De la configuration à la programmation de réseaux virtuels

Emmanuel Lavinal

Manuscrit

En vue de l'obtention de l'

HABILITATION À DIRIGER LES RECHERCHES

Délivrée par : *l'Université Toulouse III – Paul Sabatier*

Présentée et soutenue le *01/12/2022* **par :**

Emmanuel LAVINAL

De la configuration à la programmation de réseaux virtuels

JURY

Géraldine TEXIER	Professeure, IMT Atlantique	Rapportrice
Stefano SECCI	Professeur, Cnam Paris	Rapporteur
Olivier FESTOR	Professeur, Université de Lorraine	Rapporteur
Philippe OWEZARSKI	Directeur de Recherche, LAAS-CNRS	Examinateur
Emmanuel CHAPUT	Professeur, Toulouse INP	Examinateur
Michelle SIBILLA	Professeure, Université Toulouse III	Examinatrice

---

École doctorale et spécialité :
MITT : Domaine STIC : Réseaux, Télécoms, Systèmes et Architecture

Unité de Recherche :
Institut de Recherche en Informatique de Toulouse (UMR 5505)

Résumé

Les travaux présentés dans cette habilitation à diriger les recherches sont centrés sur des problèmes de conception et de gestion de réseaux de communication. Nos contributions s'intéressent, d'une part, à automatiser certaines fonctions de gestion (la reconfiguration en particulier) et, d'autre part, à faciliter la conception de réseaux virtuels ainsi que leur déploiement sur l'infrastructure réelle.

Pouvoir garantir que la configuration du système ou du réseau n'introduise pas d'erreur à l'exécution ou réponde bien aux exigences de l'opérateur est indispensable pour pouvoir automatiser la mise en production des configurations. Dans ce contexte, une partie des travaux que nous avons entrepris s'intéresse à de la vérification en cours d'exécution afin de pouvoir prendre en compte l'état courant du système dans le processus de vérification. Des propositions ont été faites basées sur des méthodes d'ingénierie dirigée par les modèles et d'autres sur de la génération automatique de paquets de tests. Enfin, nous nous sommes intéressés à des méthodes basées sur de l'apprentissage automatique supervisé pour détecter et localiser des erreurs de configuration dans des réseaux d'opérateur, pour des services de réseaux privés virtuels.

En ce qui concerne nos travaux sur la virtualisation de réseaux, ils ont d'abord ciblé l'interface nord de l'architecture SDN (*Software Defined Networking*), interface capitale pour les concepteurs et développeurs d'applications de gestion et de contrôle du réseau. Il en résulte la proposition d'un modèle d'abstraction de réseau virtuel et d'un langage de contrôle basé sur ce modèle. Une démarche complète de pilotage du réseau de haut en bas est présentée, de la spécification de la topologie virtuelle et des politiques de contrôle, jusqu'à la génération des règles déployées sur l'infrastructure physique. La suite de nos travaux sur la virtualisation s'est concentrée sur le concept de *slice* réseau dans les réseaux mobiles 5G. Nous avons mené des travaux sur la continuité de service dans un contexte multi-domaines, en proposant une architecture et des mécanismes de couture pour intégrer un lien satellite comme réseau de transport dans une *slice* 5G tout en préservant les exigences de qualité de service de la *slice*.

Le projet de recherche que nous développons pour les années à venir s'inscrit dans la mutation profonde qui s'opère dans le domaine des réseaux depuis une dizaine d'années autour des réseaux programmables. Un premier axe de recherche porte sur des problématiques de mise en relation et d'accès à des services situés en bordure du réseau (*edge computing*), avec notamment des enjeux sur l'adaptation de l'infrastructure réseau en cas de mobilité de l'utilisateur. Un deuxième axe de recherche concerne la poursuite des travaux sur les réseaux virtuels dans lesquels nous souhaitons revisiter le modèle d'abstraction que nous avons proposé afin d'y intégrer des récents travaux sur le chaînage de fonctions réseaux ou encore sur la programmation du plan de données. Nous avons également l'ambition de pouvoir générer et insérer des instructions de télémétrie au plus près des flux des utilisateurs afin de vérifier que les exigences définies au niveau du réseau virtuel sont bien respectées de bout en bout dans le monde physique.

Table des matières

Chapitre 1

Introduction

En 2001, la vision d'IBM sur l'« Autonomic Computing », initialement exposée dans un manifeste de Paul Horn (vice-président d'IBM Research) [Hor01], puis repris dans l'article de référence de J.O. Kephart et D.M. Chess [KC03], a façonné le paysage d'une partie de la recherche dans le domaine de la gestion des réseaux, des systèmes et des services. Bien que l'objectif sous-jacent était de concevoir des solutions permettant de réduire les coûts de maintenance des infrastructures informatiques devenus excessifs, le message mis en avant était plus attractif : s'appuyer sur le fonctionnement du système nerveux autonome du corps humain pour encourager le monde de l'informatique à reproduire ce concept afin de concevoir des systèmes capables de se contrôler eux-mêmes et de s'adapter à un ensemble varié de circonstances. Cette vision prônait un système « auto-géré » qui devait obéir aux quatre propriétés d'*auto-configuration* (pouvoir se configurer et s'adapter dynamiquement lors de changements dans l'environnement), d'*auto-réparation* (pouvoir détecter, diagnostiquer et réparer automatiquement toute anomalie rencontrée), d'*auto-optimisation* (pouvoir s'ajuster automatiquement pour améliorer continuellement ses performances) et enfin d'*auto-protection* (pouvoir détecter et se protéger contre toute attaque malveillante). Dans cette mouvance, sur la période 2003-2006, j'ai réalisé ma thèse sur la thématique de la gestion autonome des réseaux et des services. L'objectif était de s'appuyer sur une approche de système multi-agents (SMA) pour tendre vers ce paradigme de gestion autonome. Pour cela, j'ai proposé un modèle d'organisation du SMA (basé sur les notions de groupe et de rôle), un modèle d'information (pour représenter les ressources et les actions de gestion) et enfin une démarche et des algorithmes pour permettre aux agents de déduire des situations coopératives pour atteindre leur but, c'est-à-dire la réalisation d'une opération de gestion [LDR06].

Vingt ans après, cette vision de gestion autonome des réseaux et des services n'est toujours pas une réalité dans les systèmes complexes qui nous entourent. Force est de constater que les solutions issues des travaux de recherche n'ont pas été adoptées par les entreprises, cette vision était probablement en avance sur son temps. Parmi les raisons qui peuvent expliquer cela, il y avait sans doute un facteur humain (l'administrateur ne voulant pas confier toute la gestion de son infrastructure à un « pilote automatique »), mais surtout un manque crucial dans les technologies sous-jacentes : elles n'avaient que des moyens limités pour mettre en œuvre l'objectif de gestion sur l'infrastructure réseau. Ajouter d'avantage d'autonomie dans le plan de gestion n'est en effet pas suffisant pour permettre au réseau de s'auto-gérer. Le plan de gestion est la partie visible de l'iceberg, il faut pouvoir ensuite instancier l'objectif de l'opérateur (ou la décision issue d'une application) dans le plan de contrôle du réseau, voire dans le plan de données. Or, seuls les équipementiers contrôlaient vraiment le comportement interne des routeurs, commutateurs, et autres *middleboxes* ; et non les opérateurs ou administrateurs qui ne pouvaient s'appuyer que sur des interfaces de gestion rudimentaires (CLI, SNMP, etc.) fournies par les constructeurs de matériels. De plus, la complexité de configuration inhérente aux réseaux de grande taille (en particulier en matière de routage, de filtrage ou de qualité de service (QoS)) [BAM09] a freiné

l'adoption d'une vision de gestion autonome.

Une des raisons qui limitait le pilotage fin du comportement du réseau, et l'innovation d'une manière générale, était liée au couplage fort qui existait (et qui existe encore dans certains réseaux actuels) entre le plan de contrôle et le plan de données au sein d'un même équipement propriétaire et fermé. Cela a rendu difficile l'évolution du matériel, des protocoles et des architectures [Gho+11 ; Rag+12]. Cette rigidité architecturale, associée à un besoin d'expérimentation sur des réseaux réels (c.-à-d. non simulés) a été un des facteurs de l'apparition du paradigme SDN (*Software Defined Networking*) [McK+08]. Ce paradigme a ouvert la voie à des innovations scientifiques et technologiques importantes dans le monde de la R&D des réseaux informatiques en général (ACM SIGCOMM, IEEE ComSoc) et en particulier au sein de la communauté de la gestion de réseaux et de services (IEEE CNOM et IFIP WG6.6). Le principe général d'un réseau SDN est de découpler le plan de contrôle du plan de données en le centralisant dans un point logique appelé contrôleur SDN, ce dernier pouvant être reprogrammé via une interface ouverte, permettant ainsi de mettre à jour le comportement du réseau [Kre+15]. Les travaux sur SDN ne sont pas restés à l'état de recherche théorique ou de prototype en laboratoire mais ont bien donné lieu à des applications réelles, par exemple dans les réseaux de centres de données [Jai+13 ; Fer+21] ou encore dans des points d'échange Internet [Gup+14].

En parallèle de l'essor du paradigme SDN, des travaux de recherche sur la virtualisation de réseaux se sont développés, d'abord comme un moyen de réaliser des expérimentations sur des bancs de test virtuels [And+05], puis entraînés par le déploiement massif du *cloud computing* [Arm+09]. La virtualisation réseau s'est appuyée fortement sur des travaux passés, d'une part sur les mécanismes d'encapsulation et de recouvrement pour créer des liens virtuels, et d'autre part sur l'isolation de ressources système permettant à un nœud physique d'accueillir plusieurs nœuds virtuels [CB10]. En complément du réseau en lui-même, les fonctions de traitement des flux et des paquets, initialement fournies par des équipements matériels dédiés, ont été implémentées dans du logiciel s'exécutant sur des serveurs de commodité. Cette approche NFV (*Network Functions Virtualization*) est née de discussions entre les principaux opérateurs de télécommunications sur la façon d'améliorer les opérations et la gestion de réseaux à l'ère de contenus multimédia de masse. Ces discussions ont abouti à la publication en 2012 du premier livre blanc sur NFV [ETS12]. Peu de temps après, ont suivi les spécifications sur son architecture de référence [ETS14a] et sur le cadre de gestion et d'orchestration des fonctions réseau virtuelles [ETS14b]. Depuis, le paradigme NFV a suscité un grand nombre de travaux de recherche sur des problématiques de performance, d'allocation de ressources, de gestion et d'orchestration, de consommation énergétique ou encore de sécurité et de confiance [Mij+16].

Plus récemment, la programmation du plan de données, principalement à l'aide du langage P4 [Bos+14], étend encore le concept de programmabilité du réseau jusqu'aux circuits intégrés. Ces travaux se sont concrétisés notamment grâce à l'apparition de composants matériels spécifiques programmables comme la puce Tofino d'Intel [1] ou, depuis peu, les puces Intel® IPU 200G SoC ou NVIDIA® BlueField-3 DPU. Le language P4, standardisé par l'ONF [Lan22], permet de décrire précisément comment un paquet est construit, traité et acheminé au sein d'un équipement d'interconnexion, indépendamment des spécificités du matériel. Un compilateur se charge ensuite de compiler le programme vers un jeu d'instructions dépendant de la plateforme cible.

Contrairement au paradigme de la gestion autonome (*self-management*), qui était probablement trop ambitieux pour l'époque, les paradigmes de programmation du réseau (SDN, NFV, P4) ont incontestablement pu percer la sphère de la recherche scientifique pour s'inscrire dans un cycle de développement industriel. La recherche et le développement dans ce domaine restent néanmoins encore très actifs, les spécifications issues de groupes de standardisation et les projets *open source* continuent de se multiplier, signe de la vitalité de ces thématiques.

1. Initialement *Barefoot Networks* avant son acquisition en 2019 par Intel

1.1 Thèmes de recherche

J'ai réalisé mes activités de recherche dans le contexte présenté précédemment, avec une ligne directrice qui consistait soit à automatiser des fonctions de gestion (configuration, déploiement), soit à construire des réseaux virtuels qui répondaient à un besoin ou un usage particulier. La vue globale de mes différentes activités est illustrée dans la figure 1.1.

Contribuer à automatiser des fonctions de gestion s'est fait principalement à travers des travaux sur de la vérification de configurations. Pouvoir garantir que la configuration du système ou du réseau réponde bien aux exigences de haut niveau ou n'introduise pas d'erreur à l'exécution est indispensable pour pouvoir automatiser la mise en production des configurations. Nous avons travaillé sur de la « vérification en ligne » de configurations (c'est-à-dire nécessitant des données d'état du système en cours d'exécution), basée une méthode d'ingénierie dirigée par les modèles, dans le cadre de la thèse de Ludi Akue [ALS10 ; Aku+13b ; Aku+13a ; Aku+14]. Ce thème a été de nouveau investigué dans un projet de recherche avec des partenaires industriels sur la définition d'une méthode de construction d'une architecture réseau répondant à des exigences de communication et de sécurité. La vérification des exigences s'est faite par génération de paquets de tests exécutés dans un réseau virtuel basé sur une architecture SDN [IRE16 ; IRE18]. Enfin, nous nous sommes également intéressés à de la vérification de configurations chez des opérateurs de télécommunications pour des services de réseaux privés virtuels (VPN) interconnectant plusieurs sites client. L'originalité de notre approche est de s'appuyer sur une méthode, non pas à base de règles, mais utilisant une fonction d'apprentissage supervisé dans laquelle un modèle est entraîné sur des milliers de configurations déjà labellisées. Ce travail est en cours dans le cadre de la thèse de El-Heithem Mohammedi [MLF20 ; MLF22].

J'ai commencé à m'intéresser à la virtualisation de réseaux dès le post-doctorat à Télécom Paris avec la conception de services de réseau mobile personnalisés. La chaîne de services était construite par composition de « sous-services » de grain plus fin (que l'on pourrait qualifier aujourd'hui de micro-services), en s'appuyant sur un réseau de recouvrement et sur le protocole de session SIP [HLS07 ; LS08 ; Lav+09]. Nos premiers travaux sur le paradigme SDN avaient aussi un objectif de définition de réseaux virtuels. La possibilité de spécifier un réseau virtuel et d'y associer des politiques de contrôle indépendamment du réseau physique était en effet une des motivations de la thèse de Messaoud Aouadj. La contribution principale de ce travail a été de définir un langage dédié à la conception de réseaux virtuels et à la spécification de politiques d'orchestration de services réseaux déployés sur de telles topologies virtuelles [Aou+16b ; Aou+16a]. La suite de nos travaux sur la virtualisation s'est concentrée sur le concept de *slice* réseau, qui n'est autre qu'un réseau virtuel de bout en bout répondant aux exigences d'un service de communication particulier (essentiellement en matière de QoS et de sécurité). La thèse de Youssouf Drif s'est attachée à mettre en œuvre ce concept de *network slicing* dans le cadre des réseaux mobiles 5G intégrant un réseau de *backhaul* satellite [Dri+21a].

La spécification d'un réseau virtuel répond à l'expression d'un besoin mais, pour s'exécuter, le réseau virtuel doit, in fine, utiliser des resources de l'infrastructure physique. C'est dans ce cadre que nos travaux sur la programmation du réseau se sont déroulés. Les thèses de M. Aouadj et Y. Drif, évoquées précédemment, ont contribué à cet axe de recherche à travers la programmation du plan de contrôle et du plan de données. Dans le premier cas, la conception et le développement d'un « hyperviseur » ont permis de transformer les politiques du réseau virtuel en fonctions logicielles et règles OpenFlow exécutées sur un contrôleur SDN [Aou+17]. Dans le deuxième cas, la programmation du réseau s'est effectuée via l'introduction d'un composant particulier (*slice classifier*) implémentant des mécanismes de couture et de QoS dans le plan de données [Dri+21b]. Nos travaux sur la programmation du réseau se poursuivent actuellement dans le cadre de la thèse de Louis Royer qui se focalise sur des problèmes de redirection de trafic dans les réseaux 5G vers des applications situées en bordure du réseau (*edge*).

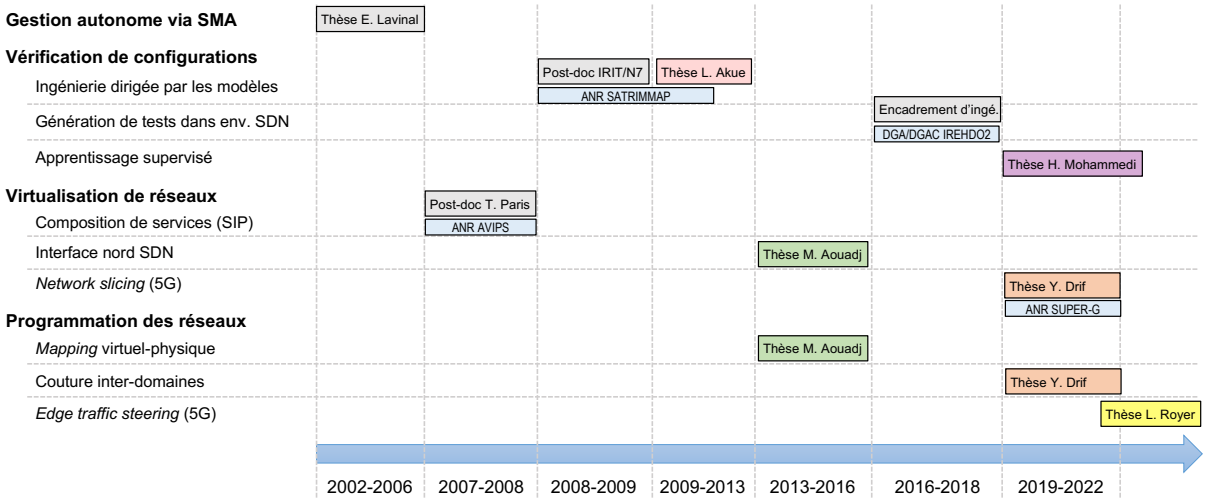


FIGURE 1.1 – Vue générale des activités de recherche

1.2 Plan du manuscrit

Dans la suite de ce manuscrit je présenterai les activités de recherche que j'ai réalisées ou encadrées, essentiellement depuis l'obtention de mon poste d'enseignant-chercheur en 2009 à l'Université Toulouse III – Paul Sabatier.

Le chapitre 2 regroupe les travaux que nous avons menés sur la vérification de configurations système et réseau. Une partie de ces travaux s'intéresse à de la vérification en cours d'exécution (*at runtime*) afin de pouvoir prendre en compte l'état courant du système dans le processus de vérification. Des propositions ont été faites basées sur des méthodes d'ingénierie dirigée par les modèles et d'autres, ciblant davantage les réseaux de communication, sur de la génération de paquets de tests dans un environnement SDN. Plus récemment, nous nous sommes intéressés à de la vérification de configurations déployées sur des routeurs d'un réseau d'opérateur en s'appuyant sur des techniques d'apprentissage automatique supervisé.

Les premiers travaux que nous avons réalisés sur la virtualisation et la programmation des réseaux dans un contexte SDN sont présentés dans le chapitre 3. Le point de départ de ces contributions a été d'étudier l'interface nord de l'architecture SDN. Il en résulte la proposition d'un modèle d'abstraction de réseau virtuel et d'un langage de contrôle basé sur ce modèle. Une démarche complète de pilotage du réseau de haut en bas est présentée, de la spécification de la topologie virtuelle et des politiques de contrôle, jusqu'à la génération des règles déployées sur l'infrastructure physique.

Toujours dans un contexte de virtualisation des réseaux, le chapitre 4 synthétise nos contributions sur la conception de réseaux virtuels dédiés à un usage particulier. Notre intérêt pour cette thématique a démarré dès le post-doctorat avec la conception de services réseau personnalisés par composition dynamique d'autres services. Cette problématique s'inscrit aussi de manière pertinente dans le concept de *network slicing* porté par les réseaux mobiles de 5$^{\text{ème}}$ génération. C'est dans ce cadre que nous avons entrepris des travaux sur la continuité de service dans un contexte multi-domaines, en proposant une architecture et des mécanismes de couture pour intégrer dans la *slice* un lien satellite comme réseau de transport.

Mon projet de recherche pour les années à venir est exposé dans le chapitre 5. Il est axé sur la construction dynamique de réseaux virtuels personnalisés en fonction d'un besoin particulier et pouvant s'adapter à certains évènements perçus de l'environnement. D'abord dans le contexte des réseaux mobiles 5G avec un besoin grandissant de faire de plus en plus de traitement en bordure

du réseau (*edge*). Des problématique liées à l'intégration des architecture MEC et 5G, ainsi que la réorientation de flux en cas de mobilité m'intéressent particulièrement. Puis, de manière plus générale, je souhaite poursuivre et étendre les travaux sur les modèles d'abstraction des réseaux virtuels (notamment pour prendre en compte les avancées récentes sur le chaînage de fonctions réseaux et sur la programmation du plan de données) et sur la génération automatique de tests (basés sur des instructions de télémétrie) pour vérifier, sur le système réel cible, le respect des exigences spécifiées au niveau du réseau virtuel.

Chapitre 2

Vérification de configurations système et réseau

2.1 Contexte des travaux

Lors de mon second post-doctorat, en 2008-09, dans le cadre du projet ANR SATRIMMAP, j'ai travaillé sur la conception et la configuration d'applications distribuées et embarquées sur une architecture avionique modulaire intégrée (IMA, *Integrated Modular Avionics*). Une des ambitions du projet était d'offrir des services et des outils pour automatiser partiellement la configuration du système et des applications, tout en prenant en compte des contraintes temporelles, de déterminisme et de fiabilité. Dans ce contexte, j'ai pu développer des compétences dans le domaine de la vérification de configurations système, une préoccupation majeure des systèmes embarqués critiques. Ces travaux m'ont amené à proposer et encadrer un sujet de thèse sur la vérification en cours d'exécution (cf. section 2.2), domaine qui était alors très peu développé dans les systèmes embarqués qui se concentraient principalement sur de la vérification lors de la phase de conception. La contribution visait la vérification de systèmes distribués et intergiciels (*middleware*) avec une méthode basée sur de l'ingénierie dirigée par les modèles, couplée à des mécanismes de supervision, pour évaluer les contraintes sur l'état courant. Par la suite, j'ai eu d'autres activités de recherche dans le domaine de la vérification de configurations, mais cette fois en ciblant les réseaux de communication. Ces travaux s'inséraient dans le cadre du projet IREHDO2, avec toujours une composante de vérification en cours d'exécution, mais par une méthode basée sur des tests (génération de paquets) dans un environnement réseau virtualisé (cf. section 2.3). Enfin, mes dernières activités se rapportant à de la vérification de configurations se déroulent actuellement dans le contexte d'un encadrement d'une thèse CIFRE, avec une collaboration avec l'entreprise IMS Networks. Ces travaux ciblent des configurations d'un réseau d'opérateur, en proposant un service de vérification basé sur des techniques d'apprentissage automatique (cf. section 2.4).

2.2 Vérification en cours d'exécution basée sur de l'ingénierie dirigée par les modèles

2.2.1 Vérification en ligne de configurations

Traditionnellement, la vérification de configurations est plutôt effectuée en phase de conception du système ou, de façon générale, avant sa mise en exploitation [DJ07]. Ce type de vérification contrôle la conformité structurelle de la configuration : présence de propriétés fonctionnelles définies par l'utilisateur, évaluation de contraintes sur les types de données, sur les valeurs autorisées ou encore sur les dépendances entre paramètres. Cette vérification de conformité structurelle est indépendante de l'état opérationnel du système géré.

Dans le cadre des travaux de la thèse de Ludi Akue [Aku14], nous nous sommes concentrés sur « l'applicabilité opérationnelle » d'une configuration. En d'autres termes, nous avons essayé de répondre à la question : cette configuration peut-elle être déployée à l'instant t sur le système cible ? Cette problématique dépasse le contexte d'une vérification structurelle pour considérer l'état opérationnel courant du système géré. Par exemple, dans le cadre de la migration d'une machine virtuelle vers un nouveau serveur physique, il ne suffit pas de vérifier dans la nouvelle configuration que l'adresse IP de l'hôte est conforme au format attendu, mais il faut également vérifier que l'hôte est dans un état « actif » et qu'il dispose de suffisamment de ressources. Ce type de vérification nécessite que le système de reconfiguration ait accès à des données d'état en cours d'exécution. Dans la suite, nous désignerons ce type de vérification par « vérification en ligne » de configurations.

La vérification de l'applicabilité opérationnelle de configurations répond notamment à la problématique des erreurs légales définies dans le travail de [Yin+11]. Dans cet article, les auteurs distinguent les erreurs de configurations *illégales* des erreurs *légales*. Une erreur de configuration illégale enfreint des contraintes explicites ou implicites de format, de consistance interne de valeurs ou de consistance vis-à-vis d'autres valeurs de configurations. En revanche, une erreur de configuration légale n'est pas due à une violation de ce type de contraintes mais provient d'inconsistances créées par les conditions environnementales courantes. Par exemple, une valeur de configuration qui pénalise la performance du système est considérée comme une erreur légale. Ces erreurs sont difficilement décelables à moins que des objectifs de service soient clairement spécifiés et puissent être évalués sur les configurations.

Une des principales motivations qui nous a conduit à travailler sur de la vérification en ligne de configurations était de pouvoir automatiser la phase de transition entre la décision de gestion et son application sur le système cible dans la boucle de gestion MAPE (*Monitor-Analyze-Plan-Execute*) [KC03]. La figure 2.1 présente une version simplifiée de cette boucle de gestion en regroupant les composants d'analyse et de planification en un unique composant de décision. L'intégration de cette fonction de vérification dans la boucle de gestion repose sur trois grandes étapes :

1. Demande d'évaluation de configurations : un besoin de reconfiguration a été détecté et le module de décision a choisi une nouvelle configuration à appliquer, il la soumet au module de vérification en ligne pour évaluation.
2. Collecte des données d'état courant : pour les besoins de la vérification en ligne, le module de vérification requiert des informations liées à l'état courant du système géré, le module de supervision les lui fournit en réponse.
3. Exécution de la vérification : le module de vérification évalue la configuration candidate, à la fois structurellement et en fonction des informations opérationnelles récupérées à l'étape 2. Il renvoie ensuite les résultats de cette vérification au module décisionnel.

2.2.2 Définition d'un métamodèle dédié

Pour concevoir le service de vérification en ligne, nous avons suivi une démarche d'ingénierie dirigée par les modèles (IDM), ce qui nous a permis de nous affranchir des spécificités techniques et protocolaires des plateformes de gestion et de nous concentrer essentiellement sur les concepts propres à la vérification de configurations dans un cadre de reconfiguration dynamique.

La vue globale des travaux réalisés dans ce contexte est illustrée dans la figure 2.2. Le point de départ a été la définition d'un métamodèle appelé MeCSV (*Metamodel for Configuration Specification and Verification*). Ce métamodèle fournit des constructeurs pour la représentation d'informations de configuration, d'informations d'état opérationnel et de contraintes à vérifier. MeCSV peut ainsi être utilisé pour concevoir des modèles de référence propres à un domaine applicatif donné. Une fois le modèle de référence construit, le service de vérification se base sur

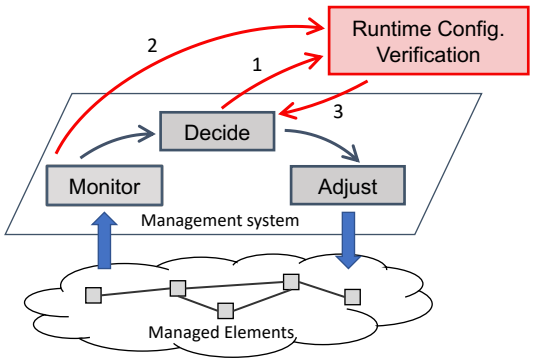


FIGURE 2.1 – Intégration d'un service de vérification au sein de la boucle de gestion

ce dernier pour évaluer les configurations et leur applicabilité opérationnelle sur le système géré (en intégrant un moteur d'évaluation de contraintes).

L'architecture permet également le développement de composants « adaptateurs » en vue de favoriser l'intégration avec des systèmes de gestion existants. Deux types d'invocation du service de vérification peuvent ainsi être utilisés : interaction directe entre le système de gestion et le service de vérification (étapes 1 et 2 sur la figure 2.2) ou interaction indirecte via un adaptateur qui assure une traduction entre le modèle de référence et le modèle d'information existant (étapes 1a/1b et 2a/2b sur la figure 2.2).

La définition du modèle de référence se fait en phase de conception du système mais peut être complétée en phase d'exécution, par exemple par l'ajout de nouvelles contraintes. La vérification de configurations se fait quant à elle obligatoirement en phase d'exécution puisqu'elle nécessite la collecte de données d'état courant.

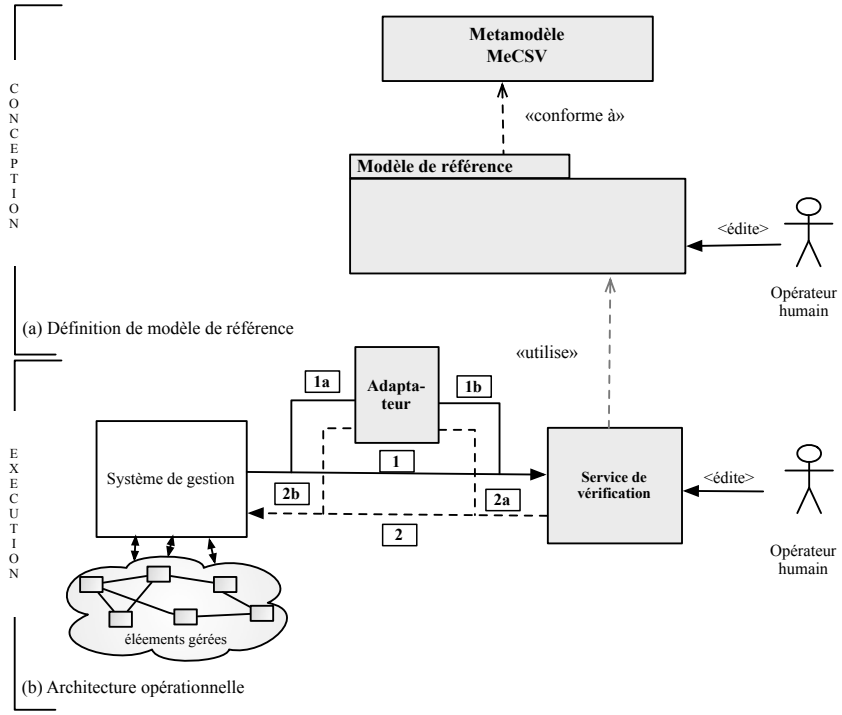


FIGURE 2.2 – Cadre de vérification en ligne de configurations

Le métamodèle MeCSV

Le métamodèle MeCSV a été conçu comme un langage de modélisation dédié (DSML, *Domain-Specific Modeling Language*) à la vérification opérationnelle de configurations. Il est inspiré de standards [CIM12 ; YAN10] et de langages de configuration existants [Kan06 ; Gol+09]. Il fournit des concepts organisés en trois catégories, chacune couvrant un objectif spécifique, avec des constructeurs permettant de définir (cf. figure 2.3) :

— des éléments et paramètres de configuration (partie a) ;

— des paramètres d'état opérationnel (partie b) ;

— des contraintes hors-ligne et en ligne (partie c).

Les contraintes hors ligne ne sont liées qu'à des données de configuration alors que les contraintes en ligne dépendent des paramètres d'état dont les valeurs sont collectées par le module de supervision.

En phase de conception, l'utilsateur utilise MeCSV pour définir le modèle de configuration du système géré (incluant les informations d'état à superviser) et exprimer les contraintes qui doivent être respectées. Le résultat de cette modélisation donne le *modèle de référence*. Ce dernier est utilisé tel quel en phase d'exécution pour vérifier les configurations candidates (qui sont des instances du modèle de référence).

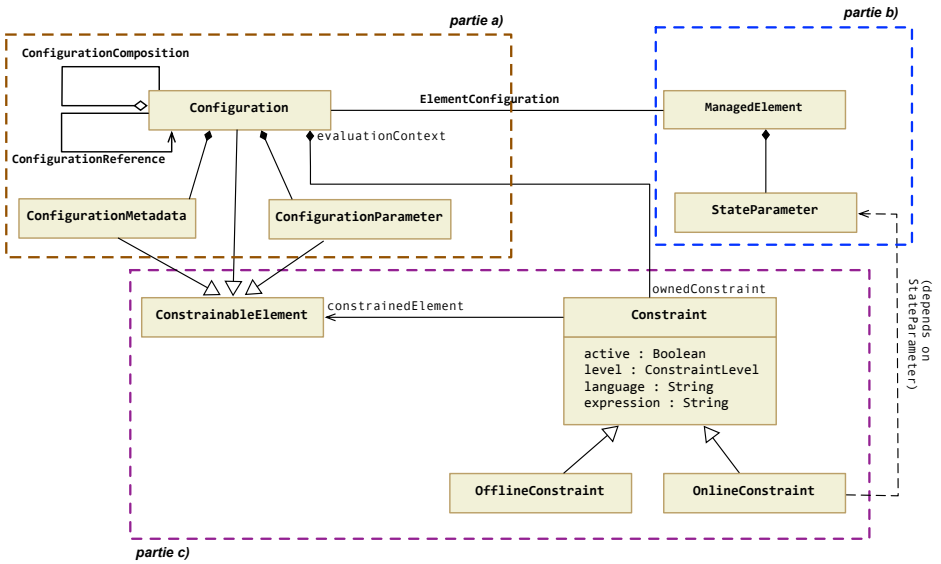


FIGURE 2.3 – Principaux constructeurs du métamodèle MeCSV

Nous allons illustrer l'application du métamodèle MeCSV sur un exemple de configuration simplifiée d'un intergiciel orienté messages (MOM, *Message-Oriented Middleware*). Nous avons testé ce modèle de configuration sur la plateforme `JORAM`, une implémentation *open source* en Java d'un MOM JMS (*Java Message Service*).

La première étape est de modéliser les éléments de configuration. Dans notre exemple nous considérons un serveur de message (`ServerConfig`) caractérisé par ses paramètres de configuration internes tels que son identifiant (`serverId`) ou son nom (`serverName`) et une métadonnée (`configType`) qui qualifie son type, par exemple « configuration par défaut ». La configuration du serveur contient également une sous-configuration, celle de ses files de messages (`QueueConfig`) et référence la configuration de la machine-hôte (`HostConfig`) qui l'héberge.

La deuxième étape concerne la spécification des informations d'état opérationnel qui découle de la nécessité d'avoir accès à l'état courant du système géré pour la vérification de l'applicabilité opérationnelle. Dans notre exemple nous considérons le statut opérationnel d'un serveur

de messages (`operationalStatus`), l'état d'utilisation de ses files de messages (`receiving`) et le nombre courant de messages stockés en mémoire (`pendingMessageCount`). Ces informations nous serviront à exprimer les contraintes en ligne.

Enfin, la troisième étape permet d'exprimer les contraintes à vérifier sur les configurations. Les contraintes hors ligne (*offline*) restreignent les valeurs possibles d'un élément de configuration par rapport à d'autres éléments de configuration ou à des constantes fixées. Les contraintes en ligne (*online*) confrontent les valeurs possibles d'un élément de configuration à des données d'état opérationnel. Dans notre exemple, nous considérons deux contraintes : une hors ligne (`QueueOffIntegrity`) qui impose de renseigner le nom de la file de messages et le nombre maximal de messages qu'elle peut stocker en mémoire (et qui doit être supérieur à 0) ; et une en ligne (`QueueOpValidity`) qui empêche les configurations candidates d'avoir un nombre maximal de messages trop bas par rapport au nombre courant de messages (dans l'exemple cette limite doit être supérieure d'un facteur 1,25 au nombre courant de messages, en d'autres termes la file de messages ne doit pas être chargée à plus de 80%).

Un extrait du modèle de référence résultant est présenté dans la figure 2.4.

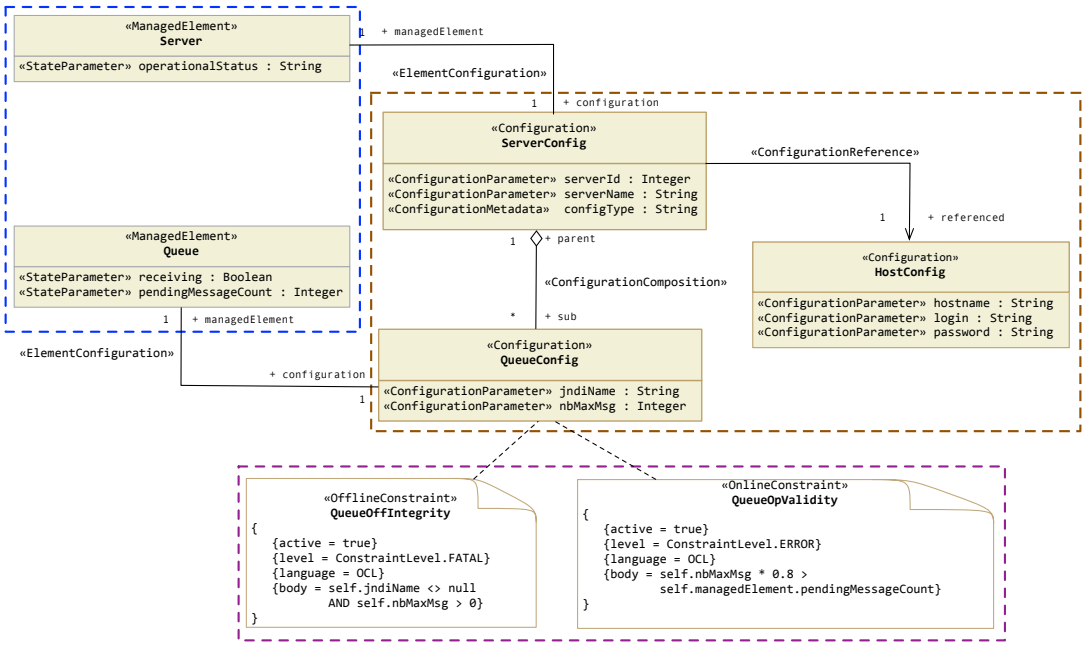


FIGURE 2.4 – Extrait du modèle de référence d'une configuration d'un MOM

Production de modèles de référence

Le métamodèle MeCSV a été défini sous forme d'un profil UML (*Unified Modeling Language*) puis implémenté dans la plateforme EMF (*Eclipse Modeling Framework*). Le résultat est un éditeur de modèles disponible comme *plugin* du logiciel `Eclipse MDT`. Quant à la spécification des contraintes, elle est réalisée en utilisant le langage OCL (*Object Constraint Language)*. Les informations techniques sur cette réalisation peuvent être consultées dans [Aku14].

Comme indiqué précédemment, le modèle de référence comprend trois parties : les informations de configuration, les informations d'état et les contraintes. Les deux premières parties peuvent être obtenues soit en les spécifiant manuellement de toute pièce via l'éditeur, soit par transformation de modèles de gestion existants. Pour ce deuxième cas, les approches d'ingénierie dirigée par les modèles se basent sur des règles de *mapping* au niveau métamodèle ce qui permet d'opérer des transformations automatiques entre modèles conformes à ces métamodèles.

Nous avons effectué ce travail entre le standard CIM du DMTF (plus précisément le patron `CIM_SettingData` qui permet de décrire la configuration) et le métamodèle MeCSV. Pour écrire ces règles de transformation, nous nous sommes appuyés sur le standard QVT (*Query/View/-Transformation*) de l'OMG. Nous avons réalisé une validation expérimentale de ces règles sur un modèle CIM tiré du profil de gestion « Virtual System » du DMTF (modèle de configuration de machines virtuelles et de l'allocation des ressources matérielles des hôtes). Le résultat a été l'obtention de la vue structurelle du modèle de référence [Aku+13a]. La dernière étape est ensuite de compléter le modèle en rajoutant les contraintes hors ligne et en ligne.

2.2.3 Architecture du service de vérification en ligne

Pour mettre en œuvre le service de vérification en ligne de configurations, nous avons défini l'architecture présentée dans la figure 2.5. L'implémentation du service comprend deux composants internes principaux : un moteur d'exécution de la vérification des contraintes (basé sur les bibliothèques *open source* `Dresden OCL`) et une base de modèles où sont stockés les modèles de référence. De plus, deux interfaces du service existent :

— Une interface de vérification (plan opérationnel ou plan de données) qui est utilisée pour exécuter la vérification des configurations candidates. Cette interface offre des méthodes d'accès à un moteur d'exécution de la vérification qui est capable d'analyser des instances de modèles conformes à la structure d'un modèle de référence MeCSV, d'évaluer les contraintes hors ligne et en ligne et de notifier les violations constatées.

— Une interface d'édition des modèles (plan de contrôle) qui est utilisée pour modifier dynamiquement les modèles de référence, en particulier l'ajout ou la suppression de contraintes, ou encore la modification de leur sévérité.

Afin de permettre l'intégration du service auprès de systèmes de gestion existants, un composant intermédiaire, appelé « adaptateur », doit être développé. Ce dernier assure une fonction de traduction entre les formats des données de configuration et d'état du système de gestion et les instances de modèles de référence MeCSV. Comme indiqué dans la figure 2.5, l'implémentation de ce composant est spécifique à la plateforme de gestion existante (CIM/WBEM, NETCONF/YANG...). En guise de preuve de concept, nous avons développé un prototype d'adaptateur pour les environnements de gestion CIM/WBEM appelé `CIM2MeCSVAdapter`. Il a été implémenté en tant que client WBEM (*Web-Based Enterprise Management*) afin de pouvoir accéder à des informations de supervision sur un serveur WBEM.

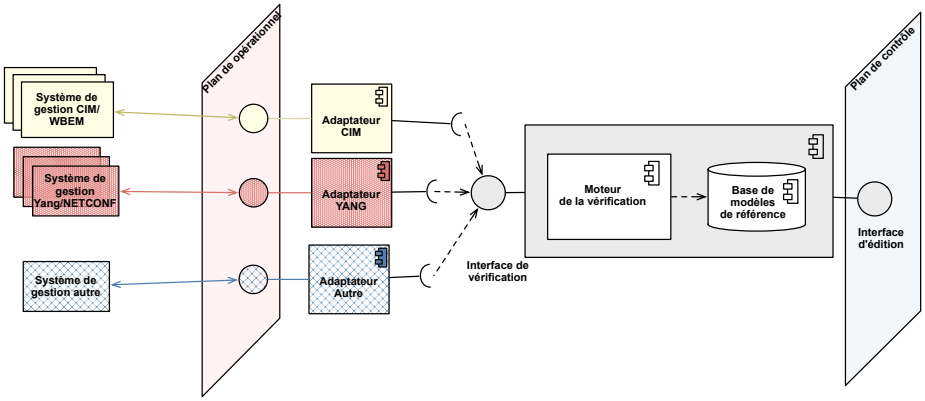


FIGURE 2.5 – Architecture du service de vérification

Ces travaux ont été validés expérimentalement sur deux cas d'étude : un premier sur la reconfiguration de l'intergiciel orienté messages JORAM (avec un appel direct du service de véri-

fication via un système de gestion *ad hoc* JMX) et un second sur la reconfiguration de machines virtuelles en environnement CIM/WBEM (avec un appel indirect du service de vérification via l'adaptateur dédié `CIM2MeCSVAdapter`). Dans les deux cas nous avons utilisé un jeu de configurations prédéfinies dont certaines valeurs variaient de manière aléatoire. Entre 10 et 100 contraintes étaient présentes dans les deux modèles de référence. Ces expérimentations ont apporté une validation fonctionnelle de l'approche : le service de vérification a pu évaluer les configurations et renvoyer des erreurs pour celles qui étaient invalides. Par ailleurs, nous avons mesuré le surcoût temporel de cette vérification. Ce dernier augmente linéairement en fonction de la taille de la configuration et du nombre de contraintes. Il est à noter toutefois que 60 à 80% du surcoût total de la vérification est lié à la consultation des valeurs d'état opérationnel sur le système géré pour l'évaluation des contraintes en ligne [Aku+13a].

2.3 Vérification basée sur des tests dans un environnement réseau virtualisé

Après des travaux de vérification de configurations basée sur de l'ingénierie dirigée par les modèles, nous nous sommes intéressés à d'autres approches avec notamment celles basées sur de la génération de tests. Bien que l'objectif global reste toujours dans un contexte de vérification du comportement d'un système, le point de départ n'était pas sur la configuration de ce système mais plutôt sur les exigences qu'il devait respecter. Par ailleurs ces travaux ont davantage ciblé le domaine des réseaux, moins la vision système et intergiciel privilégiée dans la section précédente.

Ces travaux se sont inscrits dans le cadre du projet IREHDO2 (2016-2019) financé par la DGA (Direction Générale de l'Armement) et regroupant les partenaires AIRBUS, THALES, ISAE, INPT, UPS et ONERA. Dans ce projet, notre équipe a travaillé sur une méthodologie d'ingénierie des exigences de sécurité pour un réseau industriel de type avionique. L'équipe a proposé une méthodologie d'ingénierie des exigences de sécurité réseau basée sur les niveaux d'abstraction définis dans l'approche SABSA [SCL05], à savoir les points de vue *business*, *architect*, *designer*, *builder* et *tradesman*. Les niveaux *business* et *architect* correspondent aux phases de définition des besoins et objectifs de sécurité stratégiques ; les niveaux *designer* et *builder* sont en charge des phases de conception de la sécurité du système ; et enfin le niveau *tradesman* est responsable de l'implémentation finale des services et mécanismes de sécurité. Les travaux auxquels j'ai contribué ont porté sur le niveau *builder* et ont consisté à construire et tester une architecture réseau répondant aux exigences de communication et de sécurité (travail réalisé dans le cadre d'un co-encadrement de trois ingénieurs contractuels).

2.3.1 Méthodologie pour vérifier les exigences de communication et sécurité

Suite à la spécification et au raffinement des exigences de sécurité réseau de haut niveau, notre objectif était de pouvoir concevoir une architecture réseau concrète (intégrant les hôtes, les équipements d'interconnexion, le plan de contrôle...) et vérifier en cours d'exécution si elle respectait ou non ces exigences. L'ambition était d'avoir une solution flexible (que l'on puisse déployer rapidement dans des environnements matériels génériques) et modulaire (pour pouvoir l'enrichir dans le temps). Dans ce contexte, notre proposition a été de s'appuyer sur une démarche reposant sur le paradigme SDN, illustrée dans la figure 2.6. Les deux premières phases ont consisté à concevoir un réseau « minimal » à partir de la spécification des exigences. Pour cela nous nous sommes appuyés sur des travaux existants dans le domaine de la conception sûre de réseaux SDN (cf. section 2.3.2). Les trois dernières phases ont porté sur la configuration et le déploiement d'un tel réseau dans un environnement entièrement virtualisé, ainsi que sur la vérification des exigences par génération de paquets de tests et analyse des résultats (cf. section 2.3.3).

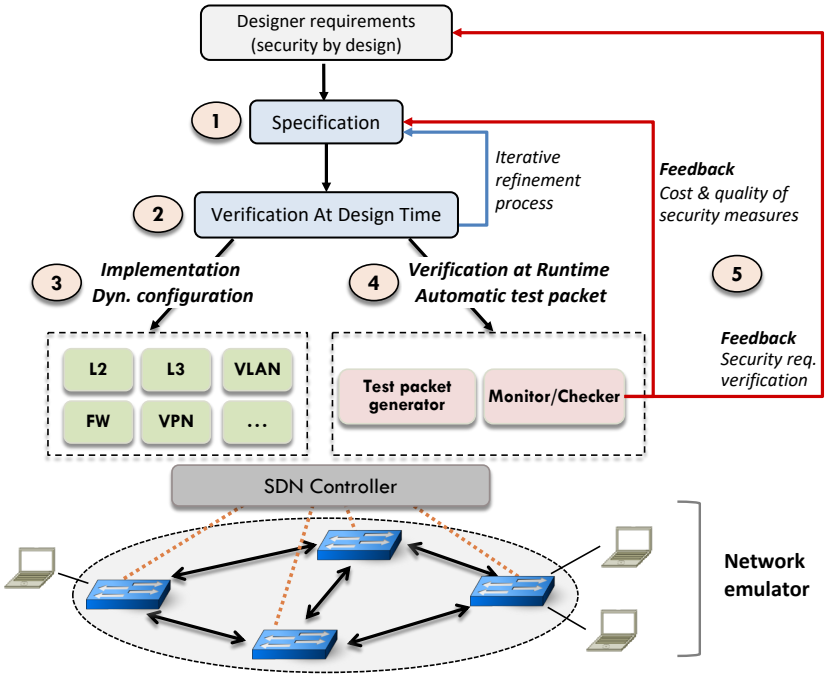


FIGURE 2.6 – Méthodologie pour la génération de tests dans un environnement SDN

2.3.2 Architecture réseau correcte par construction

La méthodologie d'ingénierie des exigences de sécurité réseau proposée par les vues *architect* et *designer* s'appuie sur de l'analyse de risque et débouche sur une modélisation basée sur le concept de segmentation réseau ou *network zoning* [Lab+19]. Des zones de sécurité représentent différents niveaux de confiance et regroupent des entités qui sont soumises à des exigences de protection similaires. Les exigences du concepteur peuvent être classifiées en deux catégories. La première couvre des exigences liées à l'architecture générale du système, à travers les concepts d'`Agent`, de `Zone` et de `Medium`. La seconde catégorie couvre des exigences de communication à travers les concepts de `Flow`, `Access Control Filter` et `Filter Rule`.

Une exigence de sécurité ne décrit pas comment la sécurité doit être mise en œuvre concrètement mais décrit le comportement attendu. L'exigence n'intègre donc pas les choix techniques en termes d'architecture physique ou de protocoles à mettre en place. Par exemple, une exigence peut spécifier que les applications `A` et `B` ne doivent pas pouvoir communiquer ensemble mais d'un point de vue technique cela peut se concrétiser en isolant les hôtes dans des VLAN différents ou encore en déployant un pare-feu avec les règles adéquates. Notre objectif n'est donc pas de proposer une configuration réseau unique pour répondre aux exigences de sécurité mais plutôt de construire une première architecture vérifiant ces exigences, que le concepteur pourra ensuite enrichir ou modifier en fonction de ses besoins ou autres contraintes non exprimées initialement (coût, complexité de la configuration...). Pour répondre à cet objectif, nous nous sommes appuyés sur les travaux de [Ryz+17] pour garantir que l'architecture réseau construite à partir des exigences soit conforme à ces dernières. Ces travaux proposent le *framework* `Cocoon` qui s'appuie sur une approche correcte par construction dans laquelle chaque étape est issue de la précédente par raffinement et vérification à partir d'outils de *model checking*.

Dans ce contexte nous avons commencé par une première spécification de haut niveau dans le *framework* `Cocoon` pour représenter les flux entre applications issus des concepts d'`Agent` et de `Flow`. Puis, un premier raffinement pour prendre en compte le concept de `Zone` en regroupant les hôtes d'une même zone dans un même domaine de diffusion associé à un réseau IP particulier

et une passerelle d'entrée-sortie de la zone. C'est sur cette passerelle que les règles de filtrage sur les flux sont positionnés (concept de `Filter Rule`). Un deuxième raffinement a permis de représenter les communications intra-zone avec un équipement dédié et des règles de filtrage intra-zone. Enfin, un troisième raffinement a porté sur le concept de `Medium` pour mettre en œuvre les communications inter-zones (choix d'une topologie en étoile par défaut). Le résultat de ces différents raffinements a permis d'obtenir une première architecture réseau conforme aux exigences du départ (cf. figure 2.7). Chaque étape du processus est spécifiée dans le langage `Cocoon` et vérifiée avec le *model checker* Corral [LQL12]. Les détails d'implémentation peuvent être trouvés dans le rapport [IRE18].

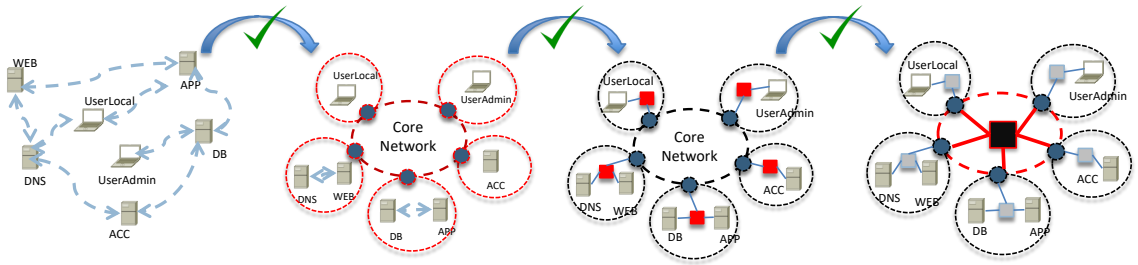


FIGURE 2.7 – Processus de raffinement pour une architecture réseau conforme aux exigences

2.3.3 Génération de tests dans un réseau SDN

L'architecture réseau issue du processus de raffinement précédent respecte bien les exigences définies au départ car elle a été vérifiée formellement lors de sa construction. Cependant, d'une part il est possible que cette architecture soit modifiée par le concepteur pour respecter d'autres exigences (coût, énergie...), d'autre part le passage d'une architecture vers une configuration technique peut introduire des erreurs, ce qui nécessite d'effectuer des tests complémentaires en cours d'exécution. Dans le cadre du projet IREHDO2, pour générer et exécuter ces tests, nous avons choisi de nous appuyer sur un environnement virtualisé (indépendant de matériels spécifiques) et suivant une architecture SDN afin de pouvoir réutiliser des modules de contrôle existants (L2, L3...) et centraliser la génération des tests.

La virtualisation du réseau est réalisée dans l'émulateur *Mininet* [Min15]. Des fichiers de configuration sont automatiquement générés à partir de l'architecture réseau spécifiée à l'étape précédente. Ces derniers incluent la liste des hôtes et des commutateurs, la topologie et le plan d'adressage. Les fonctions de contrôle du réseau (VLAN, routage, ACL) sont quant à elles mises en œuvre dans le contrôleur SDN *Faucet* [Fau18] dont la configuration est également générée à partir de l'architecture.

Pour les tests en cours d'exécution, nous nous sommes intéressés à des tests de connectivité entre les hôtes et entre les applications en mode « supervision active » (génération de flux de tests dédiés). L'objectif était de vérifier si les exigences en matière de flux autorisés et interdits étaient bien respectées sur la configuration du réseau virtuel. Cette proposition s'appuie sur des travaux existants dans le domaine de la génération automatique de paquets de test dans un environnement SDN [Zen+12 ; LVB14]. En fonction des `Zone` et des `Filter Rule` définis dans les exigences de sécurité, des scripts de génération et de collecte de paquets (ICMP, UDP, TCP) sont déployés sur certains hôtes du réseau. Les résultats de ces tests sont ensuite remontés à un module de contrôle qui produit un rapport d'exécution.

L'ensemble de cet environnement virtuel de tests réseau est illustré dans la figure 2.8. Les détails techniques sont décrits dans le rapport [IRE18]. Bien que ces travaux ne se soient pas concrétisés par des publications scientifiques, ils nous ont permis de consolider et étendre des

compétences techniques sur l'automatisation et la programmabilité des réseaux, ainsi que de développer notre expérience d'encadrement d'ingénieurs.

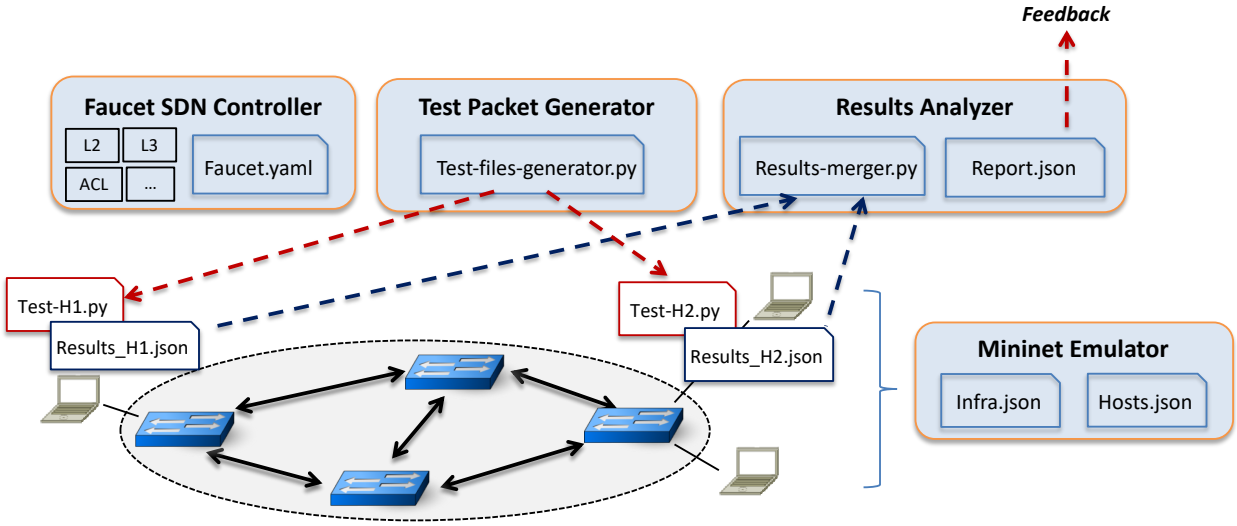


FIGURE 2.8 – Environnement SDN utilisé pour les tests réseau et sécurité

2.4 Vérification basée sur de l'apprentissage supervisé

Nous avons commencé à nous intéresser à l'application de l'apprentissage automatique au domaine des réseaux en 2018, d'abord par l'encadrement d'un stage en L3 informatique, puis par l'encadrement d'une thèse en contrat CIFRE avec l'opérateur de télécoms *IMS Networks*[1] à partir de fin 2019. Les travaux actuels de la thèse de El-Heithem Mohammedi portent sur la détection et la localisation d'erreurs de configuration dans le contexte d'un réseau d'opérateur fournissant, en particulier, des services de réseau privé virtuel (VPN, *Virtual Private Network*).

Comme nous en avons discuté dans la section 2.2, la vérification de configurations est effectuée traditionnellement en phase de conception à partir de règles prédéfinies encadrant la conformité structurelle des données, par exemple des contraintes sur les types ou sur des dépendances entre paramètres. Une des limites de ces approches est d'assurer la complétude de ces règles ainsi que leur mise à jour. De plus, il est très difficile de détecter des oublis avec une solution à base de règles. Une configuration peut en effet être valide d'un point de vue syntaxique et structurel mais incomplète de par l'objectif visé. Par exemple, omettre d'activer la table de routage d'un client dans un routeur de bordure du fournisseur de service va exclure le trafic vers ce site client bien que tout le reste de la configuration soit correct. En conséquence, l'ambition de nos travaux dans ce domaine est d'explorer un paradigme différent basé sur de l'apprentissage supervisé dans lequel un modèle est entraîné sur des centaines de configurations déjà labellisées. L'objectif est de pouvoir détecter si une configuration donnée est incorrecte ou incomplète, ainsi que de localiser précisément l'erreur.

2.4.1 Contexte des réseaux VPN IP BGP/MPLS

Le service de réseau privé virtuel de niveau 3 (L3 VPN ou encore IP VPN) permet à un opérateur d'interconnecter plusieurs sites client tout en garantissant une qualité de service et un isolement des flux entre clients [RR06]. C'est un service réseau très répandu au sein des opérateurs

1. https://www.imsnetworks.com/

de télécoms disposant d'une infrastructure MPLS [Ghe06]. La figure 2.9 illustre un exemple simple d'une architecture L3 VPN composée de trois clients (A, B et C), chacun disposant de deux ou trois sites. La configuration d'un tel réseau n'est pas triviale, celle-ci dépendant de plusieurs protocoles et d'un grand nombre de paramètres. Chaque site connecté au VPN doit avoir un routeur CE (*Customer Edge*) attaché à un ou plusieurs routeurs PE (*Provider Edge*). Le routeur CE est responsable de l'acheminement du trafic client vers le réseau de l'opérateur en utilisant du routage statique ou dynamique (via un protocole de routage tel que OSPF ou eBGP). Le routeur PE se situe à la bordure du réseau de l'opérateur, il contient une table VRF (*VPN Routing and Forwarding* ou encore *Virtual Routing and Forwarding*) pour chaque VPN. Cette table de routage permet d'isoler les routes entre les différents clients, routes qui sont apprises à partir du routeur CE connecté au PE, ainsi que des autres PE impliqués dans le VPN (via le protocole MP-BGP). La complexité d'un tel réseau augmente en fonction de sa taille, dépendante par exemple du nombre de clients ou encore du nombre de sites par client. Cette complexité peut entraîner des oublis ou des erreurs dans la configuration du réseau, avec un impact direct sur la connectivité des différents sites.

Le retour d'expérience du centre d'opérations réseau (NOC, *Network Operations Center*) d'IMS Networks a montré que la majorité des incidents réseau étaient dus à des erreurs sur la configuration des VPN en eux-mêmes et non sur la configuration du réseau dorsal (*backbone*) de l'opérateur. En effet, quand le réseau dorsal est en production, sa configuration reste stable et ne change que très rarement. En revanche, les VPN quant à eux peuvent être amenés à évoluer quotidiennement avec l'ajout, la mise à jour ou la suppression de sites clients. Face à ce constat nous avons orienté les travaux sur un sous-ensemble de la configuration du réseau ciblant tous les paramètres se rapportant au routage CE-PE, ainsi qu'à la définition et à l'affectation des VRF responsables du routage PE-PE.

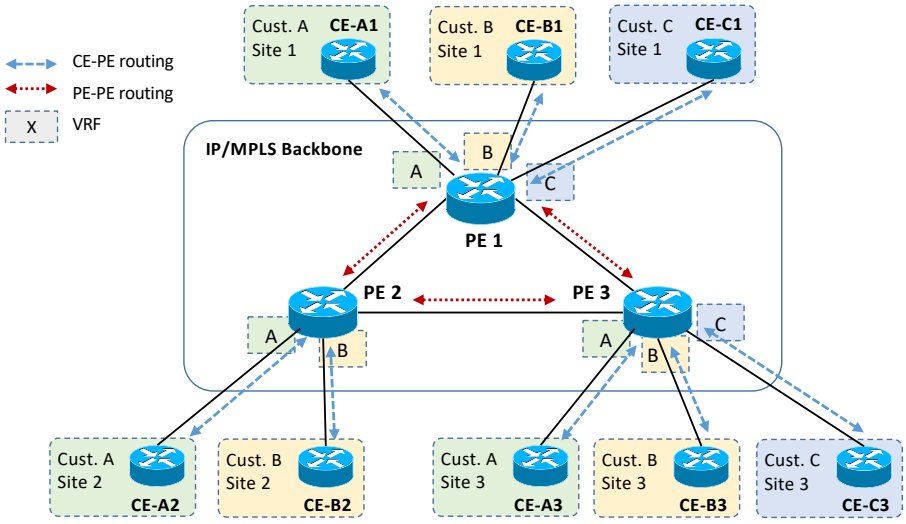


FIGURE 2.9 – Aperçu d'une architecture L3 VPN

2.4.2 Approches traditionnelles (DT, RF, MLP)

La première problématique à laquelle nous avons essayé de répondre était de savoir si un système était capable d'apprendre si une configuration cible garantissait la connectivité de tous les sites des clients. En d'autres termes, le système devait identifier si un site n'était pas joignable suite à une erreur de configuration. Cela revient à un problème de *classification* : chaque routeur CE doit être classifié dans l'une des deux catégories suivantes : « joignable » ou « non joignable »

(en ne considérant que la configuration, c'est-à-dire sans prendre en compte l'état opérationnel du réseau). Bien qu'il y ait seulement deux catégories (ou classes), ce n'est pas un problème de classification binaire puisque une configuration inclut plusieurs CE et une faute peut impacter plusieurs CE. Par exemple, dans la figure 2.9, le système doit être capable de classifier chaque CE, c'est-à-dire trouver un vecteur de 8 labels binaires (`CE-A1`, `CE-B1`, `CE-C1`, etc.), chaque label du vecteur valant 0 si le CE est joignable, 1 sinon. C'est un problème de classification multi-labels [SM18] dans lequel un vecteur binaire y en sortie (la connectivité des CE) est déduit à partir d'une entrée x (la configuration du réseau).

Génération du jeu de données

Dans le domaine des réseaux de communication, une grande quantité d'information est disponible : traces de trafic, mesures de performance, journaux d'évènements, etc. Cependant, les configurations réseau sont une exception car d'une part, elles sont spécifiques à une architecture particulière et d'autre part, lorsque la configuration est en production, il est inutile pour l'opérateur d'en maintenir plusieurs centaines supplémentaires. De plus, il est très difficile d'obtenir un grand nombre d'erreurs de configuration dans un réseau en production et il est irréaliste d'injecter des fautes dans le réseau uniquement pour avoir des données d'entraînement. Nous avons donc décidé de générer des données basées sur des configurations existantes spécifiées et déployées par l'opérateur IMS Networks. Cela nous permet de construire un jeu de données suffisamment important et diversifié, intégrant des configurations correctes et incorrectes.

Dans cet objectif, à partir de configurations valides dupliquées plusieurs milliers de fois, des erreurs sont ajoutées sur le routage CE-PE ainsi que sur le routage inter-PE. Ces erreurs sont générées en choisissant de manière aléatoire le type de faute (*route distinguisher*, peering BGP, préfixe IP...) ainsi que l'équipement sur lequel appliquer la faute. Ce tirage aléatoire a été fait de telle sorte à répartir uniformément les erreurs sur toutes les configurations avec à l'arrivée des problèmes de connectivité sur environ 50% des sites de VPN.

Algorithmes d'apprentissage supervisé

Pour mettre en œuvre notre problème d'apprentissage, nous nous sommes appuyés sur les trois méthodes suivantes : arbre de décision (*decision trees*), forêts aléatoires (*random forest*) et perceptron multicouche (*multi-layer perceptron*). Ce sont trois approches communément utilisées dans la communauté de l'apprentissage automatique pour traiter des problèmes de classification supervisée [RN19].

— L'apprentissage par arbre de décision (DT) consiste à construire un arbre depuis un ensemble de données étiquetées, dans lequel chaque nœud décrit un test sur une variable, chaque branche de l'arbre représente un résultat du test, et chaque feuille contient la classe cible. Un algorithme glouton est utilisé pour construire l'arbre [RM14].

— Les forêts aléatoires (RF) sont une extension de l'apprentissage par arbre de décision, qui contiennent plusieurs arbres entraînés sur des sous-ensembles de données. La classe choisie en sortie est celle la plus prédite par les différents arbres de décision [CCS12].

— Le perceptron multicouche (MLP) est un type de réseau de neurones à propagation directe (*feedforward neural network*) dans lequel l'information circule de la couche d'entrée vers la couche de sortie (le résultat à prédire). Les neurones de chaque couche sont connectés aux neurones de la couche suivante avec un certain poids. Ces poids sont ajustés durant l'entraînement en utilisant la rétropropagation du gradient [Sch15].

Les tests avec les arbres de décision et forêts aléatoires ont été implémentés avec le module Python Scikit-learn [Sci20] et ceux s'appuyant sur le perceptron multicouche avec Keras [Ker20], une API de réseaux de neurones de haut niveau construite sur la plateforme TensorFlow [Ten20].

Résultats

Nos expérimentations ont été réalisées sur plusieurs jeux de données dans lesquels nous avons fait varier la taille du réseau (20, 50 et 100 routeurs client) et les types d'erreurs de configuration (une dizaine d'erreurs réparties dans trois catégories). Chaque jeu de données était composé de 150000 configurations séparées en deux groupes : 70% pour l'entraînement et 30% pour les tests. Les résultats détaillés sur les temps d'entraînement et sur les performances des différents algorithmes, en fonction de la taille du réseau et du type de faute, sont présentés dans l'article [MLF20].

D'une manière générale, l'approche MLP expose de meilleurs résultats que les approches DT et RF. Nous pouvons le voir dans la figure 2.10 qui compare les performances des algorithmes pour un réseau contenant 10 routeurs PE et 100 routeurs CE. La justesse de la prédiction (*accuracy*) arrive à presque 80% avec la solution MLP, contre environ 60% et 70% pour les approches DT et RF respectivement. Néanmoins, bien que la « précision » soit élevée (métrique *precision* à presque 90%), la métrique « rappel » (*recall*) est plus faible, ce qui signifie un nombre plus important de faux négatifs, c'est-à-dire une erreur de configuration non détectée. Ce point est une limite que nous avons analysée et corrélée à un certain type de faute : des erreurs non détectées sur la configuration du routage CE-PE. Cela s'explique principalement par l'impact de l'erreur de configuration sur la connectivité des sites. Une erreur sur un routeur CE est isolée et n'impacte que la connectivité du site concerné, alors qu'une erreur sur un lien VPN PE-PE affecte potentiellement plusieurs sites, ce qui est plus facile à détecter par le système d'apprentissage.

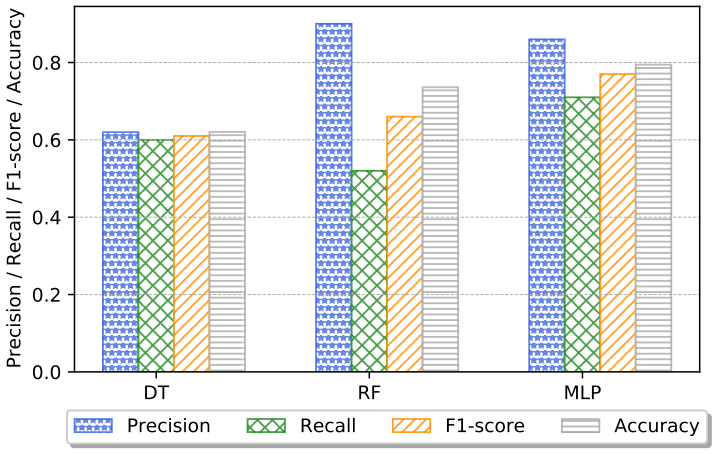


FIGURE 2.10 – Comparaison des algorithmes DT/RF/MLP pour un réseau de 100 routeurs CE

Bien que cette première approche présente des résultats intéressants, plusieurs limites demeurent. Outre le problème évoqué sur certaines erreurs de configuration sur le routeur CE qu'il est difficile de prédire, une faiblesse importante dans les trois modèles précédents réside dans la nature statique de la taille des données en entrée. En effet, les modèles sont entraînés sur un nombre maximum de clients et de routeurs, ce qui implique que la configuration cible doit respecter cette contrainte, sinon le modèle doit être entraîné de nouveau. L'alternative d'augmenter ce nombre maximum de paramètres de configuration en entrée impliquerait davantage de propriétés non représentatives, ce qui diminuerait les performances de l'apprentissage. Enfin, la dernière limite concerne l'identification des erreurs. La modélisation actuelle du problème permet de détecter qu'un site spécifique n'est pas joignable dû à une erreur de configuration mais n'indique pas la localisation précise de l'erreur. Le nombre d'erreurs considéré est par ailleurs

relativement petit (une dizaine) par rapport à la configuration complète d'un VPN (une trentaine de paramètres significatifs). Ces différentes limites nous ont encouragé à poursuivre les travaux et à étudier d'autres approches d'apprentissage supervisé.

2.4.3 Approche utilisant une structure de données en graphe (GNN)

La possibilité de pouvoir exprimer explicitement des relations entre certains paramètres de configuration présents sur des équipements différents nous a conduit à étudier les réseaux de neurones en graphes, ou GNN (*Graph Neural Networks*). C'est une classe de réseaux de neurones qui s'applique sur des données représentées sous la forme de graphes. Ils ont reçu récemment beaucoup d'intérêts de la part de la communauté scientifique travaillant sur des données dont la structure sous-jacente est non Euclidienne, comme l'étude des réseaux sociaux, des réseaux de citations bibliographiques ou encore des formes géométriques en 3D [Bro+17].

De manière générale, chaque nœud (et éventuellement chaque lien) du graphe embarque un ensemble de propriétés (*embeddings*) qui sont combinées avec celles de leurs voisins à chaque itération (processus dit de *message passing* ou *neighbourhood aggregation*). Le nombre d'itérations dépend du nombre de couches du GNN, la dernière produisant un vecteur de valeurs numériques représentant le résultat de la tâche d'apprentissage. Plusieurs objectifs peuvent être ciblés en fonction du problème, par exemple de la classification de nœuds (apprendre une nouvelle caractéristique sur un nœud, comme une recommandation de contenu pour des utilisateurs d'un réseau social) ou de la classification de graphes (déduire une propriété sur le graphe en lui même, comme l'identification d'une molécule). Les articles [Zho+20] et [Wu+21] présentent une synthèse des travaux récents sur les GNN.

Dans notre contexte, outre la représentation des liens client-fournisseur, nous souhaitons surtout représenter la topologie logique du VPN qui dicte sa configuration ainsi que les échanges possibles d'informations de routage entres les sites. Par ailleurs, contrairement aux approches précédentes (2.4.2), nous souhaitons pouvoir détecter et localiser des erreurs de configuration indépendamment de la taille du réseau, du nombre de clients et du nombre de sites par client.

Modélisation des données

D'un point de vue de sa configuration, un VPN MPLS/BGP comprend deux parties : la topologie du VPN, c'est-à-dire les politiques qui gouvernent les interactions entre les différents sites, et l'intégration des sous-réseaux du client, c'est-à-dire l'annonce des routes pour chacun des sites du client. Essayer de capturer ces deux questions dans un modèle GNN unique impliquerait une agrégation de tous les paramètres de configuration dans le même modèle, et donc un nombre plus important de propriétés non représentatives pour une tâche d'apprentissage donnée (un des écueils des approches précédentes). Par exemple, une erreur de communication entre un routeur CE particulier et son routeur PE de rattachement ne dépend que de la configuration de ces deux équipements. En revanche, une anomalie sur l'acheminement de paquets entre deux sites clients peut dépendre d'un des routeurs CE des sites ou bien d'un routeur PE impliqué dans l'établissement du VPN du client. Nous avons donc considéré deux types de graphes pour la modélisation des données : un modèle PE-PE pour le routage « inter-sites » et un modèle CE-PE pour le routage « site-operateur ».

L'objectif du premier modèle PE-PE est de détecter et d'identifier les erreurs de configuration liées aux interactions entre les routeurs PE impliqués dans un VPN. Par exemple, oublier d'activer la VRF sur un PE ou bien se tromper sur un de ses paramètres (*route distinguisher*, *route target*, etc.). Pour ce modèle, nous avons choisi de spécifier chaque VPN sur un graphe spécifique, chaque nœud représentant un routeur PE. Un nœud embarque les paramètres de configuration du VPN pour ce nœud, et une propriété binaire est rajoutée sur les liens pour indiquer si les voisins concernés doivent s'échanger des informations de routage. Cette propriété sur les liens

nous permet de capturer la topologie du VPN. La partie du haut de la figure 2.11 schématise la structure des trois graphes représentant les trois VPN de l'exemple 2.4.1. Le client `A` dispose d'un VPN maillé (*full-mesh*), les trois arêtes du graphe ont donc la propriété 1. Le client `B` a un VPN en étoile (*hub-and-spoke*), ainsi une arête a la propriété 0. Quant au client `C`, son VPN n'est présent que sur deux sites, un des trois nœuds n'a donc pas de configuration.

L'objectif du second modèle CE-PE est de détecter et d'identifier les erreurs de configuration qui affectent la communication entre un CE et son PE de rattachement. Par exemple, une erreur sur la configuration du routage statique ou du protocole de routage qui est utilisé (tel eBGP). Pour ce modèle, nous avons choisi de spécifier chaque connexion CE-PE sur un graphe spécifique, avec donc seulement deux nœuds représentant les routeurs CE et PE. Les propriétés sur les nœuds sont des paramètres de configuration qui peuvent être positionnés ou mis à jour quand un site est ajouté, modifié ou supprimé d'un VPN. La partie du bas de la figure 2.11 illustre la structure des graphes pour tous les sites des trois clients de l'exemple 2.4.1.

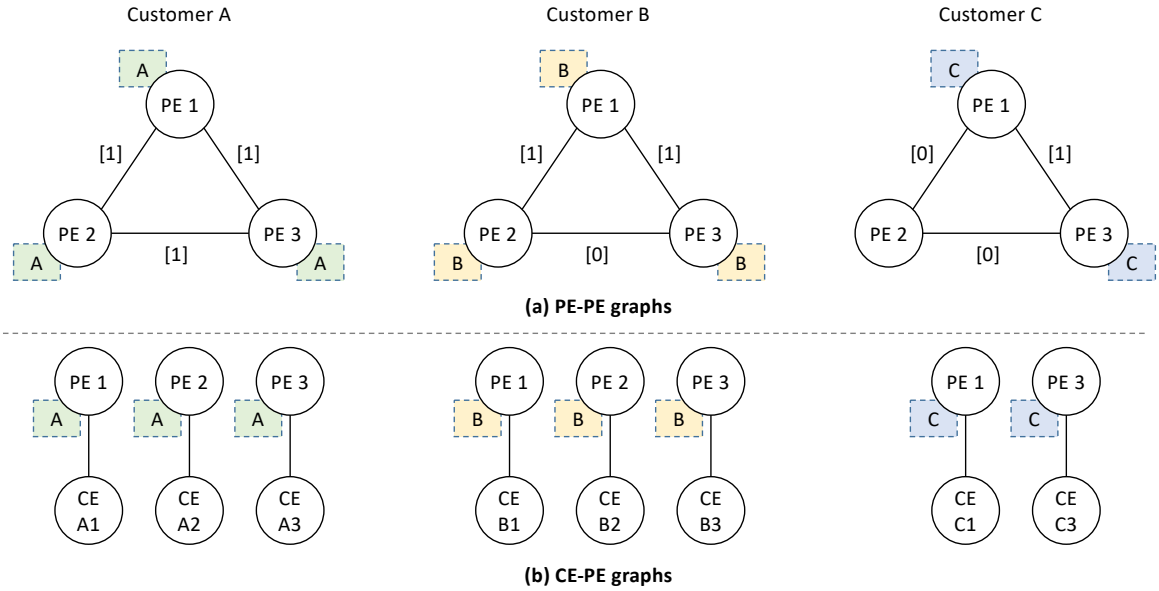


FIGURE 2.11 – Modélisation en graphes de l'exemple 2.4.1

Fonctions d'apprentissage

Notre objectif pour le modèle CE-PE est de pouvoir déterminer si chaque graphe (c'est-à-dire chaque connexion CE-PE) est correct ou non, avec par conséquent une sortie pour chaque graphe. C'est une tâche d'apprentissage qui relève de la **classification de graphes**. Puisque nous souhaitons identifier précisément l'erreur, la sortie pour chaque graphe sera un vecteur de plusieurs classes, chaque classe représentant une faute possible. Cette méthode d'apprentissage supervisé est une classification de graphes multi-classes. La propagation de l'information entre les noeuds durant la phase d'apprentissage s'appuie sur des convolutions de graphes ou GCN (*Graph Convolutional Networks*) [KW17].

Concernant le modèle PE-PE, notre objectif est d'avoir, pour chaque graphe (c'est-à-dire chaque VPN), une sortie pour chaque nœud (c'est-à-dire chaque routeur PE), indiquant si une erreur de configuration est présente sur ce nœud ou non. C'est une tâche d'apprentissage qui relève de la **classification de nœuds**. De plus, comme dans le cas précédent, puisque nous voulons identifier précisément l'erreur, la sortie pour chaque nœud sera un vecteur de plusieurs classes, chaque classe spécifiant une faute possible. Cette méthode d'apprentissage est une clas-

sification de nœuds multi-classes. Notre modèle de données ayant un label sur les liens (présent pour représenter la topologie du VPN), nous nous sommes appuyés sur des réseaux de neurones à convolution sur des graphes intégrant des filtres sur les arêtes, appelés GNN ECC (*Edge-Conditioned Convolutions*) [SK17].

Résultats

Les deux modèles ont été implémentés en utilisant Spektral [GA21], une bibliothèque Python pour de l'apprentissage profond avec des réseaux de neurones en graphes. Spektral implémente la plupart des types de couches de GNN dont les *Graph Convolutional Networks* (GCN) et *Edge-Conditioned Convolutions* (ECC) évoqués précédemment.

De manière similaire aux travaux précédents, nous avons généré des configurations sur lesquelles nous avons injecté des erreurs aléatoirement. Nous avons augmenté le nombre de types d'erreurs possibles pour atteindre une trentaine (contre une dizaine dans la première approche). Concernant la taille du réseau, nous nous sommes inspirés de celui de notre partenaire IMS Networks avec un réseau de cœur de 20 routeurs PE et entre 10 et 30 routeurs CE pour chaque client. La modélisation des données en graphe constitue un réel atout pour la phase de tests car la vérification pourra se faire par VPN ou par site. En effet, l'apprentissage se fait graphe par graphe (VPN par VPN dans le cas du modèle PE-PE, et site par site dans le cas du modèle CE-PE) et il est donc inutile d'entraîner le modèle à nouveau si on ajoute, modifie ou supprime un VPN, ou un site sur un VPN. Au final, nous avons entraîné les modèles sur 1000 configurations intégrant des VPN *full-mesh* et *hub-and-spoke*, soit 1000 graphes PE-PE et environ 20000 graphes CE-PE (c'est-à-dire tous les liens CE-PE de tous les VPN). 80% du jeu de données est utilisé pour l'entraînement (dont 10% pour la validation) et 20% pour les tests. Les résultats sont présentés dans la figure 2.12, classés par type d'erreur (2.12a et 2.12c), topologie des VPN (2.12b) et taille des VPN (2.12d).

Concernant le modèle PE-PE, le *F1-score*[2] est proche de 90%, sauf pour les classes d'erreur 5 et 7 où il est proche de 80% (cf. histogramme 2.12a). Ces classes représentent des erreurs sur des préfixes IP associés aux sous-réseaux qui doivent être importés ou exportés dans les politiques de routage pour les topologies *hub-and-spoke*. Nous expliquons cette baisse de performance principalement par deux raisons. D'abord le paramètre « sous-réseau IP » du site client n'apparaît qu'une seule fois dans le modèle PE-PE et n'a donc pas d'autres paramètres auxquels il pourrait se comparer ou se relier (ce paramètre n'est présent ailleurs que dans l'autre modèle CE-PE). De plus, les VPN *hub-and-spoke* représentent 50% des données d'entraînement donc les erreurs sur les politiques de routage sont moins représentées que les autres (classes 1 à 4 présentent sur les deux topologies). La sous-figure 2.12b confirme ces résultats qui montrent que le modèle détecte mieux les erreurs de configuration sur les VPN *full-mesh* qui n'ont pas de politique de routage.

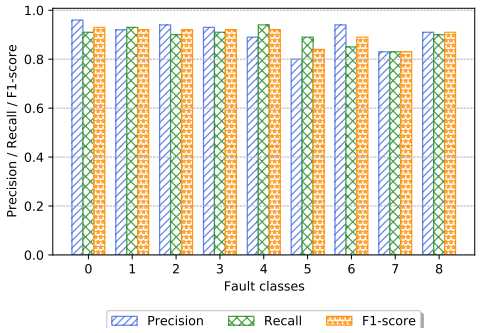
Les tests sur le modèle CE-PE affichent de très bon résultats (cf. histogramme 2.12c), quelle que soit la classe d'erreur (routage statique, routage eBGP, interface...). La nature moins complexe des graphes (2 nœuds seulement) ainsi que la taille du jeu d'entraînement (environ 16000 instances) peuvent expliquer cette performance de classification élevée.

Pour finir, l'histogramme 2.12d classe les résultats par rapport à la taille des VPN, c'est-à-dire le nombre de sites client par VPN (ces résultats sont issus de tests effectués sur de nouveaux jeux de données générés en fixant le nombre de CE). Nous pouvons voir que le *F1-score* du modèle CE-PE reste constant à presque 100%, ce qui est logique car la variation du nombre de CE ne change rien aux graphes CE-PE, cela ne fait que varier le nombre de graphes à tester. Concernant le modèle PE-PE, le *F1-score* global pour les VPN de 10, 20 et 30 CE est similaire à la moyenne des scores de la figure 2.12a. Ce résultat est cohérent puisque le modèle a été entraîné

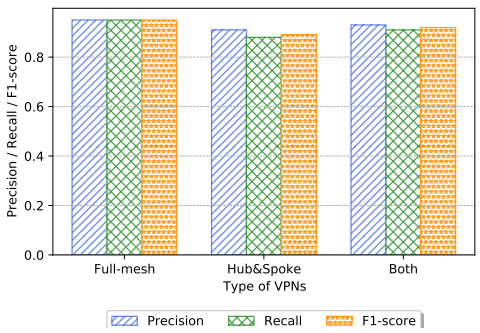
2. Le *F1 Score* est la moyenne harmonique entre les deux métriques *Precision* et *Recall*. Plus elle est élevée, moins il y a d'erreurs de prédiction (donc plus le modèle est performant)

avec des VPN de tailles comprises dans cet intervalle. Un point plus intéressant à noter est le résultat pour les VPN avec 3, 5 et 40 sites. Bien que ces tailles n'étaient pas présentes dans les données d'entraînement, la performance sur la détection des erreurs de configuration est très proche de celle des autres tailles de VPN, montrant que le processus d'apprentissage généralise correctement, sans surapprentissage (*overfitting*).

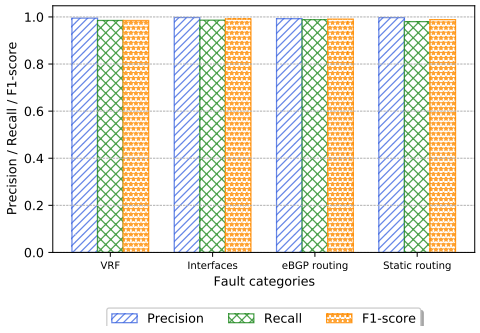
L'ensemble de ces résultats sont présentés dans l'article [MLF22]. Les travaux en cours se focalisent sur l'amélioration de la détection des erreurs sur les politiques de routage intra-VPN. Par ailleurs, d'autres types d'interconnexions client-fournisseur doivent être considérés, comme la possibilité pour un site client d'être connecté à deux routeurs de bordure pour des raisons de redondance.



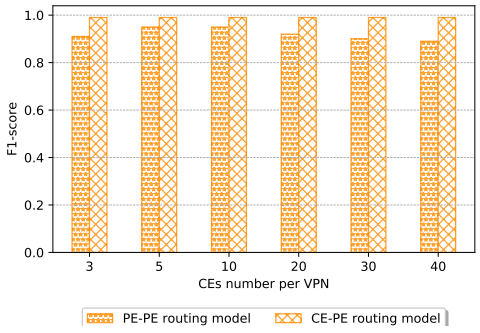
(a) PE-PE model, performance per fault type



(b) PE-PE model, performance per VPN type



(c) CE-PE model, performance per fault category



(d) F1-score according to the VPN's size

FIGURE 2.12 – *Precision/Recall/F1-score* pour les deux modèles GNN

2.5 Conclusion

Dans ce chapitre, nous avons présenté nos travaux sur la vérification de configurations système et réseau. Ils se sont appuyés sur diverses méthodes (ingénierie dirigée par les modèles, raffinement de spécifications et génération de tests, apprentissage supervisé) et ont ciblé des environnements différents (système/middleware, réseau industriel, réseau d'opérateur).

Dans notre contribution basée sur une approche d'ingénierie dirigée par les modèles, nous avons défini un métamodèle permettant de représenter des éléments de configuration, des données d'état opérationnel, et des contraintes hors-ligne et en ligne. L'intérêt est de pouvoir construire des modèles qui feront de la vérification « en ligne », c'est-à-dire qui détermineront si une configuration donnée peut être déployée sur le système cible courant sans provoquer une erreur ou dégrader les performances du système. Afin de pouvoir intégrer des plateformes de supervision existantes, une architecture modulaire et des interfaces ont été spécifiées. Une preuve de concept

a été développée et testée sur des configurations de petite taille (300 éléments gérés et 100 contraintes). Une limite de la solution reste le passage à l'échelle (essentiellement pour ce qui est du nombre de contraintes en ligne car, pour chacune d'elle, une requête de consultation d'état doit être émise) ainsi que la complétude des contraintes à vérifier.

Nos travaux réalisés dans le cadre du projet IREHDO2 se sont focalisés sur de la vérification de configurations réseau issues d'exigences de sécurité et de communication exprimées à un haut niveau d'abstraction (à travers des notions d'*agent*, de *zone* et de *medium*). La première phase a permis de générer une architecture réseau « minimale », selon une approche correcte par construction avec des règles de raffinement prédéfinies (séparation des domaines de diffusion, routeur unique d'entrée/sortie dans le domaine, topologie en étoile, etc.). Puis, une deuxième phase s'est concentrée sur la génération automatique de paquets de tests dans un environnement réseau virtualisé, afin de vérifier si les exigences sur les flux autorisés ou interdits étaient bien respectées. La partie raffinement de spécifications nous paraît pertinente dans le cadre du projet car la cible est un réseau industriel contraint, avec des configurations de faible complexité (essentiellement des VLAN et des ACL [3]). C'est un travail qu'il est probablement plus difficile de mettre en œuvre sur des réseaux ouverts, avec des configurations plus complexes. En revanche, la partie génération automatique de tests nous semble prometteuse pour d'autres types de réseaux. C'est d'ailleurs un axe que nous souhaitons poursuivre et continuer à investiguer comme nous le verrons dans le projet de recherche.

Les derniers travaux exposés dans ce chapitre concernent la vérification de configurations réseau basée sur des méthodes d'apprentissage supervisé. Les services de VPN IP offerts par un opérateur de télécommunications constituent le domaine d'application cible. L'originalité de l'approche est d'entraîner un service de vérification plutôt que de le construire à l'aide d'une base de règles prédéfinies. La méthode s'appuie sur des réseaux de neurones en graphes (GNN) qui modélisent les principaux paramètres de configuration du VPN ainsi que sa topologie logique. Les jeux de données sont générés à partir de configurations types issues de notre partenaire industriel. Le travail n'est pas terminé (thèse en cours) mais les résultats obtenus jusqu'à présent sont encourageants.

3. *Access-Control List*

Chapitre 3

Virtualisation et programmation d'un plan de contrôle SDN

En 2014, j'ai démarré un axe de recherche sur la thématique des réseaux logiciels (SDN, *Software Defined Networks*) en encadrant la thèse de Messaoud Aouadj [Aou16]. Il s'agissait là d'une des deux premières thèses sur le site Toulousain sur ces problématiques qui se sont révélées être par la suite très porteuses. La contribution principale du travail a été de définir un langage dédié à la conception de réseaux virtuels et à la spécification de politiques de contrôle et d'orchestration de services réseaux déployés sur de telles topologies virtuelles.

3.1 Enjeux de l'interface nord de l'architecture SDN

L'interface nord (*northbound API*) représente une des abstractions clés de l'écosystème SDN, étant donné que c'est l'interface qui sera utilisée par les opérateurs et les développeurs des applications de contrôle et de gestion du réseau. Cette interface a pour objectif principal d'abstraire les détails de bas niveau de l'infrastructure physique ainsi que ceux de l'implémentation du contrôleur SDN (tel le protocole utilisé par l'interface sud).

Les propositions initiales d'interface nord étaient fortement couplées au contrôleur, avec des API de bas niveau, et offrant très peu de possibilités de modularité et de réutilisabilité (par exemple [Gud+08]). Pour faire évoluer et enrichir cette interface nord, beaucoup de travaux ont été menés dans le domaine des langages de programmation dédiés au contrôle du réseau. Les premières contributions comme FML [Hin+09], Nettle [VAH10] et Procera [VKF12], ont proposé des langages fonctionnels et réactifs, masquant une grande partie de la complexité des interactions entre le programme de contrôle et le plan de données. D'autres langages comme Frenetic [Fos+10], NetCore [Mon+12] et Pyretic [Mon+13] ont été conçus dans l'objectif d'exprimer plus efficacement les règles de commutation des paquets, notamment en se basant sur des prédicats et des primitives pour spécifier des filtres et récupérer des statistiques sur le traffic. Par ailleurs, ces langages offrent plusieurs mécanismes pour résoudre les problèmes d'intersection entre les règles de différents modules de contrôle, principalement à travers l'introduction des opérateurs de composition séquentielle et parallèle. D'autres fonctionnalités plus avancées sont fournies par des langages tels que FatTire [Rei+13] qui permet d'exprimer des chemins réseau tout en positionnant des exigences de tolérance aux fautes ; ou encore le langage NetKat [And+14] qui propose un environnement de vérification formelle des programmes.

Une synthèse des travaux précédents et une analyse de leurs principales contributions ont été réalisées par [Aou16]. Un des résultats de cette étude a été de souligner l'importance du recours à de la virtualisation de réseau dans les langages les plus récents (notamment Splendid Isolation [Gut+12], Pyretic [Mon+13] et Merlin [Sou+14]). Ainsi, en utilisant ces langages, un opérateur peut définir ses propres topologies virtuelles en fonction de ses objectifs. En effet, ces approches permettent de spécifier des applications sur des visions simplifiées de l'infrastructure physique,

suivant un certain modèle d'abstraction. La figure 3.1 illustre ces vues abstraites du réseau sur lesquelles pourront être spécifiées des politiques de contrôle différentes (routage, équilibrage de charge, contrôle d'accès...) qui seront dans un second temps compilées et transposées sur l'infrastructure physique.

Un point fort de cette démarche est qu'elle facilite significativement la spécification du service réseau, étant donné que l'opérateur ne va considérer que les informations qui sont les plus pertinentes pour sa politique de configuration globale. Encore plus important, cette démarche permet d'augmenter fortement la réutilisabilité des programmes car les politiques sont spécifiées sur des topologies virtuelles et peuvent être par la suite utilisées sur des infrastructure physiques différentes.

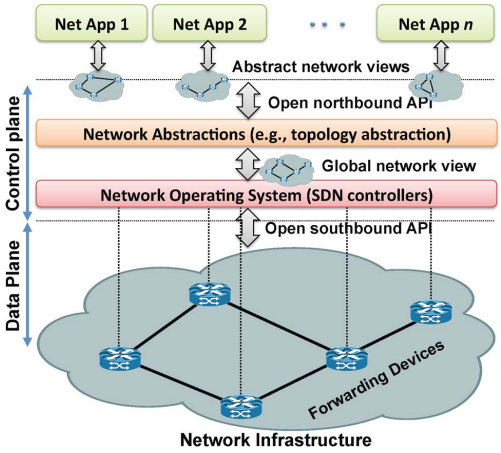


FIGURE 3.1 – Architecture SDN et ses principales abstractions [Kre+15]

Choix d'un modèle d'abstraction

Pour exploiter au mieux la virtualisation réseau et obtenir le maximum de gain en matière de facilité d'utilisation, de modularité et de réutilisabilité, le *modèle d'abstraction* représente un enjeu majeur étant donné que ses propriétés se répercuteront sur les propriétés de l'interface nord des contrôleurs SDN voire, in fine, sur l'orchestrateur du réseau.

Nous retrouvons dans la littérature deux principaux modèles d'abstraction : le modèle *One Big Switch* (*OBS*) [KR10] qui a inspiré des langages comme Merlin [Sou+14] et Maple [Voe+13], et le modèle *Overlay Network* [Cas+10] qui a été utilisé en particulier par les langages Pyretic [Mon+13] et Splendid Isolation [Gut+12]. Ces deux modèles sont schématisés dans la figure 3.2.

Le modèle *OBS* est un modèle qui a un très haut niveau d'abstraction puisqu'il abstrait la totalité de la topologie physique en un unique commutateur logique sur lequel tous les hôtes sont connectés. Un des principaux avantages est de permettre aux opérateurs de se concentrer pleinement sur la définition de leurs services réseaux complexes (règles de filtrage, politiques de QoS...) plutôt que de gérer le réseau virtuel en lui même. En revanche, toutes les fonctions réseau étant au sein d'un même conteneur logique, il sera plus difficile d'identifier la source d'un éventuel problème ou les commutateurs physiques impliqués dans un service réseau particulier. De plus, un autre inconvénient de ce modèle est qu'il ne permet pas de prendre en compte de possibles contraintes sur le réseau physique, par exemple il ne serait pas possible pour un opérateur de gérer au niveau virtuel deux groupes d'équipements physiques différents de par leur emplacement ou leur performance.

Le modèle *Overlay Network* consiste à superposer, au-dessus d'une infrastructure physique

partagée, un ou plusieurs réseaux virtuels composés de commutateurs logiques. Ces derniers sont assez semblables à des commutateurs physiques, ils exposent des tables de commutation et des ports virtuels qui sont connectés entre eux par le biais de liens virtuels. La plus grande force de ce modèle d'abstraction est sa flexibilité, étant donné qu'un opérateur a la possibilité de construire diverses topologies virtuelles en fonction de ses objectifs de haut niveau ou des contraintes physiques qu'il souhaite prendre en considération. Toutefois, ce modèle ne fait aucune distinction entre les politiques de transport et les services réseaux plus complexes, bien que ces deux types de politiques résolvent deux problématiques différentes [Cas+12]. De plus, cette approche diminue la modularité des programmes de contrôle, étant donné que les politiques de transport et les services réseau sont fortement couplés. À titre d'exemple, si un service de QoS est déplacé sur le réseau virtuel alors cela nécessitera de mettre à jour les commutateurs virtuels afin de rediriger les flux vers la nouvelle localisation du service.

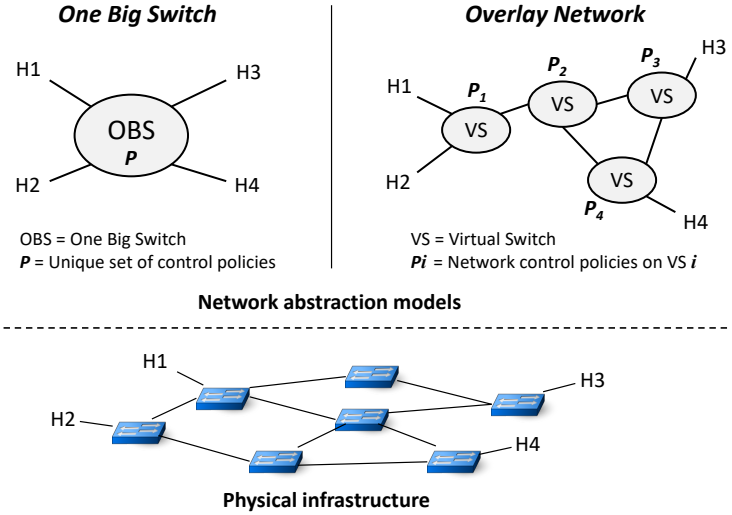


FIGURE 3.2 – Les modèles d'abstraction réseau existants

3.2 Le modèle d'abstraction *Edge-Fabric*

3.2.1 Composants virtuels du modèle

Afin de répondre aux limites des modèles d'abstraction réseau existants, nous nous sommes basés sur un concept bien connu dans la communauté scientifique des réseaux qui est de faire une séparation nette entre les équipements de cœur et de bordure du réseau, comme c'est notamment le cas dans les architectures MPLS [Ghe06]. Cette idée a été également évoquée par [Cas+12] afin de faire évoluer les architectures SDN physiques actuelles vers un modèle plus flexible. La nouveauté de notre proposition est d'utiliser cette approche de conception non pas au niveau de l'infrastructure physique mais au niveau des vues abstraites du plan virtuel. Ainsi, un *edge* sera un composant virtuel sur lequel s'exécutera du traitement avancé sur les paquets et la *fabric*[1] un composant virtuel dédié au transport des paquets. Plus précisément, le modèle d'abstraction *Edge-Fabric* que nous avons proposé s'appuie sur quatre types de composants schématisés sur la figure 3.3 :

— ***Edge*** : composant virtuel qui supporte l'exécution des fonctions et des services réseaux complexes du **plan de contrôle**.

1. Dans la suite du document, nous utilisons la terminologie et l'orthographe anglo-saxonne pour designer les composants du modèle Edge-Fabric.

— ***Data machine*** : représente un *edge* spécialisé qui effectue des opérations complexes sur les paquets au niveau du **plan de données**.

— ***Fabric*** : composant virtuel qui se charge principalement des problématiques liées au transport des paquets.

— ***Host*** et ***Network*** : représentent les sources et destinations des paquets.

Dans notre modèle d'abstraction, les ***edges*** désignent des composants virtuels polyvalents disposés à la périphérie du réseau virtuel. De ce fait, ils seront, conceptuellement, connectés aux hôtes qui représentent les sources et destinations des paquets. Ainsi, les edges ont deux principales fonctionnalités :

— Interface « hôte-réseau » : interface permettant d'identifier et d'associer les flux émanant des hôtes au réseau (et réciproquement). Un edge classifiera les flux en fonction des en-têtes des paquets, puis attachera à ces paquets une étiquette (label) qui sera par la suite utilisée pour l'acheminement au sein du cœur du réseau (la fabrique).

— Execution de fonctions réseaux : les edges offrent un support logique pour l'installation des services réseaux à valeur ajoutée (filtrage, traduction d'adresses, etc.), soit à l'entrée ou à la sortie de ce dernier. Nous considérons deux types de fonctions :

- Statiques : elles sont installées par le contrôleur SDN sur le commutateur avant que le premier paquet du flux ciblé ne soit reçu.
- Dynamiques : elles sont exécutées au niveau du contrôleur SDN (dans plan de contrôle) dès que les paquets du flux ciblé arrivent sur le commutateur.

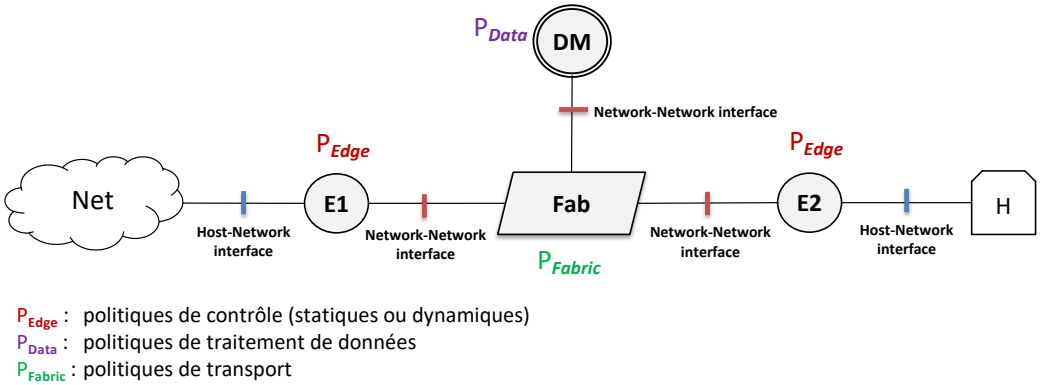


FIGURE 3.3 – Le modèle Edge-Fabric

Les ***data machines*** sont des edges particuliers dont la seule spécificité est que leur fonction est exécutée au niveau du plan de données (et donc capable d'exécuter un traitement complexe tel que de la compression, du chiffrement ou encore du transcodage). Elles constituent une abstraction des *middleboxes* que l'on retrouve communément dans les réseaux.

À l'inverse des edges classiques, les *data machines* ne participent pas au processus de prise de décision et de contrôle. En effet, ces composants opèrent exclusivement au niveau du plan de données, elles reçoivent en entrée un paquet et retournent en sortie un ou plusieurs paquets.

Les ***fabrics*** représentent des composants virtuels qui se chargent exclusivement de la logique de transport entre les edges et les data machines. Concrètement, une fabrique représente une collection d'équipements de commutation de cœur de réseau dont le premier objectif est le transport des paquets d'un point à un autre.

Pour identifier un flux de paquets, une fabrique se base exclusivement sur un label qui a été préalablement inséré par un edge, permettant ainsi de séparer les deux interfaces « hôte-réseau »

et « réseau-réseau ». Cette séparation est une des clés de la modularité de ce modèle. En effet, un opérateur peut mettre à jour les fonctions qui sont appliquées sur un flux au niveau d'un edge et tant qu'il ne change pas le label qui leur a été attribué, la politique de transport au niveau de la fabrique ne sera pas impactée. Inversement, il est aussi possible d'effectuer des mises à jour dans les politiques de transport sans que cela n'impacte ou ne nécessite un changement au niveau des politiques installées sur les edges.

En règle générale, un réseau virtuel contiendra une seule fabrique qui représentera ses capacités de transport. Néanmoins, un opérateur a aussi la possibilité d'utiliser plusieurs fabriques afin de prendre en considération des contraintes physiques ou des objectifs de contrôle de haut niveau. L'utilisation de fabriques dans le plan de contrôle peut être envisagée selon trois principales approches (cf. figure 3.4) :

— Une seule fabrique : approche similaire à celle du modèle *One Big Switch* par le fait qu'elle permet d'abstraire toutes les capacités de transport du réseau au sein d'une seule et unique fabrique.

— Plusieurs fabriques en parallèle : approche qui permet de distinguer plusieurs possibilités de transport au sein d'un réseau. Ce souhait peut être d'ordre conceptuel (pour être conforme à une politique de haut niveau) ou en rapport avec des contraintes physiques qu'un administrateur souhaite faire remonter au niveau virtuel (par exemple des réseaux physiques qui n'ont pas les mêmes performances).

— Plusieurs fabriques en séquence : approche qui permet d'expliciter un chemin de transport qui est constitué de plusieurs tronçons différents. Cette différence peut être là aussi d'ordre physique ou simplement conceptuel.

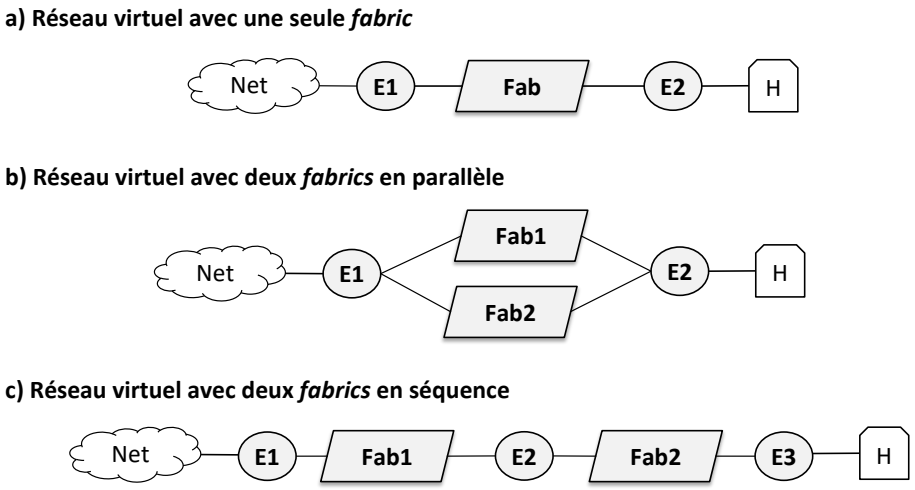


FIGURE 3.4 – Choix de conception pour le placement des *Fabrics*

3.2.2 Projection vers l'infrastructure physique

Le modèle *Edge-Fabric* étant utilisé comme modèle abstrait, ses composants pourront en pratique être « mappés » vers un ou plusieurs équipements virtuels ou physiques. Ainsi, un composant *edge* pourra être déployé sur un ou plusieurs commutateurs de bordure et, de la même manière, un commutateur de bordure pourra être associé à un ou plusieurs *edges* (cf. figure 3.5). Cette flexibilité est primordiale de sorte à être capable de considérer plusieurs contextes d'utilisation, divers choix de conception et aussi de pouvoir répondre à diverses contraintes de la topologie physique (par exemple associer tous les points d'accès sans fil à un *edge* spécifique et les accès filaires à un autre).

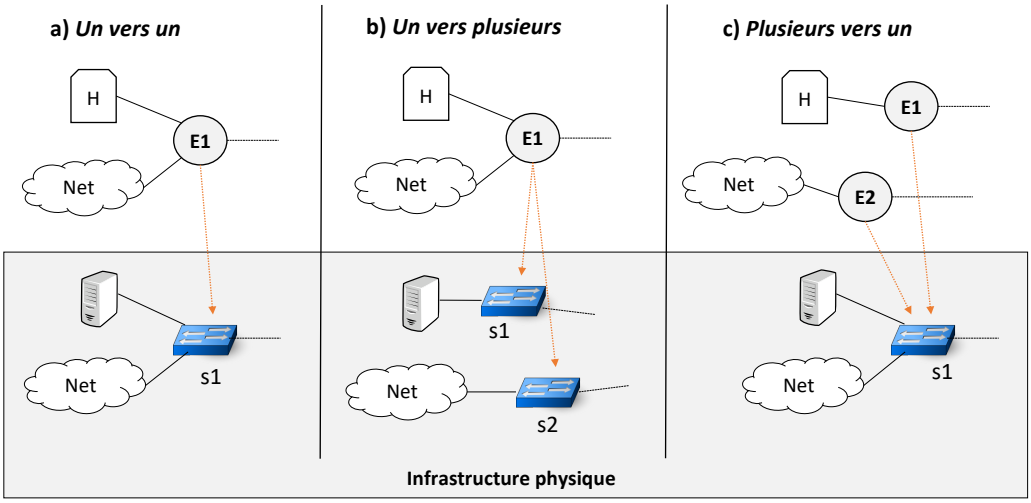


FIGURE 3.5 – Possibilités de mapping pour le composant *edge*

Comme pour les *edges*, un opérateur aura le choix entre plusieurs possibilités de mapping pour les fabriques. En particulier, deux fabriques pourront être associées à des équipements d'interconnexion distincts ou, au contraire, partager certains équipements (cf. figure 3.6). Bien entendu, cette flexibilité entraîne une plus grande complexité dans le système d'exécution sous-jacent (l'hyperviseur) qui doit assurer la projection du réseau virtuel sur l'infrastructure réelle.

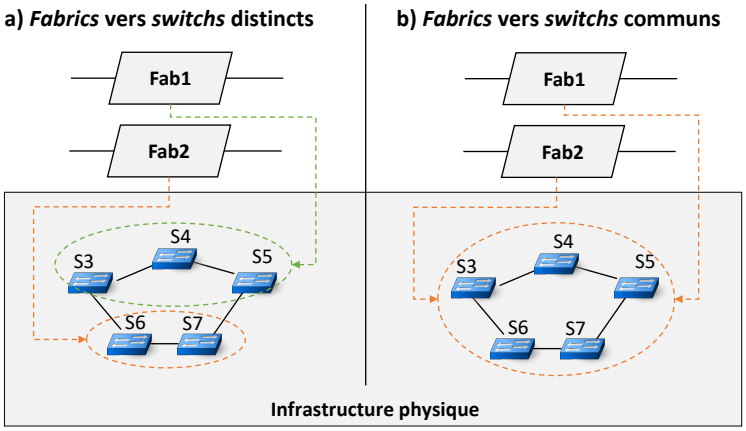


FIGURE 3.6 – Possibilités de mapping pour le composant *fabric*

3.3 AirNet : langage et modèle de programmation

Le modèle *Edge-Fabric* décrit précédemment a été formalisé à travers un langage de contrôle réseau appelé AirNet. Ce langage permet de spécifier la topologie virtuelle cible et les services réseaux souhaités. Ces services sont exprimés à l'aide des primitives du langage qui sont composées entre elles pour former des « politiques » de contrôle du comportement du réseau virtuel. Dans la section 3.4, nous verrons qu'un hyperviseur assure la composition et la projection de ces politiques sur l'infrastructure réelle.

3.3.1 Primitives du langage et opérateurs de composition

Les primitives du langage sont utilisées pour exprimer : *i)* la topologie du réseau virtuel, *ii)* les politiques associées aux *edges* et *iii)* les politiques associées aux *fabrics*.

La définition de la topologie abstraite consiste simplement à déclarer les différents composants logiques (`addEdge`, `addFabric`, `addHost`...), ainsi que les liens qui les rattachent (`addLink`). Peu de contraintes existent quant aux choix de conception de la topologie virtuelle afin de laisser de la liberté à l'utilisateur.

Conformément au modèle *Edge-Fabric* qui cantonne l'intelligence du réseau à la périphérie, le rôle des edges est de classifier des flux et d'exécuter certaines fonctions réseaux en suivant une logique *match-action*. Nous avons donc intégré au langage des primitives pour : filtrer du trafic (`match`) en fonction des champs de l'en-tête du paquet ; commuter un paquet vers un port de sortie (`forward`) ; rejeter un paquet (`drop`) ; modifier l'en-tête d'un paquet (`modify`) ; et attribuer une étiquette à un paquet (`tag`) pour le transport au sein de la fabrique.

Les fabriques exposent un jeu d'instructions plus restreint comparé à celui des edges. Ces instructions incluent principalement des primitives qui permettent de transporter les flux d'un edge à un autre suivant des exigences de QoS (primitives `catch` et `carry`), mais aussi des primitives qui permettent de faire passer un flux par des *data machines* (primitive `via`) puis de le récupérer et de l'acheminer vers sa destination finale.

AirNet intègre deux opérateurs de composition qui permettent d'une part, de composer les primitives de chaque entité virtuelle afin d'obtenir une politique de contrôle atomique cohérente et, d'autre part, de composer ces politiques en modules de contrôle indépendants. L'opérateur de composition séquentielle ($\gg$) applique une instruction (filtre, action ou politique existante) sur le résultat retourné par une autre instruction qui la précède dans la chaîne de composition (ainsi $P_1 \gg P_2$ signifie que la politique P_2 sera appliquée sur le résultat (ensemble de paquets) retourné par la politique P_1). L'opérateur de composition parallèle ($+$) indique quant à lui que des politiques peuvent être exécutées dans un ordre quelconque, leur évaluation étant totalement indépendante.

Davantage de détails ainsi que la spécification formelle complète de ces primitives et opérateurs de composition peuvent être trouvés dans la thèse de Messaoud Aouadj [Aou16].

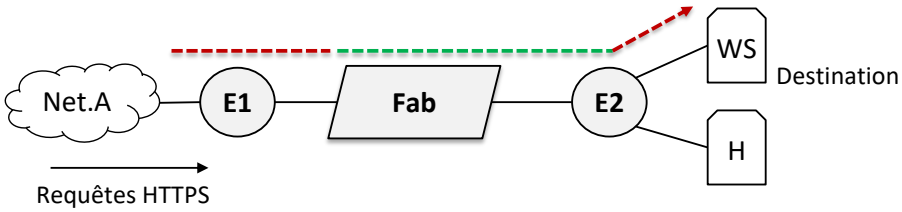
3.3.2 Politiques de contrôle statiques

Les politiques de contrôle statiques sont installées d'une manière proactive sur les équipements associés aux *edges* et *fabrics* de la topologie virtuelle. Une fois déployées, le contrôleur n'aura pas à interagir avec le plan de données pour que les règles correspondantes s'exécutent.

La figure 3.7 présente un exemple de politique statique qui spécifie l'acheminement du trafic HTTPS du réseau `Net.A` vers le serveur web `WS`. La politique globale est composée de trois règles : une sur l'edge `E1` pour classifier le trafic HTTPS et lui apposer un label, la deuxième sur la fabrique `Fab` pour acheminer le flux correspondant jusqu'à l'edge `E2` qui distribuera vers la destination grâce à la troisième règle.

3.3.3 Politiques de contrôle dynamiques

Les politiques de contrôle dynamiques sont évaluées en cours d'exécution (*at runtime*) au niveau du plan de contrôle. Pour mettre en œuvre ce concept, AirNet fournit le décorateur `@DynamicControlFct` qui permet d'annoter une fonction Python quelconque et de la composer au sein d'une politique dynamique. Il existe deux structures possibles pour un décorateur de fonction dynamique, suivant si l'utilisateur souhaite remonter au contrôleur des paquets réseau complets ou uniquement des statistiques sur un flux. L'exemple de la figure 3.8 indique à l'hyperviseur

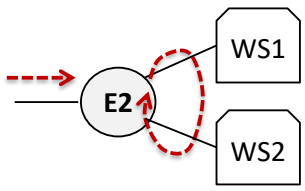



```
e1 = match(edge="E1", src="Net.A", dst="WS", tp_dst=443) >> tag("in_web") >> fwd("Fab")
t1 = catch(fabric="Fab", src="E1", flow="in_web") >> carry("E2")
e2 = match(edge="E2", dst="WS") >> fwd("WS")
return(e1 + t1 + e2)
```

FIGURE 3.7 – Exemple de politique de contrôle statique

de remonter le premier paquet (`data="packet"` et `limit=1`) d'un flux identifié par son adresse IP source (`split=["nw_src"]`). Puis la fonction exécute un simple algorithme d'équilibrage de charge de type tourniquet, en renvoyant une nouvelle politique (traduction d'adresse et transfert des paquets vers `WS1` ou `WS2`) qui sera installée sur l'équipement du plan de données uniquement pour le flux spécifié.

Cette notion de fonction dynamique est une capacité riche du langage AirNet qui autorise l'exécution de fonctions complexes au sein du contrôleur SDN.




```
match(edge="E2", nw_dst=PUB_WS, tp_dst=443) >> Dynamic_LB()

@DynamicControlFct(data="packet", limit=1, split=["nw_src"])
def Dynamic_LB(self, packet):
    my_match = match.from_packet(packet)
    if self.rrlb_token == 1:
        self.rrlb_token = 2
        return my_match >> modify(dst="WS1") >> forward("WS1")
    else:
        self.rrlb_token = 1
        return my_match >> modify(dst="WS2") >> forward("WS2")
```

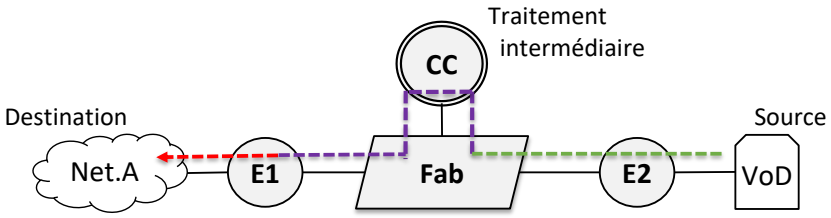
FIGURE 3.8 – Exemple de politique de contrôle dynamique

3.3.4 Politiques orientées « plan de données »

Le dernier type de politique qui peut être spécifié en utilisant le langage AirNet concerne la redirection de flux vers une fonction réseau du plan de données (matériel dédié, serveur physique ou virtuel). En effet, beaucoup de fonctions réseaux (chiffrement, compression, mise en cache, etc.) ne peuvent pas être implémentées en utilisant les instructions de base (c'est-à-dire *forward*, *modify*, *drop*) présentes sur un commutateur « traditionnel ». Afin d'offrir une solution à cette problématique, AirNet inclut la primitive `via` qui est implémentée au niveau des fabriques et qui permet de spécifier qu'un flux de paquets doit passer par une ou plusieurs *middleboxes* du plan de données, facilitant ainsi le chaînage et le déploiement de nouveaux services réseaux. Dans AirNet, nous faisons référence à ces boîtes noires par le terme de `data machines`.

La figure 3.9 illustre un exemple de fonction de conversion de codec sur un flux vidéo qui est redirigé vers cette fonction (entité `CC`) grâce à la politique de transport de la fabrique (primitive `via(dataMachine="CC")` insérée dans la règle `t1` en utilisant l'opérateur de composition séquentielle).

Cette notion de politique orientée données se rapproche de travaux plus récents sur le chaînage de fonctions réseaux (SFC, *Service Function Chaining* [Med+17]).




```
e2 = match(edge="E2", src="VoD", dst="Net.A") >> tag("vod") >> forward("Fab")
e1 = match(edge="E1", dst="Net.A") >> forward("Net.A")
t1 = catch(fabric="Fab", src="E2", flow="vod") >> via(dataMachine="CC") >> carry(dst="E1")
return(e1 + e2 + t1)
```

FIGURE 3.9 – Exemple de politique de transit par une fonction du plan de données

3.4 Architecture et expérimentations

Le langage AirNet présenté dans la section précédente a été implémenté en tant que langage dédié, embarqué dans le langage Python. Outre l'implémentation de ce langage, AirNet dispose d'un système d'exécution (que nous désignons comme un « hyperviseur ») qui permet d'assurer la composition et l'installation des politiques du réseau virtuel sur l'infrastructure physique. L'ensemble du logiciel ainsi que différents cas d'utilisation sont accessibles publiquement en licence libre [Air17].

3.4.1 Architecture de l'hyperviseur

La figure 3.10 schématise l'architecture générale de l'hyperviseur. En complément de la spécification du réseau virtuel et des politiques de contrôle, l'hyperviseur prend en entrée une description du *mapping* entre les composants virtuels et les équipements de l'infrastructure cible. Le fait de dissocier ce mapping du reste du programme AirNet permet de pouvoir réutiliser les topologies virtuelles et politiques de contrôle sur différentes infrastructures physiques.

Les principaux modules de l'hyperviseur sont décrits ci-dessous.

Le module ***infrastructure*** maintient une vision globale, cohérente et à jour de la topologie physique. Il construit un graphe du réseau réel et embarque des algorithmes de parcours et de recherche.

Le module ***language*** contient toutes les classes de base qui implémentent les primitives et les opérateurs du langage AirNet. Il assure un premier niveau de compilation des politiques pour obtenir des structures de données intermédiaires représentant des règles de contrôle basées sur des flux, similaires à des règles OpenFlow (mais avec un niveau d'abstraction plus élevé).

Le module ***classifier*** insère chacune de ces règles dans un conteneur logique ordonné appelé *classifier*. Puis, ces conteneurs sont fusionnés deux à deux d'une manière cumulative, produisant à la fin un seul classifier avec toutes les règles de contrôle ordonnées par priorité. L'objectif de cette fusion cumulative est de résoudre les problématiques d'intersection qui peuvent exister entre les règles.

Le module ***northband integration*** s'occupe de la communication avec un contrôleur SDN. Ce module assure une étanchéité entre l'hyperviseur et les différents types et versions de contrô-

leurs existants. Nous avons actuellement deux modules d'intégration : un pour le contrôleur POX [POX13] et un pour le contrôleur Ryu [Ryu14].

Le module ***runtime*** est la pièce centrale de l'hyperviseur car c'est lui qui transforme les politiques du réseau virtuel en règles physiques. Le module *runtime* prend en entrée les résultats produits par les autres modules de l'hyperviseur (infrastructure, language et classifier), applique les algorithmes de compilation, puis génère un ensemble de règles pour les équipements du plan de données. Ces règles sont passées par la suite au module d'intégration *northband* pour être transformées en règles OpenFlow et installées sur les commutateurs réels. Le module *runtime* inclut deux grandes parties (dites *core*) qui supportent l'exécution des deux principaux mode de fonctionnement d'un programme AirNet, à savoir le mode proactif et le mode réactif :

— La partie proactive s'occupe de l'installation et de l'initialisation des politiques statiques sur le plan de données ;

— La partie réactive gère toute l'exécution des politiques dynamiques et les changements qui peuvent survenir dans la topologie physique.

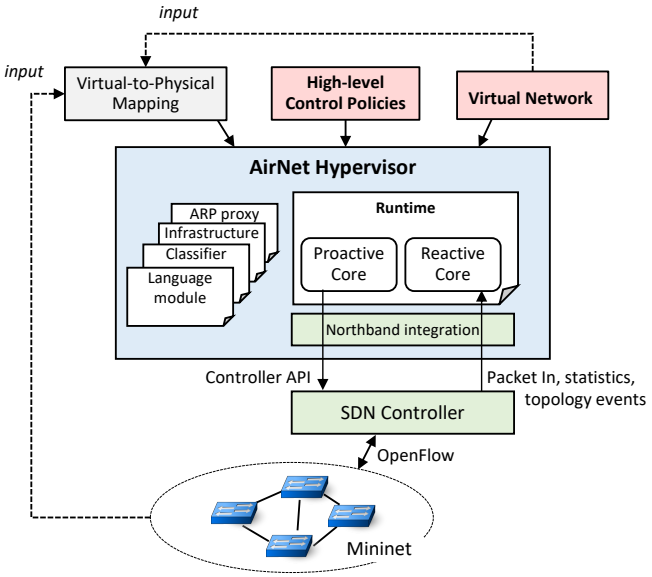


FIGURE 3.10 – Architecture de l'hyperviseur AirNet

3.4.2 Exemple de mise en œuvre

Dans cette section nous présentons une mise en pratique du langage AirNet à travers le déroulement d'un cas d'utilisation qui permettra d'illustrer les différentes étapes du processus, de la spécification jusqu'à l'exécution d'un programme AirNet.

Description du scénario

Le contexte général de ce cas d'étude consiste en la configuration d'un réseau d'entreprise qui héberge des services applicatifs déployés sur des serveurs web et bases de données. Nous considérons deux types d'utilisateurs : les employés de l'entreprise qui pourront avoir accès aux serveurs de bases de données uniquement s'ils sont connectés au réseau interne de l'entreprise et les autres utilisateurs (visiteurs, sous-traitants...), connectés à un autre réseau, qui ne devront pas avoir accès aux serveurs de bases de données. Tous les utilisateurs auront en revanche un accès aux serveurs web. Enfin, le réseau devra offrir un service de répartition de charge entre les différents serveurs d'un même type. Ce scénario est illustré dans la figure 3.11.

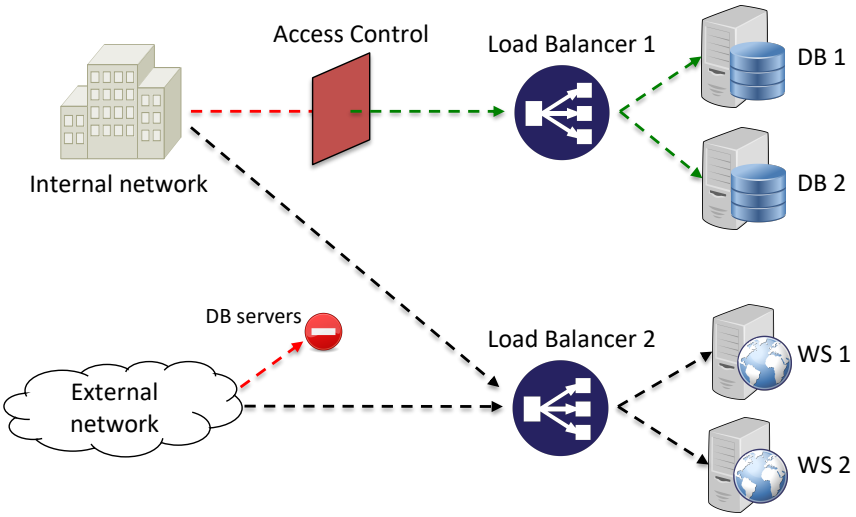


FIGURE 3.11 – Vue logique du cas d'étude

Conception de la topologie virtuelle

La première étape de spécification d'un programme AirNet est la définition de la topologie virtuelle. Cette étape se résume en des choix de conception (nombre d'*edges* et de *fabrics* par exemple) qui sont guidés principalement par les politiques de contrôle à mettre en place ou des contraintes pouvant exister dans l'infrastructure physique.

Dans le contexte de ce cas d'étude, nous avons opté pour la topologie virtuelle qui est illustrée dans la figure 3.12.

Les deux réseaux virtuels `Int.Net` et `Ext.Net` représentent respectivement les utilisateurs internes et externes de l'entreprise, et les quatre hôtes `WS1`, `WS2`, `DB1` et `DB2` représentent les serveurs web et bases de données de l'entreprise.

L'edge `IO` joue le rôle d'interface *host-network* entre les utilisateurs et le réseau. Nous avons choisi d'utiliser un seul edge comme point d'entrée au réseau afin de regrouper toutes les politiques de contrôle d'accès au niveau d'un seul composant virtuel. Néanmoins, il aurait été possible d'utiliser deux edges, un pour connecter les employés de l'entreprise et l'autre pour connecter le réseau des utilisateurs externes.

Deux fabriques (`Fab1` et `Fab2`) ont été intégrées à la topologie. Ce choix de conception a pour objectif de distinguer, conceptuellement, deux chemins de transport suivant la destination. Là aussi, une seule fabrique aurait été envisageable.

Enfin, la topologie virtuelle contient deux edges de distribution `DC1` et `DC2` pour connecter les serveurs de l'entreprise. C'est notamment sur ces edges que seront installées les fonctions de répartition de charge que nous présentons dans la sous-section suivante.

Politiques de contrôle

Après avoir défini la topologie virtuelle, les politiques de contrôle associées aux différents composants doivent être spécifiées. Considérant les objectifs de haut niveau du cas d'étude, nous définissons les politiques suivantes :

— Une politique d'authentification pour gérer l'accès aux serveurs de bases de données. Elle sera installée à l'entrée du réseau, sur l'edge `IO`, pour ne laisser passer que les flux autorisés.

— Une politique de répartition de charge entre les serveurs qui devra être installée sur les edges `DC1` et `DC2`.

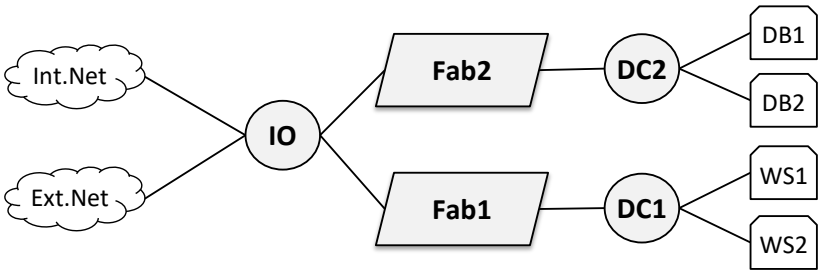


FIGURE 3.12 – Modélisation AirNet du cas d'étude

— Des politiques d'entrées/sorties statiques pour relayer les paquets vers les hôtes et les réseaux d'extrémités.

— Des politiques de transport pour configurer les deux fabriques du réseau.

La spécification AirNet de toutes ces politiques ne sera pas détaillée ici, seulement quelques-unes à titre d'exemple.

Ci-dessous, deux politiques sur l'edge `IO` : la première statique pour classifier le trafic HTTP et le relayer vers la fabrique `Fab1`, et la deuxième dynamique pour faire remonter les paquets à la fonction `authenticate` du contrôleur SDN qui pourra alors vérifier si l'adresse IP source du paquet appartient à la liste blanche établie par l'administrateur. En fonction du résultat, la fonction renverra une nouvelle politique pour le trafic autorisé (`match >> tag >> forward`) ou pour le trafic refusé (`match >> drop`).

```
match(edge=IO, nw_dst=PUB_WS_IP, tp_dst=HTTP) >> tag(in_web) >> forward(Fab1)
match(edge=IO, nw_dst=PUB_DB_IP, src=IntNet) >> authenticate()
```

Un autre exemple sur l'edge `DC1` avec les trois politiques gérant la répartition de la charge sur les deux serveurs WS (la première pour la fonction dynamique, et les deux autres pour les flux retour) :

```
match(edge=DC1, nw_dst=PUB_WS_IP) >> dynamic_LB(args=WS_Infos)
match(edge=DC1, src=WS1) >>
  modify(nw_src=PUB_WS_IP) >> modify(dl_src=PUB_WS_MAC) >>
  tag(out_web) >> forward(Fab1)
match(edge=DC1, src=WS2) >>
  modify(nw_src=PUB_WS_IP) >> modify(dl_src=PUB_WS_MAC) >>
  tag(out_web) >> forward(Fab1)
```

Enfin, un extrait de deux politiques de transport pour l'acheminement des flux en fonction des étiquettes qui ont été préalablement insérées par les edges `IO` et `DC1` :

```
catch(fabric=Fab1, src=IO, flow=in_web) >> carry(dst=DC1)
catch(fabric=Fab1, src=DC1, flow=out_web) >> carry(dst=IO)
```

Au total 14 politiques ont été spécifiées pour ce cas d'utilisation (4 sur l'edge `IO`, 4 sur les fabriques et 3 sur chaque edge `DC`).

Mapping virtuel-réel

La dernière étape dans le processus de construction d'un programme AirNet est la spécification du module de mapping. Ce module contient les informations qui permettent d'associer chaque composant virtuel à un ou plusieurs équipements de l'infrastructure réelle, ainsi que le plan d'adressage IP. La figure 3.13 présente un exemple de mapping sur une infrastructure de 12 commutateurs. On retrouve ici deux schémas de mapping : *i)* un vers un pour les edges `DC1` et `DC2`, et *ii)* un vers plusieurs pour l'edge `IO`. Concernant les deux fabriques, elles partagent seulement les commutateurs `s3` et `s10`, elles sont ensuite associées à des chemins différents.

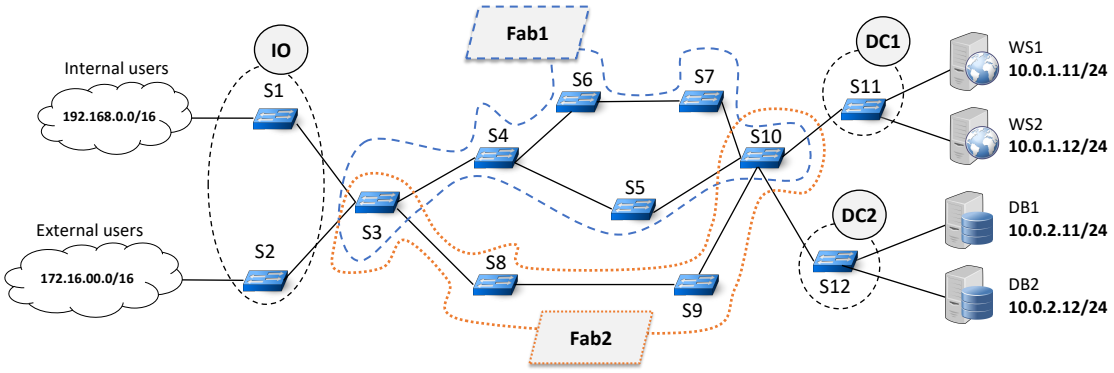


FIGURE 3.13 – Exemple de mapping virtuel-réel du cas d'étude

Exécution

L'ensemble du cas d'étude a été déployé et testé sur l'émulateur réseau *Mininet* [Min15 ; LHM10] qui orchestre des commutateurs *Open vSwitch* [Pfa+15] et des hôtes logiciels. La communication avec les contrôleurs SDN POX ou Ryu se fait en utilisant le protocole *OpenFlow* [ONF12], l'hyperviseur AirNet s'exécutant au dessus de ces derniers.

Après le lancement de Mininet et d'AirNet, la première phase pour l'hyperviseur est la phase proactive dans laquelle il compile les politiques du réseau virtuel et génère les règles OpenFlow correspondantes qui sont installées par le contrôleur SDN. Les entrées OpenFlow qui sont insérées dans les tables de flux des commutateurs peuvent être divisées en trois groupes : la gestion des paquets ARP, la commutation des paquets du plan de données, et le transfert des paquets vers le contrôleur SDN pour l'exécution des fonctions réseau dynamiques. Pour ce cas d'étude, l'hyperviseur a généré et installé un total de 44 règles OpenFlow. La répartition de ces règles sur les différents équipements de l'infrastructure est détaillée dans la ligne « init. phase » du tableau 3.1. Une seule règle (de type *table-miss*) est installée sur les commutateurs `s6` et `s7` car, par défaut, la fabrique sélectionne le plus court chemin, c'est-à-dire par `s5` dans le cas présent.

La phase réactive concerne les politiques dynamiques qui s'appuient dans le scénario présent sur les fonctions d'authentification et de répartition de charge qui sont exécutées au niveau de l'hyperviseur. Ce dernier commence par installer sur les commutateurs concernés une règle qui redirige vers le contrôleur SDN tous les flux ciblés par la politique dynamique. À la fin de l'exécution de la fonction, une nouvelle politique est générée, puis compilée et installée sur les commutateurs. Cette nouvelle règle est, d'une part, spécifique à un flux donné (identifié par le quintuplet IP classique) et, d'autre part, plus prioritaire que la règle générale qui redirige les flux vers le contrôleur SDN. Cette gestion des priorités est indispensable afin de ne faire remonter au contrôleur que les paquets des nouveaux flux. Les lignes « IntHost→WS » et « IntHost→DB » du tableau 3.1 présentent le nombre de règles ajoutées par l'hyperviseur après l'envoi d'une requête HTTP et d'un *ping* respectivement, depuis un hôte du réseau interne. Notez que ces règles sont rajoutées lors du passage du premier paquet du flux, les paquets suivants étant traités par les nouvelles règles installées dans le plan de données.

La phase réactive s'occupe également de la mise à jour des règles en cas de changement de la topologie réelle. En coupant le lien Mininet qui relie les commutateurs `s4` et `s5`, l'événement de type `LinkDown` est envoyé au contrôleur et l'hyperviseur enclenche une procédure de reconfiguration des chemins de la première fabrique. Cette reconfiguration consiste à recalculer les chemins afin de trouver un nouveau plus court chemin, réexécuter la phase de mapping afin de générer de nouveaux *classifiers*, et enfin générer les dictionnaires de différences qui contiennent les règles à ajouter, supprimer ou modifier sur les commutateurs de l'infrastructure. Dans notre exemple,

le résultat est un basculement de tous les flux qui passaient par le chemin `s4, s5, s10` vers le chemin `s4, s6, s7, s10` (cf. ligne « link s4-s5 down » du tableau 3.1).

TABLE 3.1 – Nombre de politiques sur réseau virtuel et règles physiques pour le cas d'étude

Virtual component	**IO**		**Fabric 1 et 2**								**DC1**	**DC2**	Total
Policies	4		4								3	3	14
Switch	s1	s2	s3	s4	s5	s6	s7	s8	s9	s10	s11	s12	Total
Rules (init. phase)	5	5	5	4	4	1	1	2	2	5	5	5	44
Rules (link s4-s5 down)	–	–	–	mod 1	-3	+3	+3	–	–	mod 2	–	–	47
Rules (IntHost→WS)	–	–	+1	+1	+1	–	–	–	–	+1	+1	–	49
Rules (IntHost→DB)	+1	+1	+1	–	–	–	–	+1	+1	+1	–	+1	55

Bien que ne concernant qu'un cas d'étude simple, ces chiffres attestent clairement de la différence importante existant entre le nombre de politiques spécifiées par l'opérateur et le nombre de règles réellement installées par l'hyperviseur. Pour un seul flux source, un facteur 3 existe entre le nombre de politiques et le nombre de règles installées durant la phase proactive, et un facteur 4 pour la phase réactive (accès aux serveurs de base de données). Ce nombre augmente linéairement en fonction du nombre de flux sources différents pour la fonction de contrôle dynamique.

Un autre apport de l'hyperviseur AirNet est d'abstraire toute la complexité et dynamique de l'infrastructure. Par exemple, en cas d'une défaillance d'un lien, c'est l'hyperviseur qui se charge de trouver un nouveau chemin pour les flux de paquets, si ce dernier existe.

Enfin, les politiques d'AirNet étant spécifiées au-dessus de topologies virtuelles, elles peuvent être réutilisées sur différentes topologies physiques. La seule modification qui doit être apportée par l'opérateur est de remanier le module de mapping qui contient les différentes informations sur les associations entre composants virtuels et équipements physiques.

Des tests supplémentaires, ainsi que des mesures de performance (sur le temps de compilation des politiques ou le délai introduit pas les fonctions dynamiques), peuvent être trouvés dans les articles [Aou+16a] et [Aou+17].

3.5 Conclusion

Dans ce chapitre, nous avons synthétisé nos travaux sur l'interface nord de l'architecture SDN : de l'expression du besoin à l'aide d'un réseau virtuel, jusqu'à l'implémentation des règles concrètes sur l'infrastructure physique. Le réseau virtuel est construit en s'appuyant sur un modèle d'abstraction *Edge-Fabric* dans lequel nous séparons les composants en charge de l'acheminement des paquets (dans la *Fabric*), des fonctions de traitement plus complexes (sur les *Edge*). L'innovation de notre proposition est d'utiliser cette approche de conception non pas au niveau de l'infrastructure physique mais au niveau des vues abstraites du plan virtuel. Le plan de contrôle de la topologie virtuelle est assuré par des politiques statiques (qui peuvent être installées dans le plan de données de manière proactive) ou dynamiques (qui sont évaluées par le contrôleur de manière réactive). Nous avons par ailleurs conçu et développé un hyperviseur dont le rôle principal est d'agréger les différentes politiques et de les compiler en règles OpenFlow pour être installées sur les équipements réels. Une difficulté de ce travail est de résoudre les intersections qui peuvent exister entre les règles suite au *mapping* virtuel-physique, nous y répondons en générant de nouvelles règles et en les ordonnant par priorité. Nous avons testé avec succès l'algorithme de composition sur une dizaine de cas d'étude. Cependant, si nous considérons qu'un réseau virtuel peut être dédié à un client particulier (*tenant*), alors l'hyperviseur devrait gérer en parallèle plusieurs dizaines, voire plusieurs centaines de réseaux virtuels. Même si les réseaux virtuels sont indépendants les uns des autres (le module de *mapping* s'appuyant notamment sur

des plages d'adresses différentes), il faudrait rajouter des mécanismes de contrôle sur l'utilisation des ressources physiques partagées.

La prise en compte de la QoS est une autre perspective d'amélioration de ce travail. Nous avons prévu dans le langage AirNet la possibilité de spécifier une exigence de QoS sur la politique de transport dans la fabrique, mais cette exigence n'est pas prise en compte par l'hyperviseur dans l'allocation des ressources (ce dernier fonctionnant en mode *best effort*). Ce point est l'objet d'une partie des travaux présentés dans le chapitre suivant sur la mise en œuvre d'une *slice* 5G au sein d'un réseau de transport non terrestre (NTN, *Non Terrestrial Network*).

Enfin, un autre aspect que nous souhaiterions développer est liée à la capacité du réseau virtuel de pouvoir répondre à des propriétés de bout en bout, en particulier sur des réseaux couvrant plusieurs domaines administratifs. L'objectif initial du langage AirNet n'était pas orienté sur des cas d'usage multi-domaines, de par la nature centralisée du contrôle. Malgré tout, le modèle d'abstraction *Edge-Fabric* est générique, il peut abstraire un domaine dans une fabrique, même si cette dernière dispose d'un plan de contrôle qui lui est propre. La fabrique peut en effet être vue comme une boîte noire qui doit « simplement » répondre aux exigences d'acheminement. L'enjeu est alors de pouvoir d'une part, garantir ces exigences et d'autre part, intégrer les plans de gestion et de contrôle de l'infrastructure physique des différents domaines pour permettre cette vision de bout en bout. Là encore, dans le chapitre suivant nous nous intéressons à cette problématique dans le cadre des réseaux 5G à travers la mise en place de mécanismes de couture inter-domaines.

Chapitre 4

Vers un réseau virtuel personnalisé de bout en bout

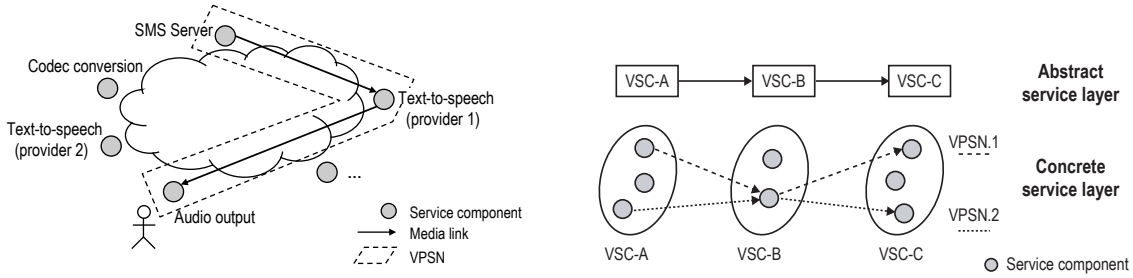
L'un des axes de recherche de l'équipe dans laquelle j'ai effectué mon premier post-doctorat à Télécom Paris en 2007 portait sur la convergence des réseaux fixes et mobiles avec des réseaux dits « de nouvelle génération ». Dans ce contexte, une problématique à laquelle nous nous sommes intéressés était de concevoir des services personnalisés, adaptés à la situation de l'utilisateur comme son terminal, son réseau d'accès, ses préférences ou encore les exigences de qualité de service (QoS) des applications. Ces travaux sont présentés dans la section 4.1 de ce chapitre. Bien que ces contributions se situaient dans le cadre d'un réseau de recouvrement de l'infrastructure IP (*overlay network*), j'étais attaché à l'idée de pouvoir construire un réseau virtuel dédié à un usage particulier, voire à une session de courte durée pour les besoins d'un utilisateur. Cet enjeu était un des facteurs de motivation des travaux sur la définition d'un langage de haut niveau pour spécifier un réseau virtuel (cf. chapitre précédent) puis, plus tard, de l'intérêt porté au *network slicing* dans les réseaux mobiles de 5$^{\text{ème}}$ génération. Mes activités sur le *network slicing* se sont concrétisées par une collaboration avec Emmanuel Chaput dans le cadre de l'encadrement de la thèse de Youssouf Drif [Dri22b], synthétisée dans les sections 4.3 et 4.4.

4.1 Réseau virtuel personnalisé par composition de services

Dans le cadre de mon post-doctorat à Télécom Paris, nous nous sommes intéressés à la construction de réseaux virtuels personnalisés en suivant une approche de conception orientée services dans laquelle un service global pouvait être construit dynamiquement par composition d'autres services modulaires. L'objectif était de permettre à des opérateurs ou des fournisseurs de services de développer et déployer dynamiquement des services pouvant s'adapter au contexte ; par exemple pour la personnalisation d'un service (*streaming* vidéo avec des sous-titres dans la langue de l'utilisateur), de la tolérance aux fautes (remplacer un composant du service en cas de panne) ou un ajustement de QoS (adapter le codec en fonction du terminal de l'utilisateur).

4.1.1 Modèle de services basé sur le concept de VPSN

Notre proposition de composition de services est basée sur le concept de *Virtual Private Service Network* (VPSN), un réseau de recouvrement (*overlay*) dédié à un utilisateur et à un usage spécifique. La figure 4.1a illustre un exemple de VPSN pour un service vocal de SMS composé de trois sous-services (*SMS Server*, *Text-to-speech*, *Audio output*). Pour mettre en œuvre ce concept de VPSN, deux verrous ont dû être considérés : *i)* la découverte et la localisation des composants de service, et *ii)* le plan de contrôle pour l'établissement, la reconfiguration et la libération du VPSN.



(a) VPSN example (vocal SMS service) (b) Abstract and concrete service layers

FIGURE 4.1 – Concept de réseau virtuel personnalisé par composition de services [LS08]

Pour le premier point, nous avons proposé de regrouper les services fonctionnellement équivalents dans des groupes appelés *Virtual Service Community* (VSC). Ces groupes contiennent des services ouverts, distribués géographiquement, accessibles via une interface bien spécifiée et pouvant avoir des propriétés non fonctionnelles différentes (performance, sécurité...). Il est de la responsabilité du VSC de gérer les membres du groupe, leur découverte, leur localisation et leur enregistrement. Dans [HLS07] nous avons implémenté ces communautés via des groupes pair à pair (P2P) dans lesquels les services sont gérés de manière décentralisée. Un mécanisme d'indexation de service basé sur des critères de QoS (temps de réponse, taux de disponibilité, taux de fiabilité...) a été proposé, ainsi qu'une extension d'un service de découverte pair à pair s'appuyant sur ces critères.

Pour le deuxième point, lié au plan de contrôle du VPSN, nous avons distingué deux niveaux d'abstraction (cf. figure 4.1b) : un niveau abstrait (*abstract service layer*) et un niveau concret (*concrete service layer*). Le niveau abstrait correspond à la logique de composition, c'est-à-dire au *workflow* fonctionnel nécessaire à la réalisation du service global. Ce niveau est indépendant de toute instance de service. Le niveau concret instancie la logique du niveau abstrait sur un chemin spécifique dédié à une session d'utilisateur. Par conséquent, notre modèle de services est défini d'abord par le niveau abstrait lors de la conception du service, puis par l'instanciation de ce niveau lors de l'exécution du service en fonction de certains critères (résolution de l'écran du terminal, exigence de QoS, préférence de l'utilisateur sur le choix d'un opérateur, etc.). Afin de garantir une composition cohérente des sous-services, nous avons affiné ce modèle en définissant deux types de sous-service :

— Un type *EndService* pour représenter un service en fin de chaîne, pouvant produire ou consommer un média (ex. un service de diffusion de VoD).

— Un type *MidService* pour représenter un service de traitement intermédiaire, qui est capable de recevoir un flux en entrée, d'effectuer un certain traitement (ex. conversion de codec) et de transmettre le flux modifié en sortie.

Plusieurs possibilités de composition peuvent être envisagées : une composition en série dans laquelle plusieurs *MidService* sont chaînés les uns à la suite des autres, ou une composition en parallèle dans laquelle plusieurs *MidService* ou *EndService* s'exécutent en parallèle (par ex. un *MidService* réalisant une fonction d'agrégation de média issus de deux *EndService* différents).

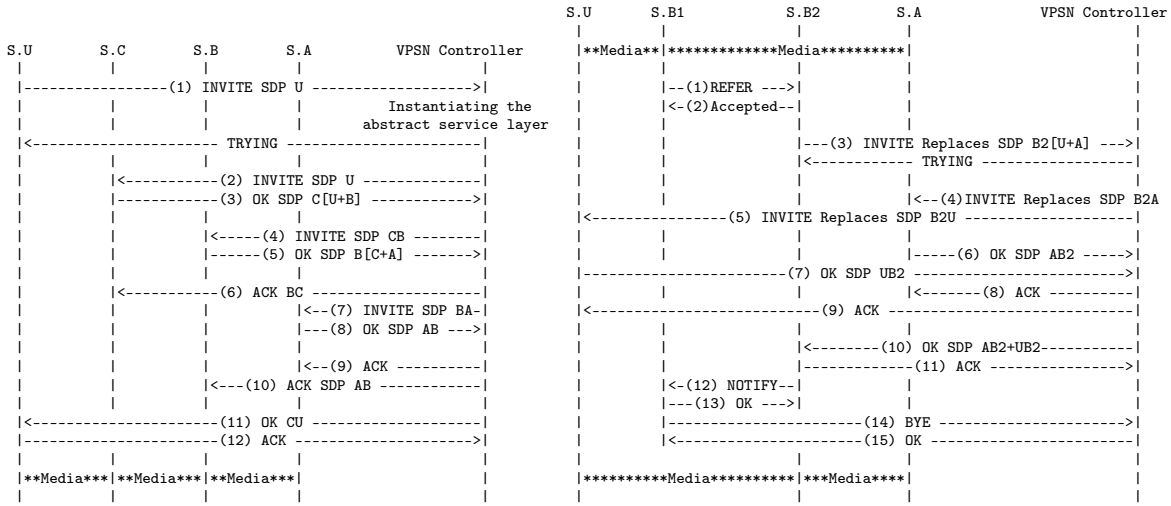
Lors de la réalisation de ces travaux de recherche, d'autres travaux existaient sur la composition de services de niveau applicatif (multimédia) [RK03 ; GN06 ; Har+06] mais ces derniers ne considéraient pas le plan de contrôle nécessaire à l'établissement et à la gestion de la chaîne de services, une fois celle-ci identifiée. Notre contribution s'est attachée à proposer des procédures basées sur le protocole de signalisation SIP (*Session Initiation Protocol*) pour gérer tout le cycle de vie du VPSN.

4.1.2 Protocole SIP pour la composition de services

Outre le fait que ce soit un standard de l'IETF [Ros+02] largement utilisé sur l'Internet, nous nous sommes appuyés sur le protocole SIP car il permet de gérer des sessions multimédia, pas seulement la VoIP. Nous avons proposé une solution impliquant une entité externe, le contrôleur du VPSN (que l'on pourrait qualifier aujourd'hui de contrôleur SDN), ainsi que des algorithmes pour automatiser le processus de composition des sous-services et le processus de reconfiguration de la chaîne dans le cas où un sous-service ne peut plus participer au VPSN (dégradation de QoS ou panne par exemple). Ces algorithmes s'appuient sur le protocole SIP pour gérer toutes les interactions entre les services. La figure 4.2 schématise le processus de signalisation SIP (*SIP call flow*), le détail des explications se trouve dans [LS08].

Plusieurs différences existent entre notre proposition et la technique exposée dans la RFC 4117 sur l'agrégation de services de transcodage en série [Cam+05]. La première différence notable est d'utiliser un contrôleur externe, et non l'agent présent sur le terminal de l'utilisateur. La logique de composition n'a donc pas à être stockée sur l'équipement d'extrémité et la signalisation sur ces équipements s'en trouve largement réduite. Une autre différence est sur la procédure de contrôle en elle-même : nous réduisons le nombre de messages SIP échangés en limitant les mises à jour de session (messages `reINVITE`), le contrôleur confirmant l'établissement de la session uniquement lorsqu'il détient sa description complète.

La validation de ce travail a été effectuée par l'implémentation d'un prototype en se basant sur l'API JAIN-SIP [NIS08].



(a) SIP call flow for a dynamic service composition (b) SIP call flow for a composition reconfiguration

FIGURE 4.2 – Signalisation SIP pour l'établissement et la reconfiguration d'une chaîne de services

4.2 Le « network slicing » dans les réseaux 5G

Le concept de *network slicing* n'est pas spécifique aux réseaux mobiles mais ce sont ces derniers qui ont su créer une dynamique autour de laquelle de nombreux travaux de recherche académiques et industriels se sont développés ces dernières années. Ce concept a été introduit dans les réseaux mobiles par l'organisme 3GPP dans la Release 15 du standard 5G [3GP21b], cependant les travaux se poursuivent toujours, et son déploiement n'est pas encore une réalité, dû en particulier à la complexité de gestion des ressources sous-jacente au *slicing*.

Le principe du *network slicing* est de mettre en place des partitions réseaux personnalisées de bout en bout allant de l'utilisateur jusqu'au serveur applicatif. Les différentes partitions, appelées

slices, sont déployées au dessus d'une seule et même infrastructure physique partagée, chaque *slice* est dédiée à un service particulier qui peut avoir des besoins très différents des autres *slices* au sein du même réseau [NGM16]. La figure 4.3 illustre des exemples de *slices* pour des services spécifiques (assistance conduite automatique, vidéo surveillance, *streaming* multimédia...) du réseau d'accès radio jusqu'au réseau de cœur, en passant par le réseau de transport.

La 5G définit les quatre services standards suivants :

- enhanced Mobile Broadband (eMBB) : service nécessitant des communications à très haut débit (de l'ordre du Gbps) ;
- Ultra-Reliable Low-Latency Communication (URLLC) : service exigeant un temps de latence particulièrement faible (< 5 ms) et une connectivité réseau extrêmement fiable (> 99.999 %) ;
- Massive Internet of Things (MIoT) : service demandant un très grand nombre de connexions avec une forte densité ;
- Vehicle-to-Everything (V2X) : service pour véhicules connectés nécessitant un temps de latence extrêmement faible, une connectivité réseaux très fiable et une bande passante élevée. L'architecture V2X est un cas spécifique du URLLC qui doit aussi permettre une communication pair à pair entre les véhicules.

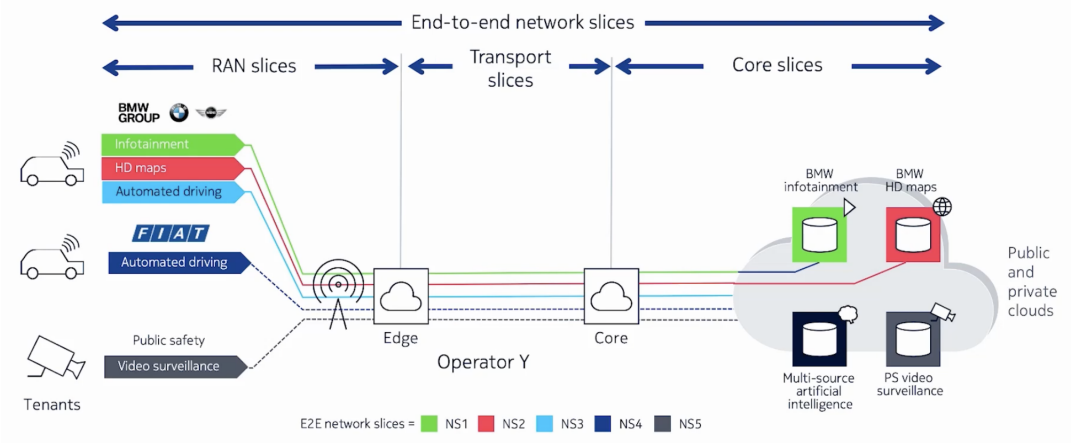


FIGURE 4.3 – Exemple de *slices* réseau de bout en bout [Nok19]

S'appuyant sur de nombreux travaux de recherche [Fou+17 ; Afo+18a ; Kha+20], plusieurs organismes de standardisation (essentiellement 3GPP, ITU, ETSI) ont travaillé sur les différentes facettes du *network slicing* : définition, architecture et orchestration, modèle de rôles, plans de contrôle et de données. Parmi les propriétés qui caractérisent les *slices* réseau, nous retenons les principales ci-dessous.

Qualité de Service Une *slice* étant dédiée à un type de service, elle doit supporter en son sein les mécanismes permettant de garantir les exigences de qualité associées au service. Dans les réseaux 5G, un UE (*User Equipement*) peut appartenir à une ou plusieurs *slices* simultanément et, dans chaque session PDU, plusieurs flux peuvent être établis avec des exigences de QoS différentes. Chaque flux appartient à un *QoS Flow Identifier* (QFI) qui utilise des valeurs normalisées de *5G QoS Identifier* (5QI) qui sont associées à des caractéristiques de QoS spécifiques (*Guaranteed/Non-Guaranteed Bit Rate*, *Packet Delay Budget*, *Packet Error Rate*...).

Bout en bout Une *slice* est instanciée au dessus d'une infrastructure mutualisée qui combine plusieurs segments réseaux. Les ressources qui sont allouées et gérées au niveau de chaque

segment doivent être orchestrées de manière cohérente pour respecter les exigences du service sur tout le chemin. Dans le cas des réseaux mobiles 5G, les différents segments réseaux au dessus desquels sont déployés les *slices* sont : le *Radio Access Network* (RAN), le *Transport Network* (TN), le *Core Network* (CN) ainsi que le *Data Network*.

Isolation Elle a pour rôle d'assurer une indépendance entre les slices. Cette isolation est présente au niveau du plan de contrôle et du plan de données, permettant de garantir la performance et la sécurité de chaque service. Les ressources allouées à la *slice* peuvent être dédiées, partagées ou mixtes. Ce choix assure une isolation plus ou moins forte et dépend du contrat de service (SLA, *Service Level Agreement*) associé à la *slice*.

Élasticité Cette propriété permet à un opérateur réseau ou fournisseur de services de modifier la quantité de ressources allouées à la *slice* (par ex. pour absorber un pic de trafic inattendu ou pour corriger une dégradation du lien radio). Cette opération peut être manuellement déclenchée par un des acteurs impliqués dans la gestion de la *slice* ou par des mécanismes automatiques.

Automatisation La création, la modification et la suppression de la *slice* doivent pouvoir s'effectuer de manière automatique, sans intervention humaine. Une interface de gestion, voire une API, doit exposer les capacités de la *slice* et permettre à des tiers d'interagir avec elle afin d'autoriser sa création ou sa mise à jour.

Sécurité En complément de l'exigence de QoS, il est nécessaire de pouvoir mettre en place au sein de la *slice* un certain niveau de sécurité par le biais de processus AAA (*Authentication*, *Authorization* and *Accounting*) pour gérer l'accès des utilisateurs ainsi que le traitement des données véhiculées. Les paquets qui circulent dans le plan de données peuvent aussi faire l'objet de chiffrement pour garantir la propriété de confidentialité.

La garantie de ces propriétés constitue un réel défi dans le déploiement de *slices* réseau et notamment dans un contexte multi-domaines qui fait intervenir plusieurs opérateurs et plusieurs technologies de communication. C'est le cas avec les réseaux de satellites auxquels nous nous sommes intéressés dans le cadre de l'encadrement de la thèse de Youssouf Drif [Dri22b].

4.3 Intégration d'un segment satellite dans les réseaux 5G

Les réseaux de satellites de télécommunications sont des réseaux longue distance (WAN, *Wide Area Network*) fournissant une connectivité très étendue, en dehors des zones de couverture des réseaux terrestres. Ils possèdent leur propre architecture et permettent de déployer plusieurs types de services et de répondre à différents cas d'usages. On retrouve plusieurs types de satellites : les satellites sur orbite géostationnaire (GEO, *Geostationary Earth Orbit*), sur orbite moyenne (MEO, *Medium Earth Orbit*) et sur orbite terrestre basse (LEO, *Low Earth Orbit*). Avec l'arrivée des mega-constellations (plusieurs milliers de satellites en orbite basse travaillant de concert), fournissant un accès à haut débit et faible latence, de nouvelles opportunités d'intégration aux réseaux terrestres se présentent.

Plusieurs projets de recherche se sont intéressés à l'intégration des réseaux satellites dans les réseaux mobiles terrestres. Nous pouvons citer le projet VITAL [Aga+16] qui a eu pour objectif d'appliquer les concepts de SDN et NFV au réseau satellite afin d'augmenter sa flexibilité et permettre une intégration étroite avec les réseaux 4G. Le projet SaT5G [Jou+18] lui a succédé, ayant pour objectif l'intégration du satellite dans les réseaux 5G, proposant différents modes d'intégration qui peuvent être classés en deux catégories : l'accès direct qui considère le satellite comme une extension radio directe aux réseaux 5G, ou l'accès indirect qui considère le réseau satellite comme un intermédiaire entre le RAN et le CN 5G. Depuis, le support des *Non Terrestrial Network* (NTN), et donc en particulier des réseaux satellites, a été intégré dans la Release 15 du standard 3GPP [3GP21b]. Dans le cadre du projet ANR Super-G sur lequel nous avons travaillé,

nous nous sommes intéressés à l'accès indirect, le réseau satellite étant considéré comme un réseau de transport (TN, *Transport Network*) fournissant un lien de *backhaul* au réseau 5G. Un des enjeux dans ce contexte est de proposer des mécanismes d'interconnexion dynamique qui puissent gérer l'établissement (et tout le cycle de vie) de la *slice* réseau de bout en bout, tout en préservant les exigences de QoS de la *slice*.

4.4 Continuité de *slice* sur un transport satellite

Une partie du projet Super-G s'est intéressée à la définition d'une architecture satellite flexible permettant de supporter le *network slicing*. Cette dernière est basée sur la mise en commun d'infrastructures satellites (mutualisation de pools de ressources), et la proposition d'une passerelle satellite « orientée services », décomposée en trois principales fonctions : fonctionnalités des couches de niveau 3 et supérieur, gestion des ressources radio sur les voies aller et retour, et unité extérieure pour émission et réception du signal vers le segment spatial. Un modèle de rôles et des composants de gestion de slice satellite ont également été définis [Dri+21a]. Bien que ce travail puisse être considéré comme prospectif, il n'en demeure pas moins nécessaire pour supporter pleinement le paradigme du *network slicing* dans les réseaux satellites.

Dans la suite de ce chapitre, nous faisons abstraction de l'architecture interne du réseau satellite et nous nous intéressons plutôt aux problématiques d'intégration d'un réseau NTN comme réseau de transport au sein de *slices* 5G. En particulier, nous identifions les interfaces entre les systèmes de gestion et de contrôle 5G et NTN, et nous introduisons un composant d'interconnexion essentiel au plan de données, le « slice classifier », qui aura pour rôle de classifier les flux en entrée/sortie du domaine satellite, et d'appliquer les mécanismes permettant d'assurer la continuité des *slices* et de leur QoS.

4.4.1 Interface entres les systèmes de gestion

L'interconnexion des systèmes de gestion satellite et 5G est la première étape afin de négocier l'établissement d'un tronçon satellite et ainsi obtenir une *slice* de bout en bout. Le réseau satellite peut être considéré comme un « sous-réseau » de la *slice*, c'est-à-dire un *Network Slice Subnet* dans la terminologie 3GPP. Il pourra alors être spécifié à l'aide du modèle d'information *Slice Profile* du 3GPP [3GP21a]. La version actuelle de cette spécification (Slice NRM v17.7.0) ne prévoit pas de profil pour le réseau de transport (uniquement pour le RAN et le CN), mais cette spécification ne cesse d'évoluer. Plus d'une vingtaine de paramètres sont prévus dans le modèle *Slice Profile*. Dans nos travaux, nous simplifions cette interface pour n'en retenir que 5, à travers la notation Theta $\Theta(\lambda, \delta, \mu, \beta, \sigma)$ que nous considérons comme les paramètres de référence du lien NTN qui seront demandés par l'opérateur 5G. Ces paramètres sont définis comme suit :

— λ : **latence** maximale en millisecondes. Ce paramètre détermine le segment spatial que l'opérateur du satellite doit utiliser pour assurer la liaison NTN (GEO, MEO ou LEO). Il influence également la politique de QoS à appliquer aux flux.

— δ : **gigue** maximale en millisecondes. Là encore ce paramètre a un impact sur la politique de QoS à appliquer aux flux, voire le choix de certaines fonctions réseau pour réduire ou compenser les retards non constants.

— μ et β : **débits descendants et montants** à garantir en Mbit/s. En fonction du débit requis, les opérateurs de satellites utiliseront certaines bandes de fréquences, de protocoles et d'équipements sur les voies aller et retour des satellites.

— σ : **partage des ressources** (booléen) avec d'autres *slices*. Ce paramètre influence le niveau d'isolement requis, donc les choix d'allocation de ressources (radio, calcul...).

Une fois que ces paramètres seront négociés, le système de gestion du réseau satellite pourra les utiliser pour déterminer les équipements à déployer, dimensionner les ressources nécessaires et les configurations optimales pour ce *slice subnet*.

4.4.2 Mécanismes de couture

En utilisant l'interface Θ précédente, nous sommes capables de dimensionner un lien NTN en fonction des besoins de la *slice* (exprimés par l'opérateur 5G) et de réserver les ressources correspondantes. Cependant, cela n'est pas suffisant pour considérer le lien NTN comme *slice-aware* car la continuité de service et de QoS doit être assurée dans le plan de données. Le réseau satellite doit en effet être capable d'identifier, pour chaque paquet, à quelle *slice* il appartient, puis ses exigences en matière de QoS.

Dans le domaine 5G, chaque UE peut établir une ou plusieurs sessions PDU et chacune d'entre elles est associée à une unique *slice*. Cependant, pour chaque session PDU, plusieurs flux avec des exigences de QoS différentes peuvent être établis (exigences identifiées grâce au QFI, *QoS Flow Identifier*). Ces éléments sont schématisés dans la partie du haut de la figure 4.4.

Le système de gestion de la QoS dans le NTN est différent de celui utilisé dans la 5G, les deux systèmes étant totalement indépendants l'un de l'autre. Nous proposons donc de mettre en place des mécanismes de couture assurant une correspondance des classes de QoS pour garantir la continuité du service. Toutes les classes de service de la 5G ne sont pas supportées par les systèmes satellite (notamment les satellites GEO) mais les méga-constellations LEO changent la donne et ouvrent la voie à davantage de prises en charge des classes standards 5QI [3GP21b].

Afin d'assurer ces mécanismes de couture, nous proposons de rajouter une fonction *slice classifier* à la frontière du domaine satellite, comme illustré dans la figure 4.4 [Dri+21b].

Le premier rôle de cette fonction est de classifier les paquets et de leur faire correspondre une politique de QoS qui pourra être mise en œuvre par le réseau satellite. Ce travail consiste à effectuer un double *mapping* : d'abord basé sur l'identification de la *slice*, puis basé sur l'identification de la classe de QoS. Pour cela le *slice classifier* maintient une table de correspondance pour les slices, et n tables de correspondance de QoS (une associée à chaque *slice*). Le *classifier* inspecte chaque paquet et fait correspondre un ensemble de champs pour identifier son appartenance à la *slice* et QoS associée. Il traduit ensuite certains champs de l'en-tête du paquet en un paquet compréhensible par le segment réseau suivant. Par exemple, les flux entrants du réseau de données (côté est) pourraient être discriminés en se basant sur l'adresse IP de l'UPF (*User Plane Function*) et le champ TEID du protocole GTP-U pour identifier la *slice*, et le champ QFI de l'en-tête d'extension de GTP-U pour identifier la classe de QoS. Le *classifier* prend ensuite des mesures en fonction de la correspondance (par ex. modifier la valeur du champ DSCP de l'en-tête IP ou insérer un label MPLS) pour traduire les flux qui seront traités par le réseau satellite. Ces tables de correspondance sont mises à jour dynamiquement par des composants du plan de contrôle que nous présentons dans la section suivante.

Le deuxième rôle du *slice classifier* est de mettre en oeuvre la politique de QoS du flux à l'entrée du réseau satellite. Appliquer la politique de QoS dès le niveau *slice classifier* nous permet de mettre en place une fonction de *traffic policing* afin de garantir que les flux respectent bien le contrat de service de la *slice* et ainsi éviter la consommation de ressources satellite non négociées. Par ailleurs, un ordonnanceur basé sur les classes de QoS assure une priorisation des flux qui l'exigent au sein de la *slice*, comme le feraient les autres composants du système satellite.

4.4.3 Architecture fonctionnelle

La procédure et les mécanismes, décrits précédemment, pour l'intégration d'un segment NTN dans une *slice* 5G sont mis en œuvre à travers un ensemble de nouveaux composants et interfaces que nous interconnectons aux fonctions 5G existantes. Cette architecture est présentée dans la figure 4.5. Nous nous concentrons ici sur l'intégration des plans de contrôle et de données, nous

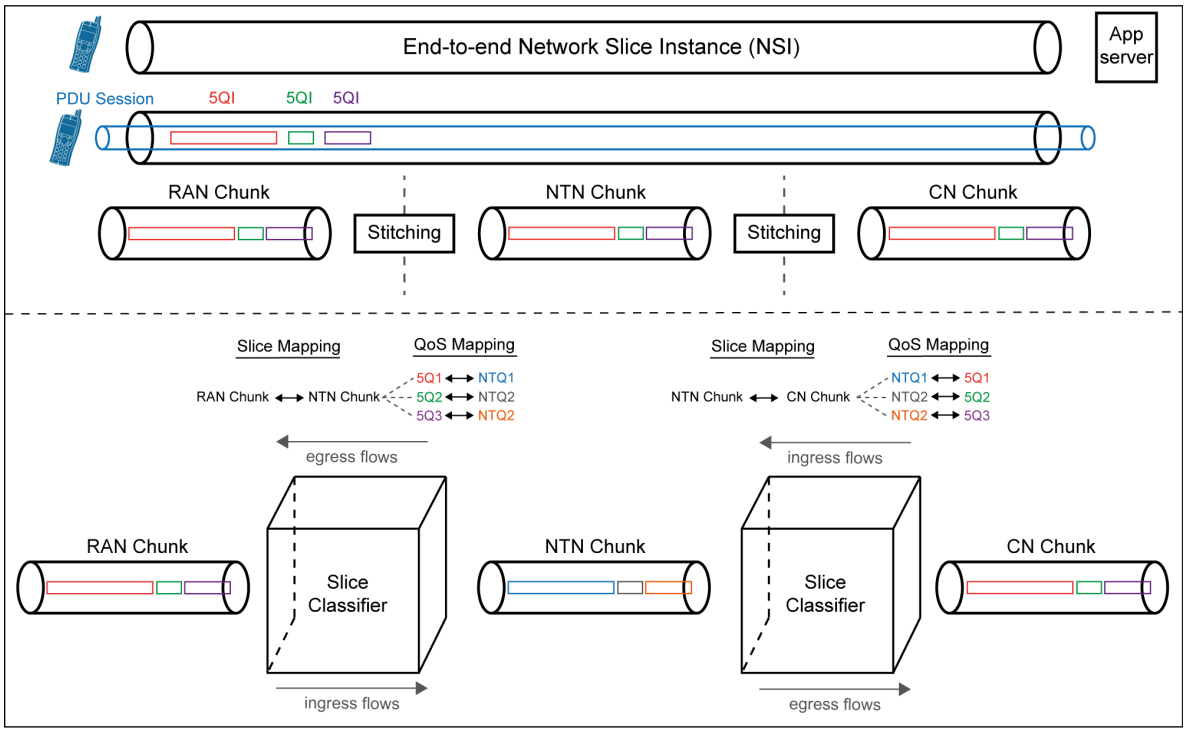


FIGURE 4.4 – Modèle de *slice* de bout en bout avec transport NTN

supposons que les paramètres de dimensionnement de la *slice* (paramètre Θ du modèle) ont été négociés en amont entre les opérateurs 5G et satellite (via l'interface de gestion).

Dans la procédure d'établissement d'une session PDU entre le terminal utilisateur (UE, *User Equipement*) et le réseau 5G, la fonction de contrôle SMF (*Session Management Function*) gère tous les paramètres associés à la session, dont l'identifiant de la *slice*, appelé S-NSSAI (*Single Network Slice Selection Assistance Information*). Cette fonction SMF est appelée pour chaque création, modification ou suppression d'une session PDU. Nous avons modifié cette fonction pour intercepter les requêtes et exécuter une sous-routine vers une nouvelle fonction QOF (*Quality of Service Function*) que nous proposons de rajouter dans le cœur 5G (cette possibilité est prévue dans le standard via des applications tierces appelées AF, *Application Function*). La fonction SMF extrait de la session PDU les informations nécessaires à la caractérisation des flux (S-NSSAI, points de terminaison du tunnel GTP, QoS Flow Identifier...) et les transmet à la fonction QOF qui effectue un pré-traitement de ces informations avant de les envoyer à son tour à la fonction NTN QOF gérée par l'opérateur satellite. Celle-ci va se charger de générer les règles de correspondance dans le plan de données (champs d'identification des *slices* et de QoS dans l'en-tête des paquets) entre les domaines 5G et satellite. Pour finir, ces règles sont poussées vers les *slice classifier*.

Les fonctions QOF et NTN QOF ont été implémentées en langage Go selon un modèle orienté services (telle l'architecture SBA (*Service-Based Architecture*) de la 5G), en exposant leurs fonctionnalités via des interfaces REST/HTTP. Quant au *slice classifier*, également développé en Go, il dispose d'un premier module qui expose une interface nord HTTP/REST vers la NTN QOF pour le plan de contrôle, et un deuxième module qui met en œuvre le plan de données à travers le *framework netfilter* du noyau Linux (et ses outils de plus haut niveau `tc` et `iptables`). Les politiques de QoS sont gérées par l'ordonnanceur de paquets du noyau grâce à une combinaison de files d'attentes *Hierarchy Token Bucket* (HTB) et *Stochastic Fairness Queueing* (SFQ).

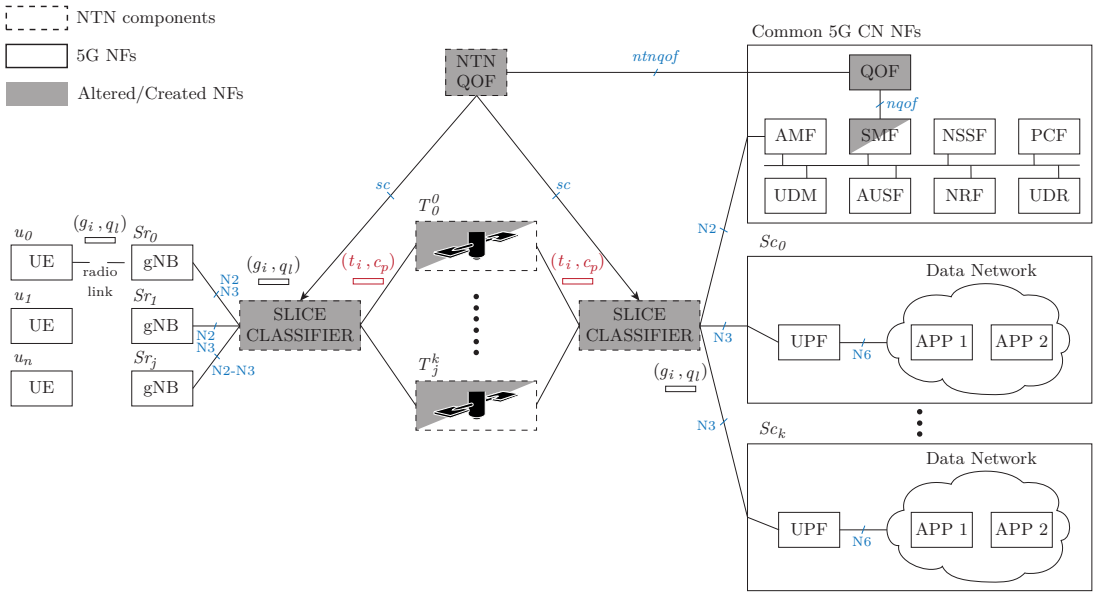


FIGURE 4.5 – Architecture fonctionnelle pour l'intégration d'un NTN *slice subnet* dans la 5G

4.4.4 Plateforme d'expérimentations

Afin de pouvoir tester et valider les propositions précédentes, la plateforme d'expérimentations COMETS a été développée [Dri22a]. L'objectif d'une telle plateforme était de pouvoir automatiser des tests à partir de la définition d'un scénario incluant sa durée, le nombre d'UE à simuler, les propriétés des liens, les slices et leurs caractéristiques. La plateforme permet également de générer le trafic des applications, sauvegarder les journaux d'exécution et tracer les performances en matière de débit, perte, délai et gigue. L'ensemble s'exécute grâce à des composants *Docker*, chacun représentant un élément fonctionnel de l'architecture décrite dans la figure 4.5. Outre l'implémentation des fonctions issues de nos contributions (*slice classifier*, QOF, NTN QOF), la plateforme se base sur des outils existants dans le monde du logiciel libre, en particulier pour le cœur 5G, la simulation du RAN et du lien satellite :

— Le cœur 5G utilise free5GC [Fre22], conforme à la Release 15 du 3GPP et choisi comme cœur de référence pour les projets de l'ONF. Nous avons modifié la fonction SMF de cette implémentation afin de pouvoir s'interfacer avec notre fonction QOF.

— Les UE et le RAN sont simulés par UERANSIM [Gün22] qui fournit une implémentation d'UE, de gNB logicielle (conforme à la Release 16 du 3GPP) et qui simule le lien radio entre les deux. Cependant, comme aucun mécanisme de QoS n'est supporté par l'outil dans le plan de données, nous avons dû développer un nouveau module dans la gNB afin de reporter le marquage du champ DSCP du paquet IP interne (celui de l'utilisateur) vers l'en-tête IP externe (celui du paquet GTP).

— Le lien satellite est simulé par l'outil Trunks [Dri22c], supportant les mécanismes de QoS de niveau 3 et capable de simuler une dégradation de lien avec la mise en place de l'Adaptive Coding and Modulation (ACM). Il est basé sur les outils réseaux natifs de Linux (`tc`, `iptables`).

— Les flux sont générés grâce à l'outil `iPerf`, paramétré (débit, burst, trafic isochrone...) en fonction de l'application cible (*streaming* vidéo ou VoIP par exemple).

Un fichier de spécification du scénario cible permet à la plateforme de générer automatiquement tous les fichiers de configuration *Docker* et *Docker Compose*, ainsi que les scripts de

lancement des différents conteneurs. Les détails d'implémentation de la plateforme peuvent être trouvés dans la thèse de Youssouf Drif [Dri22b] et sur le dépôt logiciel [Dri22a].

4.4.5 Scénario avec deux *slices* partageant un lien satellite

L'évaluation de l'architecture et des mécanismes que nous proposons pour assurer une continuité de service sur un transport satellite s'est faite en définissant un scénario de deux *slices* déployées sur un même lien satellite LEO, chaque *slice* proposant différentes applications avec des exigences de QoS spécifiques. Le tableau 4.1 donne les principaux paramètres du scénario.

TABLE 4.1 – Paramètres d'un scénario avec deux slices sur un lien LEO

Paramètre	**Description**
$D_0 = [0, 240]$	Slice 0 de 0s à 240s
$A_0 = \{\text{Web QFI 7, VoIP QFI 9}\}$	Applications dans la slice S_0
$\Theta_0 = (45, 5, 35, 5)$	Exigences de S_0 pour le lien NTN
$D_1 = [120, 240]$	Slice 1 de 120s à 240s
$A_1 = \{\text{Streaming QFI 6}\}$	Application dans la slice S_1
$\Theta_1 = (45, 5, 60, 15)$	Exigences de de S_1 pour le lien NTN

Le lien NTN est configuré comme une liaison LEO avec un temps d'aller-retour d'environ 50 ms, une capacité aller de 100 Mbps et une capacité retour de 25 Mbps. La *slice* S_0 est dédiée aux applications à faible latence A_0 : la VoIP (QFI 7) qui nécessite un débit constant de 128 kbps et la navigation Web (QFI 9) qui génère environ 3 Mbps de trafic. La *slice* S_1 est dédiée aux applications de streaming vidéo (QFI 6) A_1 qui consomment 10 Mbps. Les exigences de QoS pour chaque slice sur le tronçon NTN sont exprimées par l'opérateur 5G, via la notation Θ dans notre modèle : latence maximale (en ms), gigue maximale (en ms), débits descendant et montant (en Mbps) à garantir. Dans ce cas de figure, nous avons bien les deux *slices* qui peuvent théoriquement partager le même lien LEO ($45 < 50$ ms, $35{+}60 < 100$ Mbps et $5{+}15 < 25$ Mbps).

Cependant, dans le scénario testé, 9 UE sont démarrés pour chaque *slice*, tous établissent des sessions PDU et commencent à générer le trafic correspondant aux applications A_i dès le démarrage de la *slice*. Les flux applicatifs présents sur S_1 vont donc consommer davantage de ressources que l'exigence spécifiée par Θ_1 (9 UE consommant chacun 10 Mbps dépassent largement les 60 Mbps demandés).

Nous exécutons ce scénario deux fois sur la plateforme COMETS : une fois sans les mécanismes de couture (*No Slice*) et une fois en utilisant notre approche (*Slice Aware*). Les résultats sont présentés dans la figure 4.6. Pour chaque flux, nous mesurons le débit descendant, le taux de pertes et le délai unidirectionnel (*one-way latency*). Notez que la gigue est également mesurée par la plateforme mais non représentée dans la figure 4.6. Nous agrégeons les résultats par QFI et par *network slice*.

Respect du contrat de service

Lorsque l'opérateur satellite alloue des ressources pour véhiculer les données d'une *slice* 5G, il est nécessaire de mettre en place un mécanisme permettant de garantir que les flux respectent le contrat de service. C'est une des fonctions du composant *slice classifier* qui implémente un limitateur de flux, ou *traffic policing*. Cette fonction est mise en évidence dans la sous-figure a) 4.6 où le débit de S_1 atteint plus de 67 Mbps sur un contrat négocié à 60 Mbps (le débit ne va pas au-delà car la somme des débits des slices est limitée par la capacité du lien LEO à 100 Mbps). Cette anomalie n'est pas présente dans le mode *slice aware* dans lequel le trafic est limité en entrée du lien NTN en fonction du paramètre Θ de notre modèle.

Isolation inter-slices

Une propriété fondamentale à garantir avec le concept de *slice* réseau concerne l'isolation des ressources. Une *slice* ne doit pas pouvoir consommer des ressources utilisées par une autre *slice*. Dans notre scénario, S_1 va consommer davantage de ressources que prévu (à partir de $t = 120$ s), ce qui va avoir un impact sur S_0. Cela est visible sur le débit de S_0 qui est réduit de plus de 2 Mbps dès que S_1 démarre, ainsi que sur le taux de pertes qui oscille entre 3 et 20%. La latence est également fortement affectée passant de 24 à 32 ms.

Avec les mécanismes proposés, des améliorations significatives sont apportées et les erreurs ne se propagent pas de S_1 vers S_0. De 120 à 240 ms, on constate que S_0 ne subit aucune perte de paquets, que ce soit sur le trafic Web ou VoIP, le taux de pertes est nul. En revanche, les pertes de S_1 croissent comparativement au cas précédent (qui atteignent 38%) car les pertes ne sont plus partagées avec S_0. Le débit reste stable pour les deux classes de trafic (Web et VoIP), ainsi que la latence qui n'est pas affectée par la surconsommation de ressources de S_1.

Priorisation intra-slice

Un autre mécanisme implémenté par le *slice classifier* porte sur les politiques d'ordonnancement et de files d'attentes dédiées à des classes de trafic. En particulier, nous avons défini une classe VoIP prioritaire par rapport aux autres classes. Cette fonction est visible dans la sous-figure c) 4.6 dans laquelle le délai du trafic VoIP est légèrement inférieur au délai du trafic Web dans S_0 (en mode *slice aware*). Cet aspect se répercute également sur la gigue (non représentée sur la figure) du trafic VoIP qui est inférieure à la gigue du trafic Web sur toute la durée du scénario. Cela démontre que la QoS est préservée et que le trafic VoIP prime bien sur les autres.

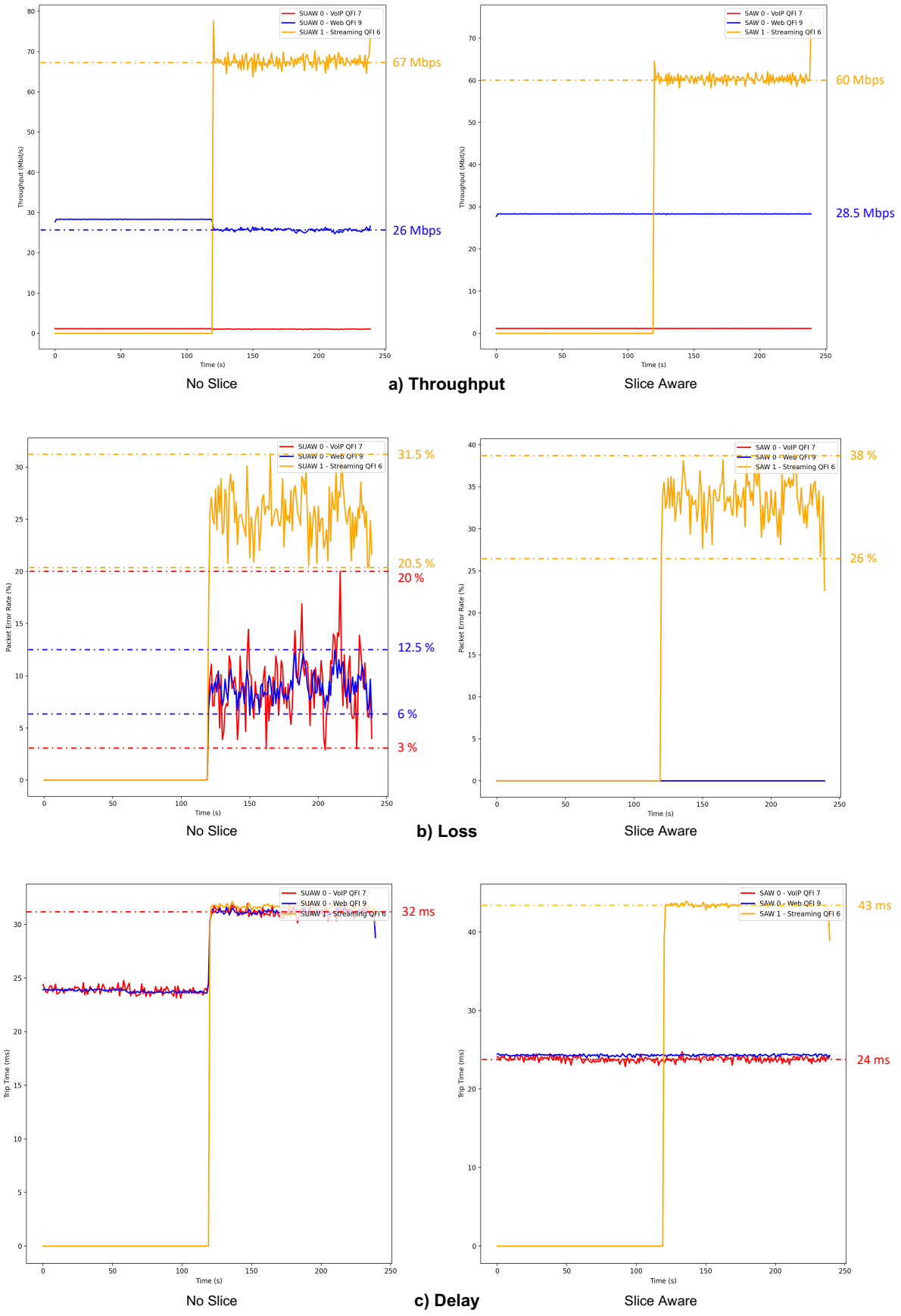


FIGURE 4.6 – Évaluation du débit (a), taux de pertes (b) et délai (c) sur les *slices* S0 et S1 instanciées sur un même lien satellite LEO

4.5 Conclusion

Les contributions présentées dans ce chapitre se sont focalisées sur la conception de réseaux virtuels dédiés à un usage particulier. D'abord à travers la composition dynamique de micro-services pour fournir, sur un réseau mobile, un service adapté aux exigences des applications ou aux préférences de l'utilisateur. Puis, dans le cadre d'un déploiement d'une *slice* 5G sur un réseau de transport satellite.

Dans le premier cas, la personnalisation du réseau se fait par choix (statique) d'une chaîne abstraite de services (de type transcodage), puis par instanciation (dynamique) de la chaîne sur des services concrets sélectionnés parmi un ensemble de services fonctionnellement équivalents. Cette sélection se fait à l'aide d'un annuaire décentralisé sur la base de critères non fonctionnels (par ex. le temps de réponse du service) et le chaînage est réalisé par le protocole de signalisation SIP. Le réseau virtuel résultant est un réseau de recouvrement dont le chemin passe par une série de services. La pertinence de cette démarche a été confirmée quelques années plus tard par l'essor du paradigme de chaînage de fonctions réseaux SFC (*Service Function Chaining*) [HP15]. Néanmoins, malgré une sélection de micro-services basée sur des propriétés non fonctionnelles, un réseau virtuel sous la forme d'un réseau de recouvrement reste une limite qui entrave les propriétés de bout en bout, du fait qu'il n'y ait aucune garantie sur les réseaux de transport entre les services.

Dans le deuxième cas, nous nous sommes intéressés au concept de *slice* dans un réseau 5G, qui, de par ses propriétés (d'isolation en particulier), peut être vu comme un réseau virtuel de bout en bout dédié à un service particulier. Contrairement à l'approche précédente, la personnalisation ne se fait pas à une granularité aussi fine que la session de l'utilisateur mais plutôt sur le type de service demandé (plusieurs flux utilisateurs pouvant être associés à la même *slice*). Cependant, la *slice* n'est pas un réseau de recouvrement, sa complexité de mise en œuvre est bien plus importante, et réside principalement dans la gestion des ressources nécessaires pour respecter les exigences associées au service, sur tout le chemin. Ce problème est d'autant plus complexe lorsqu'un sous-réseau appartient à un domaine administratif non géré par le cœur 5G. C'est l'objet des travaux que nous avons présentés sur l'intégration d'un lien satellite comme réseau de *backhaul* du réseau 5G. Une interface de gestion pour spécifier les exigences de QoS requises par la *slice* a été proposée, ainsi qu'une architecture intégrant de nouveaux composants interagissant d'un côté avec le plan de contrôle 5G, et de l'autre avec le plan de données du réseau satellite. Les mécanismes implémentés dans le plan de données, en entrée/sortie du domaine satellite, assurent la couture inter-domaines des flux (via des techniques d'encapsulation ou de traduction de champs) ainsi que l'exécution des politiques de QoS associées aux flux de la *slice* (via des files d'attentes dédiées à des classes de trafic).

Chapitre 5

Projet de recherche

Dans ce dernier chapitre je présente des perspectives de recherche pour les années à venir, que j'ai regroupées en deux grands axes scientifiques. Le premier axe, présenté dans la section 5.1, porte sur « l'informatique de bordure » (*edge computing*) et ses problématiques d'intégration dans les réseaux 5G. Nous nous intéressons ici aux solutions architecturales et au problème de continuité de service dans le cas de la mobilité. Le second axe, présenté dans la section 5.2, s'inscrit dans la mutation profonde qui s'opère dans le domaine des réseaux de communication depuis une dizaine d'années autour des réseaux logiciels ou programmables. Ce domaine est très riche, en plein essor à la fois dans la recherche académique et industrielle. Je cible mes activités dans ce contexte sur la modélisation des réseaux virtuels et sur la génération d'instructions de télémétrie pour vérifier les exigences associées à ces réseaux virtuels.

5.1 Intégration d'une approche *edge* dans les réseaux mobiles 5G

Les évolutions technologiques permanentes des terminaux des utilisateurs, les exigences de performance toujours plus élevées de certaines applications (notamment en matière de latence et de débit), la multiplication des objets connectés ou encore les besoins temps-réel des communications véhiculaires ont participé ces dernières années au développement de nombreux projets de recherche académiques et industriels autour de l'*edge computing* [Tal+17 ; MB17]. Cette approche consiste à pousser des capacités de calcul et de stockage depuis le *cloud* vers les consommateurs de données, c'est-à-dire au plus près des utilisateurs, à la périphérie des réseaux, et notamment des réseaux de mobiles. L'objectif principal étant d'améliorer la rapidité de réaction (*responsiveness*) des applications pour l'utilisateur final. Ce besoin est amplifié dans le cas de l'utilisation de réseaux de transport contraints telles les communications par satellite (bande passante limité, latence, accès partagé...). Ce concept d'*edge computing* a évolué d'un simple service de distribution répartie de contenu (type CDN, *Content Distribution Network*) vers une plateforme ouverte, en interaction forte avec l'infrastructure réseau, et pouvant héberger des services tiers.

L'effort de standardisation du paradigme *edge computing* est mené principalement par l'ETSI à travers les spécifications MEC (*Multi-access Edge Computing*), parmi lesquelles on trouve les traditionnels cas d'utilisation [ETS22b] et l'architecture de référence [ETS22a]. Cette architecture définit les éléments fonctionnels et interfaces nécessaires au développement d'une plateforme MEC, ainsi qu'une variante pour un déploiement dans un environnement de virtualisation NFV (*Network Functions Virtualisation*).

En parallèle de cette évolution, les réseaux mobiles de 5$^{\text{ème}}$ génération ont vu le jour et constitueront bientôt le principal réseau d'accès pour les équipements mobiles. Comme évoqué dans la section 4.2, ce réseau doit supporter les services URLLC et V2X nécessitant un temps de latence extrêmement faible. De plus, comme indiqué dans sa spécification technique, le système 5G doit pouvoir mettre en œuvre de l'*edge computing*, au moins dans un scénario de non *roaming* (cf. clause 5.13 de [3GP21b]). Pourtant, de manière surprenante, jusqu'à aujourd'hui peu de travaux

de recherche ou d'implémentations *open source* ont étudié les problématiques d'intégration d'une plateforme MEC dans une architecture 5G. Des recommandations ont été émises par l'ETSI [ETS20] et l'initiative la plus avancée, à notre connaissance, est le projet SD-Fabric porté par l'ONF [SD-21]. Dans ce contexte, nous souhaitons développer deux axes de recherche : d'une part, une étude sur les alternatives d'architecture permettant l'intégration des concepts MEC dans un système 5G et, d'autre part, les méthodes et mécanismes qui assurent la continuité de service en cas de mobilité de l'utilisateur.

Architecture et mécanismes de contrôle sous-jacents

Dans sa spécification, le standard 5G intègre le système MEC comme un ou plusieurs composants AF (*Application Function*) qui peuvent, soit être autorisés à interagir directement avec les fonctions du plan de contrôle 5G (cas d'une AF de confiance), soit passer par un service exposé via le composant NEF (*Network Exposure Function*). Ces interactions entre les *API framework* MEC et 5G (enregistrement et découverte de services, authentification, autorisation...) nous paraissent relever davantage de problèmes d'ingénierie que de problématiques de recherche. En revanche, le rôle clé du composant UPF (*User Plane Function*) dans le déploiement d'une solution MEC nous semble être un point d'étude important dans la définition de l'architecture globale du système. Pour rappel, l'UPF est une fonction essentielle du plan de données de la 5G, elle sert de routeur IP, agit comme un point d'ancrage stable de l'UE auprès des réseaux externes, implémente des mécanismes de traduction d'adresses et de QoS, mesure la consommation des données des utilisateurs et bien d'autres fonctions encore. Lors du déploiement de services MEC, c'est elle qui doit se charger de rediriger le trafic utilisateur (ou seulement un partie) vers un réseau local ou vers une application spécifique. Plusieurs questions ouvertes se posent alors : où déployer les UPF par rapport aux services MEC ? plusieurs hôtes MEC peuvent-ils partager le même UPF ? comment gérer plusieurs applications MEC (associées à différents UPF) avec un minimum d'impact sur le réseau de cœur (CN) 5G ? Par exemple, le projet SD-Fabric [SD-21] propose une abstraction *one-big UPF* dans laquelle toutes les fonctions UPF distribuées sur différents sites (ou dupliquées pour des raisons de passage à l'échelle ou de tolérance aux pannes) sont gérées par un unique composant UPF logiquement centralisé sur un contrôleur SDN. Cette proposition permet à l'infrastructure physique ou virtuelle sous-jacente au MEC d'évoluer indépendamment du CN 5G et aussi de s'affranchir du protocole de contrôle PFCP [1] pour pouvoir s'appuyer sur d'autres implémentations d'UPF, logicielles ou matérielles (tel [Mac+21] qui propose une implémentation d'UPF en langage P4 avec un contrôle basé sur le protocole P4Runtime).

Mobilité et continuité de service

En fonction de la localisation de l'utilisateur (définie essentiellement par son point d'accès), plusieurs options peuvent être envisagées pour accéder à une instance *edge* d'un service. Cela peut-être une désignation explicite du nom ou de l'adresse du service MEC par l'UE (nécessite un UE *MEC-aware*), une technique de redirection (comme une redirection applicative ou une redirection basée sur la résolution DNS), ou encore un réacheminement géré entièrement par l'infrastructure réseau à travers du *traffic steering*. Dans un réseau 5G, ce dernier cas est opéré en grande partie par la fonction UPF via des règles issues du plan de contrôle et orientant le trafic vers les réseaux de bordure hébergeant les services MEC. Une fois la session de l'utilisateur établie, sa mobilité est un grand défi pour les services MEC qui doivent maintenir la continuité du service tout en préservant ses exigences de performance.

Pour le service MEC, la mobilité de l'utilisateur implique que l'instance du service doit être soit transférée vers la nouvelle localisation, soit une nouvelle instance répliquée doit être démarrée [FK18]. Dans le cas des services avec état (*statefull*), le contexte de l'utilisateur doit également

1. *Packet Forwarding Control Protocol*, protocole de contrôle standard 3GPP

être transféré. Plusieurs travaux existent sur les algorithmes de décision de migration de service [MIR22] tout en essayant d'optimiser la latence ou la consommation de ressources [Kim+22].

En complément de la migration du service, les flux réseaux doivent être réorientés, c'est cette problématique qui nous intéresse avant tout. Un groupe de travail de l'ETSI [ETS20] a identifié cette problématique de mise à jour des chemins du plan de données à la fois dans un contexte intra-opérateur mais aussi inter-opérateurs comme illustré dans le figure 5.1. Des pistes de solution sont avancées qui s'appuient sur un processus de signalisation entre le plan de contrôle 5G et la plateforme MEC. Ce processus cible la sélection et la configuration d'un nouvel UPF, ainsi que la mise à jour du chemin entre lui et l'UE. Néanmoins, l'acheminement des flux entre l'UPF et les services applicatifs ne sont pas du ressort du plan de contrôle 5G. Une perspective que nous souhaitons investiguer est liée aux modèles de *traffic steering* qu'il est possible de mettre en place pour acheminer les flux vers les services MEC tout en préservant les propriétés de QoS exigées par l'application ou associées à la session PDU. Des modèles de traduction d'adresses, de tunnels, de reprogrammation du plan de données ou de routage par segments [Ven+21] pourront être évalués et comparés. De telles opérations peuvent devenir encore plus complexes dans des cas de multi-connectivité où des applications peuvent utiliser des chemins d'accès multiples afin de maximiser l'utilisation des ressources (par exemple avec l'utilisation de protocoles de transport comme MPTCP [For+20] ou QUIC [IT21]).

En complément de l'aspect multi-opérateurs, un enjeu supplémentaire se présente lorsque des utilisateurs consomment plusieurs applications en parallèle, associées à des plateformes MEC distinctes (c'est-à-dire opérées par des fournisseurs des services MEC différents). Cela peut paraître exceptionnel à l'heure actuelle mais clairement envisageable lorsque les offres de services MEC se démocratiseront.

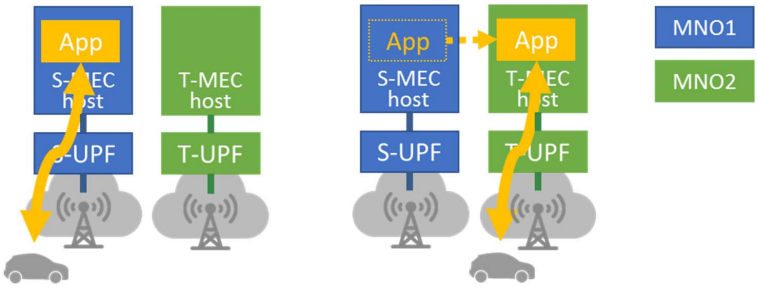


FIGURE 5.1 – Mobilité MEC inter-opérateurs par resélection d'UPF [ETS20]

5.2 Vers des réseaux programmables de bout en bout

Depuis plusieurs années la programmabilité dans les réseaux se développe et devient incontournable, poussée par des projets de recherche d'envergure et certains grands industriels qui dominent le marché. La meilleure démonstration de cette évolution réside dans le lancement, début 2021, du projet PRONTO [Pro21] qui a reçu une aide de 30 millions de dollars de l'agence DARPA (*Defense Advanced Research Projects Agency*), soit un des plus lourds investissements du gouvernement des États-Unis dans le domaine des réseaux informatiques depuis la création d'ARPANET. Le projet regroupe les universités de Stanford, Princeton et Cornell, ainsi que le consortium ONF (*Open Networking Foundation*). Son objectif général est de soutenir la recherche et le développement pour constuire une plateforme informatique distribuée (*distributed computing platform*), s'appuyant sur des solutions *open source*, et permettant de programmer le plan de contrôle et le plan de données du réseau de bout en bout, du système d'exploitation de l'hôte jusqu'au serveur, en passant par les cartes réseaux, les commutateurs et les *middleboxes*.

Cette vision d'une programmabilité extrême du réseau a pour ambition d'encourager l'innovation vers de nouvelles applications, avec toujours plus de performance, fiabilité et sécurité.

Le schéma de la figure 5.2, issu de la présentation du projet Pronto, illustre bien l'objectif d'un réseau programmable de bout en bout, spécifié de haut en bas (*top-to-bottom*), et entièrement construit à partir de logiciel [McK21]. Cette vision fait écho à certains de nos travaux de recherche et en particulier ceux présentés dans le chapitre 3, issus de le thèse de Messaoud Aouadj. L'interface nord sur laquelle nous avons travaillé permet à un opérateur de spécifier son réseau virtuel et les politiques de contrôle associées selon un modèle *Edge-Fabric*. Cette spécification peut être vue comme l'expression d'un objectif de haut niveau (*specified behavior*) qui va piloter le comportement du réseau. La partie compilation et projection vers une infrastructure physique est également présente dans nos travaux par le biais de l'hyperviseur AirNet (cf. figure 3.10) dont le rôle est de transformer les politiques de contrôle en règles physiques pouvant s'exécuter sur un équipement OpenFlow. Néanmoins, une différence importante existe entre notre contribution et la vision du projet Pronto, à savoir que nous avons ciblé exclusivement la programmation du plan de contrôle à travers des fonctions s'exécutant sur le contrôleur SDN. Nous ne modifions pas le *pipeline* du plan de données : soit nous utilisons celui sous-jacent à l'interface OpenFlow, soit nous abstrayons des fonctions existantes du plan de données dans des composants appelés *data machines* dans le langage AirNet. Par ailleurs, malgré un module de *mapping* entre les composants du réseau virtuel et des équipements de l'infrastructure physique, nous n'avons pas une projection complète de bout en bout puisque nous limitons le déploiement des fonctions au cœur et à la périphérie du réseau, sans considérer les hôtes d'extrémité.

L'ambition de pouvoir construire dynamiquement un réseau virtuel personnalisé en fonction d'un besoin particulier, intégrant toutes les fonctions nécessaires à la performance et à la sécurité du service, de bout en bout, tout en minimisant la consommation des ressources, peut être apparentée à une « quête du Graal » dans le domaine des réseaux de communication. Les verrous scientifiques sont multiples : expression du service attendu à un haut niveau d'abstraction, transformation de ce besoin en une composition cohérente de fonctions système et réseau, partitionnement de ces fonctions sur les différents domaines et segments du réseau, placement sur l'infrastructure, configuration (ou compilation) sur les ressources réelles, et enfin gestion du cycle de vie de ce réseau virtuel dédié (supervision, adaptation, libération). Dans ce dessein, nous souhaitons nous concentrer sur deux perspectives de recherche : l'une sur les modèles d'abstraction des réseaux virtuels pour la spécification des services, l'autre sur la génération automatique de tests basée sur de la télémétrie pour vérifier les exigences associées à ces services.

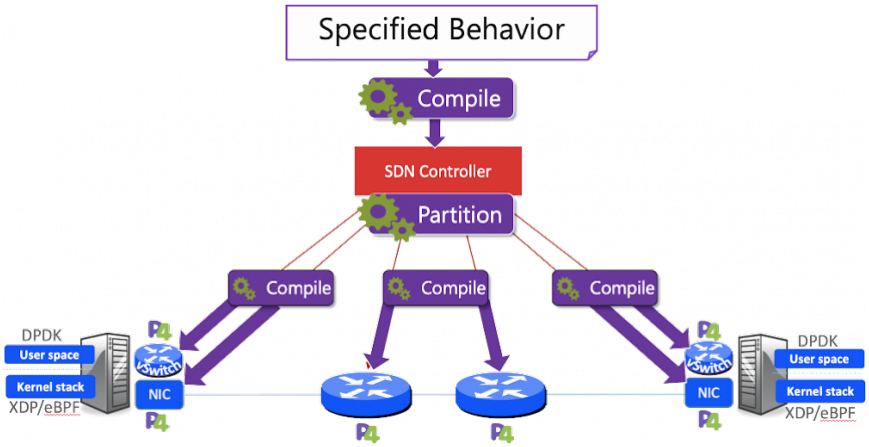


FIGURE 5.2 – Réseau programmable de bout en bout selon le projet Pronto [Pro21]

Modèle d'abstraction de services réseau

La façon dont l'opérateur réseau (ou fournisseur de service, ou développeur) spécifie son besoin est primordiale, à la fois pour lui présenter le niveau d'abstraction adapté (indépendant des technologies mais capturant toutes les informations nécessaires à la réalisation du service de bout en bout) et aussi pour que les algorithmes d'allocation de ressources puissent s'appuyer sur ce modèle (et sur le modèle des ressources sous-jacentes) pour résoudre les problèmes de placement de fonctions virtuelles et de routage/chaînage (cf. *VNF Placement and Traffic Routing problem* [Yan+21]). C'est le travail que nous avons mené sur l'interface nord des contrôleurs SDN (cf. chapitre 3). Le langage AirNet que nous avons défini permet d'exprimer une topologie virtuelle selon une abstraction *Edge-Fabric*, ainsi que des politiques de contrôle appliquées aux flux de cette topologie. Nous souhaitons revisiter ce modèle compte tenu de l'expérience acquise, des récents travaux sur le *slicing*, sur le chaînage de fonctions réseaux et sur la programmation du plan de données. En particulier, nous voyons deux axes d'amélioration :

— La premier point est lié aux objets manipulés dans le modèle *Edge-Fabric* en lui-même. Nous pensons que maintenir une séparation entre des fonctions complexes de traitement des paquets (*Edge*) et des politiques de transport (*Fabric*) reste pertinent, mais il nous paraît nécessaire de rajouter la possibilité de faire de l'abstraction hiérarchique, propriété présente dans le concept de *network slicing* [Afo+18b]. C'est une forme de virtualisation récursive dans laquelle une fabrique pourrait être raffinée pour faire apparaître un niveau d'abstraction plus fin intégrant une autre topologie virtuelle. Outre l'intérêt de restreindre la complexité du modèle, cela offrirait un moyen de faire intervenir plusieurs acteurs dans le processus de définition du réseau virtuel (chacun responsable d'un niveau hiérarchique). Toujours sur les objects manipulés, nous souhaitons proposer une refonte des composants *Edge* et *Data Machine* afin de prendre en compte des fonctions de traitement sur les paquets, indépendamment de leur mise en œuvre technique, se rapprochant ainsi du paradigme SFC (*Service Function Chaining*) [HP15]. Cette fonction générique couvrirait donc différentes implémentations, que ce soit sur un contrôleur externe, ou dans le plan de données sur un matériel dédié ou reprogrammable, ou encore dans du logiciel sur un serveur de commodité.

— La deuxième évolution concerne les propriétés non fonctionnelles associées aux composants du modèle. Dans la proposition existante, il n'est pas possible d'exprimer une propriété non fonctionnelle sur un composant de type *edge* ou *data machine* (telle une exigence de débit) et, concernant la *fabric*, nous avions prévu un paramètre pour spécifier une exigence de QoS sur la politique de transport *carry*, mais ce dernier n'était spécifié que partiellement et n'a pas été utilisé par la suite. Il nous semble important de pouvoir définir ce type de propriétés et de les rajouter au modèle, à la fois sur les fonctions de traitement des flux, et aussi sur les politiques d'acheminement de la fabrique (par exemple de manière similaire à la spécification du contrat de *slice subnet* du lien NTN que nous avons présentée dans la section 4.4.1, et que nous pourrions généraliser sur les segments RAN, TN et DN).

Génération de tests et télémétrie

L'ajout croissant de logiciels pour le contrôle et la définition même du réseau amène un défi fondamental : pouvoir vérifier les programmes pour prévenir les erreurs et effets indésirables sur le réseau. La vérification de programmes de contrôle ou du plan de données dans un contexte SDN fait l'objet de nombreux travaux de recherche depuis une dizaine d'années, par exemple Veriflow [Khu+12], NetKat [And+14], Cocoon [Ryz+17], ou encore p4v [Liu+18] pour n'en citer que quelques-uns. Nous nous sommes nous-mêmes intéressés à certaines approches de vérification décrites dans le chapitre 2. Outre les approches formelles de vérification, il existe d'autres approches basées sur la génération de tests que nous avons évoquées dans la section 2.3.3. Ces approches nous intéressent particulièrement à plusieurs titres.

Tout d'abord, un manque que nous pouvons souligner sur la vérification utilisant des méthodes formelles en phase de conception est lié à la portée de la vérification. Celle-ci se limite au programme en lui-même, voire à un sous-ensemble de propriétés du programme, sans prendre en compte l'environnement d'exécution (le *runtime*). De plus, même si le modèle formel peut prouver la correction d'un programme déployé sur plusieurs équipements, le périmètre reste limité au domaine administratif géré par l'opérateur. Il n'est donc pas possible de vérifier une propriété de bout en bout, ou sur un segment réseau de transit géré par un autre opérateur. C'est pourquoi, de manière complémentaire aux méthodes formelles, nous souhaitons continuer à nous intéresser aux méthodes basées sur des tests, et en particulier à la génération de paquets de tests [Zen+12 ; LVB14 ; Nöt+18]. La possibilité de générer, mesurer et analyser des paquets de tests nous permet de capturer des propriétés sur tout l'environnement d'exécution du programme réseau, dont l'état courant du système et les interactions éventuelles avec les autres flux.

De plus, une autre raison qui nous pousse à nous intéresser à la génération de tests, est le succès des méthodes de développement piloté par les tests (ou TDD, *Test-Driven Development*) rencontré ces dernières années dans le domaine du logiciel. Ces méthodes consistent à écrire des tests dès la conception du programme, de façon progressive, en alignant chaque test à une exigence fonctionnelle (par opposition à la création des tests après la phase de développement) [Bec03]. Outre le fait de pouvoir détecter une erreur rapidement dans le processus de développement, un des avantages du TDD est d'encourager le développeur a bien préciser le besoin et le comportement souhaité du programme. Plus nous ajouterons de logiciels qui piloteront nos infrastructures système et réseau, plus il nous semble important de s'appuyer sur des méthodes existantes qui ont été éprouvées dans le monde de l'ingénierie logicielle.

Enfin, dans le cadre des travaux sur le langage P4, la télémétrie constitue une application phare de la programmation du plan de données qui s'est concrétisée par la spécification INT (*In-band Network Telemetry*) [INT20]. L'INT fait référence à un paradigme de supervision réseau qui consiste à faire rapporter des éléments d'information sur l'état du réseau par le plan de données, c'est-à-dire par les paquets des flux transitant sur le réseau. Cette collecte s'opère donc sans l'intervention directe du plan de gestion. Sa mise en œuvre et son utilisation peuvent permettre de répondre à des questions difficiles dans le domaine de la supervision telles que : par quel chemin est passé ce paquet ? quelle est la latence introduite par chaque nœud du chemin ? par quelle règle le paquet a-t-il été traité ? etc. Les informations de télémétrie peuvent être insérées dans des paquets de flux dédiés aux mesures, ou directement dans l'en-tête des paquets du trafic des utilisateurs. Nous souhaitons explorer le couplage d'une approche INT avec de la génération automatique de tests. Un paquet du plan de données n'ayant, par définition, pas de frontière entre un point A et un point B (dans le sens où il peut parcourir plusieurs systèmes autonomes, contrairement aux plans de contrôle et de gestion qui nécessitent des protocoles ou interfaces spécifiques pour assurer une intégration multi-domaines), il nous paraît être le bon vecteur pour vérifier certaines exigences de bout en bout. Par exemple, l'exigence d'acheminer du trafic par un chemin spécifique pourrait se traduire par la génération automatique d'un cas de test qui se chargerait d'insérer dans les paquets du flux un en-tête INT approprié (intégrant les bonnes méta-données) qui serait mis à jour lors de son parcours et vérifié à la réception. Certains travaux ont déjà proposé ce type d'approche (par ex. [LVB14]) avec des paquets de sonde forgés selon le besoin, mais l'intérêt dans notre démarche est de pouvoir s'appuyer sur la spécification INT pour insérer les instructions de télémétrie au plus près des flux des utilisateurs. Un autre exemple, lié à nos contributions sur le *slicing* 5G traversant un lien satellite (cf. section 4.4), serait de pouvoir vérifier en temps réel le respect des exigences de QoS sur ce lien, voire de pouvoir tester en parallèle plusieurs flux issus de différentes *slices* pour apporter la garantie d'un traitement différencié en fonction des *slices*.

Bibliographie

[3GP21a] 3GPP. *Management and orchestration ; 5G Network Resource Model (NRM) ; Stage 2 and stage 3*. Technical specification (TS) 28.541. Version 17.5.0. 3rd Generation Partnership Project (3GPP), 2021 (cf. p. 52).

[3GP21b] 3GPP. *System architecture for the 5G System (5GS)*. Technical specification (TS) 23.501. Version 17.3.0. 3rd Generation Partnership Project (3GPP), 2021 (cf. p. 49, 51, 53, 61).

[Afo+18a] Ibrahim Afolabi, Tarik Taleb, Konstantinos Samdanis, Adlen Ksentini et Hannu Flinck. « Network Slicing and Softwarization : A Survey on Principles, Enabling Technologies, and Solutions ». In : *IEEE Communications Surveys & Tutorials* 20.3 (2018), p. 2429-2453. doi : 10.1109/COMST.2018.2815638 (cf. p. 50).

[Afo+18b] Ibrahim Afolabi, Tarik Taleb, Konstantinos Samdanis, Adlen Ksentini et Hannu Flinck. « Network Slicing and Softwarization : A Survey on Principles, Enabling Technologies, and Solutions ». In : *IEEE Communications Surveys & Tutorials* 20.3 (2018), p. 2429-2453. doi : 10.1109/COMST.2018.2815638 (cf. p. 65).

[Aga+16] George Agapiou, Ramon Ferrús, Harilaos Koumaras, Oriol Sallent Roig, Tinku Rasheed, Nicolas Kuhn et Toufik Ahmed. « SDN and NFV for satellite infrastructures ». In : *IEICE Information and Communication Technology Forum (IEICE ICTF 2016)*. 2016 (cf. p. 51).

[Air17] AirNet. *A virtual network control language built on top of an Edge-Fabric abstraction model*. https://github.com/sdnhype/airnet, Last accessed on August 2022. 2017 (cf. p. 39).

[Aku+13a] Ludi Akue, Emmanuel Lavinal, Thierry Desprats et Michelle Sibilla. « Integrating an Online Configuration Checker with Existing Management Systems : Application to CIM/WBEM Environments ». In : *International DMTF Academic Alliance Workshop on Systems and Virtualization Management : Standards and New Technologies (SVM 2013)*. Oct. 2013 (cf. p. 9, 18, 19).

[Aku+13b] Ludi Akue, Emmanuel Lavinal, Thierry Desprats et Michelle Sibilla. « Runtime Configuration Validation for Self-configurable Systems (short paper) ». In : *IFIP/IEEE International Symposium on Integrated Network Management (IM 2013)*. Ghent, Belgium : IEEE, mai 2013 (cf. p. 9).

[Aku+14] Ludi Akue, Emmanuel Lavinal, Thierry Desprats et Michelle Sibilla. « Generic and adaptable online configuration verification for complex networked systems ». In : *International Journal On Advances in Systems and Measurements* vol. 7.n° 1 (2014), pp. 168-178 (cf. p. 9).

[Aku14] Ludi Akue. « Un cadriciel pour la vérification en ligne, générique, flexible et évolutive de configurations de systèmes communicants complexes ». (Soutenance le 12/02/2014). Thèse de doctorat. Toulouse, France : Université de Toulouse III - Paul Sabatier, fév. 2014 (cf. p. 14, 17).

[ALS10] Ludi Akue, Emmanuel Lavinal et Michelle Sibilla. « Towards a validation framework for dynamic reconfiguration (short paper) ». In : *IEEE/IFIP International Conference on Network and Service Management (CNSM 2010)*. Niagara Falls, Canada : IEEE, oct. 2010, p. 314-317 (cf. p. 9).

[And+05] T. Anderson, L. Peterson, S. Shenker et J. Turner. « Overcoming the Internet impasse through virtualization ». In : *Computer* 38.4 (2005), p. 34-41. doi : 10.1109/MC.2005.136 (cf. p. 8).

[And+14] Carolyn Jane Anderson, Nate Foster, Arjun Guha, Jean-Baptiste Jeannin, Dexter Kozen, Cole Schlesinger et David Walker. « NetKAT : Semantic Foundations for Networks ». In : *SIGPLAN Not.* 49.1 (jan. 2014), p. 113-126. issn : 0362-1340. doi : 10.1145/2578855.2535862 (cf. p. 31, 65).

[Aou+16a] Messaoud Aouadj, Emmanuel Lavinal, Thierry Desprats et Michelle Sibilla. « AirNet : The edge-fabric model as a virtual control plane ». In : *IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), SWFAN 16 : International Workshop on Software-Driven Flexible and Agile Networking*. 2016. doi : 10.1109/INFCOMW.2016.7562059 (cf. p. 9, 44).

[Aou+16b] Messaoud Aouadj, Emmanuel Lavinal, Thierry Desprats et Michelle Sibilla. « Composing data and control functions to ease virtual networks programmability ». In : *IEEE/IFIP Network Operations and Management Symposium (NOMS 2016) : Mini-Conference*. IEEE, 2016. doi : 10.1109/NOMS.2016.7502844 (cf. p. 9).

[Aou+17] Messaoud Aouadj, Emmanuel Lavinal, Thierry Desprats et Michelle Sibilla. « AirNet : An Edge-Fabric abstraction model to manage software-defined networks ». In : *International Journal of Network Management* 27.6 (2017). doi : 10.1002/nem.1983 (cf. p. 9, 44).

[Aou16] Messaoud Aouadj. « AirNet : le modèle de virtualisation "Edge-Fabric" comme plan de contrôle pour les réseaux programmables ». (Soutenance le 08/11/2016). Thèse de doctorat. Toulouse, France : Université de Toulouse III - Paul Sabatier, nov. 2016 (cf. p. 31, 37).

[Arm+09] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy H. Katz, Andrew Konwinski, Gunho Lee, David A. Patterson, Ariel Rabkin, Ion Stoica et Matei Zaharia. *Above the Clouds : A Berkeley View of Cloud Computing*. Technical Report No. UCB/EECS-2009-28. University of California at Berkeley, 2009 (cf. p. 8).

[BAM09] Theophilus Benson, Aditya Akella et David Maltz. « Unraveling the Complexity of Network Management ». In : *Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation*. NSDI'09. USENIX Association, 2009, p. 335-348 (cf. p. 7).

[Bec03] Kent Beck. *Test Driven Development : By Example*. Pearson Education, 2003 (cf. p. 66).

[Bos+14] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese et David Walker. « P4 : Programming Protocol-Independent Packet Processors ». In : *SIGCOMM Comput. Commun. Rev.* 44.3 (juill. 2014), p. 87-95. doi : 10.1145/2656877.2656890 (cf. p. 8).

[Bro+17] Michael M. Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam et Pierre Vandergheynst. « Geometric Deep Learning : Going beyond Euclidean data ». In : *IEEE Signal Processing Magazine* 34.4 (2017), p. 18-42. doi : 10.1109/MSP.2017.2693418 (cf. p. 26).

[Cam+05] G. Camarillo, E. Burger, H. Schulzrinne et A. van Wijk. *Transcoding Services Invocation in the Session Initiation Protocol (SIP) Using Third Party Call Control (3pcc)*. IETF RFC 4117. Juin 2005 (cf. p. 49).

[Cas+10] Martin Casado, Teemu Koponen, Rajiv Ramanathan et Scott Shenker. « Virtualizing the Network Forwarding Plane ». In : *Proceedings of the Workshop on Programmable Routers for Extensible Services of Tomorrow*. PRESTO'10. Philadelphia, Pennsylvania : ACM, 2010, 8 :1-8 :6. isbn : 978-1-4503-0467-2. doi : 10.1145/1921151.1921162 (cf. p. 32).

[Cas+12] Martin Casado, Teemu Koponen, Scott Shenker et Amin Tootoonchian. « Fabric : A Retrospective on Evolving SDN ». In : *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*. HotSDN'12. Helsinki, Finland : ACM, 2012, p. 85-90. isbn : 978-1-4503-1477-0. doi : 10.1145/2342441.2342459 (cf. p. 33).

[CB10] N.M. Mosharaf Kabir Chowdhury et Raouf Boutaba. « A survey of network virtualization ». In : *Computer Networks* 54.5 (2010), p. 862-876. doi : https://doi.org/10.1016/j.comnet.2009.10.017 (cf. p. 8).

[CCS12] A. Cutler, D.R. Cutler et J.R. Stevens. « Random Forests ». In : *Ensemble Machine Learning : Methods and Applications*. Sous la dir. de C. Zhang et Y. Ma. Boston, MA : Springer US, 2012, p. 157-175 (cf. p. 24).

[CIM12] CIM. *Common Information Model (CIM) Metamodel*. Specification. Distributed Management Task Force (DMTF), 2012 (cf. p. 16).

[DJ07] Thomas Delaet et Wouter Joosen. « PoDIM : A Language for High-Level Configuration Management ». In : *21th Large Installation System Administration Conference (LISA)*. USENIX, nov. 2007 (cf. p. 13).

[Dri+21a] Youssouf Drif, Emmanuel Chaput, Emmanuel Lavinal, Pascal Berthou, Boris Tiomela Jou, Olivier Gremillet et Fabrice Arnal. « An Extensible Network Slicing Framework for Satellite Integration into 5G ». In : *International Journal of Satellite Communications and Networking* 39.4 (juill. 2021), p. 339-357. doi : 10.1002/sat.1387 (cf. p. 9, 52).

[Dri+21b] Youssouf Drif, Emmanuel Lavinal, Emmanuel Chaput, Pascal Berthou, Boris Tiomela Jou, Olivier Grémillet et Fabrice Arnal. « Slice Aware Non Terrestrial Networks ». In : *46th IEEE Conference on Local Computer Networks (LCN 2021)*. Oct. 2021. doi : 10.1109/LCN52139.2021.9524938 (cf. p. 9, 53).

[Dri22a] Youssouf Drif. *COMETS, COntainer MEsh for neTwork Slicing*. https://github.com/shynuu/slice-aware-ntn, Last accessed on August 2022. 2022 (cf. p. 55, 56).

[Dri22b] Youssouf Drif. « Le « network slicing » dans les réseaux 5G : intégration d'un lien satellite avec continuité de service ». (Soutenance le 13/07/2022). Thèse de doctorat. Toulouse, France : Institut National Polytechnique de Toulouse (INP Toulouse), juill. 2022 (cf. p. 47, 51, 56).

[Dri22c] Youssouf Drif. *Trunks, a simple satellite network simulator. v2.0.1*. https://github.com/shynuu/trunks, Last accessed on August 2022. 2022 (cf. p. 55).

[ETS12] ETSI. *Network Functions Virtualisation – An Introduction, Benefits, Enablers, Challenges & Call for Action*. NFV Introductory White Paper. Oct. 2012 (cf. p. 8).

[ETS14a] ETSI. *Network Functions Virtualisation (NFV) ; Architectural Framework*. ETSI GS NFV 002 v1.2.1 (2014-12). 2014 (cf. p. 8).

[ETS14b] ETSI. *Network Functions Virtualisation (NFV) ; Management and Orchestration*. ETSI GS NFV-MAN 001 v1.1.1 (2014-12). 2014 (cf. p. 8).

[ETS20] ETSI. *Multi-access Edge Computing (MEC) - MEC 5G Integration*. ETSI GR MEC 031 v2.1.1 (2020-10). 2020 (cf. p. 62, 63).

[ETS22a] ETSI. *Multi-access Edge Computing (MEC) - Framework and Reference Architecture*. ETSI GS MEC 003 v3.1.1 (2022-03). 2022 (cf. p. 61).

[ETS22b] ETSI. *Multi-access Edge Computing (MEC) - Phase 2 : Use Cases and Requirements*. ETSI GS MEC 002 v2.2.1 (2022-01). 2022 (cf. p. 61).

[Fau18] FAUCET. *Open source SDN Controller for production networks*. https://faucet.nz/, Last accessed on January 2022. 2018 (cf. p. 21).

[Fer+21] Andrew D. FERGUSON et al. « Orion : Google's Software-Defined Networking Control Plane ». In : *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*. USENIX Association, avr. 2021, p. 83-98 (cf. p. 8).

[FK18] Pantelis A. FRANGOUDIS et Adlen KSENTINI. « Service migration versus service replication in Multi-access Edge Computing ». In : *14th International Wireless Communications & Mobile Computing Conference (IWCMC)*. 2018. DOI : 10.1109/IWCMC.2018.8450284 (cf. p. 62).

[For+20] Alan FORD, Costin RAICIU, Mark J. HANDLEY, Olivier BONAVENTURE et Christoph PAASCH. *TCP Extensions for Multipath Operation with Multiple Addresses*. RFC 8684. Mars 2020. DOI : 10.17487/RFC8684 (cf. p. 63).

[Fos+10] Nate FOSTER, Michael J. FREEDMAN, Rob HARRISON, Jennifer REXFORD, Matthew L. MEOLA et David WALKER. « Frenetic : A High-level Language for OpenFlow Networks ». In : *Proceedings of the Workshop on Programmable Routers for Extensible Services of Tomorrow*. PRESTO '10. Philadelphia, Pennsylvania : ACM, 2010, 6 :1-6 :6. ISBN : 978-1-4503-0467-2. DOI : 10.1145/1921151.1921160 (cf. p. 31).

[Fou+17] Xenofon FOUKAS, Georgios PATOUNAS, Ahmed ELMOKASHFI et Mahesh K. MARINA. « Network Slicing in 5G : Survey and Challenges ». In : *IEEE Communications Magazine* 55.5 (2017), p. 94-100. DOI : 10.1109/MCOM.2017.1600951 (cf. p. 50).

[Fre22] FREE5GC. *Open source 5G core network base on 3GPP R15. v3.2.1*. https://github.com/free5gc/free5gc, Last accessed on August 2022. 2022 (cf. p. 55).

[GA21] Daniele GRATTAROLA et Cesare ALIPPI. « Graph Neural Networks in TensorFlow and Keras with Spektral [Application Notes] ». In : *IEEE Computational Intelligence Magazine* 16.1 (2021), p. 99-106. DOI : 10.1109/MCI.2020.3039072 (cf. p. 28).

[Ghe06] Luc De GHEIN. *MPLS Fundamentals*. 1st. Cisco Press, 2006 (cf. p. 23, 33).

[Gho+11] Ali GHODSI, Scott SHENKER, Teemu KOPONEN, Ankit SINGLA, Barath RAGHAVAN et James WILCOX. « Intelligent Design Enables Architectural Evolution ». In : *Proceedings of the 10th ACM Workshop on Hot Topics in Networks*. HotNets-X. 2011. DOI : 10.1145/2070562.2070565 (cf. p. 8).

[GN06] Xiaohui GU et Klara NAHRSTEDT. « On Composing Stream Applications in Peer-to-Peer Environments ». In : *IEEE Transactions on Parallel and Distributed Systems* 17.8 (août 2006), p. 824-837 (cf. p. 48).

[Gol+09] Patrick GOLDSACK, Julio GUIJARRO, Steve LOUGHRAN, Alistair COLES, Andrew FARRELL, Antonio LAIN, Paul MURRAY et Peter TOFT. « The SmartFrog Configuration Management Framework ». In : *SIGOPS Operating Systems Review* 43.1 (jan. 2009), p. 16-25. DOI : 10.1145/1496909.1496915 (cf. p. 16).

[Gud+08] Natasha GUDE, Teemu KOPONEN, Justin PETTIT, Ben PFAFF, Martin CASADO, Nick MCKEOWN et Scott SHENKER. « NOX : Towards an Operating System for Networks ». In : *SIGCOMM Comput. Commun. Rev.* 38.3 (juill. 2008). DOI : 10.1145/1384609.1384625 (cf. p. 31).

[Gün22] Ali GÜNGÖR. *UERANSIM, open source 5G UE and RAN (gNodeB) implementation. v3.2.6.* https://github.com/aligungr/UERANSIM, Last accessed on August 2022. 2022 (cf. p. 55).

[Gup+14] Arpit GUPTA, Laurent VANBEVER, Muhammad SHAHBAZ, Sean Patrick DONOVAN, Brandon SCHLINKER, Nick FEAMSTER, Jennifer REXFORD, Scott SHENKER, Russ CLARK et Ethan KATZ-BASSETT. « SDX : A Software Defined Internet Exchange ». In : *Proceedings of the 2014 ACM Conference on SIGCOMM*. SIGCOMM '14. 2014, p. 579-580. DOI : 10.1145/2619239.2631473 (cf. p. 8).

[Gut+12] Stephen GUTZ, Alec STORY, Cole SCHLESINGER et Nate FOSTER. « Splendid Isolation : A Slice Abstraction for Software-defined Networks ». In : *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*. HotSDN '12. Helsinki, Finland : ACM, 2012, p. 79-84. ISBN : 978-1-4503-1477-0. DOI : 10.1145/2342441.2342458 (cf. p. 31, 32).

[Har+06] Frank HARTUNG, Norbert NIEBERT, Andreas SCHIEDER, René REMBARZ, Stefan SCHMID et Lars EGGERT. « Advances in Network-Supported Media Delivery in Next-Generation Mobile Systems ». In : *IEEE Communications Magazine* 44.8 (août 2006), p. 82-89 (cf. p. 48).

[Hin+09] Timothy L. HINRICHS, Natasha S. GUDE, Martin CASADO, John C. MITCHELL et Scott SHENKER. « Practical Declarative Network Management ». In : *Proceedings of the 1st ACM Workshop on Research on Enterprise Networking*. WREN'09. Barcelona, Spain : ACM, 2009, p. 1-10. ISBN : 978-1-60558-443-0. DOI : 10.1145/1592681.1592683 (cf. p. 31).

[HLS07] Hanane HUYNH, Emmanuel LAVINAL et Noëmie SIMONI. « A Dynamic QoS Management for Next Generation of Services ». In : *Third International Conference on Autonomic and Autonomous Systems (ICAS'07)*. 2007 (cf. p. 9, 48).

[Hor01] Paul HORN. *Autonomic Computing : IBM's Perspective on the State of Information Technology*. IBM White Paper. 2001 (cf. p. 7).

[HP15] Joel M. HALPERN et Carlos PIGNATARO. *Service Function Chaining (SFC) Architecture*. RFC 7665. Oct. 2015. DOI : 10.17487/RFC7665 (cf. p. 59, 65).

[INT20] INT. *In-band Network Telemetry (INT) Dataplane Specification, v2.1*. P4.org Applications Working Group. Mai 2020 (cf. p. 66).

[IRE16] IREHDO2. *Specification of a requirement engineering methodology for networks. F. Barrere, A. Benzekri, B.T. Sravani, T. Desprats, R. Laborde, F. Lavaud, E. Lavinal, M. Sibilla, S. Wazan.* IREHDO2 Project. Deliverable D.1.2.7.4. IRIT/UPS, juill. 2016 (cf. p. 9).

[IRE18] IREHDO2. *From specification of network security requirements to verification of network configurations. F. Barrere, A. Benzekri, B.T. Sravani, T. Desprats, S. Kallel, R. Laborde, E. Lavinal, M. Sibilla, S. Wazan.* IREHDO2 Project. Deliverable D.5.2.9.2. IRIT/UPS, juin 2018 (cf. p. 9, 21).

[IT21] Jana IYENGAR et Martin THOMSON. *QUIC : A UDP-Based Multiplexed and Secure Transport*. RFC 9000. Mai 2021. DOI : 10.17487/RFC9000 (cf. p. 63).

[Jai+13] Sushant JAIN et al. « B4 : Experience with a Globally-Deployed Software Defined Wan ». In : *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*. SIGCOMM '13. 2013, p. 3-14. DOI : 10.1145/2486001.2486019 (cf. p. 8).

[Jou+18] Boris Tiomela Jou, Oriol Vidal, Joe Cahill, Fabrice Arnal, Jean-Michel Houssin, Mael Boutin et Duy Kha Chau. « Architecture Options for Satellite Integration into 5G Networks ». In : *2018 European Conference on Networks and Communications (EuCNC)*. 2018. doi : 10.1109/EuCNC.2018.8442436 (cf. p. 51).

[Kan06] Luke Kanies. « Puppet : Next-generation configuration management ». In : *The USENIX Magazine* 31.1 (fév. 2006) (cf. p. 16).

[KC03] J. O. Kephart et D. M. Chess. « The vision of autonomic computing ». In : *IEEE Computer* 36.1 (2003), p. 41-50 (cf. p. 7, 14).

[Ker20] Keras. *Keras : Simple, Flexible, Powerful.* https://keras.io/, Last accessed on January 2022. 2020 (cf. p. 24).

[Kha+20] Latif U. Khan, Ibrar Yaqoob, Nguyen H. Tran, Zhu Han et Choong Seon Hong. « Network Slicing : Recent Advances, Taxonomy, Requirements, and Open Research Challenges ». In : *IEEE Access* 8 (2020), p. 36009-36028. doi : 10.1109/ACCESS.2020.2975072 (cf. p. 50).

[Khu+12] Ahmed Khurshid, Wenxuan Zhou, Matthew Caesar et P. Brighten Godfrey. « VeriFlow : Verifying Network-Wide Invariants in Real Time ». In : *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*. HotSDN '12. 2012, p. 49-54. doi : 10.1145/2342441.2342452 (cf. p. 65).

[Kim+22] Taejin Kim, Sandesh Dhawaskar Sathyanarayana, Siqi Chen, Youngbin Im, Xiaoxi Zhang, Sangtae Ha et Carlee Joe-Wong. « MoDEMS : Optimizing Edge Computing Migrations for User Mobility ». In : *IEEE INFOCOM 2022 - IEEE Conference on Computer Communications*. 2022. doi : 10.1109/INFOCOM48880.2022.9796680 (cf. p. 63).

[KR10] Eric Keller et Jennifer Rexford. « The "Platform As a Service" Model for Networking ». In : *Proceedings of the 2010 Internet Network Management Conference on Research on Enterprise Networking*. INM/WREN'10. San Jose, CA : USENIX Association, 2010, p. 4-4. url : http://dl.acm.org/citation.cfm?id=1863133.1863137 (cf. p. 32).

[Kre+15] Diego Kreutz, Fernando M. V. Ramos, Paulo Esteves Veríssimo, Christian Esteve Rothenberg, Siamak Azodolmolky et Steve Uhlig. « Software-Defined Networking : A Comprehensive Survey ». In : *Proceedings of the IEEE* 103.1 (2015), p. 14-76. doi : 10.1109/JPROC.2014.2371999 (cf. p. 8, 32).

[KW17] Thomas N. Kipf et Max Welling. *Semi-Supervised Classification with Graph Convolutional Networks*. arXiv 1609.02907. 2017. arXiv : 1609.02907 [cs.LG] (cf. p. 27).

[Lab+19] Romain Laborde, Sravani Teja Bulusu, Ahmad Samer Wazan, François Barrère et Abdelmalek Benzekri. « Logic-Based Methodology to Help Security Architects in Eliciting High-Level Network Security Requirements ». In : *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*. SAC '19. Limassol, Cyprus, 2019 (cf. p. 20).

[Lan22] P4 Language Consortium. *$P4_{16}$ Language Specification*. Version 1.2.3. Juill. 2022 (cf. p. 8).

[Lav+09] Emmanuel Lavinal, Noemie Simoni, Meng Song et Bertrand Mathieu. « A next-generation service overlay architecture ». In : *Annals of Telecommunications* 64.3-4 (2009). Springer, p. 175-185. doi : 10.1007/s12243-008-0082-x (cf. p. 9).

[LDR06] Emmanuel Lavinal, Thierry Desprats et Yves Raynaud. « A generic multi-agent conceptual framework towards self-management ». In : *IEEE/IFIP International Network Operations and Management Symposium (NOMS 2006)*. Avr. 2006, p. 394-403 (cf. p. 7).

[LHM10] Bob Lantz, Brandon Heller et Nick McKeown. « A Network in a Laptop : Rapid Prototyping for Software-defined Networks ». In : *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*. Hotnets-IX. Monterey, California : ACM, 2010, 19 :1-19 :6. isbn : 978-1-4503-0409-2. doi : 10.1145/1868447.1868466 (cf. p. 43).

[Liu+18] Jed Liu, William Hallahan, Cole Schlesinger, Milad Sharif, Jeongkeun Lee, Robert Soulé, Han Wang, Călin Caşcaval, Nick McKeown et Nate Foster. « P4v : Practical Verification for Programmable Data Planes ». In : *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*. SIGCOMM '18. 2018, p. 490-503. doi : 10.1145/3230543.3230582 (cf. p. 65).

[LQL12] Akash Lal, Shaz Qadeer et Shuvendu Lahiri. « Corral : A Solver for Reachability Modulo Theories ». In : *Computer-Aided Verification (CAV)*. Juill. 2012. url : https://www.microsoft.com/en-us/research/publication/corral-a-solver-for-reachability-modulo-theories-2/ (cf. p. 21).

[LS08] Emmanuel Lavinal et Noëmie Simoni. « Dynamic and Adaptive Composition of SIP-Based Services ». In : *IEEE International Conference on Communications (ICC 2008)*. 2008 (cf. p. 9, 48, 49).

[LVB14] David Lebrun, Stefano Vissicchio et Olivier Bonaventure. « Towards test-driven software defined networking ». In : *2014 IEEE Network Operations and Management Symposium (NOMS)*. 2014. doi : 10.1109/NOMS.2014.6838225 (cf. p. 21, 66).

[Mac+21] Robert MacDavid, Carmelo Cascone, Pingping Lin, Badhrinath Padmanabhan, Ajay ThakuR, Larry Peterson, Jennifer Rexford et Oguz Sunay. « A P4-Based 5G User Plane Function ». In : *Proceedings of the ACM SIGCOMM Symposium on SDN Research (SOSR)*. SOSR '21. 2021, p. 162-168. doi : 10.1145/3482898.3483358 (cf. p. 62).

[MB17] Pavel Mach et Zdenek Becvar. « Mobile Edge Computing : A Survey on Architecture and Computation Offloading ». In : *IEEE Communications Surveys & Tutorials* 19.3 (2017), p. 1628-1656. doi : 10.1109/COMST.2017.2682318 (cf. p. 61).

[McK+08] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker et Jonathan Turner. « OpenFlow : Enabling Innovation in Campus Networks ». In : *SIGCOMM Comput. Commun. Rev.* 38.2 (mars 2008), p. 69-74. doi : 10.1145/1355734.1355746 (cf. p. 8).

[McK21] Nick McKeown. *The Network will be programmed by many, operated by a few*. The Networking Channel, https://youtu.be/-cvNw1g5kJc, Last accessed on August 2022. Mars 2021 (cf. p. 64).

[Med+17] Ahmed M. Medhat, Tarik Taleb, Asma Elmangoush, Giuseppe A. Carella, Stefan Covaci et Thomas Magedanz. « Service Function Chaining in Next Generation Networks : State of the Art and Research Challenges ». In : *IEEE Communications Magazine* 55.2 (2017). doi : 10.1109/MCOM.2016.1600219RP (cf. p. 39).

[Mij+16] Rashid Mijumbi, Joan Serrat, Juan-Luis Gorricho, Niels Bouten, Filip De Turck et Raouf Boutaba. « Network Function Virtualization : State-of-the-Art and Research Challenges ». In : *IEEE Communications Surveys & Tutorials* 18.1 (2016), p. 236-262. doi : 10.1109/COMST.2015.2477041 (cf. p. 8).

[Min15] MININET. *Emulator for rapid prototyping of Software Defined Networks*. http://mininet.org, Last accessed on September 2021. 2015 (cf. p. 21, 43).

[MIR22] Atri MUKHOPADHYAY, George IOSIFIDIS et Marco RUFFINI. « Migration-Aware Network Services With Edge Computing ». In : *IEEE Transactions on Network and Service Management* 19.2 (2022), p. 1458-1471. DOI : 10.1109/TNSM.2021.3139857 (cf. p. 63).

[MLF20] El-Heithem MOHAMMEDI, Emmanuel LAVINAL et Guillaume FLEURY. « Configuration faults detection in IP Virtual Private Networks based on machine learning ». In : *3rd International Conference on Machine Learning for Networking (MLN 2020)*. Nov. 2020 (cf. p. 9, 25).

[MLF22] El-Heithem MOHAMMEDI, Emmanuel LAVINAL et Guillaume FLEURY. « Detecting and locating configuration errors in IP VPNs with Graph Neural Networks ». In : *34th IEEE/IFIP Network Operations and Management Symposium (NOMS 2022). Workshop on Analytics for Network and Service Management (ANNET)*. Avr. 2022. DOI : 10.1109/NOMS54207.2022.9789800 (cf. p. 9, 29).

[Mon+12] Christopher MONSANTO, Nate FOSTER, Rob HARRISON et David WALKER. « A Compiler and Run-time System for Network Programming Languages ». In : *SIGPLAN Not.* 47.1 (jan. 2012), p. 217-230. ISSN : 0362-1340. DOI : 10.1145/2103621.2103685 (cf. p. 31).

[Mon+13] Christopher MONSANTO, Joshua REICH, Nate FOSTER, Jennifer REXFORD et David WALKER. « Composing Software-defined Networks ». In : *Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation*. nsdi'13. Lombard, IL : USENIX Association, 2013, p. 1-14. URL : http://dl.acm.org/citation.cfm?id=2482626.2482629 (cf. p. 31, 32).

[NGM16] NGMN. *Description of Network Slicing Concept*. NGMN 5G Project Requirements & Architecture, Version 1.0. Next Generations Mobile Networks, 2016 (cf. p. 50).

[NIS08] NIST. *JSIP : Java SIP specification Reference Implementation. v1.2*. https://github.com/usnistgov/jsip, Last accessed on August 2022. 2008 (cf. p. 49).

[Nok19] NOKIA. *5G network slicing : automation, assurance and optimization of 5G transport slices*. https://youtu.be/beM1FYnOBqI, Last accessed on June 2022. 2019 (cf. p. 50).

[Nöt+18] Andres NÖTZLI, Jehandad KHAN, Andy FINGERHUT, Clark BARRETT et Peter ATHANAS. « P4pktgen : Automated Test Case Generation for P4 Programs ». In : *Proceedings of the Symposium on SDN Research*. SOSR '18. 2018. DOI : 10.1145/3185467.3185497 (cf. p. 66).

[ONF12] ONF. *OpenFlow Switch Specification Version 1.3.0*. Open Networking Foundation. Juin 2012 (cf. p. 43).

[Pfa+15] Ben PFAFF, Justin PETTIT, Teemu KOPONEN, Ethan J. JACKSON, Andy ZHOU, Jarno RAJAHALME, Jesse GROSS, Alex WANG, Jonathan STRINGER, Pravin SHELAR, Keith AMIDON et Martin CASADO. « The Design and Implementation of Open VSwitch ». In : *Proceedings of the 12th USENIX Conference on Networked Systems Design and Implementation*. NSDI'15. Oakland, CA : USENIX Association, 2015 (cf. p. 43).

[POX13] POX. *The POX network software platform*. https://github.com/noxrepo/pox, Last accessed on September 2021. 2013 (cf. p. 40).

[Pro21] Project PRONTO. *Creating an open source end-to-end programmable and verifiable network*. https://prontoproject.org/, Last accessed on August 2022. 2021 (cf. p. 63, 64).

[Rag+12] Barath Raghavan, Martin Casado, Teemu Koponen, Sylvia Ratnasamy, Ali Ghodsi et Scott Shenker. « Software-Defined Internet Architecture : Decoupling Architecture from Infrastructure ». In : *Proceedings of the 11th ACM Workshop on Hot Topics in Networks*. HotNets-XI. 2012, p. 43-48. doi : 10.1145/2390231.2390239 (cf. p. 8).

[Rei+13] Mark Reitblatt, Marco Canini, Arjun Guha et Nate Foster. « FatTire : Declarative Fault Tolerance for Software-defined Networks ». In : *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*. HotSDN '13. Hong Kong, China : ACM, 2013, p. 109-114. isbn : 978-1-4503-2178-5. doi : 10.1145/2491185.2491187 (cf. p. 31).

[RK03] Bhaskaran Raman et Randy H. Katz. « An architecture for highly available wide-area service composition ». In : *Computer Communication Journal* 26.15 (sept. 2003), p. 1727-1740 (cf. p. 48).

[RM14] L. Rokach et O. Maimon. *Data Mining With Decision Trees : Theory And Applications*. 2nd. World Scientific, 2014 (cf. p. 24).

[RN19] S. Russell et P. Norvig. *Artificial Intelligence : A Modern Approach*. 4th. Pearson, 2019 (cf. p. 24).

[Ros+02] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley et E. Schooler. *SIP : Session Initiation Protocol*. IETF RFC 3261. Juin 2002 (cf. p. 49).

[RR06] E. Rosen et Y. Rekhter. *BGP/MPLS IP Virtual Private Networks (VPNs)*. RFC 4364. RFC Editor, fév. 2006. url : https://www.rfc-editor.org/rfc/rfc4364.txt (cf. p. 22).

[Ryu14] Ryu. *Ryu component-based software defined networking framework*. https://ryu-sdn.org, Last accessed on September 2021. 2014 (cf. p. 40).

[Ryz+17] Leonid Ryzhyk, Nikolaj Bjørner, Marco Canini, Jean-Baptiste Jeannin, Cole Schlesinger, Douglas B. Terry et George Varghese. « Correct by Construction Networks Using Stepwise Refinement ». In : *Proceedings of the 14th USENIX Conference on Networked Systems Design and Implementation*. NSDI'17. Boston, MA, USA : USENIX Association, 2017 (cf. p. 20, 65).

[Sch15] Jürgen Schmidhuber. « Deep learning in neural networks : An overview ». In : *Neural Networks* 61 (2015), p. 85-117. issn : 0893-6080. doi : 10.1016/j.neunet.2014.09.003 (cf. p. 24).

[Sci20] Scikit-learn. *Scikit-learn : Machine Learning in Python*. https://scikit-learn.org/, Last accessed on January 2022. 2020 (cf. p. 24).

[SCL05] John Sherwood, Andrew Clark et David Lynas. *Enterprise Security Architecture : A Business-Driven Approach*. CRC Press, 2005 (cf. p. 19).

[SD-21] Project SD-Fabric. *Open source full-stack programmable network fabric optimized for edge cloud, 5G, and industry 4.0 applications*. ONF, https://opennetworking.org/sd-fabric/, Last accessed on August 2022. 2021 (cf. p. 62).

[SK17] Martin Simonovsky et Nikos Komodakis. « Dynamic Edge-Conditioned Filters in Convolutional Neural Networks on Graphs ». In : *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2017)*. Juill. 2017 (cf. p. 28).

[SM18] Seema Sharma et Deepti Mehrotra. « Comparative Analysis of Multi-label Classification Algorithms ». In : *First International Conference on Secure Cyber Computing and Communication (ICSCCC)*. 2018. doi : 10.1109/ICSCCC.2018.8703285 (cf. p. 24).

[Sou+14] Robert Soulé, Shrutarshi Basu, Parisa Jalili Marandi, Fernando Pedone, Robert Kleinberg, Emin Gun Sirer et Nate Foster. « Merlin : A Language for Provisioning Network Resources ». In : *Proceedings of the 10th ACM International on Conference on Emerging Networking Experiments and Technologies*. CoNEXT '14. Sydney, Australia : ACM, 2014, p. 213-226. isbn : 978-1-4503-3279-8. doi : 10.1145/2674005.2674989 (cf. p. 31, 32).

[Tal+17] Tarik Taleb, Konstantinos Samdanis, Badr Mada, Hannu Flinck, Sunny Dutta et Dario Sabella. « On Multi-Access Edge Computing : A Survey of the Emerging 5G Network Edge Cloud Architecture and Orchestration ». In : *IEEE Communications Surveys & Tutorials* 19.3 (2017), p. 1657-1681. doi : 10.1109/COMST.2017.2705720 (cf. p. 61).

[Ten20] TensorFlow. *An end-to-end open source machine learning platform*. https://www.tensorflow.org/, Last accessed on January 2022. 2020 (cf. p. 24).

[VAH10] Andreas Voellmy, Ashish Agarwal et Paul Hudak. *Nettle : Functional Reactive Programming for OpenFlow Networks*. Rapp. tech. YALEU/DCS/RR-1431. Yale University, juill. 2010 (cf. p. 31).

[Ven+21] Pier Luigi Ventre, Stefano Salsano, Marco Polverini, Antonio Cianfrani, Ahmed Abdelsalam, Clarence Filsfils, Pablo Camarillo et Francois Clad. « Segment Routing : A Comprehensive Survey of Research Activities, Standardization Efforts, and Implementation Results ». In : *IEEE Communications Surveys & Tutorials* 23.1 (2021), p. 182-221. doi : 10.1109/COMST.2020.3036826 (cf. p. 63).

[VKF12] Andreas Voellmy, Hyojoon Kim et Nick Feamster. « Procera : A Language for High-level Reactive Network Control ». In : *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*. HotSDN '12. Helsinki, Finland : ACM, 2012, p. 43-48. isbn : 978-1-4503-1477-0. doi : 10.1145/2342441.2342451 (cf. p. 31).

[Voe+13] Andreas Voellmy, Junchang Wang, Y Richard Yang, Bryan Ford et Paul Hudak. « Maple : Simplifying SDN Programming Using Algorithmic Policies ». In : *SIGCOMM Comput. Commun. Rev.* 43.4 (août 2013), p. 87-98. issn : 0146-4833. doi : 10.1145/2534169.2486030 (cf. p. 32).

[Wu+21] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang et Philip S. Yu. « A Comprehensive Survey on Graph Neural Networks ». In : *IEEE Transactions on Neural Networks and Learning Systems* 32.1 (2021), p. 4-24. doi : 10.1109/TNNLS.2020.2978386 (cf. p. 26).

[Yan+21] Song Yang, Fan Li, Stojan Trajanovski, Ramin Yahyapour et Xiaoming Fu. « Recent Advances of Resource Allocation in Network Function Virtualization ». In : *IEEE Transactions on Parallel and Distributed Systems* 32.2 (2021), p. 295-314. doi : 10.1109/TPDS.2020.3017001 (cf. p. 65).

[YAN10] YANG. *M. Bjorklund. YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)*. RFC 6020. RFC Editor, oct. 2010. url : https://rfc-editor.org/rfc/rfc6020.txt (cf. p. 16).

[Yin+11] Zuoning Yin, Xiao Ma, Jing Zheng, Yuanyuan Zhou, Lakshmi N. Bairavasundaram et Shankar Pasupathy. « An Empirical Study on Configuration Errors in Commercial and Open Source Systems ». In : *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*. SOSP '11. Cascais, Portugal, 2011 (cf. p. 14).

[Zen+12] Hongyi ZENG, Peyman KAZEMIAN, George VARGHESE et Nick MCKEOWN. « Automatic Test Packet Generation ». In : *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies*. CoNEXT '12. Nice, France : Association for Computing Machinery, 2012, p. 241-252. ISBN : 9781450317757. DOI : 10.1145/2413176.2413205 (cf. p. 21, 66).

[Zho+20] Jie ZHOU, Ganqu CUI, Shengding HU, Zhengyan ZHANG, Cheng YANG, Zhiyuan LIU, Lifeng WANG, Changcheng LI et Maosong SUN. « Graph neural networks : A review of methods and applications ». In : *AI Open* 1 (2020), p. 57-81. ISSN : 2666-6510. DOI : 10.1016/j.aiopen.2021.01.001 (cf. p. 26).