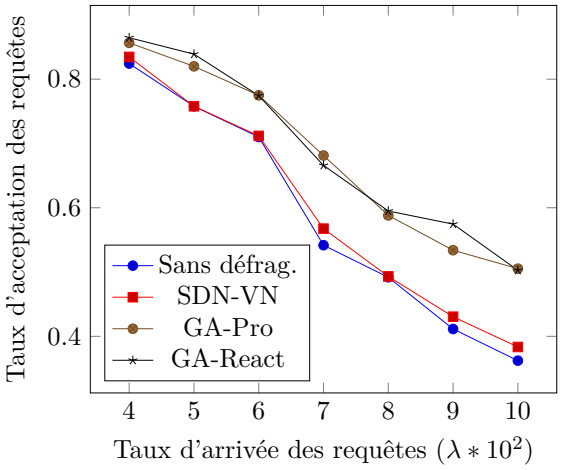
Mécanismes de défragmentation de ressources

A.1 Résultats expérimentaux sur le réseau de Campus

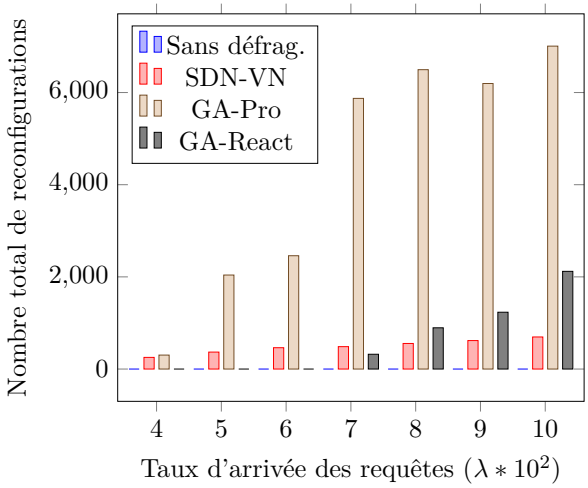
Cette section présente la synthèse des résultats d'évaluation sur le réseau de Campus.



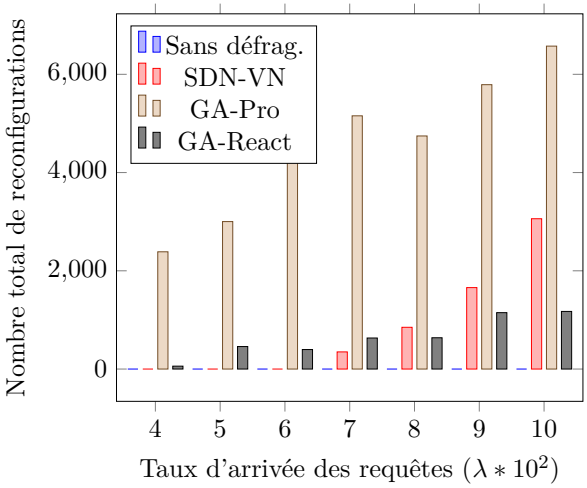
(a) Requêtes de type Unicast



(b) Requêtes de type Multicast

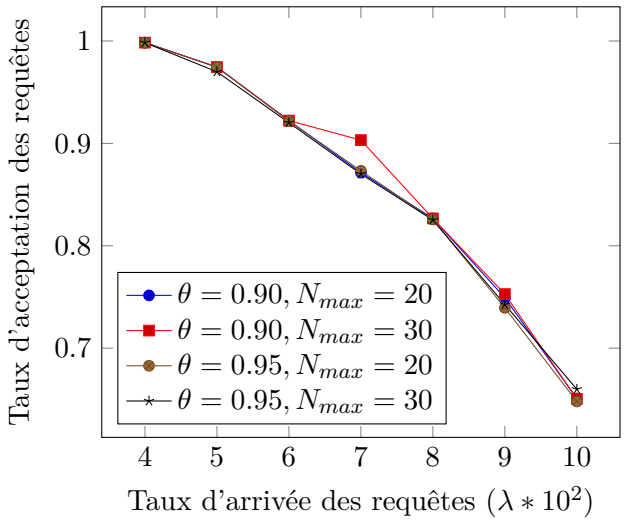


(c) Requêtes de type unicast

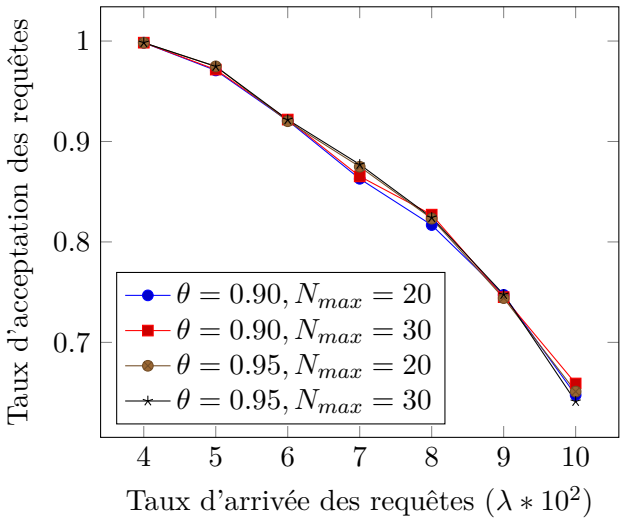


(d) Requêtes de type multicast

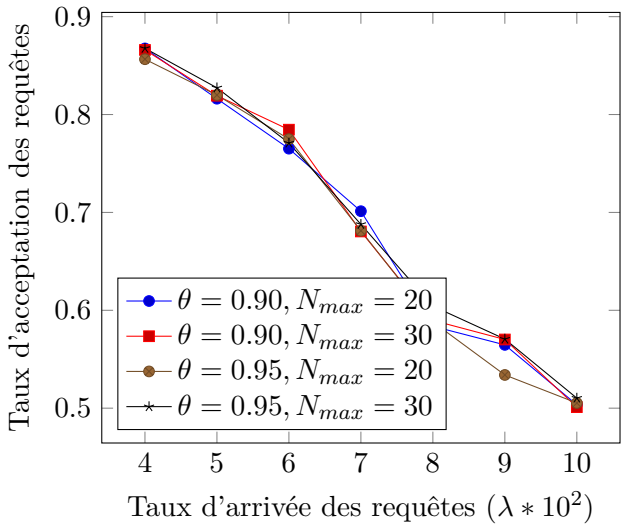
Figure A.1 – Comparaison de différents algorithmes de défragmentation



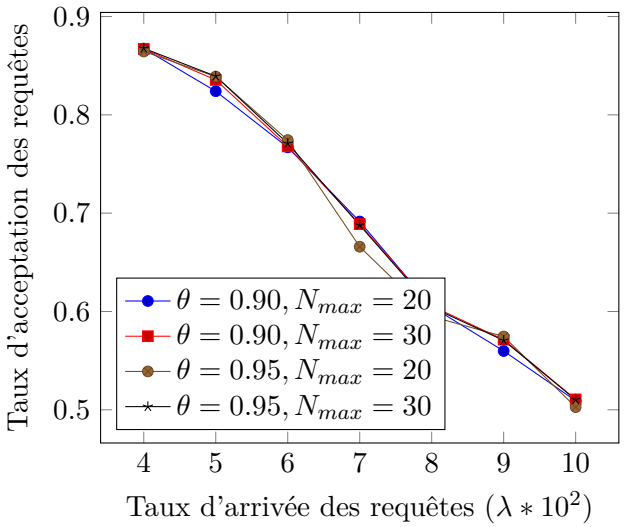
(a) Taux d'acceptation des requêtes de type **unicast** en fonction du taux d'arrivée - **proactive**.



(b) Taux d'acceptation des requêtes de type **unicast** en fonction du taux d'arrivée - **reactive**.

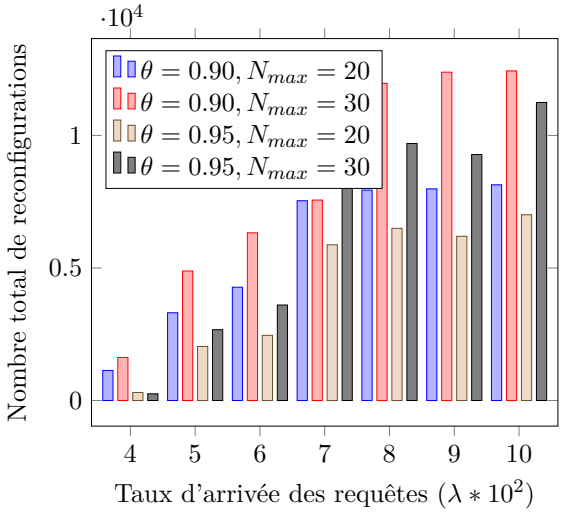


(c) Taux d'acceptation des requêtes de type **multicast** en fonction du taux d'arrivée - **proactive**.

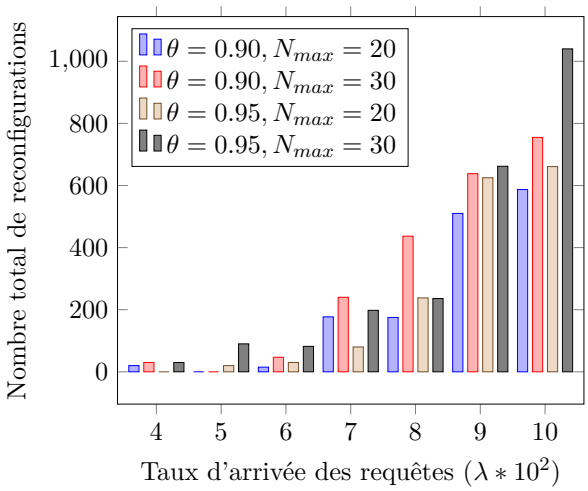


(d) Taux d'acceptation des requêtes de type **multicast** en fonction du taux d'arrivée - **reactive**.

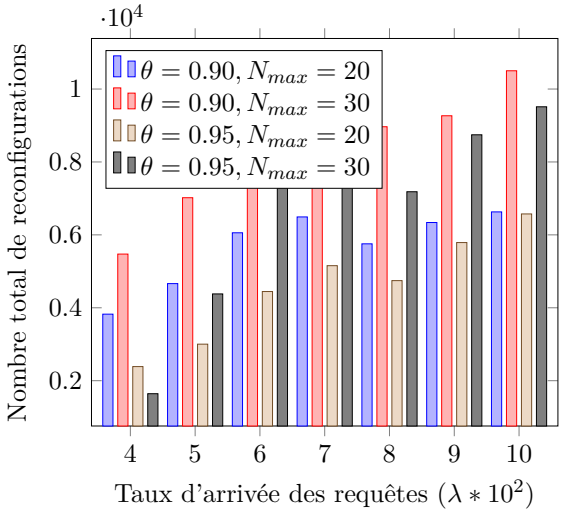
FIGURE A.2 – Taux d'acceptation des requêtes en fonction de leur taux d'arrivée et des paramètres de simulation.



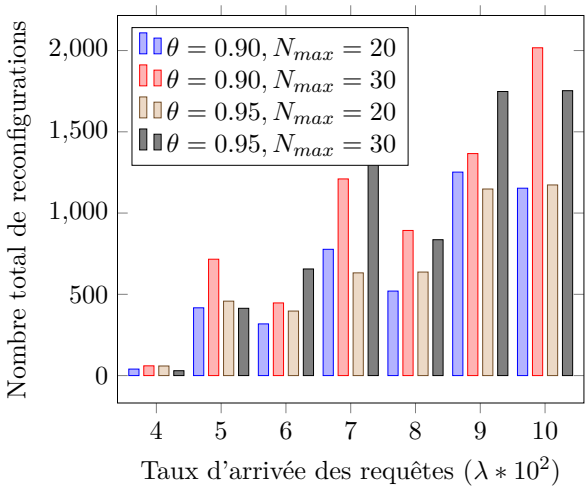
(a) Nombre total de reconfigurations en fonction du taux d'arrivée des requêtes de type **unicast - proactive**.



(b) Nombre total de reconfigurations en fonction du taux d'arrivée des requêtes de type **unicast - reactive**.



(c) Nombre total de reconfigurations en fonction du taux d'arrivée des requêtes de type **multicast - proactive**.



(d) Nombre total de reconfigurations en fonction du taux d'arrivée des requêtes de type **multicast - reactive**.

FIGURE A.3 – Nombre total de reconfigurations en fonction du taux d'arrivée des requêtes et des paramètres de simulation.

Annexe B

Démonstrateur ADN

B.1 Introduction

L'application considérée est une application de simulation distribuée impliquant plusieurs simulateurs de véhicule dirigés par des apprenants qui participent à un exercice ou une formation à la conduite en groupe inspirée d'une des solutions commercialisées par ECA FAROS (voir figure B.1). Des postes instructeurs recueillent les données de certains simulateurs et les renvoient à l'écran pour le compte d'instructeurs en charge de superviser la formation. Enfin, un serveur de simulation récupère et archive l'ensemble des données des simulateurs pour un éventuel rejeu de la simulation. Les instructeurs ont la possibilité d'établir des communications audio avec les apprenants. Ils ont également la possibilité d'activer l'envoi d'un flux vidéo depuis une caméra localisée sur le poste simulateur de véhicule et qui filme l'apprenant vers le poste instructeur.

L'application considérée repose sur l'intergiciel DDS (Data Distribution Service) de l'OMG (Object Management Group) pour distribuer les flux de données applicatives relatifs à la simulation. Les échanges audio et vidéo ne reposent pas sur DDS. Pour cette raison, nous nous focaliserons dans la suite sur les échanges de données en lien avec la simulation. Étant donnée, l'impossibilité de redévelopper nos propositions à partir des simulateurs ECA Faros, le prototype s'appuiera sur des pseudo-simulateurs (ou des émulateurs) qui chercheront à reproduire les données réelles envoyés par ces simulateurs sur le réseau.

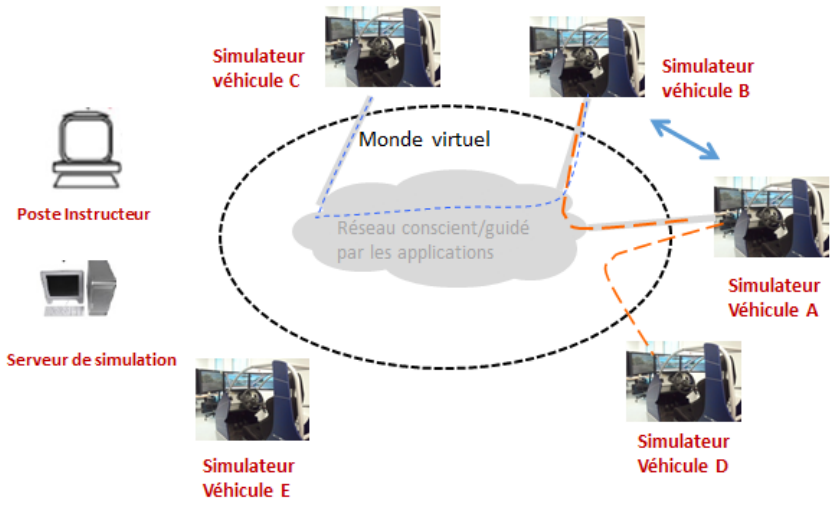


Figure B.1 – Application de simulation considérée

Le démonstrateur :

- Implique 5 pseudo-simulateurs de véhicule, un poste instructeur et un serveur de simulation ;
- Reposera sur les données issues des simulateurs de véhicule commercialisés par ECA Faros. Ces données sont disponibles au LAAS et concernent certains simulateurs commercialisés en 2011 ;
- Le réseau support considéré sera un réseau de campus équivalent à celui du LAAS/CNRS ;
- Le scénario de simulation qui sera considéré suppose un modèle de mobilité des simulateurs plutôt déterministe basé sur un plan routier (autour du LAAS) dans lequel les pseudo-simulateurs évolueront ;
- La définition des besoins instantanés en QdS relatifs à la distribution des données des pseudo-simulateurs reposera sur la comparaison de la distance entre pseudo-simulateurs à des seuils préétablis. Le choix de ces seuils ne sera pas motivé (faute d'expertise métier sur ces aspects).
- Le paramètre vitesse de certains pseudo-simulateurs pourra être varié afin de considérer plusieurs scénarios applicatifs.

Le plan de cette annexe se présente comme suit. Nous commencerons par décrire l'application DDS considérée en explicitant ses flux de données (via la définition des topics DDS) et leurs besoins en QdS. Nous décrirons ensuite l'infrastructure réseau considérée et sa mise en œuvre dans le démonstrateur. Enfin, nous préciserons le scénario applicatif de démonstration, suivi, des résultats ou observations qui illustrent le fonctionnement du réseau ADN.

B.2 Application dynamique DDS considérée

Les données échangées entre les différents acteurs de l'application (simulateurs de véhicule, poste instructeur et serveur de simulation) reprennent les données des simulateurs de véhicules produits par ECA Faros. Elles sont principalement composées de trois types : des données de simulation, des données pédagogiques et données de contrôle. Elles sont organisées en 16 Topics DDS (voir figure B.2).

B.2.1 Description des topics

La figure B.2 recense le nom des différents topics DDS de l'application en les organisant selon leur profil : périodiques, apériodiques, ou ponctuels et éphémères car échangés à l'initialisation de l'application. Comme indiqué dans la section B.2.2, seuls quelques uns de ces topics ont leurs QdS qui évoluent dans le temps, conférant ainsi l'aspect dynamique à l'application considérée. Ils sont décrits ci-après.

- **Position** : C'est un topic qui montre la position de chaque simulateur au cours de la simulation. Ce topic sera échangé entre les simulateurs afin que chaque simulateur connaisse la position relative des autres simulateurs.
- **EtatsVéhiculeSim** : Ce topic contient les données d'état des véhicules simulés (sa position, sa vitesse) participant à l'exercice de simulation en question. Ces données sont échangées régulièrement afin d'avoir un état global du système. Ce topic est produit par

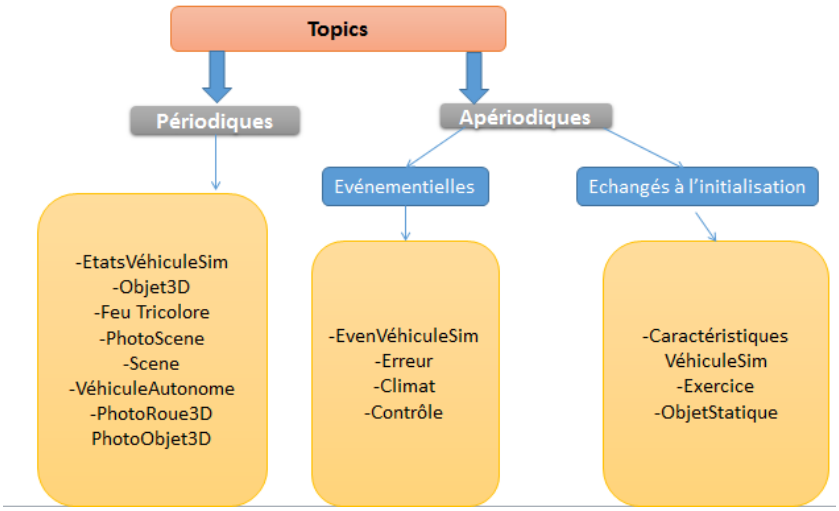


FIGURE B.2 – Topics DDS relatifs à l'application considérée

chaque simulateur pour d'autres simulateurs, le serveur de simulation et le poste instructeur.

- **EvenVéhiculeSim** : Ce topic est échangé suite à un évènement sur un véhicule simulé appartenant à l'exercice (exemple panne). Ce topic est produit par les simulateurs et consommé par les autres simulateurs, le serveur de simulation et le poste instructeur.
- **PhotoRoue3D** : Ce topic permet d'avoir régulièrement les données 3D d'état de chaque roue de chaque véhicule simulé. Ce topic est échangé entre les simulateurs.

B.2.2 Composante dynamique de l'application considérée

L'aspect dynamique de l'application considérée concerne l'évolution des abonnements/désabonnements (et des besoins en QdS associés) aux données transportant les variables d'état échangées entre les simulateurs. Ceux-ci doivent s'échanger en plus de leur position, leurs informations d'états, à une certaine fréquence (notée f ci-avant) et avec un délai plus ou moins contraignant, selon leur distance relative. En effet la logique applicative suppose que, plus les simulateurs sont proches, plus ils peuvent mutuellement influencer le comportement des apprenants. Ainsi, afin de restituer au mieux un rendu réaliste, une fine précision de leurs informations d'état les plus importantes est requise dans les plus brefs délais pour permettre aux apprenants d'être rapidement mis en situation et d'exécuter les actions recommandées par l'apprentissage. C'est ce mode de consommation de ces informations d'état qui confère à cette application le caractère dynamique dans la mesure où les exigences d'acheminement liées à cet échange évoluent au cours du temps en fonction de la distance inter simulateur.

Faute d'expertise métier sur la manière de choisir la fréquence de distribution en fonction de la distance entre simulateurs, les choix présentés dans ce paragraphe ne seront pas motivés. Elles correspondent à une mise en situation des apprenants en milieu urbain avec comme hypothèse sous-jacente que les véhicules simulés auront des difficultés pour dépasser une vitesse de 100

Distance relative	**]0 à 25m]**	**]25m à 50m]**
Fréquence de production f (paramètre DDS : Deadline)	40 Hz	20 Hz
Délai de distribution maximal (paramètre DDS : Latency_budget)	25 ms	50 ms

TABLE B.1 – Caractérisation de la dynamique de l'application

Km/h. Le tableau B.1 explicite la fonction que nous utilisons pour déduire les besoins en QdS des flux de données relatifs aux variables d'état des simulateurs. Ces derniers sont définis par les topics DDS : *Position*, *EtatsVehiculeSim*, *EvenVéhiculeSim* et *PhotoRoue3D*. Au delà de 50 m de distance, les simulateurs ne s'échangent plus leurs variables d'état (seul le topic *Position* est échangé à une fréquence f' de 10 Hz). Les autres données périodiques seront transmises à une fréquence fixe de 10 Hz. Pour les variables événementielles, elles seront supportées au niveau réseau par un même lien virtuel avec des besoins de QdS statiques.

B.3 Infrastructure réseau considérée

L'infrastructure réseau considérée pour ce démonstrateur reprend le réseau du campus du LAAS. C'est la volonté d'avoir une topologie réseau réaliste qui nous amène à l'infrastructure réseau de la figure B.3. Nous retrouvons une topologie hiérarchique avec redondance, classique des réseaux d'entreprise.

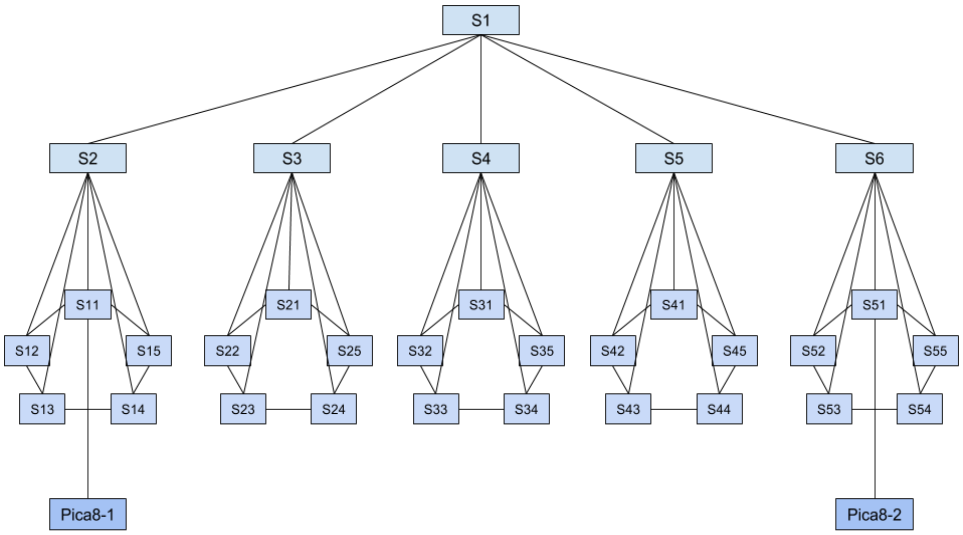


FIGURE B.3 – Topologie du réseau de campus du LAAS

Le démonstrateur n'utilisera pas le réseau du LAAS mais ce dernier sera émulé en utilisant l'émulateur "Mininet" connecté à trois commutateurs OpenFlow Pica8 P-3295. Pour les

nœuds émulés avec Mininet, ils seront de type OVS car ils implémentent les groupes OpenFlow de type *select* et *all*. OVS offre également un support expérimental à OpenFlow 1.4 avec la possibilité d'utiliser les *Bundles* OpenFlow. Les caractéristiques évoquées ci-avant sont des pré-requis pour l'algorithme du « VNET Deployer ». Au moment du démarrage des développements (juillet 2015), OVS était le seul commutateur logiciel OpenFlow à les supporter et qui proposait une intégration assez stable avec les contrôleurs SDN existants, notamment avec le contrôleur "Floodlight". La plateforme mise en place est décrite dans la figure B.4.

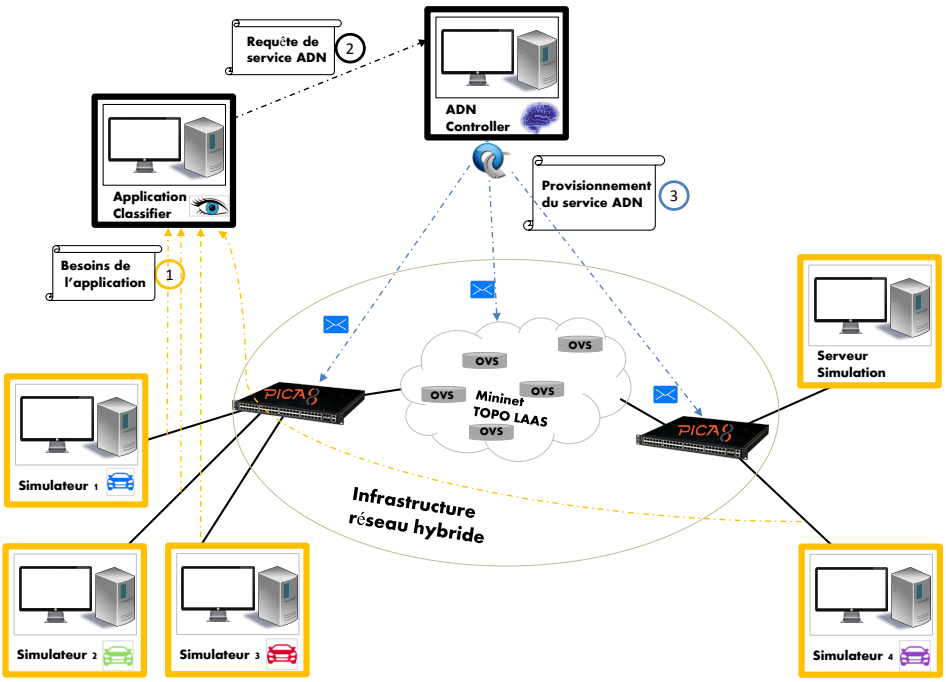


FIGURE B.4 – Plateforme mise en place

B.4 Scénario applicatif et résultats obtenus

B.4.1 Description du scénario applicatif

L'objectif du scénario considéré est d'illustrer le fonctionnement du réseau ADN, notamment, en cas de changement de besoins applicatifs. Quatre cas de figure illustrant ces changements peuvent être considérés : rajout de flux applicatifs, changement des besoins en QdS de flux applicatifs vers des garanties plus strictes, changement des besoins en QdS à la baisse et le retrait de flux. Même si l'implémentation du démonstrateur considère tous ces cas de figure, seul le *rajout de flux applicatif* sera illustré dans la suite de ce manuscrit.

Le scénario utilise le plan routier autour du LAAS avec 3 simulateurs de véhicule S1, S2 et S3 qui se déplacent en cortège tout au long de l'exécution du scénario. Comme décrit ci-après, un quatrième simulateur S4 viendra à la rencontre du cortège, parcourra un petit trajet à leur

proximité avant de s'en éloigner. Tout au long de l'exécution du scénario, les flux de données échangés entre les simulateurs S1, S2, S3 auront des besoins inchangés. Il en est de même pour les flux de données qu'ils échangent avec le serveur de simulation et le poste instructeur. La dynamique de l'application concernera les flux de données échangées entre le simulateur S4 et les simulateurs du cortège.

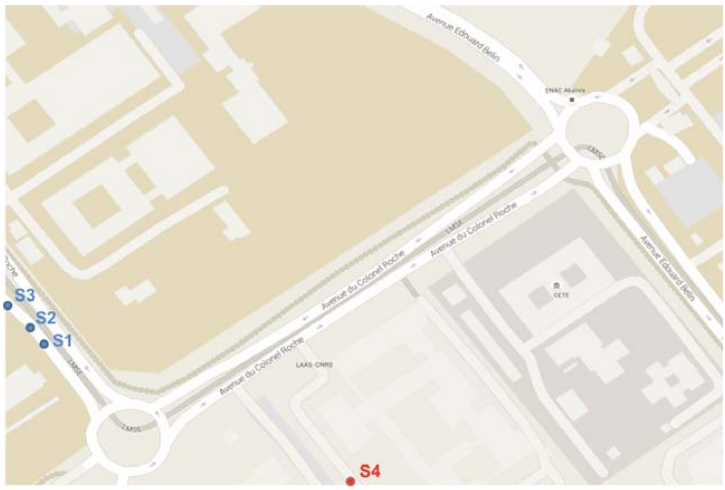


FIGURE B.5 – Illustation du scénario à l'intant t0

A l'instant t0, le simulateur S4 est à une distance supérieure à 50 m du cortège. S4 n'est pas abonné aux variables d'état des simulateurs S1, S2 et S3 (et inversement). En revanche les simulateurs S1, S2 et S3 sont mutuellement abonnés à leurs informations d'états.

A l'instant t1, le simulateur S4 se trouve à une distance du cortège qui est légèrement inférieure à 50 m (voir Figure 24). Une phase de souscription aux topics *EtatsVehiculeSim* et *PhotoRoue3D* des simulateurs S1, S2 et S3 s'engage. Similairement, les simulateurs S1, S2 et S3 s'abonnent à leurs tours aux données des mêmes topics issues de S4. Cela se traduit au niveau du réseau ADN par la réception :

- d'une requête VNET de rajout d'un lien virtuel correspondant aux données de type *Etats-VehiculeSim* produites par S4 et deux mises à jour de liens virtuels en y intégrant de nouvelles destinations relatives aux nœuds réseau de S1, S2, S3. Ces demandes émanent du « Application Classifier » et sont traités par le « Requets Handler » qui après avoir sollicité le « Flow Aggregator » soumet les requêtes au » VNET Resource Allocator ».
- Similairement, et après l'intervention des composants « Application Classifier » et « Requets Handler », trois requêtes de mise à jour de liens virtuels sont soumises au « VNET Resource Allocator » pour inclure le simulateur S4 comme réceptionnaire des variables d'état des simulateurs S1, S2 et S3.



FIGURE B.6 – Illustation du scénario à l'intant t1

B.4.2 Résultats du démonstrateur

La figure B.7 et la figure B.8 illustrent le résultat des allocations des ressources sur l'infrastructure réseau considérée pour les différents liens virtuels à provisionner aux instants t0 et t1 respectivement.

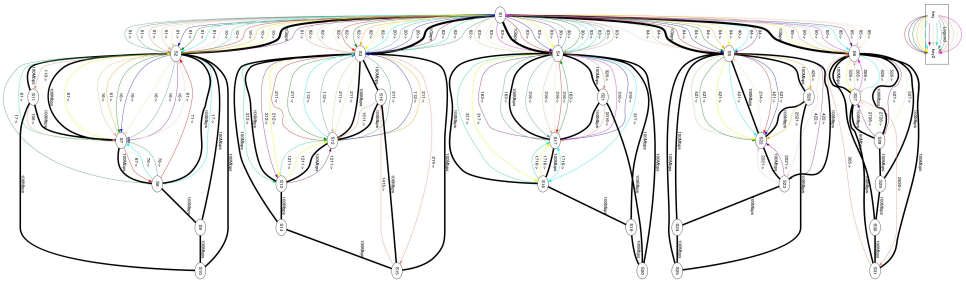


FIGURE B.7 – Résultats allocations à l'instant t0

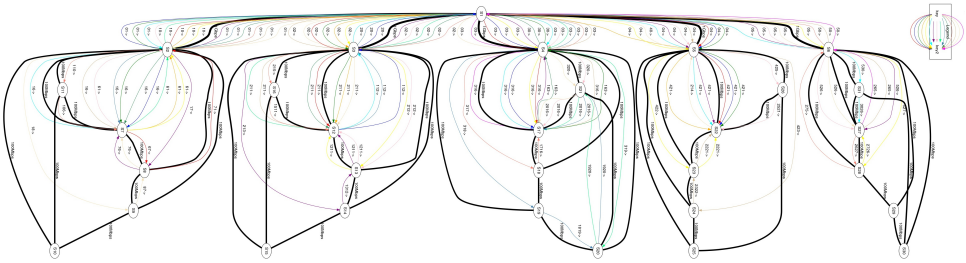


FIGURE B.8 – Résultats allocations à l'instant t1

B.5 Conclusion

Cette annexe a décrit le démonstrateur ADN qui vise à illustrer le service qu'offre un réseau ADN à une application DDS dynamique. Ce démonstrateur illustre, notamment, le niveau de finesse pouvant être atteint par l'approche ADN dans la prise en compte des besoins applicatifs. C'est l'un des points caractéristiques de l'approche ADN, même si celle-ci, est également applicable dans un contexte plus général qui n'impose pas une telle description des besoins applicatifs.

Bibliographie

[Abbes 2004] T. Abbes, A. Bouhoula et M. Rusinowitch. *Protocol analysis in intrusion detection using decision tree.* Dans International Conference on Information Technology : Coding and Computing, 2004. Proceedings. ITCC 2004., volume 1, pages 404–408 Vol.1, April 2004. (Cité en page 46.)

[Accenture 2015] Accenture. *The future of applications.* White paper, June 2015. (Cité en page 1.)

[Bays 2016] L. R. Bays, L. P. Gaspary, R. Ahmed et R. Boutaba. *Virtual network embedding in software-defined networks.* Dans NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium, pages 10–18, April 2016. (Cité en page 62.)

[Blenk 2016] A. Blenk, A. Basta, M. Reisslein et W. Kellerer. *Survey on Network Virtualization Hypervisors for Software Defined Networking.* IEEE Communications Surveys Tutorials, vol. 18, no. 1, pages 655–685, Firstquarter 2016. (Cité en page 97.)

[Capgemini 2017] Capgemini. *Le Cloud combiné à l'IoT, le Big Data et l'IA, révolutionne l'économie.* White paper, July 2017. (Cité en page 1.)

[Choi 2016] Hyon-Young Choi, A. L. King et I. Lee. *Making DDS really real-time with OpenFlow.* Dans 2016 International Conference on Embedded Software (EMSOFT), pages 1–10, Oct 2016. (Cité en page 23.)

[Chowdhury 2009] N. M. M. K. Chowdhury, M. R. Rahman et R. Boutaba. *Virtual Network Embedding with Coordinated Node and Link Mapping.* Dans IEEE INFOCOM 2009, pages 783–791, April 2009. (Cité en pages 58, 59 et 60.)

[Chowdhury 2012] M. Chowdhury, M. R. Rahman et R. Boutaba. *ViNEYard : Virtual Network Embedding Algorithms With Coordinated Node and Link Mapping.* IEEE/ACM Transactions on Networking, vol. 20, no. 1, pages 206–219, Feb 2012. (Cité en page 58.)

[Cisco 2013] Cisco. *Application Centric Infrastructure (ACI).* White paper, June 2013. (Cité en page 17.)

[Citrix 2015] Citrix et Cisco. *Agile application-driven networks with Cisco ACI and NetScaler.* White paper, June 2015. (Cité en page 18.)

[Claise 2004] Benoit Claise. *Cisco Systems NetFlow Services Export Version 9.* RFC 3954, oct 2004. (Cité en page 46.)

[Computing 2006] Aaron Computing. *An architectural blueprint for autonomic computing.* Technical report, 2006. (Cité en page 48.)

[Estrada 2010] Ernesto Estrada et Desmond J. Higham. *Network Properties Revealed through Matrix Functions.* SIAM Review, vol. 52, no. 4, pages 696–714, jan 2010. (Cité en pages 102 et 107.)

[Fajjari 2011] I. Fajjari, N. Aitsaadi, G. Pujolle et H. Zimmermann. *VNR Algorithm : A Greedy Approach for Virtual Networks Reconfigurations.* Dans Global Telecommunications Conference (GLOBECOM 2011), 2011 IEEE, pages 1–6, Dec 2011. (Cité en pages 99, 102 et 104.)

[Fallon 2011] L. Fallon et Y. Huang. *GSQR : A generic service quality reporting protocol for terminals.* Dans 12th IFIP/IEEE International Symposium on Integrated Network Management (IM 2011) and Workshops, pages 1122–1128, May 2011. (Cité en page 46.)

[Farooq Butt 2010] Nabeel Farooq Butt, Mosharaf Chowdhury et Raouf Boutaba. Topology-awareness and reoptimization mechanism for virtual network embedding, pages 27–39. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010. (Cité en pages 102 et 103.)

[Ferguson 2013] Andrew D. Ferguson, Arjun Guha, Chen Liang, Rodrigo Fonseca et Shriram Krishnamurthi. *Participatory Networking : An API for Application Control of SDNs.* SIGCOMM Comput. Commun. Rev., vol. 43, no. 4, pages 327–338, août 2013. (Cité en page 24.)

[Fielder 2012] Justin Fielder et Thierry Grenot. *Killer Apps 2012.* Research study, 2012. (Cité en page 2.)

[Fischer 2013] A. Fischer, J. F. Botero, M. T. Beck, H. de Meer et X. Hesselbach. *Virtual Network Embedding : A Survey.* IEEE Communications Surveys Tutorials, vol. 15, no. 4, pages 1888–1906, Fourth 2013. (Cité en page 101.)

[Frei 2005] C. Frei, Boi Faltings et M. Hamdi. *Resource allocation in communication networks using abstraction and constraint satisfaction.* IEEE Journal on Selected Areas in Communications, vol. 23, no. 2, pages 304–320, Feb 2005. (Cité en pages 59 et 60.)

[Halpern 2010] Joel M. Halpern, Robert HAAS, Avri Doria, Ligang Dong, Weiming Wang, Hormuzd M. Khosravi, Jamal Hadi Salim et Ram Gopal. *Forwarding and Control Element Separation (ForCES) Protocol Specification.* RFC 5810, mars 2010. (Cité en page 6.)

[He 2008] Jiayue He, Rui Zhang-shen, Ying Li, Cheng yen Lee, Jennifer Rexford et Mung Chiang. *DaVinci : Dynamically Adaptive Virtual Networks for a Customized Internet.* Dans in Proc. CoNEXT, 2008. (Cité en pages 102 et 104.)

[Heckerman 2013] David Heckerman, Dan Geiger et David Maxwell Chickering. *Learning Bayesian Networks : The Combination of Knowledge and Statistical Data.* CoRR, vol. abs/1302.6815, 2013. (Cité en page 46.)

[Horn 2001] Paul Horn. *Autonomic Computing : IBM's Perspective on the State of Information Technology.* Technical report, 2001. (Cité en page 48.)

[Hsu 2012] W. H. Hsu, Y. P. Shieh, C. H. Wang et S. C. Yeh. *Virtual Network Mapping through Path Splitting and Migration.* Dans 2012 26th International Conference on Advanced Information Networking and Applications Workshops, pages 1095–1100, March 2012. (Cité en pages 58 et 60.)

[Jin 2014] Xin Jin, Hongqiang Harry Liu, Rohan Gandhi, Srikanth Kandula, Ratul Mahajan, Ming Zhang, Jennifer Rexford et Roger Wattenhofer. *Dynamic Scheduling of Network Updates*. SIGCOMM Comput. Commun. Rev., vol. 44, no. 4, pages 539–550, août 2014. (Cité en page 102.)

[Kephart 2003] J. O. Kephart et D. M. Chess. *The vision of autonomic computing*. Computer, vol. 36, no. 1, pages 41–50, Jan 2003. (Cité en page 48.)

[Kerravala 2015] Zeus Kerravala. *Top Five Considerations for Building a Cloud-Ready Network for Distributed Enterprises*. Research study, January 2015. (Cité en page 2.)

[King 2014] A. L. King, S. Chen et I. Lee. *The MIDdleware Assurance Substrate : Enabling Strong Real-Time Guarantees in Open Systems with OpenFlow*. Dans 2014 IEEE 17th International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing, pages 133–140, June 2014. (Cité en page 23.)

[Kucharzak 2012] M. Kucharzak et K. Walkowiak. *Maximum flow trees in overlay multicast : Modeling and optimization*. Dans 2012 2nd Baltic Congress on Future Internet Communications, pages 260–267, April 2012. (Cité en pages 59 et 60.)

[Li 2014] Y. Li et J. Li. *MultiClassifier : A combination of DPI and ML for application-layer classification in SDN*. Dans The 2014 2nd International Conference on Systems and Informatics (ICSAI 2014), pages 682–686, Nov 2014. (Cité en page 46.)

[Lischka 2009] Jens Lischka et Holger Karl. *A Virtual Network Mapping Algorithm Based on Subgraph Isomorphism Detection*. Dans Proceedings of the 1st ACM Workshop on Virtualized Infrastructure Systems and Architectures, VISA '09, pages 81–88, New York, NY, USA, 2009. ACM. (Cité en pages 54, 58 et 60.)

[Liu 2013] Hongqiang Harry Liu, Xin Wu, Ming Zhang, Lihua Yuan, Roger Wattenhofer et David A. Maltz. *zUpdate : updating data center networks with zero loss*. Dans ACM SIGCOMM 2013 Conference, SIGCOMM'13, Hong Kong, China, August 12-16, 2013, pages 411–422, 2013. (Cité en page 102.)

[Masri 2011] H. Masri, S. Krichen et A. Guitouni. *An ant colony optimization metaheuristic for solving bi-objective multi-sources multicommodity communication flow problem*. Dans 2011 4th Joint IFIP Wireless and Mobile Networking Conference (WMNC 2011), pages 1–8, Oct 2011. (Cité en pages 59 et 60.)

[McKeown 2008] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker et Jonathan Turner. *OpenFlow : Enabling Innovation in Campus Networks*. SIGCOMM Comput. Commun. Rev., vol. 38, no. 2, pages 69–74, mars 2008. (Cité en page 6.)

[Melo 2013] M. Melo, J. Carapinha et S. Sargento. *Network Virtualization : A step closer for seamless resource mobility*. Dans 2013 IFIP/IEEE International Symposium on Integrated Network Management (IM 2013), pages 692–695, May 2013. (Cité en pages 57, 58 et 60.)

[Mijumbi 2014] R. Mijumbi, J. Serrat, J. Rubio-Loyola, N. Bouten, F. D. Turck et S. Latré. *Dynamic resource management in SDN-based virtualized networks.* Dans 10th International Conference on Network and Service Management (CNSM) and Workshop, pages 412–417, Nov 2014. (Cité en pages 97, 99, 102, 103, 104 et 118.)

[Monsanto 2012] Christopher Monsanto, Nate Foster, Rob Harrison et David Walker. *A Compiler and Run-time System for Network Programming Languages.* SIGPLAN Not., vol. 47, no. 1, pages 217–230, janvier 2012. (Cité en page 10.)

[Monsanto 2013] Christopher Monsanto, Joshua Reich, Nate Foster, Jennifer Rexford et David Walker. *Composing Software Defined Networks.* Dans 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13), pages 1–13, Lombard, IL, 2013. USENIX Association. (Cité en page 33.)

[Montazerolghaem 2017] Ahmad Reza Montazerolghaem, Mohammad Hossein Yaghmaee Moghaddam et Alberto Leon-Garcia. *OpenSIP : Toward Software-Defined SIP Networking.* CoRR, vol. abs/1709.01320, 2017. (Cité en page 22.)

[MRV 2013] MRV. *Application-Aware Networking at A Glance.* While paper, 2013. (Cité en page 2.)

[Nakagawa 2013] Yukihiro Nakagawa, Kazuki Hyoudou, Chunghan Lee, Shinji Kobayashi, Osamu Shiraki et Takeshi Shimizu. *DomainFlow : practical flow management method using multiple flow tables in commodity switches.* Dans Conference on emerging Networking Experiments and Technologies, CoNEXT '13, Santa Barbara, CA, USA, December 9-12, 2013, pages 399–404, 2013. (Cité en page 62.)

[Nogueira 2011] J. Nogueira, M. Melo, J. Carapinha et S. Sargento. *Virtual network mapping into heterogeneous substrate networks.* Dans 2011 IEEE Symposium on Computers and Communications (ISCC), pages 438–444, June 2011. (Cité en pages 58 et 60.)

[Nunes 2014] B. A. A. Nunes, M. Mendonca, X. N. Nguyen, K. Obraczka et T. Turletti. *A Survey of Software-Defined Networking : Past, Present, and Future of Programmable Networks.* IEEE Communications Surveys Tutorials, vol. 16, no. 3, pages 1617–1634, Third 2014. (Cité en pages ix et 6.)

[OMG 2007] OMG. *The Data Distribution Service Specification*, 2007. (Cité en page 47.)

[ONF 2012] Open Networking Foundation ONF. *Software-Defined Networking : The New Norm for Networks.* Rapport technique, 2012. (Cité en page 5.)

[ONF 2013a] Open Networking Foundation ONF. *OpenFlow Switch Specification – version 1.4.0 (Wire Protocol 0x05).* Rapport technique, October 2013. (Cité en pages ix, 6, 7, 8 et 9.)

[ONF 2013b] Open Networking Foundation ONF. *SDN Architecture Overview.* Rapport technique, December 2013. (Cité en pages ix et 10.)

[ONF 2013c] Open Networking Foundation ONF. *Software-Defined Networking : The New Norm for Networks*. Rapport technique, April 2013. (Cité en pages ix et 7.)

[Panchen 2001] Sonia Panchen, Neil McKee et Peter Phaal. *sFlow : A Method for Monitoring Traffic in Switched and Routed Networks*. RFC 3176, sep 2001. (Cité en page 46.)

[Qazi 2013] Zafar Ayyub Qazi, Jeongkeun Lee, Tao Jin, Gowtham Bellala, Manfred Arndt et Guevara Noubir. *Application-awareness in SDN*. SIGCOMM Comput. Commun. Rev., vol. 43, no. 4, pages 487–488, août 2013. (Cité en pages 21 et 46.)

[Rathore 2010] M. S. Rathore, M. Hidell et P. Sjödin. *Performance evaluation of open virtual routers*. Dans 2010 IEEE Globecom Workshops, pages 288–293, Dec 2010. (Cité en page 56.)

[Santos 2017] Ricardo Santos, Zdravko Bozakov, Simone Mangiante, Anna Brunstrom et Andreas Kassler. *A NEAT framework for application-awareness in SDN environments*. 2017. (Cité en page 22.)

[Schaffrath 2012] G. Schaffrath, S. Schmid et A. Feldmann. *Optimizing Long-Lived CloudNets with Migrations*. Dans 2012 IEEE Fifth International Conference on Utility and Cloud Computing, pages 99–106, Nov 2012. (Cité en page 97.)

[Singh 2018] Varun Singh, Rachel Huang, Roni Even, Dan Romascanu et Deng Lingli. *Considerations for Selecting RTCP Extended Report (XR) Metrics for the WebRTC Statistics API*. Internet-Draft draft-ietf-xrblock-rtcweb-rtcp-xr-metrics-08, Internet Engineering Task Force, février 2018. Work in Progress. (Cité en page 47.)

[Sinnreich 2010] Henry Sinnreich, Amy Pendleton, Alan Clark et Alan Johnston. *Session Initiation Protocol Event Package for Voice Quality Reporting*. RFC 6035, nov 2010. (Cité en page 46.)

[Solaris 2015] Solaris et Pluribus Networks. *Enabling Application-Driven SDN in the Cloud*. White paper, March 2015. (Cité en pages 19 et 21.)

[Tegueu 2017a] A. F. Simo Tegueu, S. Abdellatif, T. Villemur et P. Berthou. *A dynamic resource defragmentation scheme for virtualized SDN-enabled networks*. Dans 2017 IEEE 6th International Conference on Cloud Networking (CLOUDNET), pages 1–6, Sept 2017. (Cité en pages 104 et 106.)

[Tegueu 2017b] A. F. Simo Tegueu, S. Abdellatif, T. Villemur et P. Berthou. *A reactive resource defragmentation method for Virtual Links Mapping in software-defined networks*. Dans 2017 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN), pages 1–6, June 2017. (Cité en pages 104 et 111.)

[Tran 2012] Phuong Nga Tran, L. Casucci et A. Timm-Giel. *Optimal mapping of virtual networks considering reactive reconfiguration*. Dans 2012 IEEE 1st International Conference on Cloud Networking (CLOUDNET), pages 35–40, Nov 2012. (Cité en pages 102 et 103.)

[Tran 2013] Phuong Nga Tran et A. Timm-Giel. *Reconfiguration of virtual network mapping considering service disruption.* Dans 2013 IEEE International Conference on Communications (ICC), pages 3487–3492, June 2013. (Cité en pages 102 et 104.)

[Vmware 2013] Vmware. *The VMware NSX Network Virtualization Platform.* White paper, June 2013. (Cité en page 17.)

[Voellmy 2012] Andreas Voellmy, Hyojoon Kim et Nick Feamster. *Procera : A Language for High-level Reactive Network Control.* Dans Proceedings of the First Workshop on Hot Topics in Software Defined Networks, HotSDN '12, pages 43–48, New York, NY, USA, 2012. ACM. (Cité en page 34.)

[Wang 2016] Yi Wang, Dong Lin, Changtai Li, Junping Zhang, Peng Liu, Chengchen Hu et Gong Zhang. *Application Driven Network : Providing On-Demand Services for Applications.* Dans Proceedings of the 2016 ACM SIGCOMM Conference, SIGCOMM '16, pages 617–618, New York, NY, USA, 2016. ACM. (Cité en page 20.)

[Wanis 2013] B. Wanis, N. Samaan et A. Karmouch. *Substrate network house cleaning via live virtual network migration.* Dans 2013 IEEE International Conference on Communications (ICC), pages 2256–2261, June 2013. (Cité en pages 99, 102, 103 et 104.)

[Wei 2010] Yongtao Wei, Jinkuan Wang, Cuirong Wang et Cong Wang. *Bandwidth allocation in virtual network based on traffic prediction.* Dans 2010 International Conference On Computer Design and Applications, volume 5, pages V5–304–V5–307, June 2010. (Cité en pages 59 et 60.)

[Xi 2010] Y. Xi et E. M. Yeh. *Distributed Algorithms for Minimum Cost Multicast With Network Coding.* IEEE/ACM Transactions on Networking, vol. 18, no. 2, pages 379–392, April 2010. (Cité en pages 59 et 60.)

[Y.2770 2012] UIT-T Y.2770. *Spécifications relatives au contrôle approfondi des paquets dans les réseaux de prochaine génération*, 2012. (Cité en page 45.)

[Yu 2008] Minlan Yu, Yung Yi, Jennifer Rexford et Mung Chiang. *Rethinking Virtual Network Embedding : Substrate Support for Path Splitting and Migration.* SIGCOMM Comput. Commun. Rev., vol. 38, no. 2, pages 17–29, mars 2008. (Cité en pages 58, 99, 102, 103 et 104.)

[Zhang Gong 2016] Huo Dawei Zhang Gong et Li Yingtao. *All about the user with ADN.* White paper, June 2016. (Cité en page 20.)

[Zhou 2010] Y. Zhou, Y. Li, G. Sun, D. Jin, L. Su et L. Zeng. *Game Theory Based Bandwidth Allocation Scheme for Network Virtualization.* Dans 2010 IEEE Global Telecommunications Conference GLOBECOM 2010, pages 1–5, Dec 2010. (Cité en pages 102 et 104.)

[Zhu 2006] Y. Zhu et M. Ammar. *Algorithms for Assigning Substrate Network Resources to Virtual Network Components.* Dans Proceedings IEEE INFOCOM 2006. 25TH IEEE International Conference on Computer Communications, pages 1–12, April 2006. (Cité en pages 102, 103 et 104.)