UNIVERSITE BRETAGNE LOIRE MATHSTIC

UNIVERSITÉ DE RENNES 1

THESE DE DOCTORAT DE

L'UNIVERSITE DE RENNES 1
COMUE UNIVERSITE BRETAGNE LOIRE

ECOLE DOCTORALE N° 601
Mathématiques et Sciences et Technologies de l'Information et de la Communication

Par
Johan PELAY

« Garantir la cohérence applicative lors des changements de configuration réseau »

« Etude de cas sur la connectivité dans les Software Defined Infrastructures »

Thèse présentée et soutenue à Rennes, le 13 janvier 2020
Unité de recherche : Network Architecture, b<>com & DiverSE, IRISA
Thèse N° :

Rapporteurs avant soutenance :

Chantal Taconet	Maitre de conférences – HDR, Telecom Sud Paris
Pascal Lorenz	Professeur, université de Haute Alsace

Composition du Jury :

Président :	Jean-Louis Pazat	Professeur, INSA Rennes
Examinateurs :	Chantal Taconet	Maitre de conférences – HDR, Telecom Sud Paris
	Pascal Lorenz	Professeur, université de Haute Alsace
	Fabrice Guillemin	Directeur de recherche – HDR, Orange Labs
	Olivier Barais	Professeur, Université de Rennes 1
	Kevin Corre	Ingénieur de recherche, EXFO Solutions
Dir. de thèse :	Fabrice Guillemin	Directeur de recherche – HDR, Orange Labs
	Olivier Barais	Professeur, Université de Rennes 1

REMERCIEMENTS

La réalisation de cette thèse m'a permis d'échanger et de travailler avec beaucoup de monde. Je ne les remercierai pas tous dans ces quelques lignes, mais nombreux sont celles et ceux qui ont contribué à l'aboutissement de cette thèse.

Cependant je tiens particulièrement à remercier mes directeurs de thèse, Olivier Barais et Fabrice Guillemin qui m'ont guidé durant ces années. Leurs conseils, critiques et idées ont nourris mes travaux et ma découverte du monde de la recherche.

Je souhaite aussi présenter mes remerciements à Chantal Taconet, Pascal Lorenz, Kevin Corre et Jean-Louis Pazat pour avoir accepté de faire partie du jury et pour les réflexions que nous avons eu autour de mes travaux.

Je garde un très bon souvenir de mes années passées au sein de l'équipe Network Architecture de b<>com et de l'équipe DiverSE de l'IRISA. C'est un plaisir de travailler avec elles et eux.

Je suis aussi reconnaissant d'avoir pu bénéficier du soutien de b<>com lorsque j'étais en difficulté.

Je remercie également mes amis et ma famille, j'ai pu compter sur leur soutien indéfectible et plus particulièrement sur celui de Lucas, Maximilien et Julie qui m'ont supporté au quotidien.

Pour finir, ce manuscrit serait moins agréable à lire sans les relectures attentives de Françoise Ille.

TABLE DES MATIÈRES

TABLE DES FIGURES

GLOSSAIRE

** **

EPC : Evolved Packet Core : cœurs de réseau mobile

ETSI : Institut européen des normes de télécommunications

FAI : Fournisseur d'Accès Internet

KAT : Algèbre de Kleene avec Test

MANO : Open Source NFV Management and Orchestration : Standard d'ochestrateur NFV

NFPD : Network Forwarding Path Descriptor

NFVO : NFV Orchestrator : il est en charge de l'orchestration des ressources et des services

NFV : Network Functions Virtualization : concept de virtualisation de fonctionnalité réseau

NSD : Network Service Descriptor : il est l'élément central qui rassemble tous les paramètres nécessaires à l'orchestrateur pour les déploiements

NS : Network Service assure un service complet, il peut être composé de plusieurs VNFs

ONAP : Open Networking Automation Platform : plateforme d'orchestration open source née de la fusion de OpenECOMP et Open-Orchestrator

ONOS : Open Network Operating System : contrôleur SDN Open Source

PEP : Policy Enforcement Points

SDN : Software-Defined Networking

VIM : Virtualized Infrastructure Manager : il gère les ressources physiques d'une infrastructure

VM : Machine Virtuelle

VNFFGD : VNF Forwarding Graph Descriptor

VNFM : Virtual Network Function Manager : il fournit et adapte les modèles de configuration des différents services réseau

VNF : Virtual Network Function : application du concept de NFV à une fonctionnalité

WAN : Wide Area Network

CHAPITRE 1

INTRODUCTION

1.1 Contexte

1.1.1 Des réseaux indispensables

Depuis les débuts d'Internet dans les années 1970, les usages et les besoins ont grandement évolué. Le nombre d'appareils connectés augmente de plus en plus rapidement. Après l'arrivée des ordinateurs personnels, des portables et des smartphones, les objets connectés (Internet of Things) sont à l'origine de la prochaine évolution technologique qui va provoquer une arrivée massive d'appareils connectés pour atteindre les 100 milliards en 2025 d'après GiV [1]. Cette augmentation est loin d'être terminée, actuellement à peine plus de la moitié de la population mondiale a accès à Internet (54,4% en 2017 [2]).

Cette démocratisation d'Internet s'est accompagnée d'une modification des usages, de nouveaux services sont apparus et les services déjà présents ont adapté leur modèle. De ce fait, Internet est devenu indispensable dans de nombreux domaines (par exemple : le commerce, les médias ou les transports) et les problèmes réseau provoquent rapidement de lourds dommages.

En 2016, la compagnie aérienne Delta a été coupée de son infrastructure de traitement durant 5 heures provoquant une annulation d'environ 1 000 vols le jour de la panne et 1 000 vols supplémentaires au cours des deux jours suivants. La compagnie estime que ce problème lui a coûté 150 millions de dollars [3].

Les problèmes réseaux arrivent même à des entreprises dont c'est le cœur de métier. Amazon Web Services, qui domine le marché du cloud computing, est utilisé par de nombreuses entreprises pour héberger leurs services sur internet. Mais suite

1. http://www.huawei.com/minisite/giv/en/download/whitebook.pdf
2. https://www.internetworldstats.com/emarketing.htm
3. http://money.cnn.com/2016/09/07/technology/delta-computer-outage-cost/index.html

à une modification réseau mal exécutée les services hébergés ont été inaccessibles pendant 5 à 36 heures, coupant ainsi d'Internet de nombreux sites dont certains parmi les plus consultés comme Reddit, le New York Times ou FourSquare.

D'importantes évolutions matérielles des équipements composant le réseau et de nouveaux protocoles ont été mis en place pour faire face à ces évolutions des usages et de la demande. Mais récemment deux nouvelles manières de gérer les infrastructures et les services réseau ont gagné en popularité : le Software Defined Networking (SDN) et le Network Functions Virtualization (NFV). Elles donnent une plus grande importance au logiciel et permettent de passer outre certaines contraintes matérielles.

Nous nous sommes intéressés dans cette thèse à ce nouveau paradigme pour garantir la cohérence applicative lors des changements de configuration réseau.

Vocabulaire

Nous employons ici le terme configuration réseau dans son sens le plus large. Elle comprend la topologie des équipements réseau et des services qui l'utilisent ainsi que les règles qui définissent la façon dont est traité le trafic réseau.

1.1.2 Diminution des contraintes matérielles

Dans les équipements réseau avec le SDN

Dans les réseaux classiques, chaque équipement a une fonctionnalité propre et des optimisations matérielles lui permettant de traiter efficacement les données qu'il reçoit. Ils peuvent modifier les paquets reçus et les rediriger en fonction de certaines règles. Leur administration se fait en se connectant à l'équipement et en utilisant des protocoles qui varient en fonction des équipements, de leurs constructeurs ou de leur version. L'idée clé du SDN est mettre en place un contrôleur chargé de l'administration et du monitoring de tous ces équipements. Dans les équipements non SDN ces fonctions étaient effectuées par ce qui est appelé le plan de gestion, les équipements de type SDN se contentent d'agir sur les paquets en fonction des règles qu'ils ont reçues (ce sont les fonctions du plan de transfert). Ce changement apporte différent avantages :

— Le contrôleur a un point de vue central, il connaît l'état de tous les équipements et peut raisonner sur l'intégralité de son réseau.

— L'administration et la configuration peuvent se faire depuis le contrôleur avec des protocoles standardisés communs à tous les équipements.

— Le contrôleur est entièrement programmable, ainsi de nombreuses tâches peuvent être automatisées et cela permet une gestion du trafic plus élaborée.

— L'ajout de nouvelles fonctionnalités est simplifié, plus rapide et moins coûteux. Elles ne dépendent plus du constructeur, il suffit de mettre à jour le programme du contrôleur sans avoir à changer les autres équipements.

— La dépendance aux constructeurs est fortement réduite : des appareils génériques peuvent servir de contrôleurs et de commutateurs, diversifier les constructeurs ne rend pas l'administration plus complexe.

— Les coûts d'achat d'équipement et de maintenance devraient donc aussi être réduits.

Dans les services réseau avec le NFV

Le NFV applique les techniques de virtualisation aux services réseaux pour créer des VNFs afin d'éviter certaines contraintes matérielles. Les services réseau sont composés de différentes VNFs qui réalisent les différentes fonctions nécessaires au service.

Vocabulaire

Les termes NFV et VNF seront utilisés en fonction du contexte : le terme Network Functions Virtualization est utilisé lorsque nous parlons du concept de virtualisation de fonctionnalité réseau alors que le terme Virtual Network Function est l'application de ce concept à une fonctionnalité.

Cela peut s'appliquer à presque tous les services réseaux comme la mise en cache (CDN), les pare-feux, les cœurs de réseau mobile (Evolved Packet Core) [1] ou la résolution de noms de domaine (DNS). La virtualisation de ces fonctionnalités réseau à plusieurs buts :

— Gagner en flexibilité : Les VNFs ne nécessitent pas de matériel dédié, elles peuvent être instanciées en quelques secondes au plus proche de la demande, adapter la capacité et le nombre de leurs instances en fonction de la charge et cela sur des centres de données qui peuvent être gérés par différentes entreprises.

— Accélérer le déploiement de nouveaux services : en évitant les longues phases de développement et de tests matériels (nécessaire afin d'assurer leur fiabilité et car leur évolution est complexe), le temps pour proposer un nouveau service peut être grandement réduit.

— Faciliter l'administration : de nombreuses tâches peuvent être automatisées (comme l'adaptation en fonction de la charge) et ne nécessite pas d'interventions physiques.

— Réduire les coûts : Tous les services pouvant être instanciés sur les mêmes types d'équipement cela permet d'éviter l'achat de matériel dédié. De plus cela permet aussi d'éviter le surdimensionnement matériel de chaque service en prévision de montées en charge : les ressources sont réparties entre les services en fonction de leurs besoins et les fournisseurs de services ne payent que les ressources qu'ils utilisent. Cette gestion plus souple amène une réduction des coûts d'exploitation en diminuant la consommation électrique et l'espace nécessaire ainsi qu'en simplifiant l'administration.

Les concepts du NFV ont été proposés en 2012 lors d'une conférence sur le SDN [2]. Ces deux technologies sont complémentaires, la tendance à l'intégration du SDN avec le NFV pour atteindre divers objectifs de contrôle et de gestion du réseau a connu une croissance notable.

Une architecture NFV couplée à des réseaux gérés par SDN peut aider à relever les défis de la gestion dynamique des ressources et de l'orchestration intelligente des services. Grâce à NFV, SDN est capable de créer dynamiquement un environnement virtuel adapté à un service réseau spécifique, par conséquent le matériel dédié et le travail complexe pour effectuer le déploiement réseau peuvent être évités.

Ces nouvelles possibilités peuvent par exemple servir à répondre à une montée en charge dû à un événement localisé. Lors d'une manifestation culturelle ou sportive si le cœur de réseau mobile n'est pas dimensionné pour tenir une telle charge, il est possible de déployer automatiquement d'autres instances de manière prédictive [3].

Les services nécessaires à son fonctionnement peuvent être répartis sur plusieurs sites, c'est généralement le cas à cause de l'authentification des utilisateurs qui ne reposent que sur un nombre réduit de serveurs.

1.2 Problématiques abordées

Ces changements amènent de nombreuses problématiques de recherche autour de ces technologies, certaines sont nouvelles comme l'optimisation du nombre de contrôleurs ainsi que leur placement ou la programmation des équipements (OpenFlow est le protocole le plus utilisé mais des solutions abordant le problème différemment existent comme P4 [4]). D'autres cherchent à résoudre des problématiques préexistantes grâce à ces nouvelles possibilités comme le calcul de chemin, la détection d'attaque ou l'utilisation optimale des ressources physiques.

Cette thèse abordera la problématique de la vérification de configuration réseau dans le contexte de l'arrivée des technologies Software Defined Networking et Network Functions Virtualization.

Nous avons fait le choix de nous concentrer sur la vérification des changements apportés à une configuration valide pour rechercher comment garantir que, malgré des changements, la configuration demeure cohérente vis-à-vis des besoins des services réseaux qu'elle impacte.

Nous nous sommes principalement intéressés aux situations demandant des changements fréquents, comme les réseaux étendus ou les grands centre de données. Des solutions de vérification efficaces nous semblent indispensables afin d'assurer la continuité des services en place et le déploiement de nouveaux services. Pour cela il faut prendre en compte à la fois les reconfigurations qui viennent de modifications du réseau (par exemple lors d'ajout de nouveau équipements) et celles venant directement des Network Services qui l'utilisent (comme l'ajout d'instances ou le changement de micro-services).

Cette problématique ne se limite donc pas aux techniques de vérification, nos travaux portent aussi sur la formalisation des besoins car si les critères de vérifications sont incorrects les preuves formelles perdent leur intérêt.

1.3 Organisation du document

Dans le chapitre 2 nous allons revenir en détail sur le SDN, le NFV et sur les notions utilisés dans nos travaux. Puis nous dresserons un état de l'art des recherches dans notre domaine dans le chapitre 3.

Ainsi nous pourrons continuer avec les chapitres sur les contributions de la thèse.

Les solutions de vérification au moment du début de nos travaux étaient trop lentes pour être applicables sur des réseaux de grande taille. Nous avons cherché à répondre à cette problématique de vérification de configuration de réseau étendu (chapitre 4). Nous avons proposé d'effectuer ces vérifications uniquement à des points stratégiques du réseau notamment pour limiter l'impact du SDN et de la vérification sur les performances du réseau. Nous avons outillé ce concept avec un langage qui a aussi été pensé pour faciliter l'écriture des règles réseau.

Lors de l'écriture du langage de configuration d'un tel réseau nous avons essayé de limiter les erreurs en nous éloignant du langage des commutateurs et en rapprochant notre langage de ce que voulait faire l'administrateur réseau. Nous avons cherché comment vérifier la cohérence entre le comportement réseau attendu par un Network Service et la configuration du réseau sur lequel il va être déployé. Cela nous a amené à formaliser ces besoins dans le chapitre 5. Pour cela nous avons créé des contrats de comportement réseau qui regroupe les besoins réseau d'une fonctionnalité et de ses différents composants, nous avons aussi formalisé la description de la topologie et le déploiement des composants. Nous utilisons un outil de vérification de modèle, qui grâce à ces informations peut s'assurer de la cohérence entre le déploiement d'une fonctionnalité et ses besoins.

Effectuer cette vérification de configuration réseau au sein de NFV MANO nous a semblé être un défi intéressant. Plusieurs des contraintes que nous avions rencontrées dans nos travaux (ex : changements très fréquents, grande réactivité, diversité des acteurs impliqués) étaient exacerbées dans cet environnement. Afin de proposer une méthode permettant d'appliquer les techniques de vérification au sein d'orchestrateurs NFV gérant des Network Services déployés sur des infrastructures réseau différentes, nous avons étudié le standard de NFV MANO dans le chapitre 6. Cela nous a permis de présenter une solution de vérification dans la continuité des pratiques décrites dans le standard. Nous avons aussi proposé une extension à la modélisation des NS prévu dans le standard pour faciliter l'adoption des techniques de vérification dans les

orchestrateurs et réfléchi à leur implémentation.

Nous conclurons dans le chapitre 7 en discutant des suites possibles à nos travaux.

CHAPITRE 2

CONTEXTE ET CHALLENGES

Les services réseau actuels reposent sur des équipements propriétaires conçus pour effectuer des fonctions précises. De plus le coût élevé de l'infrastructure réseau et la nécessité d'un accord entre un grand nombre d'organisations ayant souvent des intérêts divergents induit un problème appelé l'*ossification du réseau*, qui fait obstacle à toute évolution et à l'ajout de nouveaux services. On peut citer comme exemple les difficultés du déploiement du multicast ou de l'IPv6 [5] qui 20 ans après sa spécification [6] n'est utilisé que par 24% du trafic IP [1]).

Il est possible de résoudre ce problème, et réduire les dépenses en capital (CapEx) et les dépenses d'exploitation (OpEx) [7], avec la *softwarization* qui permet de dissocier le traitement réseau et les applications de leur équipement dédié. Nous allons revenir sur le Software Defined Networking (SDN) et le Network Functions Virtualization (NFV) en détaillant principalement les notions sur lesquelles reposent nos travaux. Certaines notions spécifiques à une contribution sont abordées dans les chapitres qui leur sont dédiées.

2.1 Software Defined Networking

Le fonctionnement des équipements actuels en charge de diriger le trafic à travers le réseau peut être séparé en deux plans (voir figure 2.1) :

— Le plan de contrôle, qui effectue les calculs nécessaires aux protocoles de routage utilisés et qui définit les règles qui devront être appliquées.

— Le plan de transfert, qui applique la première règle dont les critères correspondent au paquet reçu.

1. https://www.google.com/intl/en/ipv6/statistics.html

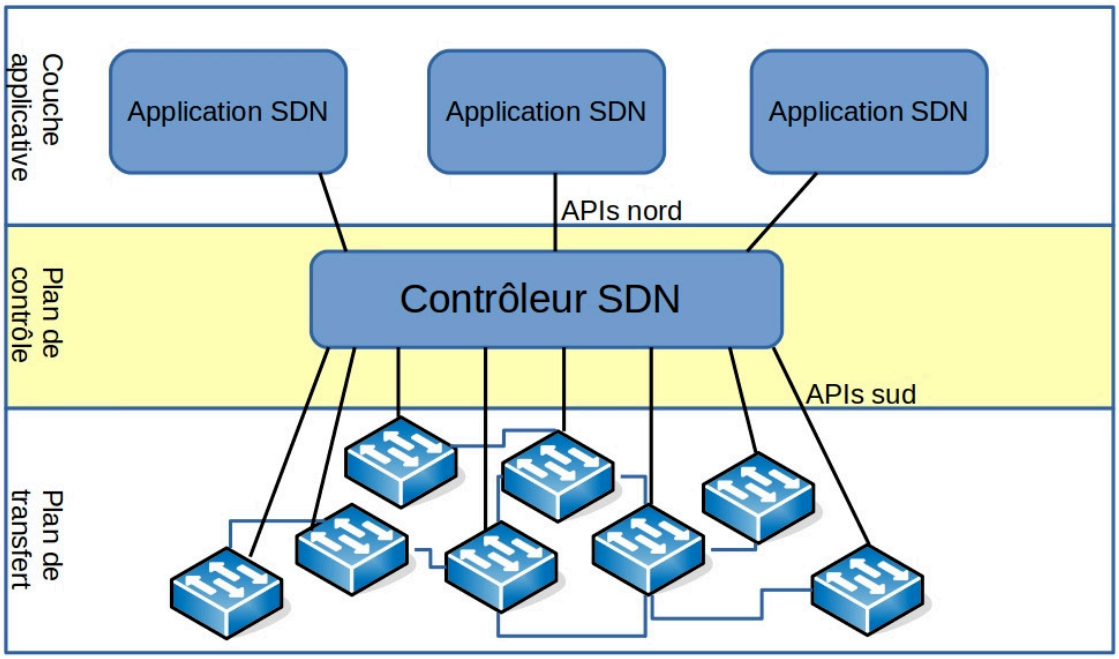

FIGURE 2.1 – Le contrôleur SDN administre le plan de transfert des commutateurs de son réseau

L'idée clé du SDN est de découpler ces deux plans afin d'avoir un plan de contrôle commun pour tous les équipements. Le contrôleur transmettra les décisions de ce plan de contrôle aux plans de transfert des commutateurs.

À cet effet l'Open Networking Foundation[2] a créé un protocole permettant aux contrôleurs d'avoir accès aux plans de transfert de ses commutateurs : OpenFlow. Il est possible d'utiliser d'autres protocoles pour contrôler les commutateurs, certains travaux de recherche portent sur ce point, par exemple P4 [8]. Cependant pour l'instant OpenFlow est utilisé dans presque toutes les implémentations et c'est celui que nous avons utilisé dans tous nos travaux.

OpenFlow permet au contrôleur d'administrer les tables de transfert d'un commutateur en ajoutant, modifiant et supprimant les règles de correspondance de paquets ainsi que les actions à leur appliquer. Les paquets qui ne correspondent à aucune règle peuvent être transmis au contrôleur, qui peut alors décider de modifier les règles de la table de transfert existantes ou de déployer de nouvelles règles.

Ce changement facilite les tâches de configuration des réseaux, qui pouvaient s'avérer ardues d'autant que les grands réseaux sont souvent hétéroclites : les opéra-

2. `https://www.opennetworking.org/`

teurs utilisent des équipements venant de différents constructeurs pour éviter d'avoir une trop grande dépendance à un seul d'entre eux ; ce qui serait dommageable en cas de bogues et empêcherait de mettre en concurrence les tarifs des constructeurs. Cela participe à rendre les réseaux plus complexes à administrer, chaque constructeur ayant son propre langage de configuration (qui lui-même diffère entre les versions de ces équipements).

Cette idée n'est pas totalement nouvelle et a déjà été étudiée par exemple dans le cadre du protocole COPS (Common Open Policy Service) [9] qui proposait une solution centralisée pouvant modifier les règles de routage des équipements réseaux. Dans ce protocole les règles sont stockées sur des serveurs (*Policy Decision Points* – PDP) et appliquées au niveau de *Policy Enforcement Points* (PEP). Deux modèles différents sont proposés pour sa mise en œuvre. Dans le premier modèle, toutes les stratégies sont stockées dans le PDP. Chaque fois que le PEP doit prendre une décision, il questionne le PDP qui analyse les informations remontées, prend une décision et la transmet au PEP. Le PEP se contente ensuite d'appliquer la décision.

Le second modèle repose sur les capacités de décision du PEP. Le PDP envoie des règles sur le PEP en fonction de la situation. Le PEP les stocke dans sa PIB (Policy Information Base) et prend ses propres décisions sur la base de ces politiques.

Dans notre situation nous nous rapprochons plus du deuxième modèle comme nous le verrons plus en détail notamment lors de nos travaux détaillés dans le chapitre 4.
Aujourd'hui de nombreux fournisseurs d'équipements sont prêts à offrir des interfaces ouvertes pour la configuration des éléments de réseau, ce qui explique en partie la meilleure adoption du SDN.

Les architectures réseau de type SDN ont d'abord été déployées dans des campus universitaires, comme celui de Stanford en 2011 [3] [10], dans un objectif de recherche et d'expérimentation. Un des premiers déploiement d'envergure est celui de Google au sein du réseau interne de ses centres de données en 2012. Ses équipes estiment que ce changement leur à permis d'utiliser leurs liens à prés de 100 % de leur capacité, ainsi cette nouvelle gestion du réseau s'est révélée deux à trois fois plus efficace qu'avant l'utilisation de SDN [11] [12].

3. `http://groups.geni.net/geni/wiki/OpenFlow/CampusTopology`

Contrôleur SDN

Le contrôleur, qui a une visibilité sur l'ensemble du réseau est le point de contrôle central. Il contient généralement un ensemble de modules qui effectuent les tâches de base comme l'inventaire des équipements du réseau et des capacités de chacun ou la collecte des statistiques. Des extensions peuvent être ajoutées pour améliorer ces fonctionnalités et prendre en charge des fonctionnalités plus avancées, telles que l'analyse des statistiques ou l'exécution d'algorithmes pour définir le comportement du réseau.

Les interfaces de programmation (API) nord sont utilisées pour la communication entre le contrôleur SDN et les services et applications utilisant le réseau (voir figure 2.1). Ces applications peuvent définir le comportement du réseau par l'intermédiaire du contrôleur. Cela facilite l'innovation et permet une orchestration et une automatisation efficaces du réseau en fonction des besoins des différentes applications.

L'interface sud sert à la communication entre le contrôleur et les commutateurs SDN. Des adaptateurs existent pour permettre au contrôleur d'échanger avec d'autres type d'équipements. Nos travaux se sont principalement concentrés sur cette interface, car nous cherchons à vérifier les règles réseaux qu'elles soient écrites par un administrateur via le contrôleur ou via ses APIs nord.

Il existe de nombreux contrôleurs open source regroupant différents acteurs du domaine (opérateurs, constructeurs, fournisseurs de contenu) et de nouveaux contrôleurs sont régulièrement développés pour répondre à des besoins spécifiques ou pour des travaux de recherche. L'un des plus importants actuellement est OpenDaylight [4].

Les constructeurs d'équipements de réseaux veulent fournir leurs propres contrôleurs SDN pour orchestrer leurs propres équipements (et potentiellement ceux d'autres fournisseurs). Cisco, HP, IBM, VMWare, Lumina Networks et Juniper ont lancé sur le marché leurs propres contrôleurs SDN. Certains d'entre eux proposent maintenant une version commerciale de contrôleur SDN open source, c'est par exemple le cas de HP, Cisco et IBM qui se basent maintenant sur OpenDaylight ou de Juniper qui a fait l'acquisition de Contrail pour l'ajouter à leur catalogue à la fois en version open source et en version commerciale.

4. `https://www.opendaylight.org/`

2.2 Network Function Virtualization

Dans les réseaux actuels, un service réseau comprend un ensemble d'équipements réseau dédiés offrant des fonctionnalités tels que les répartiteurs de charge, les pare-feux ou la détection d'intrusion (IDS), pour prendre en charge les traitements nécessaires au service. Si le nouveau service à besoin de nouvelles fonctionnalités, ou que les équipements en place ne suffisent plus, de nouveaux doivent être installés, configurés et connectés, ce qui est long, complexe, coûteux et sujet aux erreurs.

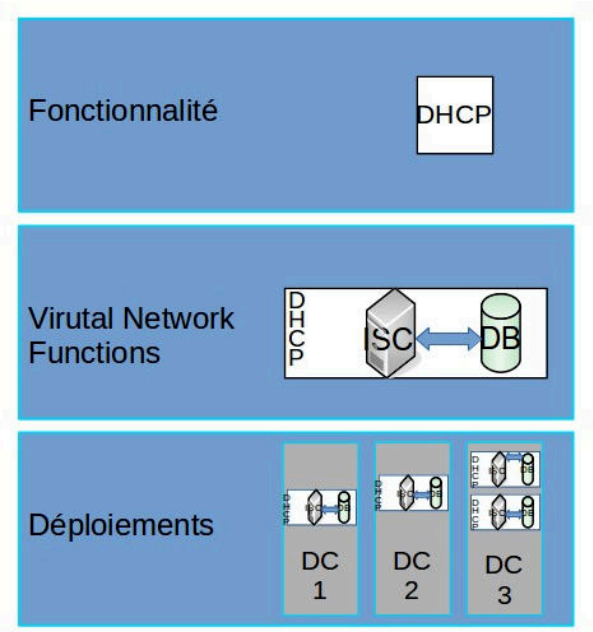

FIGURE 2.2 – Décomposition de la fonctionnalité DHCP en VNFs

NFV s'attaque à ce problème en remplaçant les équipements physiques par des briques logicielles pouvant être exécutées dans des machines virtuelles sur des serveurs classiques.

Par exemple un service de DHCP, qui fournit des adresses IP aux machines qui lui en demande, peut être décomposé en deux VNFs comme sur la figure 2.2 : un serveur qui reçoit les requêtes et les traite (ex : ICP) et une base de donnée qui contient les adresses et qui sera interrogée par le serveur (ex : MariaDB)

Les deux VNFs composant le service peuvent être instanciées dans n'importe quel centre de données pouvant accueillir des machines virtuelles. Pour répondre au mieux à la demande, elles peuvent être instanciées dans les centres de données les plus proches du besoin et lancer ou arrêter des instances en fonction de la charge.

Vocabulaire

Un Network Service rend un service complet, comme le vEPC, il peut être composé de plusieurs VNF (dans le cas du vEPC : DHCP, P-GW, MME, HSS, ...). Une VNF peut être composée de plusieurs micro-services nécessaires à la fonctionnalité, comme dans le cas du DHCP présenté ci-dessus.

Ces concepts ont rapidement été adoptés, l'année suivant le premier papier sur le sujet, des constructeurs [5] avaient adapté leurs catalogues pour répondre aux besoins des opérateurs. Actuellement cet intérêt peut se remarquer dans l'engouement autour de projets open source communs (comme Open Source MANO [6] ou ONAP [7]) sur lesquels travaillent ensemble des opérateurs télécoms, des géants du web, des agences gouvernementales, des centres de recherche, des éditeurs de logiciel et des équipementiers.

Ces travaux donnent lieu à de nombreux déploiements effectués chez les opérateurs réseaux (ex : 80 % du trafic mobile de SFR [8] passe par leur plateforme NFV) et chez les principaux gestionnaires d'infrastructures cloud (Amazon Web Services, Google cloud, Microsoft Azure, OVH). Les opérateurs réseau y voient un moyen de récupérer des marges attaquées par les services OTT (over-the-top) qui se basent sur une infrastructure déjà mise en place par les opérateurs (réseau IP jusqu'aux clients) pour concurrencer certaines sources de revenus des opérateurs comme la téléphonie et ou la télévision.

Architecture NFV et SDN

Ensemble, le SDN et le NFV peuvent combiner leurs forces pour permettre aux opérateurs de services de surveiller et de contrôler avec précision le trafic réseau. D'une part, NFV déplace les fonctions réseau d'équipements matériels dédiés vers des logiciels pouvant être exécutés sur du matériel générique. D'autre part, le SDN déplace les fonctions du plan de contrôle hors des équipements et les place dans un

5. https://www.nec.com/en/press/201310/global_20131022_03.html
6. https://osm.etsi.org/
7. https://www.onap.org/
8. https://www.lemagit.fr/etude/Altice-virtualise-ses-reseaux-mobiles-avec-les-plates-formes-NFV-Ope

contrôleur logiciel. Par conséquent, le déploiement de services peut être piloté par le contrôleur. L'implémentation logicielle des fonctions réseau requises signifie que le déploiement de service réseau ne nécessite plus l'acquisition d'équipements dédiés.

Dans une plate-forme combinant SDN et NFV, l'orchestrateur NFV s'appuie sur des hyperviseurs pour la gestion des VMs qui implémentent les VNFs. Le contrôleur SDN et le système d'orchestration NFV forment le contrôle logique. Le système d'orchestration NFV est chargé du choix de la topologie du déploiement et des règles qui s'y appliquent. Le contrôleur SDN applique ces choix au sein des commutateurs dont il a la charge.

Ces changements apportés par le SDN et NFV amènent des solutions mais aussi de nouvelles problématiques. Nous nous sommes intéressés à la vérification de configuration réseau, car ces nouveaux concepts offrent à la fois des outils pouvant faciliter la vérification et permettent des usages qui augmentent les risques d'erreurs de configuration.

2.3 Un modèle OSI brouillé

Le modèle OSI est souvent utilisé pour représenter les différentes fonctionnalités réseaux sous formes de 7 couches (voir schéma 2.3). Chaque couche du modèle communique avec une couche adjacente : une couche sert la couche au-dessus et est servie par la couche en dessous. Le but de cette représentation est de séparer le problème en différentes couches selon leur niveau d'abstraction. Les services déployés en utilisant SDN et NFV peuvent impacter les couches 2 à 7. Par exemple lors du déploiement du Network Service d'un vEPC, le service agira sur les couches hautes mais aussi sur les couches dites « matérielles » en modifiant certaines règles appliquées par les équipements réseau afin d'ouvrir un tunnel pour les utilisateurs correctement authentifiés. C'est pourquoi on peut dire que les séparations du modèle OSI sont

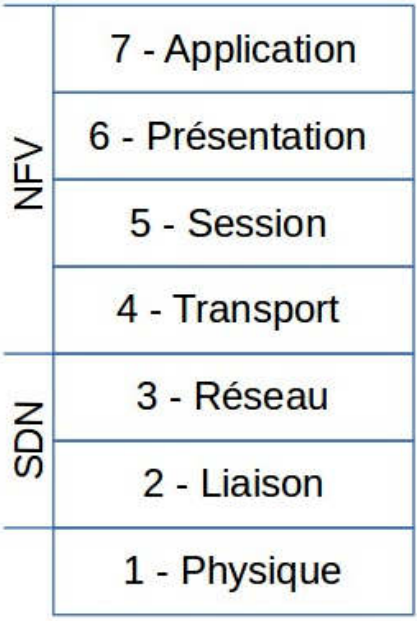

FIGURE 2.3 – Les couches du modèle OSI

moins nettes dans ce domaine.

2.4 Besoin de vérification

2.4.1 Dans le SDN

Les configurations réseau ne sont pas figées, elles évoluent souvent pour répondre par exemple à l'ajout de nouveaux équipements ou à une panne. Ces modifications sont la principale cause d'erreurs ; d'après l'IT Process Institute [9], 80 % des arrêts imprévus sont attribuables à des changements mal planifiés effectués par les administrateurs. Les outils utilisés pour l'administration réseau ne suffisent pas à éviter toutes ces erreurs. 60% des professionnels interrogés, lors de l'enquête menée par Dimensional Research [10], rapportent que les solutions de supervision échouent à prédire les interruptions réseaux ou les problèmes de performances dans plus d'un cas sur deux.

Il est possible d'utiliser la vérification de configuration réseau de type SDN pour éviter ces coupures inattendues après un changement de configuration. Même s'il existait des techniques de vérification avant le SDN, la centralisation du raisonnement dans le contrôleur a facilité leur mise en en place. De plus le SDN fournit une excellente architecture pour embarquer des outils de vérification à l'exécution.

2.4.2 Dans le NFV

Après ces premiers travaux se concentrant sur le SDN, nous avons trouvé dans le NFV un cas d'application des techniques de vérification intéressant. Les risques d'erreurs évoqués plus haut sont plus fréquents, car les architectures de type NFV ont continuellement besoin d'effectuer des modifications de leurs règles réseau, afin par exemple, de lancer un nouveau service ou lorsque la charge d'un service augmente et qu'il faut déployer de nouvelles instances.

Les services réseaux virtualisés forment un ensemble de fonctionnalités connectées pour rendre un service global. Lorsque ces services reposaient sur des équipements spécifiques, c'était aux constructeurs d'assurer la cohérence de l'écosystème vendu. La mise en place d'un nouveau service peut faire appel à de nombreux corps de

9. https://itpi.org/the-visible-ops-book-series/
10. https://ci96.actonsoftware.com/acton/attachment/29444/f-000f/1/-/-/-/-/survey.pdf

métiers avec des visions différentes et des problématiques qui leur sont propres (voir figure 2.4). Par exemple une entreprise qui veut proposer un service à ses clients, utilisera des briques logicielles existantes et en fera développer d'autres pour répondre à ses besoins. Ces fonctionnalités devront ensuite être configurées et connectées entre elles pour composer le service final. Puis l'entreprise fera appel à un gestionnaire de cloud pour héberger le service, les différentes VNFs seront déployées dans les centres de données. Durant ces différentes étapes, il est important de pouvoir vérifier la cohérence entre les besoins réseaux de chaque brique du service et le réseau de leur déploiement effectif.

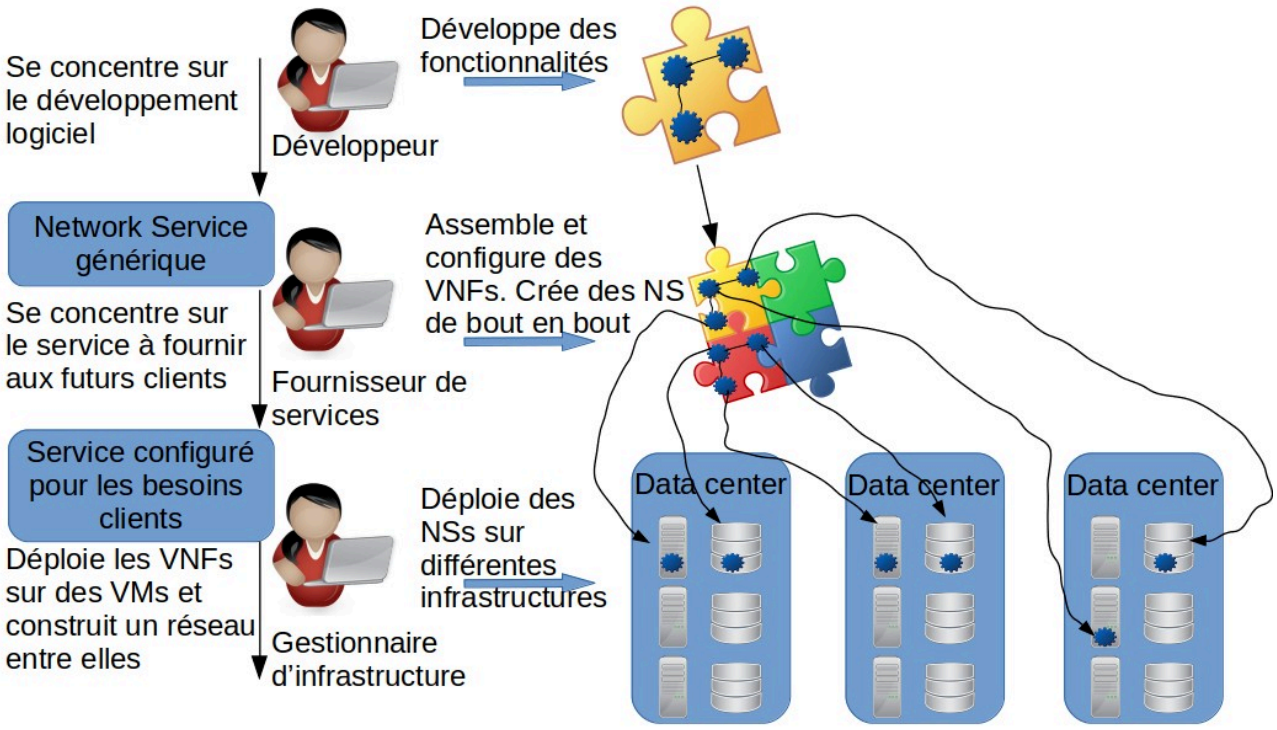

FIGURE 2.4 – Création et déploiement d'un service réseau

Nous avons donc plusieurs métiers qui utilisent des outils variés, dans des buts différents pour créer des fonctionnalités qui pourront être assemblées de diverses manières puis déployées dans des architectures différentes. Cette hétérogénéité rend complexe le déploiement et la flexibilité augmente le nombre de situations où des problèmes peuvent se produire.

La plupart des outils de gestion de configuration et des orchestrateurs réseau utilisés pour déployer des Network Services n'ont pas d'abstraction pour exprimer des

contrats entre la configuration de VNFs et la configuration réseau. Par conséquent, le déploiement d'un Network Service peut être incompatible avec la configuration du réseau auquel il est relié et provoquer des problèmes sur le nouveau service mais aussi des conflits avec les Network Services déjà déployés.

2.4.3 Dans les orchestrateurs

Pour gérer les infrastructures NFV de tailles importantes, des orchestrateurs ont été créés. L'European Telecommunications Standards Institute (ETSI) a standardisé la gestion de toutes les ressources des centres de données (calcul, réseau, stockage et de machines virtuelles). Le standard défini les rôles des différents composants et les informations qu'ils échangent afin de déployer de manière automatisée les Network Services.

L'orchestrateur, grâce à la vue globale qu'il a sur l'état du réseau, peut être l'endroit idéal pour vérifier les configurations réseaux où seront déployés les composants des Network Services. Cependant cet aspect n'a pas était pris en compte dans le standard.

2.4.4 Vérification ou Intent based networking

Une des techniques visant à s'assurer que le réseau réagit comme les opérateurs le souhaitent, utilise les intentions (intent based networking). C'est-à-dire qu'au lieu de créer des règles réseau, c'est le résultat attendu qui est décrit et les règles dictant au réseau son comportement sont générées automatiquement en fonction de ces intentions. Cette méthode n'est pas totalement nouvelle ; son principe rappelle les *goal-based policies*, mais elle a gagné en popularité récemment, notamment sous l'impulsion de Cisco.

Cette approche n'exclut cependant pas la vérification, et les outils de vérification peuvent servir à vérifier les règles générées de la même manière qu'ils peuvent être utilisés pour les règles écrites par les opérateurs.

2.4.5 Vérifier une configuration réseau étendue

Les critères utiles à la vérification pouvant caractériser un réseau et son fonctionnement sont multiples. La topologie du réseau donne naissance à une carte indiquant

quels équipements sont reliés physiquement. Ces équipements peuvent être de différents type : commutateurs SDN, middleboxes (par exemple un pare-feux, un répartiteur de charge ou un système de détection d'intrusion) ou des ordinateurs qui sont en bout du réseau et s'en servent pour communiquer. Les informations qu'ils s'envoient traverseront le réseau jusqu'à leur destinataire si la topologie et si les règles appliquées au trafic par les divers équipements le permettent.

Grâce à ces informations nous pouvons raisonner sur le comportement du réseau pour y effectuer des vérifications sur les services qui l'utilisent. Il nous semble indispensable d'intégrer les services dans la démarche de vérification car le réseau n'est pas une finalité en soi mais un moyen utilisé par les services pour fonctionner. Intégrer les services permet de savoir quels critères doivent être vérifiés. Ces critères peuvent varier en fonction des services rendus, des protocoles utilisés et des logiciels pour les mettre en œuvre.

2.5 Contributions et publication

2.5.1 FlowKAT

Le besoin de flexibilité et de rapidité dans les déploiements est un moteur dans l'adoption des technologies de type SDN. Cela augmente la fréquence des modifications de configurations réseau et donc les besoins de moyens pour les vérifier.

Afin de pouvoir valider une nouvelle configuration, ou les modifications d'une configuration existante, il faut que le réseau et que le comportement attendu soient correctement définis. Nous avons pour cela réfléchi à la création d'un langage dédié fournissant des abstractions devant permettre aux opérateurs d'exprimer des politiques réseau dynamiques de manière concise et intuitive pour ensuite les vérifier. Ce langage est exécuté dans le contrôleur pour envoyer des règles aux commutateurs qu'il dirige (voir le schéma 2.5). Cette contribution est détaillée dans le chapitre 4.

2.5.2 Contrat de comportement réseau

Nous pensons que les techniques de vérification peuvent être très utile au NFV du fait de l'hétérogénéité des acteurs qui interviennent dans la mise en place d'un

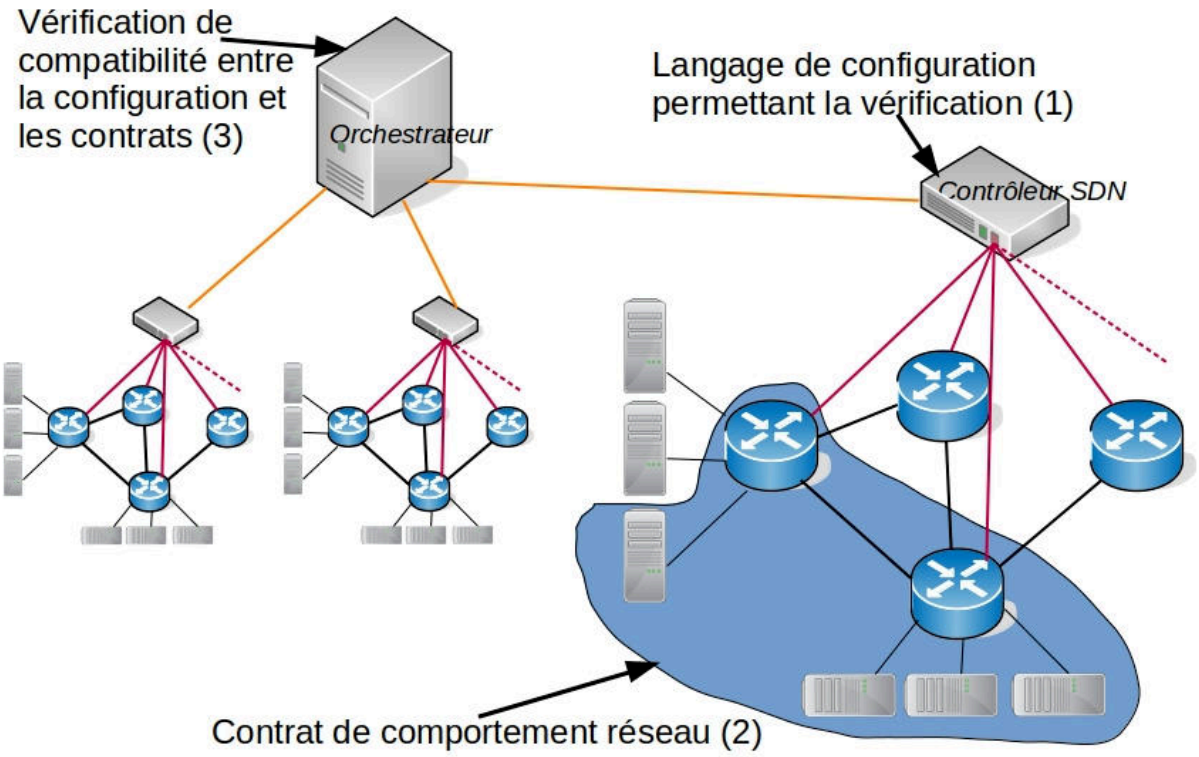

FIGURE 2.5 – Les contributions de la thèse.

service. Cela rend la coordination complexe et augmente le nombre de situations où des problèmes peuvent se produire.

En nous appuyant sur nos travaux se concentrant sur le SDN et la programmation du contrôleur nous avons élargi le champ d'applications en travaillant sur des contrats de comportement réseau qui ne définissaient plus la manière dont les équipements devaient agir mais le résultat attendu.

Cette deuxième contribution, approfondie dans le chapitre 5, propose une approche pour vérifier la cohérence entre la description d'une VNF, faite à partir d'un ensemble de modèles structurels et de modèles d'organigramme, et un déploiement proposé sur une infrastructure de type SDN réelle avec son propre gestionnaire de configuration. Pour cela nous proposons une abstraction afin de mieux traiter les préoccupations des différents acteurs en créant des contrats pouvant exprimer les besoins réseau d'une fonctionnalité. Nous utilisons ensuite ces contrats pour vérifier la cohérence entre la description d'un service réseau et un déploiement proposé sur une infrastructure de type SDN/NFV.

Nous avons synthétisé ces travaux dans l'article « Verifying the configuration of virtualized network functions in software defined networks » [13].

2.5.3 Vérification dans les architectures NFV MANO

Pour gérer ces infrastructures NFV, des orchestrateurs respectant le standard défini par l'ETSI peuvent être utilisés (voir *(3)* dans la figure 2.5). Nous avons étudié ce standard dans le but d'intégrer la vérification de configuration dans l'architecture des orchestrateurs NFV. Dans le chapitre 6, nous détaillons notre analyse de ce standard ainsi que les informations manquantes pour effectuer des vérifications sur la configuration réseau. Nous proposons ensuite une extension du standard palliant ces manques.

Ainsi une fois que les informations nécessaires sont remontées grâce à l'orchestrateur et que les contrats de comportement sont définis, il devient possible de vérifier la cohérence entre les attentes des services déployés et la configuration du réseau sur lequel ils sont déployés.

CHAPITRE 3

ÉTAT DE L'ART

3.1 Méthodologie

Notre état de l'art est séparé en deux parties correspondant à deux aspects de nos travaux : nous verrons d'abord la modélisation des Software Defined Infrastructures puis comment ces abstractions sont utilisées pour la vérification de configuration réseau. Dans cette seconde partie, nous nous concentrerons sur les outils permettant, au minimum, de vérifier qu'une machine A connectée à un réseau SDN peut bien communiquer avec une machine B (l'atteignabilité). D'autres propriétés découlent de celle-ci :

— L'absence de boucles : si les paquets sont coincés dans une boucle du réseau, ils n'atteindront jamais leur cible.

— L'isolation : si les règles ou la topologie empêchent les paquets de la machine A d'atteindre la machine B, nous pouvons dire qu'elles sont isolées. Cela est recherché pour certaines des politiques de confidentialité ou de sécurité.

Ces critères de tri ont été appliqués aux outils présentés dans : [14] [15] [16] puis la liste a été complétée diverses études.

3.2 Modélisation de réseau SDN

Les outils présentés ici utilisent des abstractions pour limiter la complexité du système sur lequel ils raisonnent. Ces abstractions diffèrent en fonction de l'objectif de l'outil. La majorité de ceux étudiés dans le cadre de cet état de l'art modélise le réseau avec sensiblement les mêmes informations : les nœuds (clients et commutateurs), liens qui les relient et les règles qui s'appliquent au trafic. Certains aspects peuvent être modélisés plus finement, par exemple la modélisation d'un lien peut se conten-

ter de décrire quels équipements il relie mais aussi inclure des informations sur sa capacité à analyser des problématiques de congestion.

Nous allons présenter ici les modèles qui incluent des informations différentes de cette modélisation classique, les autres seront vus en détail dans la deuxième partie de cet état de l'art qui s'attardera sur la façon dont ils sont utilisés pour la vérification.

Dans « Analytical model for SDN signaling traffic » [17], Metter *et al.* proposent de modéliser le nombre de paquets dans un flux en fonction de l'application à laquelle il est destiné afin de calculer les effets du temps de conservation des règles dans les tables des commutateurs. Pour limiter la taille des tables des commutateurs, il existe des mécanismes permettant de supprimer les règles inutilisées. Quand de nouveaux paquets ayant besoin d'une de ces règles pour être traités arriveront, le commutateur devra interroger le contrôleur pour à nouveau recevoir cette règle. Le but de cette modélisation est de calculer le temps optimal de conservation des règles dans le commutateur en fonction du type de flux.

Sato *et al.* axent leur modélisation sur le contrôleur et ses différents modules dans « Abstract Model of SDN Architectures Enabling Comprehensive Performance Comparisons » [18]. Elle permet à des architectures SDN d'être évaluées et comparées. Un modèle de fonction décrit toutes les fonctions du contrôleur SDN et les détails de chacun de ses composants. Puis un modèle de traitement décrit les différents types de procédures. Ils permettent de comparer plusieurs architectures SDN selon une condition d'évaluation unifiée et de discuter de l'architecture SDN et de ses variations pour trouver la conception optimale.

Sethi *et al.* [19] reprennent une modélisation classique du réseau avec un contrôleur, des commutateurs et des clients disposants de ports qui permettent de les relier entre eux. À cela, ils ajoutent une modélisation des paquets qui traversent le réseau. Ceux-ci sont représentés à l'aide de leur en-tête mais aussi des informations (*payload*) qu'ils transportent. Ils sont utilisés dans l'abstraction du réseau pour tester le comportement du contrôleur après la mise à jour de règles.

Soenen *et al.* [20] abordent le problème du placement des VNFs en étudiant leurs relations avec différentes métriques pour considérer à la fois les attentes des clients et celles des gestionnaires d'infrastructure. Leur modélisation reprend les informations classiques du réseau en ajoutant des informations sur les ressources de chaque équipement.

Modélisation d'architecture

Haleplidis *et al.* [21] partent du principe que le SDN et le NFV forment un ensemble plus vaste, celui du cycle de vie complet des dispositifs du réseau et peuvent tirer parti de la définition d'un modèle abstrait commun, tant pour le modèle de transmission que pour les fonctions du réseau. Leur but avec ce modèle est de permettre l'interopérabilité et l'homogénéité, pour le contrôle, la gestion, l'orchestration et les fonctions du réseau.

Vocabulaire

Une chaîne de service constitue une liste, généralement ordonnée, des fonctionnalités (VNF) qui échangent des informations pour rendre un service (NS) complet.

Dans [22], Giotis *et al.* décrivent une architecture de management NFV modulaire permettant à des services d'opérateurs télécom (OSS/BSS) d'être mis en œuvre en tant que chaîne de services grâce à une liste de politiques définies. Elles sont basées sur un *Information Model* qui décrit les ressources du réseau, leurs fonctions de contrôle et propose une manière générique d'exprimer les politiques de contrôle et de gestion des ressources.

3.3 Vérification des règles réseau

Comme les erreurs de configuration sont la première cause d'interruption réseau et que les outils de supervision ne permettent que rarement de détecter ces erreurs (voir [23]), de nombreux travaux ont été menés dans ce domaine [24]. Peu d'entre eux datent d'avant le SDN, le contrôleur SDN facilite ces vérifications et donne accès à de nouvelles méthodes grâce à sa vue globale du réseau et sa programmabilité.

La majorité des outils présents dans cette partie de l'état de l'art fournissent des preuves formelles, seul trois utilisent des tests. Les tests ont l'avantage d'être peu gourmands en ressources mais garantissent uniquement le cas testé dans les conditions du test. Pour ces deux catégories, nous allons décrire les outils et les méthodes utilisés.

Nom	**Informations contenues dans l'abstraction**	**Objectifs**
« Analytical model for SDN signaling traffic » [17]	Le nombre de paquets en fonction du type de flux	Calculer les effets du temps de conservation des règles dans les tables des commutateurs
« Abstract Model of SDN Architectures Enabling Comprehensive Performance Comparisons » [18]	Les fonctions du contrôleur SDN et ses composants	Comparer plusieurs architectures selon une condition d'évaluation unifiée
« Abstractions for Model Checking SDN Controllers » [19]	Abstraction réseau classique et les paquets (entête et données)	Tester le comportement du contrôleur après la mise à jour de règles par l'envoi de paquets dans l'abstraction du réseau
« A Model to Select the Right Infrastructure Abstraction for Service Function Chaining » [20]	Abstraction réseau détaillant les ressources de chaque équipement	Permettre aux gestionnaires de choisir le meilleur placement pour un scénario donné
« Towards a Network Abstraction Model for SDN » [21]	Abstraction des fonctions réseau commune au SDN et au NFV	Permettre une gestion homogène du réseau et des fonctions du réseau
« Policy-based Orchestration of NFV Services inSoftware-Defined Networks » [22]	Abstraction réseau détaillant les ressources de chaque équipement	Mettre en place des services réseau en respectant des politiques prédéfinies

TABLE 3.1 – Modélisations

3.3.1 Preuves formelles

"Testing shows the presence, not the absence of bugs"
Dijkstra (1969)

Pour fournir une preuve formelle tous les cas possibles sont pris en considération et vérifiés, les résultats obtenus sont "absolus", ce qui signifie que si la propriété "aucune boucle dans ce réseau" a été validée, il est impossible qu'une boucle se produise avec cette configuration. Le principal défi de la vérification des modèles de configuration réseau SDN est l'explosion de l'espace d'état résultant des facteurs suivants :

— Il peut y avoir un grand nombre de paquets parcourant le réseau, de plus ces paquets peuvent déclencher des changements de l'état du réseau ce qui augmente encore le nombre d'états.

— Les tables de flux de commutation stockent une correspondance entre l'en-tête du paquet, le port d'entrée et le port de sortie, ce qui donne un grand nombre d'états possibles pour chaque commutateur (les commutateurs modernes peuvent avoir des dizaines de milliers d'entrées dans leurs tables de flux).

Cela rend la validation très complexe et très coûteuse en temps de calcul. Certains des outils présentés tentent de limiter ce problème en utilisant différentes techniques, d'autres s'orientent vers les tests qui permettent d'avoir des résultats plus rapidement en se concentrant sur des cas précis mais qui n'apportent aucune preuve absolue.

Diagramme de décision binaire

Les premiers outils de vérification ont utilisé des diagrammes de décision binaire pour modéliser de manière compacte, en comparaison par exemple aux arbres de décision, les règles des commutateurs SDN.

FlowChecker [25], puis ConfigChecker [26] [27], modélisent le comportement réseau de bout en bout en représentant le réseau comme une machine à états où l'entête des paquets et leurs emplacements déterminent l'état. Les règles réseaux sont écrites sous la forme de fonctions booléennes à l'aide de diagrammes de décision binaires (BDD). Ils utilisent ensuite la Computation Tree Logic (CTL) et la vérification du modèle pour étudier tous les états d'un paquet dans le réseau et vérifier le respect d'invariants. Ces invariants sont définis à l'aide du Security Content Automation Protocol

(SCAP) et peuvent concerner par exemple l'atteignablilité ou des exigences de sécurité. Cet outil permet aussi d'analyser l'impact de nouvelles configurations en étudiant des hypothèses (*what-if analysis*).

Modélisation géométrique

Un des premiers outils utilisant la modélisation géométrique est Header Space Analysis (HSA) [28]. Il vérifie de manière statique les spécifications et les configurations du réseau afin d'identifier des problèmes d'atteignabilité, de boucles ou d'isolation du trafic. Il s'agit d'un modèle géométrique où les paquets sont des points dans un espace réseau et les commutateurs SDN sont des fonctions qui transforment les points dans l'espace réseau défini. Ces techniques ont donné lieu à la création d'une bibliothèque appelée Header Space Library (Hassel) et plus récemment à une extension : Stateful Header Space Analysis (SHSA) [29] pour détecter et résoudre les problèmes sur les équipements à états (*stateful*).

Deux autres outils utilisent HSA pour effectuer leurs vérifications : NetPlumber [30] (par les mêmes auteurs) et Flowguard [31].

Flowguard permet la résolution des violations des politiques de pare-feu dans les réseaux SDN. Pour cela, il vérifie les espaces de chemin d'accès au réseau pour détecter les violations de la politique de pare-feu lorsque les états du réseau sont mis à jour. Il se base sur Header Space Analysis (HSA) pour la construction du mécanisme de suivi de flux.

NetPlumber vérifie des politiques en temps réel. Il agit entre le plan de contrôle et le plan de données pour détecter les changements d'état du réseau. NetPlumber vérifie progressivement la conformité des changements d'état, en utilisant un ensemble d'outils conceptuels qui maintiennent un graphe de dépendance entre les règles. Il permet de vérifier ainsi les violations d'invariants telles que les boucles, les problèmes d'atteignabilité ou les trous noirs. La topologie est construite en récupérant les informations des tables de transfert et les contraintes sont écrites à l'aide de Flowexp. Pour décrire des politiques de plus haut niveau, NetPlumber permet d'utiliser Flow-based Management Language [32] pour écrire des politiques ajoutant des contraintes comme par exemple : le trafic doit passer par un pare-feu lorsque c'est un client qui s'adresse à un

serveur.

Les travaux de Basile *et al.* [33] [34] [35] utilisent une modélisation géométrique qui leur est propre. Ils modélisent les règles réseau de manière à effectuer la détection et la résolution à partir de l'intersection d'hyper-rectangles. Ce modèle formel leur permet de détecter les conflits entre les fonctions déployées sur un même réseau.

NICE [36] est un vérificateur de modèle conçu pour les contrôleurs OpenFlow, et peut effectuer une vérification par rapport à des propriétés réseau comme la présence de boucles. Il crée un modèle à partir de la topologie puis explore tout l'espace d'état du modèle et vérifie les propriétés. Ensuite, NICE peut fournir les violations de propriété trouvées ainsi que les traces permettant de les reproduire et d'identifier le problème. Afin d'éviter d'avoir un espace d'état trop important, NICE utilise un moteur d'exécution symbolique pour identifier les paquets qui déclenchent ces événements. NICE fournit une bibliothèque de propriétés vérifiables qui peut être étendue sous forme de snippets de code Python.

Verificare [37] est une plate-forme conçue pour permettre la vérification formelle de configuration SDN. Elle a trois composantes principales : un langage de modélisation (VML), un ensemble d'exigences à vérifier et des traducteurs pour des outils de vérification déjà existant (tels que SPIN, PRISM ou Alloy). Les exigences en matière de SLA ou de sécurité peuvent être sélectionnées parmi des bibliothèques ou écrites par les développeurs. Ces exigences comprennent l'atteignabilité, les boucles, la perte et la duplication de paquets et les trous noirs du réseau. Verificare pourra fournir des contre-exemples montrant les exigences non respectées.

Parcours de graphe

Les graphes sont utilisés pour représenter la topologie et ses règles, ils sont ensuite parcourus pour connaître l'impact sur les paquets et vérifier des conditions. Différentes méthodes sont utilisées lors de la création ou lors du parcours du graphe pour diminuer le temps de calculs.

VeriFlow [38] se place entre un contrôleur SDN et les commutateurs pour vérifier dynamiquement les violations d'invariants (comme l'accessibilité) à l'échelle du réseau à mesure que chaque règle est insérée, modifiée ou supprimée. VeriFlow se sert d'algorithmes incrémentaux pour rechercher les violations potentielles des invariants en découpant le graphe réseau en un ensemble de classes d'équivalence. Il limite ainsi la

vérification aux seules parties du réseau dont les actions peuvent être impactées par la modification de règles. Cela lui permet de répondre rapidement pour les modifications qui ont peu d'incidences mais pour celles qui entraînent plus de vérifications les auteurs proposent la possibilité de les effectuer après l'application des modifications afin de ne pas bloquer le réseau.

VeriCon [39] est en mesure de garantir l'absence de conflits avec des invariants dans les configurations SDN et de fournir un contre-exemple concret en cas de violation d'un invariant. Il se sert pour parcourir son graphe d'une approche de vérification déductive Floyd-Hoare-Dijkstra utilisant le solveur Z3 [40]. Il propose une surcouche à OpenFlow permettant une description à un plus haut niveau.

Libra [41] permet de détecter les problèmes d'atteignabilité et de trous noirs dans les très grands réseaux. Pour avoir des temps de vérification réduits, malgré la taille du réseau traité, Libra divise et parallélise les opérations de vérification grâce à MapReduce [42]. Libra est plus rapide et plus scalable que les outils existants parce qu'il résout un problème plus restreint : il suppose que les paquets ne sont transférés que sur la base de préfixes IP et que les en-têtes ne sont pas modifiés en cours de route.

Le vérificateur temps réel Delta-net [43] s'attaque aussi aux difficultés liées aux très grands réseaux. Il réduit le temps nécessaire à la vérification en modélisant, de façon incrémentale, tous les flux de paquets dans un seul graphe étiqueté (single edge-labelled graph) au lieu d'avoir de multiples graphes se chevauchant pour représenter les flux. Il utilise OpenFlow pour la description des règles qui s'appliquent dans le réseau et C++ pour écrire les vérifications d'invariants et les scénarios "*what if*".

VeriSDN [44] [45] vérifie les configurations OpenFlow en les décrivant à l'aide de *packet-based ACSR*, qui est une version étendue de l'Algebra of Communicating Shared Resources (ACSR) [46] développée pour la vérification formelle des systèmes embarqués en temps réel. VeriSDN génère un Symbolic Transition Graph (STG) à partir de cette description et détecte des problèmes de configuration (comme des boucles ou des trous noirs) ou des incohérences entre différentes règles.

Machines à états finis

Les graphes de transition à états finis fournissent un formalisme adéquat pour la description des systèmes à états finis, mais aussi pour des abstractions à états finis comme des protocoles de communication. Dans notre cas, les états correspondent à la situation d'un paquet lors de son passage à travers les commutateurs composant le réseau.

NetKAT [47] [48] s'appuie sur le fait que les procédures réseau qui agissent sur les paquets peuvent être considérées comme des expressions régulières d'un certain alphabet (celui formé par les champs de paquets). Ensuite, en introduisant un historique permettant de suivre la progression d'un paquet générique à travers le réseau, notamment en fonction des décisions de chaque commutateur, il est possible de montrer que le système est décidable et prouvable. Pour cela NetKAT utilise l'algèbre de Kleene avec des tests (KAT) [49] [50] qui lui permet de décrire le réseau et les règles qui s'y appliquent. NetKAT est capable d'apporter des preuves formelles pour des propriétés données comme l'atteignabilité, l'isolation ou l'absence de boucle. Une extension de NetKAT, WNetKAT (pour Weighted NetKAT) [51], permet de raisonner avec des réseaux qui sont pondérés, permettant ainsi de prendre en compte des contraintes de coûts ou de qualité de service. Le fonctionnement de NetKAT est détaillé dans la section 5.4.2.

La plupart des outils présentés ici ne fonctionnent que sur un instantané du plan de données. Ils ne permettent pas aux opérateurs de raisonner sur les programmes des contrôleurs, ni sur la façon dont le contrôleur effectue des changements en réponse à divers événements dans le réseau. Ils ne fournissent aucun moyen pour un opérateur de réseau de trouver des erreurs dans les contrôleurs qui installent les règles dans les commutateurs.

Kinetic [52] permet de faire les deux grâce à un langage dédié et un contrôleur SDN qui permettent aux opérateurs d'exprimer des politiques réseaux dynamiques. Son langage, basé sur Pyretic [53], est structuré pour exprimer les politiques de réseau en termes de machines à états finis (FSM). Il utilise la Computation Tree Logic (CTL) pour vérifier automatiquement les politiques à l'aide de NuSMV [54] peut permettre aux opérateurs de réseau de vérifier le comportement du contrôleur avant que les modifications ne soient appliquées.

Assertions

Beckett *et al.* [55] proposent un langage d'assertions pour vérifier des conditions pouvant changer dynamiquement. Le langage permet aux programmeurs d'écrire et d'annoter des configurations de commutateurs SDN avec des assertions, d'une manière similaire aux langages de programmation conventionnels qui utilisent les assertions [56] [57] pour faciliter la recherche de bogues. Afin de pouvoir faire face aux changements fréquents Beckett *et al.* proposent une structure de données incrémentale avec un moteur de vérification pour éviter de re-vérifier l'ensemble du plan de données à mesure que ces conditions de vérification changent. Ce langage a été implémenté en tant que bibliothèque de débogage sur une version modifiée de VeriFlow.

Solveurs

Spinozo *et al.* [58] vérifient que les fonctionnalités mises en œuvre dans une VNF ne sont pas perturbées par les modifications apportées par les middleboxes ou les autres VNFs. Pour ce faire, ils utilisent un solveur SMT (satisfiability modulo theories) qui fournit des preuves formelles avant un déploiement. La tâche de vérification formelle est divisée en plusieurs sous-tâches, pour que l'ensemble du processus soit plus simple et que les temps de réponses soient plus courts. Les chaînes de VNFs composant le service réseau sont calculées puis, pour chacune d'entre elles, un modèle formel est généré, incluant le modèle de toutes les VNFs impliquées. Enfin, le moteur de vérification traite l'ensemble du modèle de chaîne de VNFs pour vérifier la satisfaction de propriétés d'atteignabilité.

VeriCon [39], détaillé dans la partie de notre état de l'art sur les parcours de graphe, effectue son parcours de graphe avec un solveur Z3 (solveur SMT).

3.3.2 Simulation et tests

Les tests, suivant leur complexité, peuvent ne demander qu'un temps calcul réduit ; c'est un avantage pour faire face aux reconfigurations fréquentes et à des réseaux de taille très importante.

Kuai [59] permet de vérifier certaines propriétés du réseau en utilisant un vérificateur de modèle énumératif distribué conçu pour les réseaux SDNs. Pour améliorer

le temps de vérification Kuai utilise un contrôleur SDN écrit en Murphi et une version simplifiée des commutateurs OpenFlow ainsi qu'un ensemble de techniques de *partial order reduction* pour réduire l'espace d'état en exploitant la concurrence entre les différents processus.

Sethi *et al.* [19] s'appuient sur des techniques d'abstraction pour prouver certaines propriétés (absence de boucle, trous noirs) en utilisant la vérification de modèle. Étant donné la topologie d'un réseau, l'exactitude d'une configuration SDN est prouvée par un nombre arbitraire de paquets. Ils font abstraction de l'état des données et de l'état du réseau en se basant sur le maintien d'un seul paquet dans le réseau et sur l'injection de paquets avec des valeurs d'en-tête arbitraires (paquets d'environnement). Cela permet de limiter les problèmes d'explosion de l'espace d'état.

Toolkit for Automated Sdn TEsting (TASTE) [60] propose une méthodologie pilotée par les tests, inspirée par les principes du génie logiciel. Elle utilise le langage Data Path Requirement Language (DPRL), qui étend le Flow-based Management Language (FML) [32], pour formaliser les exigences. TASTE génère des paquets de test à partir de règles DPRL, puis injecte ces paquets dans un réseau émulé et analyse les traces des paquets de test afin de vérifier le comportement.

3.4 Description de configuration étendue

Les critères utiles à la vérification du bon fonctionnement de services réseau sont multiples comme expliqué dans la section 2.4.5. Certains des outils décris précédemment permettent une description prenant en compte certains d'entre eux.

Spinozo *et al.* [58] vérifient que les fonctionnalités mises en œuvre dans une VNF ne sont pas perturbées. Les chaînes de VNFs composant le service réseau sont calculées puis, pour chacune d'entre elles, un modèle formel est généré, incluant le modèle de toutes les VNFs impliquées. Le moteur de vérification traite l'ensemble du modèle de chaîne de VNFs pour vérifier l'atteignabilité.

Dans sa description étendue de la configuration en VML, Verificare [37] permet de spécifier les agents actifs pouvant démarrer les interactions (ex : clients et serveurs) et l'environnement passif (ex : les commutateurs) et d'utiliser leur rôles dans la rédaction des exigences.

Certains langages de description permettent de décrire des politiques plus fines prenant en compte des rôles, des groupes ou les interactions des clients. C'est le cas de Flow-based Management Language [32], utilisé par NetPlumber [30], Stateful Header Space Analysis (SHSA) [29] et Header Space Analysis (HSA) [28], ou des langages propres aux outils comme ceux utilisés par ConfigChecker [27] [25], Spinozo *et al.* [58], ou Basile *et al.* [33].

3.5 Conclusion

La vérification ne peut valider que des critères spécifiques et non pas tous les paramètres du réseau qui pourraient provoquer des erreurs, des latences ou des incompatibilités entre les serveurs qui souhaitent communiquer. Il y a donc des risques de faux positifs : par exemple des paquets de test ou une simulation peut valider une configuration sans remarquer que certains périphériques ne sont pas configurés pour accepter les trames jumbo et ceci n'apparaîtra que lorsque le service sera en cours d'exécution. Certains appareils prennent en compte le contexte (les paquets déjà traités) pour décider quelles règles s'appliquent au trafic. Une configuration ne peut donc être validée que pour un état donné. Il est également possible que certains événements puissent entraîner un changement dans la gestion globale du trafic de l'infrastructure. Par exemple, suite à la détection d'une attaque, les politiques de sécurité peuvent conduire à un changement dans les règles du réseau et certains paquets seront traités différemment en fonction de modèles de trafic suspects (IPs sources, HTTP HEADER, ...).

De même dans de grands réseaux avec des changements fréquents, un instantané (*snapshot*) peut être incohérent parce que l'état du réseau change pendant que celui-ci est capturé ou pendant le temps de la validation. Kinetic s'attaque à ce problème en agissant au niveau du contrôleur et des commutateurs. D'autres outils agissent avant l'envoi de modifications mais leurs performances rendent difficile de traiter de nombreuses modifications dans un temps raisonnable. Les outils plus récents ont réussi à diminuer le temps nécessaire pour apporter des preuves sur un grand réseau. Parmi eux Delta-net [43] est le plus efficace, d'après leurs propres expérimentations, pour effectuer en temps réel des vérifications sur des réseaux de grande taille.

Même lorsqu'il est techniquement possible de prouver certaines propriétés dans un temps acceptable, il reste à définir correctement les propriétés qui doivent être

testées. Des oublis peuvent amener à valider une configuration qui rencontrera des problèmes et inversement un critère mal défini ou superflu peut bloquer un déploiement en invalidant une configuration correcte.

Il nous semble plus sûr que les propriétés nécessaires au bon fonctionnement d'un service soient écrites par l'équipe conceptrice du service et non par l'équipe qui le déploie. Cela doit pouvoir faire partie d'une description étendue de la configuration réseau (voir le chapitre 2.4.5) comme le font Verificare [37], NetPlumber [30], ConfigChecker [27] [25], Spinozo *et al.* [58], ou Basile *et al.* [33].

Ces propriétés peuvent être rassemblées dans des bibliothèques pour être ensuite réutilisées comme le propose par exemple Verificare [37], Beckett *et al.* [55] ou NICE [36]. Cela évite le travail fastidieux de les réécrire et possiblement d'y ajouter des erreurs. Les communications entre différentes entités à travers un réseau sont très souvent standardisées et suivent un protocole qui sera le même quels que soient les logiciels utilisés pour les mettre en place.

Ces travaux apportent de puissants outils de vérification de configurations réseau, cependant nous avons identifié deux freins à leur déploiement à grande échelle : le temps de réponse et les erreurs dans les critères de vérification. Les réseaux sont dynamiques, leurs configurations changent fréquemment. Pour qu'un outil de vérification soit utilisable il faut qu'il puisse répondre suffisamment rapidement pour ne pas bloquer une modification importante ou que les modifications à vérifier ne s'accumulent.

Pour que la vérification soit efficace il faut que le réseau et les critères à vérifier soient correctement décrit. Pour cela nous avons commencé à créer un langage permettant une description à haut niveau de la configuration réseau et d'éviter ainsi certaines erreurs (voir le chapitre 4). Puis pour mieux prendre en compte les VNFs, nous avons travaillé sur un formalisme permettant de décrire leurs besoins lorsqu'elles sont conçues pour s'assurer que leurs déploiements sont en accord avec les attentes de leurs concepteurs. Cela peut permettre de limiter les erreurs en supprimant les incohérences dues aux différents acteurs qui agissent de la création au déploiement tout en permettant de réutiliser des invariants déjà définis pour certains types de services (voir le chapitre 5).

Le cas des middleboxes

Les middleboxes ne se conforment pas au modèle de transfert des commutateurs SDN, leur modèle dépend non seulement de leurs règles de transfert, mais aussi de la séquence des paquets précédemment traités. Ceci empêche l'utilisation des techniques de vérification, car le contrôle des comportements des paquets n'est plus centralisé dans le plan de contrôle, mais peut dépendre de l'historique des paquets vus par chaque middlebox. Panda *et al.* [23] proposent des pistes pour inclure les middleboxes dans les modèles de vérification en partant d'un modèle abstrait que les constructeurs pourraient fournir avec leurs équipements. Le modèle abstrait pourrait par exemple pour un système de détection d'intrusion (IDS) se contenter de définir comment sont transférés les paquets selon qu'ils soient considérés comme suspect ou non sans détailler le mécanisme permettant de décider si le trafic est suspect. Spinozo *et al.* [58] prennent ce problème en compte dans le framework proposé.

Limites du classement

La méthode de classement des tailles de réseaux cibles montre quelques limites. Les solutions testées ont des fonctionnalités et des buts différents, certaines peuvent avoir des résultats rapides car elles ne se concentrent que sur un nombre limité de critères (comme Libra qui ne travaille qu'avec des header statiques). Les tests de ces solutions sont rarement effectués dans des mêmes conditions similaires. De plus deux solutions avec un même classement ne sont pas forcément équivalentes. Néanmoins ce classement permet d'avoir un aperçu rapide des fonctionnalités et des réseaux qui peuvent être ciblés par ces outils, pour ensuite tester celles qui correspondent à nos critères.

Travaux	**Techniques**	**Langage**	**Configuration étendue**	**Taille du réseau cible**
NICE [36]	statespace search	api NOX	non	+
Kuai [59]	vérificateur de modèle énumératif distribué	implémentation partielle d'OF	non	++
Beckett et al. [55]	assertions	langage d'assertions	non	++++
Sethi et al. [19]	Murphi Model checking	surcouche faible	non	+++
TASTE [60]	paquets de test sur un réseau émulé	langage de formalisation	non	+++
ConfigChecker [27] [25]	diagrammes de décision binaires et CTL	SCAP	oui	+++++
Basile et al. [33]	modélisation géométrique	pas de détail	oui	++
Header Space Analysis (HSA) [28]	modélisation géométrique	FML	non	+++
Stateful Header Space Analysis (SHSA) [29]	modélisation géométrique	FML	non	+++
FLOWGUARD [31]	Header Space Analysis (HSA)	parefeux	non	++++

Travaux	**Techniques**	**Langage**	**Configuration étendue**	**Taille du réseau cible**
NetPlumber [30]	Header Space Analysis (HSA)	FML prolog haut niveau	oui	+++++
Libra [41]	MapReduce	Snapshot	non	+++++
VeriCon [39]	Floyd-Hoare-Dijkstra + SMT solveur Z3	langage dédié de haut niveau	non	++++
VeriFlow [38]	parcours de graphe	OF et C++	non	++++
Delta-net [43]	parcours de graphe	OF et C++	non	+++++
NetKAT [47]	Kleene Algebra with Test	surcouche à OF	non	+++
WNetKAT (pour Weighted NetKAT) [51]	Kleene Algebra with Test	surcouche à OF	non	+++
Kinetic [52]	machines à états finis (FSM)	Pyretic, surcouche logique à OF	non	+++
Verificare [37]	Model checking avec différents outils externes	Verificare Modeling Language	oui	(dépend du vérificateur externe)
Spinozo et al. [58]	solveur SMT	Langage de description dédié	oui	++
VeriSDN [44]	packet-based pACSR + ST-Graph	pACSR	non	(pas de données)

CHAPITRE 4

FLOWKAT

4.1 Introduction

Le SDN ne couvre pas seulement les aspects réseau, mais aussi la configuration des centres de données et plus généralement le contrôle des équipements virtuels qui peuvent être instanciés pour supporter des fonctions spécifiques (par exemple les fonctions réseau virtuelles). Le SDN offre ainsi la possibilité de couvrir tous les aspects des réseaux : transmission de l'information, stockage et traitement des données.

Avec le SDN, le contrôle des éléments du réseau est effectué par des contrôleurs externes capables de configurer plusieurs éléments à travers le même type d'interface. Il est à noter que même si OpenFlow n'est pas supporté par un équipement, le même concept de contrôleur externe est également pertinent avec d'autres protocoles de configuration déjà existants (par exemple NetConf).

Un seul contrôleur est certainement suffisant pour configurer un petit réseau (comme un réseau de campus composé de quelques commutateurs), mais pour un réseau étendu, plusieurs contrôleurs seront nécessaires (voir [61]) et il sera très rapidement nécessaire de concevoir un plan de contrôle et de gestion d'un réseau basé sur une hiérarchie de contrôleurs. La conception d'un tel plan de contrôle pour les réseaux étendus peut être complexe et sujette aux erreurs. De plus, fournir des abstractions pour l'élaboration d'un tel plan de contrôle est un défi. Dans ce chapitre, nous abordons précisément la conception d'un plan de contrôle pour un réseau étendu. Notre but est d'identifier les exigences d'un langage de programmation.

Nous proposons de conserver certaines composantes de base d'Internet, notamment le routage distribué et le contrôle de bout en bout, et nous introduisons le concept de *Policy Enforcement Point* (PEP) agissant sur les paquets. Nous nous concentrons en particulier sur les exigences pour la conception de politiques agissant non seulement sur les paquets dans un PEP mais aussi sur les flux. Parmi ces exigences, nous considérons qu'un langage spécifique à un domaine fournissant des abstractions de-

vrait permettre aux opérateurs d'exprimer des politiques dynamiques d'une manière concise et intuitive et devrait permettre leur vérification formelle. Nous soulignons également qu'un langage dédié à la programmation du plan de contrôle d'un réseau étendu devrait fournir des abstractions qui gèrent les flux dans le réseau. Sur la base de cette analyse, nous dessinerons les bases de FlowKAT, un langage spécifique au domaine pour programmer le plan de contrôle du réseau étendu.

4.2 Création d'un langage de programmation pour les réseaux étendus

Dans la conception d'un langage de programmation ou plus généralement d'un système d'exploitation réseau pour un réseau étendu, nous devons d'abord identifier les tâches qui doivent être effectuées dans ce réseau. Ces tâches sont généralement organisées selon trois plans :

— plan de données : transfert de paquets, destruction de paquets, mise en file d'attente de paquets, partage de bande passante ou limitation de bande passante, etc.

— plan de contrôle : activation de certaines fonctions dans le plan de données, réservation de bande passante, routage, etc.

— plan de gestion : configuration des éléments du réseau, configuration des fonctions du plan de données (par exemple, réglage des paramètres dans les fonctions de gestion du trafic, etc). Ce plan, qui servait d'interface de gestion aux administrateurs, est inclus dans le plan de contrôle dans beaucoup de représentations du SDN, car ce rôle est assuré par le contrôleur SDN.

Le défi introduit par le SDN est d'englober les trois plans mentionnés ci-dessus dans un plan global mettant en œuvre le système d'exploitation du réseau [62]. En particulier, le système d'exploitation réseau a une vision globale du réseau, depuis la couche physique jusqu'à la couche applicative lorsque certains serveurs manipulant la couche 7 sont implémentés dans le réseau (par exemple, les serveurs proxy).

Si nous partons du principe que les réseaux étendus (les segments radio, les réseaux de collecte et de transit) convergeront vers IP pour transporter l'information segmentée en paquets IP, alors les fonctions du plan de données, de contrôle et de gestion

devraient être orientées IP. Cela implique que la conception d'un système d'exploitation réseau doit être orienté IP dès que le transport de l'information est concerné, notamment pour la configuration des fonctions de gestion du trafic. Cela implique que le système d'exploitation réseau global devrait être capable de configurer tous les éléments du réseau, mais dès que des actions sur l'information doivent être effectuées, ces actions devraient être basées sur IP. Cela n'exclut pas l'introduction de fonctions dans les couches supérieures. Dans ce chapitre, nous nous concentrerons sur le transport de paquets IP.

En ce qui concerne le routage, le cadre du SDN offre la possibilité de configurer des tables transfert de sorte que les paquets arrivant sur un port donné d'un équipement réseau soient transférés vers un port de sortie, éventuellement après un traitement dans l'équipement réseau. Avec ce principe, les paquets peuvent voyager à travers le réseau de leur source à leur destination. Avec OpenFlow, les tables de transfert peuvent être configurées par les contrôleurs OpenFlow tandis que les réseaux IP ont été conçus pour permettre un routage distribué à travers des protocoles de passerelle interne et externe (comme OSPF, ISIS ou BGP).

Fondamentalement, le paradigme SDN propose de revenir au routage centralisé. Bien que cela présente l'avantage de contrôler et d'optimiser le transfert de paquets sur le réseau, le routage centralisé présente de nombreux inconvénients (vulnérabilité, temps de réponse, etc.). C'est pourquoi nous préconisons que le routage des paquets devrait reposer sur un routage distribué, qui est beaucoup plus résistant aux défaillances, plus évolutif et nécessite moins d'échange d'informations. Il convient également de noter que les réseaux IP offrent déjà la possibilité d'effectuer un routage contraint en utilisant MPLS ; il est possible au moyen de MPLS-TE [63] d'indiquer l'itinéraire d'un tunnel MPLS à travers le réseau.

Si l'on admet que le routage se fait par les protocoles IP, un système d'exploitation réseau devrait se concentrer sur le traitement des paquets pour ce qui concerne la couche IP. Cette tâche devrait être divisée en au moins deux sous-tâches :

— la configuration des éléments du réseau (par exemple le paramétrage de certaines fonctions ou la gestion des files d'attente)

— la programmation des fonctions de traitement des paquets qui modifient certains champs des paquets.

Dans ce qui suit, nous nous concentrons sur cette deuxième sous-tâche, qui manipule les paquets. La première tâche couvre la configuration et se situe davantage dans

le plan de gestion du réseau. Pour le développement d'un système d'exploitation de réseau complet, les deux tâches devront être abordées.

Pour la programmation des fonctions de traitement des paquets, nous utilisons le concept de Policy Enforcement Point (PEP). Ce type d'élément apparaît naturellement dans la conception des réseaux. Les paquets entrent dans le réseau à certains endroits, comme les passerelles ou les routeurs de peering. Par exemple, l'architecture des réseaux mobiles est très centralisée et les paquets passent par les P-Gateways avant d'entrer dans le réseau IP (voir la figure 5.1). Même s'il est possible d'envisager que les architectures soient plus distribuées, les paquets entreront nécessairement dans le réseau par certaines passerelles. La même situation se produit dans les réseaux fixes (voir la figure 4.1) où les paquets doivent traverser certaines passerelles, c'est généralement le rôle des serveurs d'accès au réseau à large bande (Broadband Access Server). Les différentes passerelles sont des emplacements remarquables pour la mise en œuvre des fonctions de traitement des paquets.

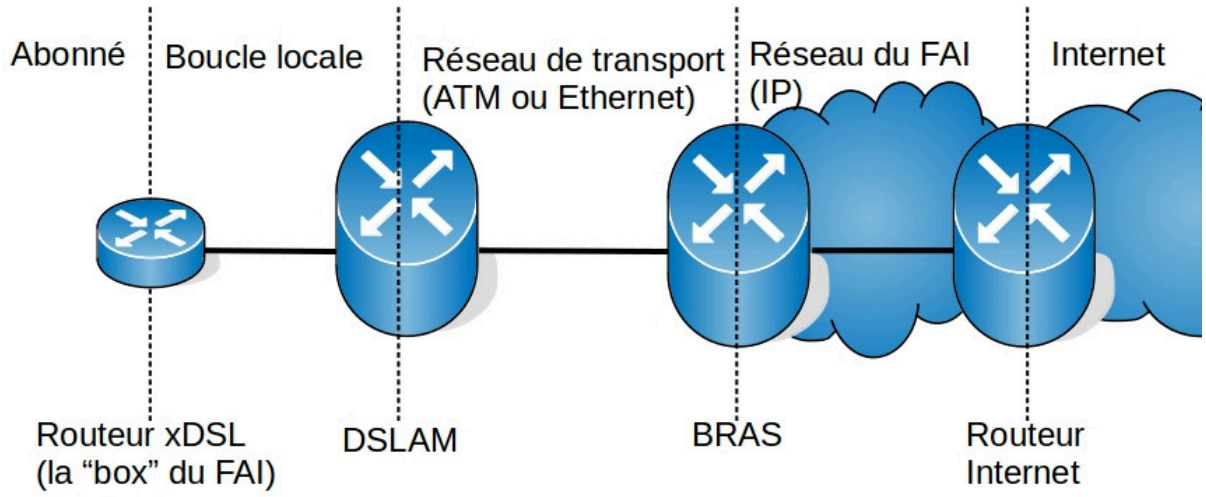

FIGURE 4.1 – Équipements permettant à un client ADSL de se connecter à Internet.

En raison de la grande vitesse des routeurs de coeur de réseau et du routage IP distribué, nous constatons que la portée du SDN dans les réseaux étendus est plutôt à la frontière du réseau pour configurer les passerelles d'accès et les éléments de peering, garantissant que le trafic délivré par d'autres réseaux (par exemple, les réseaux de transit) est conforme à certaines règles.

Dans les Policy Enforcement Points des politiques de traitement des paquets doivent être mises en œuvre. Le défi consiste à concevoir une couche de programmation pour

configurer ces éléments. Il convient de noter que les passerelles d'accès peuvent elles-mêmes être instanciées comme des ensembles de machines virtuelles. Il y a donc en fait deux couches de programmation : une en charge de la configuration des passerelles d'accès (par exemple via OpenDaylight et Openstack) et une autre pour la mise en œuvre des politiques de paquets. C'est cette deuxième couche en charge du transport des paquets que nous traiterons dans ce chapitre.

La programmation du PEP pourrait se faire par la mise en œuvre de politiques agissant sur des paquets, mais le défi est de vérifier la cohérence entre les politiques. L'approche la plus appropriée est de concevoir un langage de programmation fournissant des abstractions qui permettent de vérifier les configurations installés dans les différents PEPs du réseau. Les règles mises en œuvre dans le PEP doivent être cohérentes afin de répondre à certaines exigences de haut niveau de l'opérateur de réseau ou d'un tiers accédant au réseau via une API.

Les PEPs sont configurés par le système d'exploitation réseau, pouvant être hébergés par un seul contrôleur SDN ou répartis sur plusieurs contrôleurs. En fonction des exigences de haut niveau de l'opérateur, le système d'exploitation du réseau configure les politiques sur le ou les PEPs concernés. Notez que ce système d'exploitation doit acquérir la topologie du réseau ainsi que les informations de routage, qui évoluent fréquemment. Il peut également imposer certaines décisions de routage, via l'utilisation par exemple de tunnels MPLS avec routage contraint. La traduction d'une exigence de haut niveau en termes de politiques à mettre en œuvre dans le PEP n'entre pas dans le cadre des travaux présentés dans ce chapitre. Une exigence de haut niveau se traduit dans la pratique par un ensemble de politiques mises en œuvre dans les PEPs du réseau. Ces PEPs sont interconnectés entre eux par des routes déterminées par l'algorithme de routage du réseau.

Cas d'usage

Certains services ont des contraintes de qualité de service (QoS) très fortes qui sont complexes à maintenir, notamment quand le trafic passe par des réseaux qui ne sont pas gérés par le fournisseur de services (comme ceux des FAI). Prenons l'exemple de la recherche vocale en temps réel : répondre à une recherche vocale, comme le propose les assistants, nécessite une connexion rapide et à faible latence de l'appareil d'un utilisateur à la périphérie du réseau du fournisseur de service, et

de la périphérie de son réseau à un de ses centres de données. Une fois à l'intérieur d'un centre de données, des centaines, voire des milliers de serveurs individuels doivent consulter de grandes quantités de données pour établir la correspondance d'un enregistrement audio avec des phrases possibles. La phrase résultante est ensuite transmise à un autre cluster pour effectuer une recherche en consultant un index. Les résultats sont ensuite rassemblés et renvoyés à la périphérie du réseau du fournisseur pour être retransmis à l'utilisateur final.

Répondre à ce genre de requêtes en temps réel implique de coordonner des dizaines de routeurs Internet et des milliers de serveurs à travers le monde, souvent en l'espace de moins d'une seconde.

Agir sur les flux en bordure du réseau en raisonnant avec une vue globale du réseau, comme nous le proposons avec FlowKAT, offre un avantage de taille : choisir de manière dynamique l'endroit par lequel faire sortir le trafic en fonction de mesures de performance des connexions réseau en temps réel. Plutôt que de choisir un point statique pour connecter les utilisateurs simplement en fonction de leur adresse IP, cela permet de sélectionner dynamiquement le meilleur point en fonction des performances réelles. De même, cela permet de réagir en temps réel aux pannes et à la congestion, tant au sein de son propre réseau que sur les autres réseaux par lesquels passe notre trafic.

L'exemple de Google Espresso

Google utilise des techniques du SDN à différents endroits dans son réseau et a récemment déployé Espresso [64] à la bordure de son réseau pour gérer ses connexions de peering. Google présente Espresso comme la partie la plus ambitieuse de sa stratégie SDN, étendant son approche jusqu'à la bordure où il se connecte aux autres réseaux à travers le monde. Cette solution permet aussi à Google de ne pas subir les limitations des protocoles Internet existants qui ne sont pas en mesure d'utiliser toutes les options de connectivité que leur proposent les FAI et ne sont donc pas en mesure d'offrir la meilleure disponibilité et expérience utilisateur à ses utilisateurs finaux.

4.3 Raisonner sur les flux avec FlowKAT

4.3.1 Les flux de paquets

À un certain niveau d'abstraction, un réseau peut être vu comme un graphe où les sommets sont des éléments de réseau reliés par des liens de transmission qui sont les liens du graphe. Du point de vue de la programmation, les nœuds des graphes sont les éléments du réseau qui agissent sur les paquets en les transférant mais aussi en modifiant leurs champs ; ces nœuds sont les PEPs. Le fonctionnement de la transmission de paquet est fixé par le protocole de routage choisis dans le réseau. De plus, le système d'exploitation réseau global doit configurer tous les éléments du réseau pour un traitement approprié des paquets tels que définis par les PEPs.

Le réseau est traversé par des flux qui suivent des routes. Plus précisément, un flux est un ensemble de paquets partageant certaines informations communes, telles que les adresses IP source et destination, les numéros de port source et destination du protocole et le type de protocole. C'est le quintuplet habituel des champs de paquets utilisés par exemple pour mettre en place une gestion des ressources basée sur les flux dans les réseaux. Dans IPv6, un flux pourrait également être identifié par une étiquette de flux. Les flux ont des attributs supplémentaires, par exemple les Differentiated Services Code Points (DSCP) ou le bit ECN (Explicit Congestion Notification). Ces attributs peuvent être modifiés le long du trajet du flux à travers le réseau. Il convient également de noter que certains éléments peuvent être modifiés par des équipements réseau en fonction de leur état. Par exemple, le bit ECN sera modifié par un élément de réseau pour prévenir un problème de congestion.

L'introduction des flux dans Internet a notamment été préconisée par Roberts et Oueslati-Boulahia en proposant une administration réseau capable de raisonner sur des flux [65]. La détection de la logique qui lie des paquets entre eux pour en faire des flux est très utile notamment pour les questions de gestion des ressources. Cela peut par exemple permettre de faire de la prédiction de performances sur la totalité d'un flux, et de qualité de service (par exemple en assurant la même route et les même conditions à tous les paquets d'un même flux). De plus, certains paquets sont intrinsèquement corrélés puisqu'ils appartiennent à la même connexion TCP ou à la même session. Ainsi, pour le partage des ressources réseau, il est plus pratique de travailler avec des flux au lieu des paquets qui ne sont pas en mesure de tenir compte des corrélations intrinsèques entre certains paquets. TCP vise à parvenir à un partage

équitable de la bande passante ; il introduit nativement la notion de flux dans Internet, même si ce réseau est fondamentalement sans connexion. Enfin, le routage IP établit des routes à travers le réseau et les paquets ayant des attributs similaires suivent la même route ; c'est également le cas lorsque le MPLS est utilisé.

Nous avons choisi la programmation réseau au niveau des flux parce que la plupart des procédures de gestion des ressources sont basées sur les flux, même si des actions sont prises sur les paquets. De plus, nous comptons sur le routage IP pour créer des routes à travers le réseau. Les protocoles de routage IP sont très performants et savent, par exemple, éviter les boucles dans le réseau alors que ce problème réapparaît dans la programmation de réseau SDN (voir par exemple [66]). Enfin, pour des raisons d'héritage, il serait difficile et très coûteux de reconstruire complètement Internet. OpenFlow offre néanmoins la grande opportunité de configurer des éléments de réseau (PEP) pour y mettre en œuvre des politiques sur les paquets ou les flux.

4.3.2 Traitements des flux

Les actions et les tests effectués sur les paquets se traduisent naturellement par des traitements sur les flux. Ces procédures peuvent modifier les attributs des flux mais aussi les identifiants de flux, donnant ainsi naissance à de nouveaux flux. Ce dernier cas se produit lors d'un réacheminement imposé par l'opérateur du réseau, par exemple pour atténuer l'impact d'attaques sur le réseau, le trafic sera réacheminé vers un équipement spécialisé pour le filtrer. Une procédure peut également rejeter un flux dans le réseau. Il convient de noter qu'une procédure sur un flux est une composition non commutative de procédures sur des paquets.

Un flux ϕ est un enregistrement composé d'identificateurs et d'attributs, et est désigné par $\phi[[id_1, \ldots, id_n], [at_1, \ldots, at_m]]$. L'ensemble des flux est noté par ϕ et forme un alphabet. Lorsque nous considérons les flux, nous devons distinguer deux ensembles de procédures (comme dans le cas de NetKAT qui est orienté paquet) :

— Catégorie 1 : Les procédures qui sont des prédicats (des tests sur les champs d'un flux)

— Catégorie 2 : Les procédures qui modifient les flux : un flux est transformé en un autre flux en changeant les champs servant à l'identifier.

Certaines procédures sont externes, elles sont définies par l'opérateur réseau. Par exemple, le réglage du DSCP des paquets d'un flux est une procédure invoquée dans

le cadre de DiffServ sur Internet et implémentée dans un PEP (comme une passerelle). De même, certains flux peuvent être bloqués, par exemple dans le cadre du contrôle d'admission. Il convient de noter que le système d'exploitation du réseau devrait permettre la programmation incrémentale des procédures, le réseau ne peut pas être configuré une fois pour toutes et doit pouvoir être continuellement mis à jour.

D'autres procédures sont déclenchées par le réseau lui-même et sont appelées internes. C'est typiquement le cas du marquage ECN qui est invoqué lorsque la congestion est imminente sur le réseau. De plus, le DSCP peut être modulé pour marquer les paquets qui dépassent certaines limites de bande passante (par exemple pour déclasser un service qui consomme trop de bande passante).

4.3.3 Exigence d'un langage de programmation de flux

Pour penser notre langage nous nous sommes appuyés sur NetKAT, un langage conçu pour tester les paquets et modifier leurs champs. NetKAT est le regroupement de l'algèbre de Kleene, qui est utilisée pour penser le réseau comme un automate fini, et de l'algèbre booléenne pour tester les champs de paquets.

Considérons d'abord les procédures externes de la catégorie 1. Ce type de procédure consiste à donner des attributs sans modifier l'identification du flux et à tester les identificateurs ou les attributs.

L'ensemble des affectations d'attributs et des tests sur les flux, équipé du $+,.,*$, 0 et 1, est une algèbre Kleene. L'ensemble des tests équipé de $+,.,\bar{}$, 0, 1 est une algèbre booléenne. Il s'ensuit que les affectations d'attributs et les tests sont une Kleene Algebra with Test (voir le détail de l'algèbre de NetKAT dans la section 5.4.2).

Examinons maintenant les procédures de la seconde catégorie. Une telle procédure p transforme un ensemble de flux en un autre ensemble de flux et peut donc être considérée comme une relation binaire. Les procédures de la première catégorie peuvent alors être considérées comme un cas particulier des procédures de la seconde catégorie. L'ensemble des procédures de ces catégories forment une KAT. Les étapes suivantes dans le développement de FlowKAT seraient :

— Inclure les procédures internes (c'est-à-dire celles déclenchées par le réseau) dans le modèle équationnel.

— Prendre en compte le caractère dynamique de la configuration réseau dans le modèle équationnel.

4.4 Les concepts du langage

Bien que se voulant détaché d'OpenFlow, afin de pouvoir garder le même langage quel que que soit le protocole de communication, nous en sommes restés assez proches. Les fonctionnalités que nous avons implémentées dans notre langage ont dû se limiter à celles prises en compte par OpenFlow, car les constructeurs s'y réfèrent pour assurer la compatibilité de leurs commutateurs. Nous sommes donc partis des possibilités décrites dans le document *OpenFlow Switch Specification* pour bâtir un langage permettant de les exploiter. Nous avons priorisé l'implémentation des options de configuration qui nous étaient utiles pour les vérifications que nous souhaitions effectuer. Certaines options, comme celles qui concernent les relations entre le commutateur et le contrôleur, n'ont pas encore étaient implémentées.

Comme le montre l'extrait du meta-modèle du langage 4.2, il permet de créer une configuration complète. Chaque équipement y est nommé et ses caractéristiques sont décrites. Les règles réseau qui s'y appliquent sont composées des conditions de déclenchement, des différentes actions effectuées sur les paquets ainsi que l'ordre dans laquelle elles doivent être exécutées.

4.4.1 La topologie

La configuration commence par la description des équipements réseau puis dans un second temps par la définition des régles qui s'y appliquent. L'équipement ainsi que ses ports sont nommés et les connexions avec les ports d'autres équipements sont listées.

```
device equipement1
{
    port p1
        max_speed 10000
    port p2
        Connection AdminR1 { p1 -> routeur1.p3 }
```

Listing 4.1 – Description d'équipement avec FlowKAT.

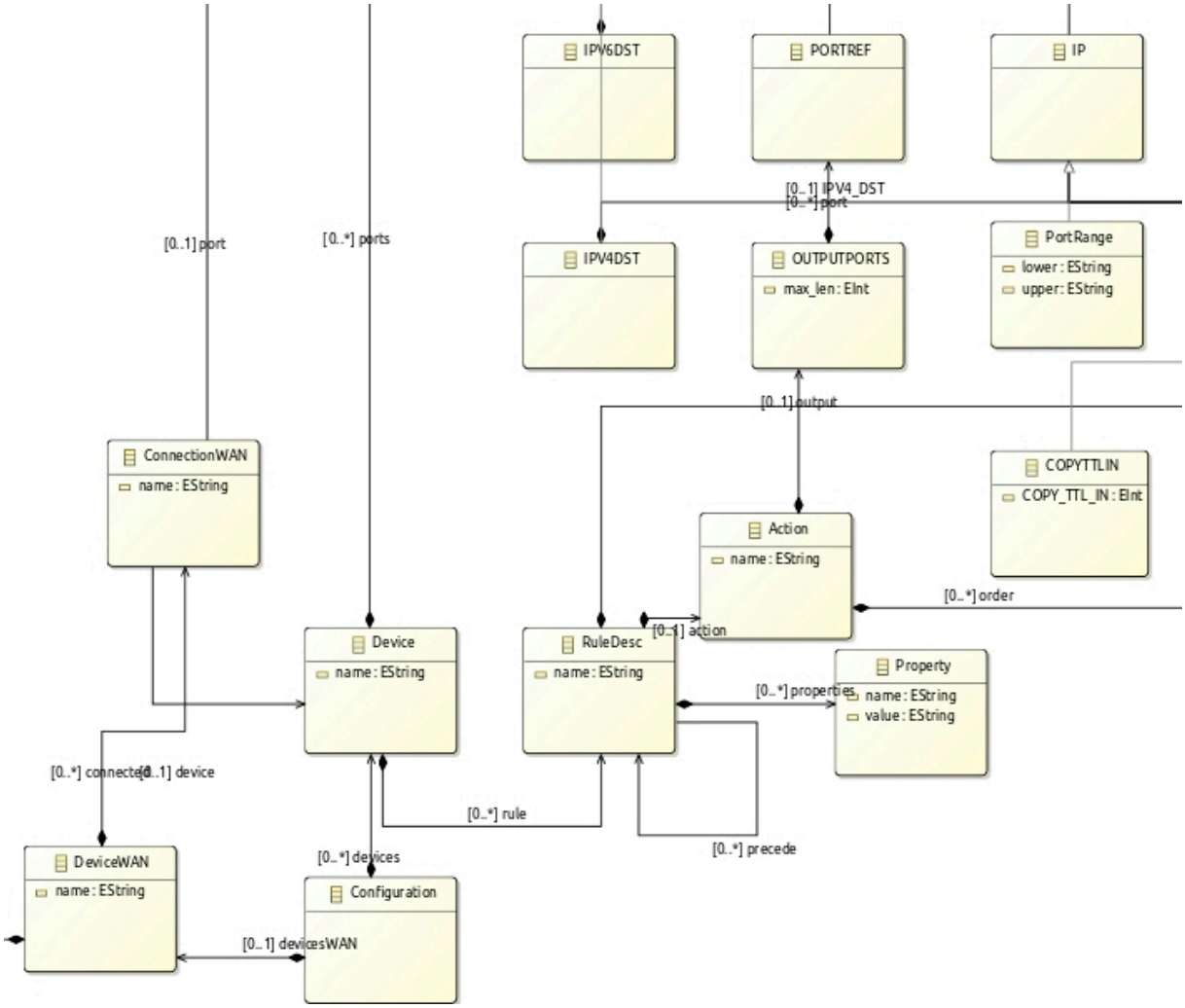

FIGURE 4.2 – Extrait du meta modèle de FlowKAT.

Réseaux externes

En agissant uniquement dans les PEPs, nous sommes amenés à interagir avec des réseaux que nous ne contrôlons pas. Il peut s'agir de réseaux internes ou de réseaux appartenant à d'autres entités.

Dans le premier cas, il est possible d'obtenir des informations comme la capacité des équipements ou leur charge (étant donné que le trafic passe par les PEPs avant d'y entrer).

Pour effectuer des vérifications il faut prendre en compte cet environnement dont nous ignorons les actions. Même si nous savons que le flux atteindra un des PEPs que nous contrôlons en sortie, nous ne connaissons pas le chemin qu'il empruntera ni quelles modifications seront effectuées sur ses paquets.

Il est possible d'influer sur le routage (par exemple grâce aux tunnels MPLS présentés précédemment) et de s'assurer que les données qui nous intéressent ne soient pas modifiées. Cela permettrait de diminuer les calculs nécessaires aux vérifications, mais cela nous priverait de certains avantages liés à la délégation du routage aux protocoles IP (comme le routage distribué, voir 4.2).

Nous avons accès à moins d'information dans le cas des réseaux externes situés à l'extérieur de la bordure des PEPs, mais il est possible d'avoir des informations en effectuant des mesures (par exemple sur le temps que met une requête à obtenir une réponse). Ces informations pourront être utilisés lors des choix de route. FlowKAT permet de prendre en compte ces informations dans la description de la topologie. Un réseau externe est vu comme un seul équipement et les différentes connexions (différentes adresses IP) vers ce réseau sont vus comme des ports de cet équipement.

```
deviceExtern equipementExt1
{
    port pVisio
    port pArchive
```

Listing 4.2 – Description d'équipement avec FlowKAT.

4.4.2 Les règles réseau

Chaque règle peut être nommée pour faciliter leur agencement logique. L'ordre de priorité peut être définit en fonction des autres règles comme le montre cet exemple :

```
Rule secIP
    {    }
Rule stream precedes secIP
    {    }
```

Listing 4.3 – Ordre des règles avec FlowKAT.

Les conditions de déclenchement

La première partie d'une règle énonce ses conditions de déclenchement, lesquelles peuvent être regroupées au sein d'un Flow, qui sera nommé. Les critères sont ensuite listés en écrivant le nom de la propriété et sa valeur.

```
Flow FluxStreaming
        Destination IPv4 90.10.0.1/24
        UDP RTSP or TCP RTSP
```

Listing 4.4 – Conditions de déclenchement avec FlowKAT.

Il est possible de définir différentes valeurs pour une même propriété en les séparant avec le mot-clé « OR » comme fait en 4.4. Les ports peuvent être définis en utilisant des plages ou les noms des protocoles ciblés.

Les actions

Les actions à réaliser sont décrites selon le même principe : le nom de l'action suivi de ses paramètres.

```
Action
                Outport port p1
                Push VLAN tag 15
```

Listing 4.5 – Actions avec FlowKAT.

Il est possible d'utiliser toutes les conditions de déclenchement et toutes les actions décrites dans le document *OpenFlow Switch Specification*.

4.5 Expérimentation

Nous avons réalisé une ébauche du futur langage avec comme objectif d'aider les administrateurs à écrire les règles de leur réseau tout en nous permettant de modéliser le comportement du réseau. Comme expliqué précédemment nous nous sommes appuyés sur le formalisme NetKAT pour apporter des preuves formelles sur le comportement du réseau.

Voici un exemple permettant d'aborder plusieurs spécificités de FlowKAT.

```
device switch1
{
    port p1
    port p2
        Connection Admins1r1 { p1 -> routeur1.p3 }
        Connection s1Ext { p2 -> equipementExt1 }


    Rule SecRule
    {
    Flow ping0
        Protocol ICMP or ICMPv6
        Action drop
    }

    Rule stream precedes SecRule
```

```
    {
    Flow Streaming1
        Destination IPv4 90.10.0.1
        UDP RTSP or TCP RTSP
        Action
                outport port p1
                Push VLAN tag 15
     }
}
```

Listing 4.6 – Exemple de code FlowKAT.

En plus de décrire le comportement des équipements nous voulions pouvoir décrire la façon dont ils sont connectés entre eux. Comme nous souhaitons agir dans les Policy Enforcement Points, il nous a semblé intéressant de permettre de décrire aussi les connexions aux réseaux que nous ne contrôlons pas. Il est possible de les nommer et de leur joindre un nombre de paramètres limité. Lors de notre vérification, les réseaux externes sont considérés comme des liens reliés à un équipement transférant tout le trafic. Nous bénéficions ainsi d'une description de la topologie que nous utiliserons pour effectuer nos vérifications.

L'exemple 4.6 décrit un commutateur avec les règles qui s'y appliquent mais aussi avec ses ports et les équipements qui lui sont connectés.

Les règles et les équipements sont nommés pour faciliter le raisonnement et la rédaction. De plus, il est aussi possible de définir si une règle ou un groupe de règles doit être appliqué avant ou après une autre règle en utilisant leurs noms. Cette solution nous a paru plus simple que le système de numérotation croissant définissant la priorité des règles dans les commutateurs. Ce système obligeait à garder en tête les numéros donnés aux autres règles pour définir la valeur de la priorité d'une nouvelle règle. Cela demandait parfois de décaler les numéros assignés à de nombreuses règles pour en intercaler une nouvelle dans l'ordre de priorité déjà existant (voir les travaux de Basile *et al.* [33] pour une autre approche de cette problématique). Ces numéros, utilisés dans les équipements OpenFlow, seront calculés lors de la compilation en fonction des ordres ainsi décrits.

Dans notre exemple 4.6, une règle de sécurité rejetant les *ping* sera appliquée avant la règle traitant le flux de streaming vidéo. Le flux de streaming, défini par le

protocole utilisé et par leur destination, sortira sur le port 1 du commutateur.

Le mot clé "OR" permet de décrire différentes possibilités de déclenchement concernant une même propriété sans avoir à rédiger une nouvelle règle pour chacune d'elles. Ici, la règle « ping0 » traite à la fois le trafic ICMPv4 et ICMPv6, l'implémentation dans un équipement OpenFlow créera deux règles comme le montre l'exemple 4.7.

Pour faciliter la rédaction des règles nous avons voulu faire un langage de plus haut niveau qu'OpenFlow, qui a été conçu pour la communication entre équipements, et utilisés par certains contrôleurs (voir 4.11). Le même code aurait pu être écrit comme ceci avec OpenFlow :

```
{"flow":
   {  "flow-name": "ping0",
      "id": 1,
      "table_id": 0,
      "priority": 2,
      "instructions":
      {"instruction":
         [ {  "order": 0,
              "apply-actions":
              { "action":
                 [ {  "order": 0,
                      "drop"   }  ] } } ] },
      "match":
      {"ICMPv4",
      "strict": false,
      "idle-timeout": 0 },
      {"flow-name": "ping1",
      "id": 2,
      "table_id": 0,
      "priority": 2,
      "instructions":
      {"instruction":
         [ { "order": 0,
             "apply-actions":
```

```
            {"action":
               [ { "order": 0,
                     "drop" } ] } } ] },
    "match":
    {"ICMPv6",
    "strict": false,
    "idle-timeout": 0 },,
  { "flow-name": "streaming1",
    "id": 3,
    "table_id": 0,
    "priority": 1,
    "instructions":
    { "instruction":
       [ {"order": 0,
          "set-field":
          {"protocol-match-fields":
             { "vlan-tag": 15 } } },
       {"order": 1,
        "apply-actions":
          {"action":
             [ { "order": 0,
                  "output-action":
                   {"output-port": "1"} } ] } } ] },
    "match":
    {"in-port": "1",
      "ethernet-match":
       {"ethernet-type":
          { "type":0800 },
          "udp-dst":554  },
       "ipv4-destination": "90.10.0.1"},
    "strict": false,
    "idle-timeout": 0} }
```

Listing 4.7 – Extrait d'une configuration OpenFlow.

```
ovs-ofctl add-flow switch1 priority=2,dl_type=0x0800,
    nw_proto=1,icmp_type=0,
    actions=drop

ovs-ofctl add-flow switch1 priority=2,dl_type=0x86dd,
    nw_proto=58,icmp_type=0,
    actions=drop


ovs-ofctl add-flow switch1 priority=1,dl_type=0x0800,
    nw_proto=6,tp_dst=554,nw_dst=90.10.0.1,
    actions=mod_vlan_vid:15,output:1

ovs-ofctl add-flow switch1 priority=1,dl_type=0x0800,
    nw_proto=17,tp_dst=554,nw_dst=90.10.0.1,
    actions=mod_vlan_vid:15,output:1

ovs-ofctl add-flow switch1 priority=1,dl_type=0x0800,
    nw_proto=6,tp_src=554,nw_dst=90.10.0.1,
    actions=mod_vlan_vid:15,output:1

ovs-ofctl add-flow switch1 priority=1,dl_type=0x0800,
    nw_proto=17,tp_src=554,nw_dst=90.10.0.1,
    actions=mod_vlan_vid:15,output:1
```

Listing 4.8 – Commandes OpenFlow pour OpenVSwitch.

```
if (ethTyp=0x800 and ip4Dst = 90.10.0.1 and
  (tcpDstPort = 554 or tcpSrcPort = 554 or
  udpDstPort = 554 or udpSrcPort = 554) and
  (ipProto=0x06 or ipProto=0x17 ))
```

```
then (port := 1; vlan :=15)

if (ethTyp=0x800 or dl_type=0x86dd) and
  (ipProto=0x01 or ipProto=0x58)
then drop
```

Listing 4.9 – Exemple de code NetKAT.

Nous avions aussi identifié de possibles améliorations lors de notre utilisation de NetKAT (par exemple la possibilité d'utiliser des plages pour les ports). La comparaison

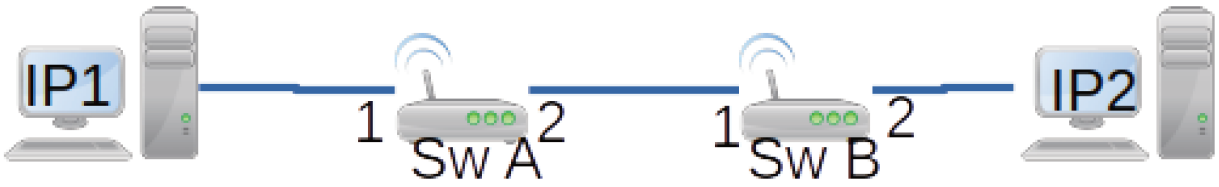

FIGURE 4.3 – Réseau composé de 2 clients communiquant via 2 commutateurs.

de l'écriture d'une règle simple permet de constater les facilités d'écriture et de lecture que FlowKAT peut apporter par rapport à des langages proches d'OpenFlow (ici celui de l'orchestrateur Ryu).

Dans notre exemple nous allons comparer une règle s'appliquant au trafic à destination de la machine IP2 (voir le schéma 4.3) et utilisant le protocole SSH ; elle envoie les paquets via le port numéro 2 avec l'ajout de l'identifiant de vlan 15.

```
        Destination IPv4 IP2
                Protocol ssh
        Action
                outport port 2
                        Push VLAN tag 15
```

Listing 4.10 – Solution avec FlowKAT.

```
match = self.ofparser.OFPMatch(ipv4_dst=IP2, tcp_dst=22)
actions = output : 2
inst = [parser.OFPInstructionActions(
    ofproto.OFPIT_APPLY_ACTIONS, actions)]
```

```
mod = parser.OFPFlowMod(datapath=datapath,
    vlantag=15, match=match,instructions=inst)
datapath.send_msg(mod)
```

Listing 4.11 – Solution avec Ryu.

```
(type = SSH; destination = IP2; Port ← 2; Vlantag ← 15)
```

Listing 4.12 – Solution avec NetKAT.

4.6 Les outils du langage

La rédaction de ce langage a été faite à l'aide de Xtext[1] qui offre une surcouche à Java pensée pour la conception de langages dédiés. Pour spécifier FlowKAT, nous avons créé sa grammaire puis généré les classes du modèle objet correspondant. La compilation génère des instructions en OpenFlow à destination des commutateurs.

Notre langage dispose de différents outils pour éviter les erreurs de compilation.

La syntaxe a été pensée pour essayer d'éviter certaines erreurs venant d'un décalage entre ce que veut écrire la personne qui est en charge de la configuration et l'effet de sa configuration, par exemple en permettant de nommer des éléments ou en spécifiant explicitement l'ordre des règles au lieu de les numéroter. De plus cette syntaxe de haut niveau facilite la relecture du code par une tierce personne qui le comprendra plus facilement et pourra faire des remarques dessus.

Grâce à Xtext, nous avons pu proposer des outils qui facilitent la rédaction (comme la coloration syntaxique ou l'auto-complétion du code) ainsi que des vérifications lors de la compilation comme le type-checking qui s'assure que les politiques sont correctement rédigées.

Outre ces facilités d'écriture et ces analyses statiques, il est possible d'utiliser un outil de vérification pour s'assurer que la configuration écrite satisfait certains critères comme l'atteignabilité entre deux points du réseau. Notre langage s'est appuyé sur le formalisme de NetKAT et peut facilement être traduit dans la syntaxe utilisée par NetKAT afin de profiter de ses capacités de vérification tout en bénéficiant avant sa traduction des facilités que nous venons de détailler.

1. https://www.eclipse.org/Xtext/

4.7 Conclusion

Dans ce chapitre nous avons abordé la façon dont les concepts du SDN pourraient être appliqués aux réseaux étendus (WAN). Nous proposons de conserver quelques éléments de base d'Internet, notamment le routage distribué et le contrôle de bout en bout, et nous utilisons le concept de PEP pouvant agir sur les flux de paquets aux points d'entrée du réseau.

À partir de cet examen, nous avons identifié les exigences d'un langage de programmation pour mettre en œuvre des politiques de paquets dans les PEPs, en particulier nous pensons qu'un tel langage doit fournir une abstraction pour les flux réseau.

Cette solution facilite la mise œuvre du SDN au sein du WAN en permettant aux deux technologies de cohabiter, cela évite des investissements très coûteux pour changer tous les équipements ou d'attendre qu'ils soient renouvelés. Mais cette concession permet quand même de traiter tout le trafic traversant le réseau.

Pour écrire les politiques de ces réseaux nous avons crée FlowKAT. Ce langage facilite la rédaction de configuration et leur relecture grâce à une syntaxe adaptée et les analyses statiques de son compilateur permettent de s'assurer que les règles sont correctement écrites. Nous l'avons conçu en nous appuyant sur le formalisme de NetKAT ; il est possible d'utiliser ce dernier afin de vérifier que la configuration créée répond bien aux attentes en prouvant formellement certains critères comme l'atteignabilité, l'isolation ou l'absence de boucles.

Cependant il existe maintenant des outils de vérification plus performant que NetKAT pour travailler sur des réseaux de grandes tailles comme Delta-net [43] qui permet de prouver formellement certaines propriétés sans que l'explosion de l'espace d'état ne rende le temps réponse trop long.

Après avoir créé ce langage nous permettant de décrire et vérifier une configuration, nous avons voulu poursuivre ces travaux en cherchant à formaliser les critères qui avaient besoin d'être vérifiés pour s'assurer de la cohérence entre les règles écrites et les besoins des services qui utilisent le réseau.

CHAPITRE 5

CONTRATS DE COMPORTEMENT RÉSEAU

5.1 Introduction

La virtualisation des fonctions réseau incite les opérateurs à modifier leurs modèles économiques. Au lieu d'offrir la connectivité et le transport de l'information, ils deviennent des fournisseurs de solutions réseau et doivent exploiter des infrastructures de stockage distribué et de calcul en plus de leur rôle traditionnel de fournisseur réseau.

La décomposition d'une VNF globale en plusieurs composants (ou micro-services [67] [68]), qui peuvent être instanciés sur des serveurs distants, pose le problème de leur interconnexion. Ceci est profondément lié à la méthode de programmation du réseau. Avec l'émergence du SDN, il devient possible de programmer un réseau au moyen de contrôleurs externes et donc de configurer complètement la façon dont les différentes entités communiquent entre elles.

Au-delà des questions de performance, qui posent le problème du placement des différents composants d'une VNF dans le réseau afin d'atteindre les objectifs de qualité de service, les opérateurs réseau devront résoudre dans le futur le problème de la cohérence entre les besoins de connectivité des composants d'une VNF et la configuration du réseau.

Un tel problème n'est pas critique pour le moment parce que la plupart des VNFs sont encore monolithiques. Les VNFs sont en effet constituées de plusieurs composants, qui peuvent être hébergés dans des machines virtuelles ou des conteneurs, mais la plupart du temps ils sont dans le même centre de données. La répartition peut permettre par exemple d'être au plus proche géographiquement de la demande pour réduire la latence de certaines fonctionnalités. La scission d'une VNF sur des sites distants soulève la question de la cohérence entre la configuration du réseau et les call flow de la VNF.

Des problématiques proches se posaient avant les VNFs, par exemple pour vérifier

un service composé de plusieurs services web [69], nous avons utilisé les outils de ce nouvel environnement, pour proposer une solution au problème de cohérence entre les call flows des services déployés et la configuration du réseau. Nous proposons une approche, où le call flow est attaché à la définition de la VNF en tant que contrat de comportement réseau.

Ensuite, sur la base d'un modèle de déploiement des micro-services d'une VNF, nous vérifions la cohérence entre les besoins réseau de la VNF et les règles SDN du déploiement.

5.2 Décomposition d'une VNF en micro-services

Une VNF est une suite logicielle complète composée de plusieurs modules et accomplissant un certain nombre de tâches. La tendance actuelle est de décomposer une VNF sous la forme de micro-services [67, 70, 68, 71, 72] interagissant entre eux, chaque micro-service exécutant un ensemble de tâches élémentaires. Une fois qu'une VNF est décomposée en micro-services, la tâche suivante est de les instancier sur une architecture virtualisée.

Avant de passer à la phase d'instanciation, soulignons qu'une VNF peut être constituée de briques logicielles développées par des entités indépendantes et déployées par un gestionnaire d'infrastructures (détails dans la figure 2.4).

L'écart entre les différents métiers impliqués peut provoquer des malentendus ou des erreurs de configuration entre l'architecture logique de la VNF [73] et la configuration réseau réelle. Les services sont souvent conçus sans tenir compte de l'utilisation du réseau par d'autres, ce qui peut entraîner des problèmes tels que la perte de la sécurité des communications, les temps de latence, la congestion, etc.

Il existe plusieurs outils de gestion de configuration et orchestrateurs réseau qui permettent aux opérateurs de décrire rapidement les architectures des VNFs. On peut citer Ansible [1], Chef [2], Puppet [3] ou Docker Compose [4]. Mais il n'existe pas de norme commune de description d'architecture pour ces outils de gestion de configuration. Actuellement, à notre connaissance, seuls les orchestrateurs suivant l'architecture NFV

1. `https://www.ansible.com/`
2. `https://www.chef.io/`
3. `https://puppet.com/`
4. `https://docs.docker.com/compose/`

MANO (voir chapitre 6) proposent un moyen de déclarer qu'un service doit pouvoir communiquer avec un autre service.

Les outils de gestion de configuration permettent aux opérateurs d'implémenter rapidement des architectures NFV. L'utilisation d'un orchestrateur, ne deviendra intéressante que si l'architecture devient plus complexe. Le temps consacré à la mise en place et à la configuration de l'orchestrateur peut dépasser celui du déploiement des fonctions constituant les VNFs.

Nous pensons que les langages de description d'architecture utilisés par les gestionnaires de configuration manquent de l'abstraction nécessaire pour spécifier les contraintes des VNFs concernant la configuration du réseau. Dans ce qui suit, nous allons examiner comment utiliser ces gestionnaires de configuration pour déployer les VNFs et montrer les problèmes qu'ils peuvent provoquer pendant la configuration du réseau. Nous accorderons une attention particulière à la façon dont les micro-services d'une VNF sont interconnectés.

Nous pouvons identifier deux aspects à vérifier pour s'assurer de l'exactitude de la mise en œuvre d'une VNF :

— La correction sémantique : Les différents micro-services échangent des messages entre eux selon un protocole donné. La vérification sémantique d'une VNF consiste à vérifier l'exactitude de la mise en œuvre du protocole. Ceci peut être fait hors ligne lors de l'écriture de la VNF sous forme de micro-services. Les outils habituels de vérification de modèles [74] peuvent être utilisés pour vérifier la cohérence comportementale entre les services [75].

— La fidélité de l'échange d'information : Lorsque des micro-services sont implémentés sur différents serveurs, ils doivent communiquer entre eux à travers le réseau. Les micro-services sont hébergés par un serveur relié au réseau. Le point clé est de vérifier que les messages sont correctement échangés entre les micro-services.

La première question est pertinente pour la mise en œuvre d'une VNF au moyen de micro-services consiste à vérifier sémantiquement la mise en œuvre correcte d'un protocole. Mais nous nous sommes concentrés sur la deuxième question que nous pensons moins bien outillée.

5.3 Exemple de l'implémentation d'un vEPC

5.3.1 Les différentes fonctions d'un vEPC

L'EPC est l'ensemble des fonctions d'un coeur de réseau mobile 4G. Il est composé de fonctions qui agissent sur le plan données et sur le plan de contrôle telles que décrites dans la figure 5.1. Il s'occupe des différentes fonctions pour l'accès radio cellulaire et WiFi. Dans ce qui suit, nous nous concentrons sur l'accès 4G constitué d'un HSS (Home Subscriber Server), d'un S/PGW (Serving/Packet data network Gateway) et d'un MME (Mobility Management Entity). Pour prendre en compte l'accès non cellulaire, des modules supplémentaires seront nécessaires (par exemple, une Evolved Packet Data Gateway ou des contrôleurs WiFi).

Le défi de la virtualisation est précisément d'implémenter des fonctions de contrôle et du plan de données sous la forme de modules qui peuvent être implémentés et exécutés sur du matériel générique.

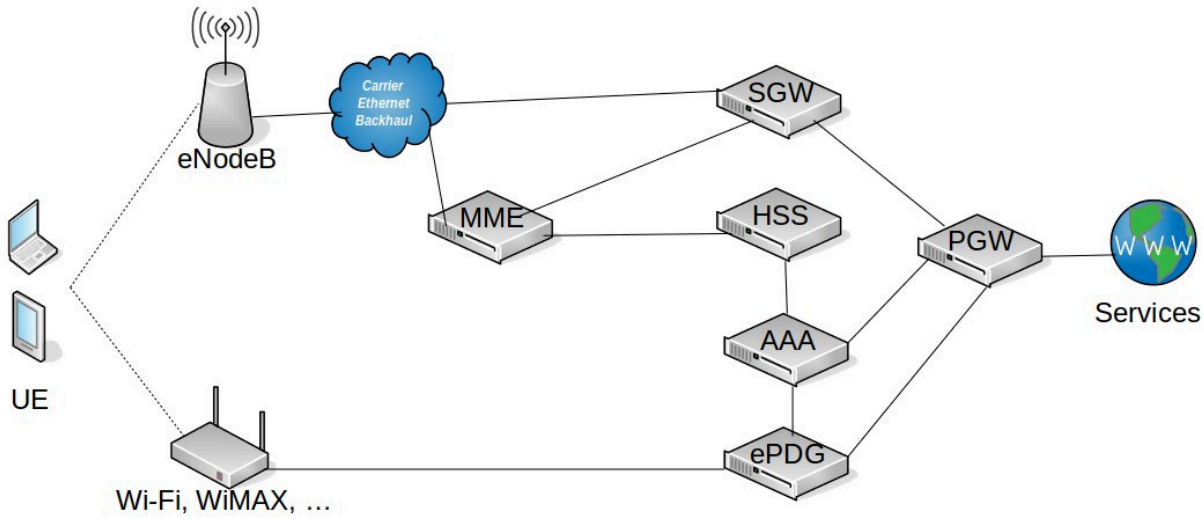

FIGURE 5.1 – Modules du plan de données et de contrôle pour l'accès radio . [76]

5.3.2 Raccordement et authentification

Pour se connecter au réseau radio, un équipement utilisateur identifié par sa carte SIM se connecte à une station de base eNodeB. Avant d'autoriser un équipement

d'utilisateur (UE) à accéder au réseau, son identité est vérifiée par le MME grâce à une base de données (le HSS) qui décrit les entités du réseau et contient la liste des utilisateurs, les droits et permissions associés ainsi que les sessions en cours .

Par souci de concision dans nos prochains exemples, nous nous concentrerons sur ces premières étapes.

Radio Bearer

La mobilité d'un UE est gérée par le plan de contrôle 4G. Un paquet est routé en utilisant une adresse qui est généralement liée à un emplacement fixe. La solution choisie pour la 4G est de faire passer le trafic par un Packet Gateway (PGW). Lorsque l'UE se déplace et change de station de base, la PGW est informée de son nouvel emplacement par le MME.

Selon la norme LTE, le S/PGW doit fournir une adresse IP au client. Cette étape est divisée en deux parties distinctes : d'une part, la demande puis la transmission de l'adresse IP et d'autre part, le choix de cette adresse IP. Ce choix peut être fait directement par le S/PGW s'il dispose d'une base de données avec le pool d'adresses IP ou par l'interrogation d'un serveur DHCP. L'adresse IP assignée peut être retournée au client via le MME.

Le déroulement du call flow du raccordement d'un UE est représenté dans la figure 5.2. Les différents éléments de l'EPC doivent échanger des informations qui sont transmises par l'intermédiaire du réseau. Dans les réseaux actuels, tous les serveurs (MME, HSS) et les éléments réseau du plan de données (eNodeB, S/PGW) ont des adresses IP fixes et le routage est statique. Le défi amené par le NFV est d'implémenter dynamiquement ces fonctions dans les centres de données.

5.3.3 Questions relatives à la mise en œuvre

Nous avons testé le déploiement d'un EPC open source (comme illustré à la figure 5.3) dans le laboratoire Network Architecture à b<>com : Open Air Interface (OAI) EPC [5] [77]. Pour déployer ce vEPC avec un simple gestionnaire de configuration, nous avons utilisé des scripts Ansible pour définir les rôles et un fichier Vagrant pour monter et connecter toutes les ressources. Un rôle est défini par une liste de fichiers à installer

5. `http://www.openairinterface.org/?page_id=864`

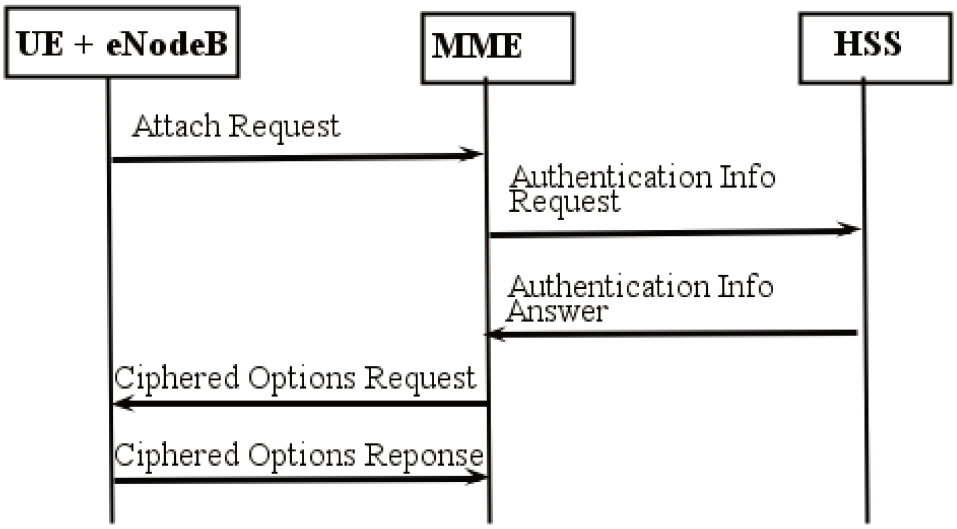

FIGURE 5.2 – Call flow du rattachement d'un UE.

ou à importer et une liste de valeurs qui changeront pour chaque ressource associée au rôle (nom, IP....). Le même type d'équipement joue généralement plusieurs rôles, un fichier les répertorie et transmet aux rôles associés les variables à appliquer pour chaque ressource. La configuration des différents services ainsi que les tables de routage sont fixées et écrites dans des fichiers qui sont copiés par Ansible après le lancement des VMs.

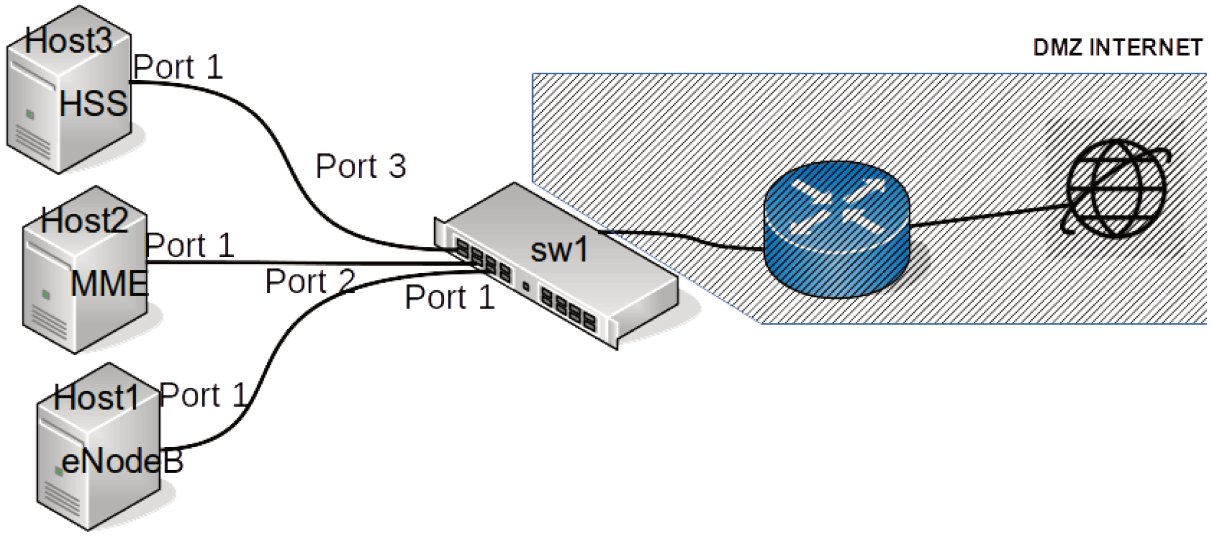

FIGURE 5.3 – Micro-services nécessaires au rattachement d'un UE.

Au total, 1047 lignes de code Ansible et 344 fichiers de configuration ont été néces-

saires pour déployer les six micro-services qui forment le vEPC ainsi que les équipements réseau et leur contrôleur. Aucun mécanisme permettant d'apporter des preuves formelles pour s'assurer que la configuration du réseau permet les échanges nécessaires n'a été mis en place, mais seulement des tests fonctionnels. Lorsque le déploiement a été effectué sur une architecture différente de celle du laboratoire de développement (c'est-à-dire lors de l'implémentation du vEPC sur la plate-forme d'un autre projet), des problèmes réseau sont apparus (boucles).

Les tests effectués au moment de l'écriture des configurations ne sont valables que sur le réseau testé à un moment donné. Lorsque le déploiement a lieu dans un autre environnement ou par exemple lorsque de nouveaux services sont déployés, de nouvelles erreurs peuvent apparaître.

Cet exemple simple nous a incité à développer une méthodologie pour tester la cohérence entre la décomposition d'une VNF en termes de micro-services et son déploiement dans l'environnement multi-cloud. Nous ne traitons ici que de l'échange d'informations entre les micro-services, et non de l'exactitude de l'implémentation du protocole.

5.4 Solution proposée

L'approche NFV promet de pouvoir encapsuler les fonctions de réseau dans des briques réutilisables qui fonctionnent de manière prévisible sans que les opérateurs de réseau n'en connaissent les détails. Pour détecter les incohérences entre les besoins réseau d'une VNF et la configuration réseau, cette section présente un modèle étendu de VNF avec des contrats. Elle montrera comment ces contrats pourraient être utilisés en combinaison avec des outils existants, tel que NetKat, pour vérifier la cohérence globale du déploiement de VNF. Nous définissons ci-dessous les quatre étapes principales de notre approche et illustrons chacune de ces étapes sur l'étude de cas du vEPC :

1. Définir un modèle de VNF réutilisable (section 5.4.1) ;
2. Définir *la topologie étendue* du réseau (section 5.4.2) ;
3. Définir un modèle de déploiement de VNF (section 5.4.3) ;
4. Vérifier la cohérence avec le modèle de déploiement de VNF (subsection 5.4.4).

5.4.1 Étape 1 - Modèle de VNF réutilisable

Commençons par présenter une définition abstraite d'une VNF.

Definition 1 *Une VNF est définie comme :*

- *Un ensemble de micro-services* $S = \{s_1, s_2, ..., s_i\}$.
- *Un ensemble de Hubs* $H = \{h_1, h_2, h_i\}$*, où* H *représente un lien logique entre un ensemble de services, pouvant être complété par des métadonnées. Par exemple,* $h1 = \{services : \{s_1, s_2\}, secured : false\}$ *et* $h_2 = \{services : \{s_2, s_3\}, secured : false\}$ *signifie que* s_1 *peut envoyer des messages à* s_2*,* s_2 *peut envoyer des messages à* s_3 *mais* s_1 *ne peut échanger des messages avec* s_3 *et ne peut pas voir les messages entre* s_2 *et* s_3*.* $h_2 = \{services : \{s_3, s_4, s_5\}, secured : true\}$ *signifie que les communications entre* s_3, s_4, s_5 *doivent être chiffrées.*
- *Un contrat de comportement réseau (voir definition 2).*
- *Un ensemble d'implémentation de micro-services.*

Dans la définition ci-dessus, nous avons utilisé le concept de contrat de comportement réseau défini ci-dessous.

Definition 2 *Le contrat de comportement réseau est défini par un ensemble de messages. Chaque message est un ensemble* $msg = (s_1, s_2, order, \{a_1, a_2, ..., a_n\})$ *où :*

- s_1 *est le micro-service source du message,*
- s_2 *est le destinataire du message du micro-service,*
- $order$ *est le numéro correspondant à l'ordre du message dans le diagramme de séquence correspondant,*
- $a_1, a_2, ..., a_n$ *sont un ensemble de règles réseau (ajout, suppression, modification) qui peuvent être déclenchées avant la réception du message ou après son émission (voir Définition 3).*

L'ordre des messages dans le call flow construit un ordre partiel. Le numéro d'un message est donné en fonction de sa position dans l'arbre. Le nœud racine correspond au micro-service de départ d'un diagramme de séquence.

Definition 3 *Une action de règle de routage a est définie en utilisant la syntaxe NetKAT, elle a un nom et elle gère les noeuds virtuels et les ports virtuels d'une VNF. Un nœud virtuel ($VN(s_i)$) mappe les nœuds où le micro-service s_i est déployé. Un port virtuel ($VP_k(s_i)$) mappe le port k du noeud où le micro-service s_i est déployé.*

Illustration de la connexion à un vEPC

Pour illustrer ce formalisme, un vEPC peut être caractérisé ainsi :

Proposition 1 *Un vEPC peut être décris comme un ensemble* $S = \{eNodeB, MME, HSS\}$ *avec :*

— *un ensemble de hubs* $H = \{h_1, h_2\}$ *avec*
$h_1 = \{services : \{eNodeB, MME\}, secured : false\}$
$h_2 = \{services : \{MME, HSS\}, secured : false\}$

— *un contrat de comportement réseau*
$BC = \{Attach\_Request, Auth\_info\_Req, Auth\_info\_Ans, Ciphered\_Req, Ciphered\_Rep\}$, *avec :*
$Attach\_Request = \{eNodeB, MME, 1\}$,
$Auth\_info\_Req = \{MME, HSS, 2\}$,
$Auth\_info\_Ans = \{HSS, MME, 3\}$,
$Ciphered\_Req = \{MME, eNodeB, 4\}$,
$Ciphered\_Rep = \{eNodeB, MME, 5\}$.

Nous pouvons voir que le vEPC dispose de trois micro-services connectés par l'intermédiaire de deux hubs non sécurisés différents (h_1, h_2). Le contrat de comportement réseau contient cinq types de messages :

— L'UE initie la procédure de raccordement : $Attach\_Request$.

— Le MME tente d'identifier l'UE en interrogeant le HHS : $Auth\_info\_Req$, qui répondra avec le message : $Auth\_info\_Ans$.

— Puis le MME et l'UE mettent en place le chiffrement de la connexion avec les messages $Ciphered\_Req$ et $Ciphered\_Rep$.

5.4.2 Étape 2 - Une topologie étendue basée sur NetKAT

Pour décrire la configuration du réseau, nous proposons l'approche utilisée par NetKAT.

Les bases de NetKAT

Dans cette section, nous rappelons les éléments de base de NetKAT, qui sont nécessaires pour notre proposition visant à vérifier la cohérence entre les call flow d'une VNF et la configuration du réseau. NetKAT a donné lieu à une série d'articles, que nous n'avons pas l'intention de résumer ici parce que cela nous mènerait trop loin dans la théorie des langages équationnels.

L'approche de base de NetKAT est de supposer que les fonctions implémentées dans un réseau peuvent être considérées comme des expressions régulières agissant sur des paquets. Dans ce cadre, un paquet pk est une série de champs, à savoir

$$pk ::= \{f_1 = v_1, \ldots, f_k = v_k\}$$

où f_i, $i = 1, \ldots, k$ sont des champs exprimés sous forme de séries de bits.

Les champs les plus couramment utilisés dans le contexte des réseaux IP sont les adresses IP source et destination, les numéros de port source et destination, les types de protocole, le DiffServ Code Point (DSCP), etc. Afin de suivre la position d'un paquet dans le réseau, NetKAT introduit deux champs supplémentaires : l'étiquette du commutateur et le numéro de port auquel le paquet se présente lorsqu'il atteint le commutateur. Le point important est que ces deux champs sont modifiés à chaque commutateur tandis que les autres champs sont pertinents de bout en bout (à l'exception du champ DSCP qui peut être mis à jour à l'intérieur du réseau dans le cas d'un marquage non fiable mais qui devrait en théorie avoir une valeur de bout en bout).

Avec la définition ci-dessus, les paquets forment un alphabet d'éléments 2^N si $N = |f_1| + \ldots + |f_k|$. Ce nombre est potentiellement très élevé, mais la plupart des procédures réseau n'agissent que sur un nombre restreint de champs (les adresses IP en cas de réacheminement à l'intérieur du réseau, les étiquettes de commutateur et de port).

L'idée fondamentale de NetKAT est de reconnaître que les fonctions réseau habituelles telles que le transfert, le reroutage, le pare-feu, etc. peuvent être considérées comme des expressions régulières qui peuvent être :

— soit des prédicats ($f = v$, où f est un champ et v une valeur donnée) ;

— ou bien des politiques, par exemple la mise à jour d'un champ ($f \leftarrow v$).

Les procédures de base (politiques et prédicats) de NetKAT sont données dans le tableau 5.1.

L'étoile de Kleene

$$p^* = \sum_{n \geq 0} \underbrace{p \ldots p}_{n \text{ fois}}$$

est la somme des itérations finies de la procédure p. L'outil *dup* est introduit pour dupliquer des paquets dans la construction des historiques (c'est-à-dire qu'il précède le même paquet dans un historique) ; les historiques sont présentés ci-dessous et sont déterminants dans la capacité de NetKAT à prouver les propriétés du réseau.

Prédicats		
$a, b ::=$	1	Élément identité
	0	Abandon
	$f = v$	Test
	$a + b$	Disjonction
	$a.b$	Conjonction
	$\neg a$	Négation
Politiques		
$p, q ::=$	a	Filtre
	$f \leftarrow v$	Modification
	$p + q$	Union
	$p \cdot q$	Composition séquentielle
	p^*	Étoile de Kleene
	dup	Duplication

TABLE 5.1 – Prédicats et politiques dans NetKAT.

L'ensemble des règles avec les procédures identity 1 (pas d'actions), null 0 (packet drop), $+$, $\cdot$, $\cdot$ et l'étoile * constituent une algèbre de Kleene. L'ensemble des prédicats avec l'identity 1 et drop 0 équipés des opérations $+$, $\cdot$ et $\neg$ est une algèbre booléenne. Les prédicats avec les opérations $+$ et $\cdot$ constituent un sous-algèbre de politiques. L'ensemble des politiques et des prédicats avec les opérations ci-dessus est une algèbre de Kleene avec test (KAT).

Avec la notation ci-dessus, un pare-feu qui abandonne (drop) tous les paquets vers une adresse IP donnée (disons, A_0) peut être écrit comme

$$(@IP_d = A_0) \cdot 0,$$

où, comme indiqué ci-dessus, 0 est la politique de filtrage qui supprime tous les paquets. De même, le transfert d'un paquet depuis le port pt_1 sur le commutateur A vers le port pt_2 sur le commutateur B peut s'écrire :

$$(\mathrm{SW} = A, pt = pt_1) \cdot (\mathrm{SW} \leftarrow B, pt \leftarrow pt_2)$$

Pour enregistrer le chemin pris par un paquet à travers le réseau, NetKAT introduit le concept d'historique qui correspond à la liste des états occupés par un paquet générique lors de la traversée du réseau. Un historique h a la forme $< pk_1, \ldots, pk_n >$ où pk_i est l'état du paquet pk au commutateur $n - i$ (la liste se lit de droite à gauche). Tous les prédicats et politiques de NetKAT agissent sur l'historique des paquets pour créer un nouvel historique (éventuellement vide si le paquet est détruit). Les politiques et les prédicats agissent sur les historiques comme détaillé dans la section 3 de [47].

Plus précisément, un prédicat sur un historique h renvoie un singleton $\{h\}$ ou l'ensemble vide $\{\}$. Une modification de champ $(f \leftarrow v)$ retourne un historique singleton dans lequel le champ f du paquet courant a été défini à v. Ainsi, les prédicats et les politiques induisent des fonctions sur les historiques. La fonction induite par la politique p est dénotée par $[[p]]$.

En utilisant le même formalisme, NetKAT décrit la topologie du réseau afin de vérifier si les règles n'essaient pas de faire des liens logiques qui n'existent pas physiquement (comme décrit dans l'exemple fourni dans la proposition 2).

Un des intérêts majeurs de NetKAT est sa prouvabilité et sa complétude. Cela signifie qu'avec les axiomes de KAT et les axiomes de NetKAT (voir Section 2 de [66]), chaque équivalence prouvable en utilisant les axiomes de NetKAT tient également dans le modèle équationnel (correction) et inversement, chaque équivalence dans le modèle équationnel est prouvable avec les axiomes de NetKAT (complétude) ; ces deux énoncés sont prouvés dans la Section 4 de [47].

En particulier, il est possible de prouver qu'un paquet suit un itinéraire donné (à savoir une séquence de commutateurs dans le réseau). Cette simple remarque nous motive à introduire une version étendue du réseau.

Topologie réseau étendue

Dans les réseaux configurés au moyen de SDN, par exemple via OpenFlow, il est possible de traduire la configuration du réseau dans NetKAT en récupérant la configuration du réseau (c'est-à-dire les règles OpenFlow poussées dans les éléments du réseau via les contrôleurs). Nous pouvons alors résumer la topologie du réseau en termes de nœuds, de ports et de liens.

Pour construire la topologie étendue de l'infrastructure virtualisée, nous considérons que les micro-services échangent des messages comme les commutateurs (virtuels) échangent des paquets. En effet, d'un point de vue réseau, les micro-services reçoivent des messages, les traitent et les transmettent à d'autres micro-services ou aux utilisateurs finaux. Cela peut être vue comme si les micro-services étaient des commutateurs et les messages étaient des paquets (virtuels).

Sur la base de la topologie (physique) du réseau et de la topologie (logique) des micro-services, nous sommes en mesure de construire la topologie étendue du réseau combinée avec les VNFs. Avec cet artefact et la puissance du formalisme NetKAT, nous sommes en mesure de vérifier que les paquets virtuels sont routés à travers la topologie étendue afin que les call flow d'une VNF puissent correctement se dérouler.

Illustration avec la connexion d'un UE

Les micro-services sont illustrés dans la Figure 5.3 et le call flow associé dans la figure 5.2. Nous avons introduit une topologie augmentée avec quatre noeuds ($host1$, $host2$, $host3$, $sw1$), chaque $host_i$ a un port, le commutateur sw_1 a trois ports. Il y a un lien entre chaque $host_i$ et sw_1. Si l'on considère les micro-services comme des commutateurs, la topologie étendue de notre exemple peut être décrite comme suit.

Proposition 2 *La topologie étendue de la procédure de connexion d'un UE pour l'exemple illustré dans la figure 5.3 est*

$$
\begin{aligned}
& t = (sw = host1 \cdot pt = 1 \cdot sw \leftarrow sw1 \cdot pt \leftarrow 1) \\
& +(sw = sw1 \cdot pt = 1 \cdot sw \leftarrow host1 \cdot pt \leftarrow 1) \\
& +(sw = host3 \cdot pt = 1 \cdot sw \leftarrow sw1 \cdot pt \leftarrow 2) \\
& +(sw = sw1 \cdot pt = 2 \cdot sw \leftarrow host3 \cdot pt \leftarrow 1) \\
& +(sw = host2 \cdot pt = 1 \cdot sw \leftarrow sw1 \cdot pt \leftarrow 3) \\
& +(sw = sw1 \cdot pt = 3 \cdot sw \leftarrow host2 \cdot pt \leftarrow 1)
\end{aligned}
$$

Nous pouvons combiner cette topologie avec le modèle de déploiement d'une nouvelle VNF pour vérifier la cohérence entre le modèle de VNF et la topologie actuelle du réseau avant d'agir sur le déploiement réel de la VNF.

5.4.3 Étape 3 - Modèle de déploiement de VNF

Pour déployer une VNF réutilisable sur un réseau, le modèle de déploiement crée un mapping entre chaque micro-services appartenant à un modèle de VNF et un nœud appartenant au modèle de configuration du réseau.

Definition 4 *Un modèle de déploiement D est défini comme $D = \{m_1, m_2, ..., m_i\}$ et m est un tuple tel que $m = \{n_i, s_j\}$.*

Dans notre cas exemple (raccordement d'un UE), le mapping proposé est le suivant :

Proposition 3 *Le modèle de déploiement de VNF pour la procédure de rattachement d'un UE illustré dans la figure 5.3 est*

$D = \{m_1, m_2, m_3\}$,
$m_1 = \{host1, eNodeB\}$,
$m_2 = \{host2, HSS\}$,
$m_3 = \{host3, MME\}$.

Le micro-service $eNodeB$ est déployé sur l'hôte $host1$. Le micro-service HSS est déployé sur l'hôte $host2$. Le micro-service MME est déployé sur l'hôte $host3$.

Une fois que la topologie et la description des VNFs sont combinées, nous pouvons écrire les règles réseau souhaitées dans NetKAT qui seront ensuite traduites en règles OpenFlow.

Proposition 4 *Les politiques NetKAT pour la procédure de rattachement d'UE de la figure 5.3 s'écrivent :*

$$p = (sw = sw1 \cdot pt = 1 \cdot sw \leftarrow MME \cdot pt \leftarrow 1)$$
$$+(sw = sw1 \cdot pt = 2 \cdot dst = HSS \cdot sw \leftarrow HSS \cdot pt \leftarrow 1)$$
$$+(sw = sw1 \cdot pt = 2 \cdot dst = eNodeB \cdot sw \leftarrow eNodeB \cdot pt \leftarrow 1)$$
$$+(sw = sw1 \cdot pt = 3 \cdot sw \leftarrow MME \cdot pt \leftarrow 1)$$
$$+(dst = HSS \cdot src = eNodeB \cdot 0)$$

5.4.4 Étape 4 - Vérification de la cohérence d'un modèle de déploiement de VNF

Exécuter le call flow sur la topologie étendue

Les micro-services sont implémentés pour envoyer des messages selon le protocole qu'ils mettent en œuvre. Après l'exécution d'une tâche, un micro-service transmet un message à un autre micro-service. Ce message est encapsulé dans un ou plusieurs paquets IP. Il est donc essentiel que le câblage des micro-services soit correctement programmé.

Si les micro-services sont hébergés dans le même centre de données, le routage Ethernet habituel de la couche 2 est en général suffisant pour assurer la connectivité entre les micro-services. Des problèmes potentiels surviennent lorsque les micro-services sont hébergés par des hôtes distants. Dans ce cas, les paquets IP transmis par les micro-services doivent être transportés en sûreté à travers le réseau.

Pour le call flow d'une VNF, un paquet virtuel est introduit et un historique est créé pour enregistrer le trajet de ce paquet virtuel dans le réseau. NetKAT peut alors être utilisé pour prouver que l'échange est bien réalisé par le paquet virtuel. Cette méthode peut notamment être utilisée pour éviter les boucles, les pertes de paquets ou les règles de transfert manquantes.

Pour vérifier la conformité entre les historiques et le call flow, nous avons construit deux traces de messages et nous vérifions qu'il existe une équivalence entre les historiques et les call flow à l'aide d'une *weak bi-simulation* [78].

Historique appliqué à notre exemple

Pour illustrer la dernière étape, nous nous concentrons sur le contrat de comportement réseau de la partie *Attach and Authentication* du call flow défini dans la figure 5.2. Dans le formalisme de NetKAT, pour un paquet provenant d'un commutateur virtuel eNodeB, nous obtenons l'historique suivant (rappelons qu'il doit être lu de bas en haut) :

$pk ::= \{$
$pk_1[src := eNodeB; dst := MME; sw := sw1; port := 2],$
$pk_2[src := eNodeB; dst := MME; sw := eNodeB; port := 1],$
$pk_3[src := MME; dst := eNodeB; sw := sw1; port := 1],$
$pk_4[src := MME; dst := eNodeB; sw := MME; port := 1],$

$pk_5[src := HSS; dst := MME; sw := sw1; port := 3]$,
$pk_6[src := HSS; dst := MME; sw := HSS; port := 1]$,
$pk_7[src := MME; dst := HSS; sw := sw1; port := 3]$,
$pk_8[src := MME; dst := HSS; sw := MME; port := 1]$,
$pk_9[src := eNodeB; dst := MME; sw := sw1; port := 2]$,
$pk_{10}[src := eNodeB; dst := MME; sw := eNodeB; port := 1]\}$

Sur la base de cet historique, nous pouvons construire deux systèmes de transition de labels (LTS) dans lesquels le label est défini en utilisant le nom de la source et le nom de la cible. Ensuite, nous vérifions l'inclusion de la trace entre le contrat de comportement réseau de la VNF et l'historique NetKAT. Pour vérifier l'inclusion des traces LTS, nous vérifions une propriété globale de weak bi-simulation qui implique l'inclusion de traces [79] comme illustrer dans la figure 5.4. Pour outiller l'approche, en plus de NetKAT, nous utilisons le vérificateur de modèle LTSA [80]. LTSA peut vérifier la simulation en représentant le contrat de comportement réseau de la VNF comme celui de son processus de vérification de propriété de sécurité.

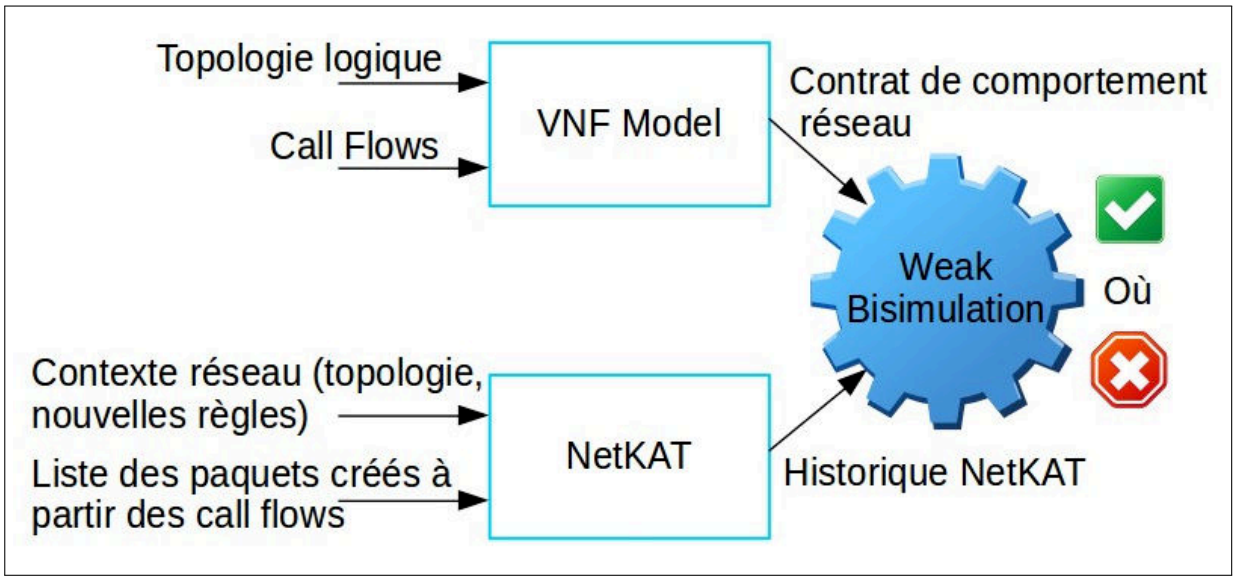

FIGURE 5.4 – Vérification de la correspondance entre les traces réseau et le comportement attendu.

5.5 Tester avant de déployer

5.5.1 Outils de démonstration

Nous avons voulu proposer une solution pour pouvoir appliquer facilement nos contrats de comportement réseau afin d'éprouver leur fonctionnement sur des cas réels. Le but étant de se concentrer sur les contrats nous avons cherché une solution qui soit légère et rapide à mettre en place.

Pour cela nous nous nous sommes tournés vers les tests, qui nécessitent beaucoup moins de ressources que la vérification et la simulation complète, pour ne représenter que les éléments réseau que nous voulions tester. Les résultats seront limités aux conditions exactes dans lesquelles se déroulent les tests, ils seront moins fiables que les méthodes utilisant la vérification formelle, mais cela nous permet de valider la conception de nos contrats et leur adéquation à des cas réels.

Nous avons créé un outil de démonstration de la mise en place de test automatisés. Pour ce faire, nous déployons un réseau constitué à partir :

— de la topologie du réseau de l'infrastructure où le déploiement aura lieu,

— des micro-services tels que décrits dans le modèle de déploiement basé sur le call flow.

— Des règles réseau existantes (facilement récupérer par le contrôleur SDN),

— des règles ajoutées pour le déploiement de la nouvelle VNF.

Nous utilisons Open vSwitch [81] pour reproduire le comportement des commutateurs SDN, il nous permet de créer des commutateurs virtuels pouvant recevoir des instructions OpenFlow. La topologie du réseau que nous voulons simuler est recréée à partir des informations extraites. Ensuite nous envoyons les règles OpenFlow extraites aux commutateurs afin qu'ils aient le même comportement.

Pour chaque micro-service un conteneur Docker est créé, il est connecté aux commutateurs en fonction des informations fournies par le modèle de déploiement de VNF (voir section 5.4.3). Nous nous sommes servis de Containernet [6] [82] pour mettre en place les commutateurs Open vSwitch et les conteneurs Docker, puis pour gérer les connections entre eux. Il étend le framework d'émulation Mininet et nous permet d'utiliser des conteneurs Docker standard comme host au sein du réseau émulé. Containernet permet d'ajouter et de supprimer des conteneurs du réseau émulé au moment

6. https://containernet.github.io/

de l'exécution, ce qui n'est pas possible dans Mininet. Ce concept nous permet d'utiliser une infrastructure de type Containernet dans laquelle nous pouvons démarrer et arrêter des instances de calcul (sous forme de conteneurs) à n'importe quel moment. Une autre caractéristique de Containernet est qu'il permet de modifier les limites de ressources, par exemple, le temps CPU disponible pour un seul conteneur, au moment de l'exécution et pas seulement lorsqu'un conteneur est démarré, comme dans les configurations normales de Docker.

Les conteneurs reçoivent la partie du modèle de VNF qui les concernent et un programme chargé de répondre aux paquets reçus. Pour cela il va forger un paquet virtuel (via la bibliothèque Scapy) en fonction du paquet reçu et du modèle de VNF. Ces paquets représentent le type de trafic que nous voulons garantir.

La condition va vérifier si les champs du paquet reçu (tel que l'adresse IP source ou le port de destination) correspondent aux champs du paquet attendu en fonction de la progression dans le déroulement du call flow. Puis il enverra le paquet correspondant à la suite du call flow. Il est possible de travailler sur tout type de champs s'ils sont détaillés dans le modèle de VNF.

Lorsque tous les paquets virtuels sont envoyés et bien reçus, nous savons que cette configuration du réseau permet aux call flow de se dérouler correctement.

Cette méthode peut notamment être utilisée pour éviter par exemple les boucles, les rejets inopportuns de paquets ou les règles de transfert manquantes.

Les tests peuvent rapidement être mis en place et offrent des temps de réponse plus court que la preuve formelle, mais ne permettent de ne vérifier qu'un nombre réduit de paramètres dans des conditions précises.

5.5.2 Exemple d'un service web

Nous nous plaçons dans le cas d'un service web qui délègue certaines de ses taches à des serveurs spécialisés, comme le font par exemple les Application delivery controllers [7].

Scénario A avec serveur de chiffrement

Le service a d'abord utilisé un serveur spécialisé dans le chiffrement qui lui permet de diminuer sa charge de calculs. Il déchiffre les requêtes envoyées au serveur et

7. `https://www.citrix.fr/glossary/adc.html`

chiffre les réponses à ces requêtes.

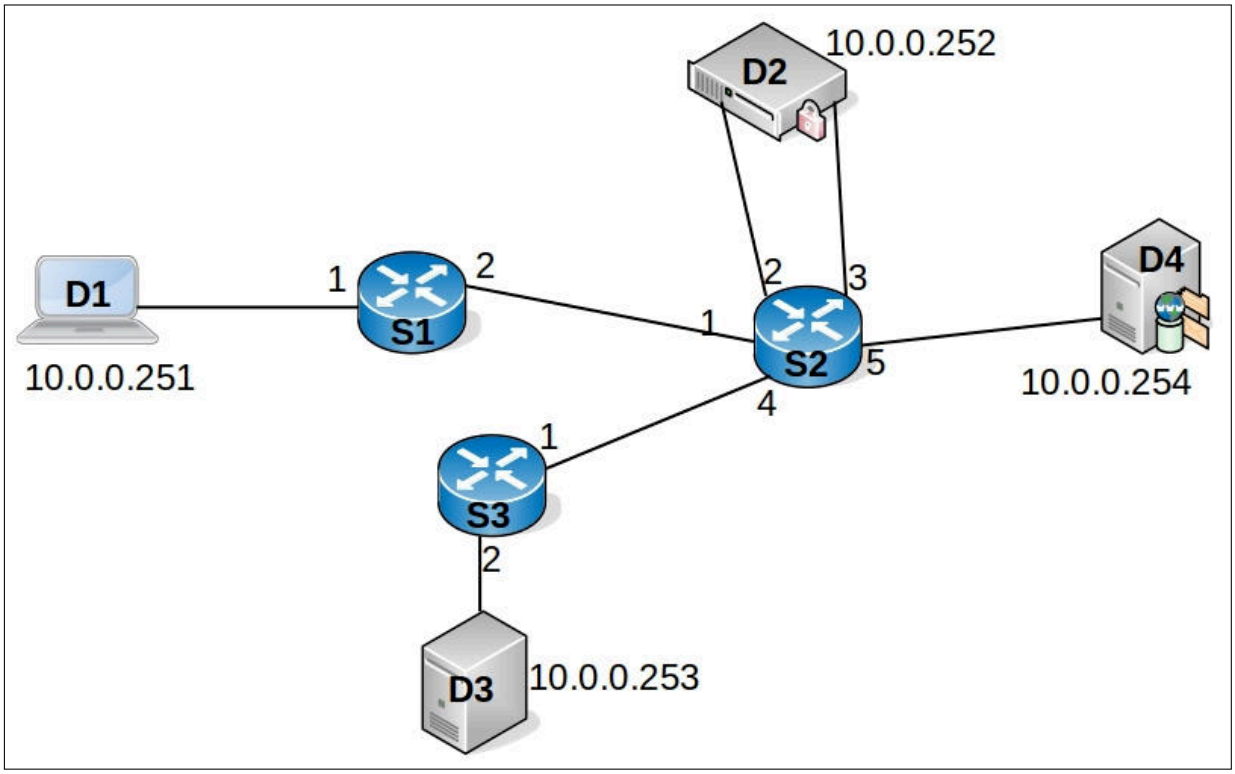

FIGURE 5.5 – Topologie de la démonstration.

L'architecture (voir 5.5) comprend un commutateur S1 qui réparti les requêtes envoyées dans le centre de données et un deuxième commutateur S2 qui gère les requêtes envoyées au service web étudié. Ce sont les règles de S2 qui s'assurent que les paquets https soit bien déchiffrés par le serveur D2 avant et après leur traitement par le serveur D4 (comme décrit dans le call flow 5.6).

Scénario B avec serveur de compression

Lorsque la charge réseau atteint un seuil défini par le gestionnaire du service, il fait appel à un serveur qui va se charger de compresser les réponses du serveur web. Ce serveur n'est utilisé que lorsque c'est nécessaire afin de limiter les coûts.

Le serveur de compression D3 est derrière un commutateur S3 qui redirige le trafic des différents services qui font appel à lui (voir 5.5). Lorsque la charge devient trop importante pour le service, une règle est ajoutée au commutateur S2 afin de faire passer le service sortant par D3.

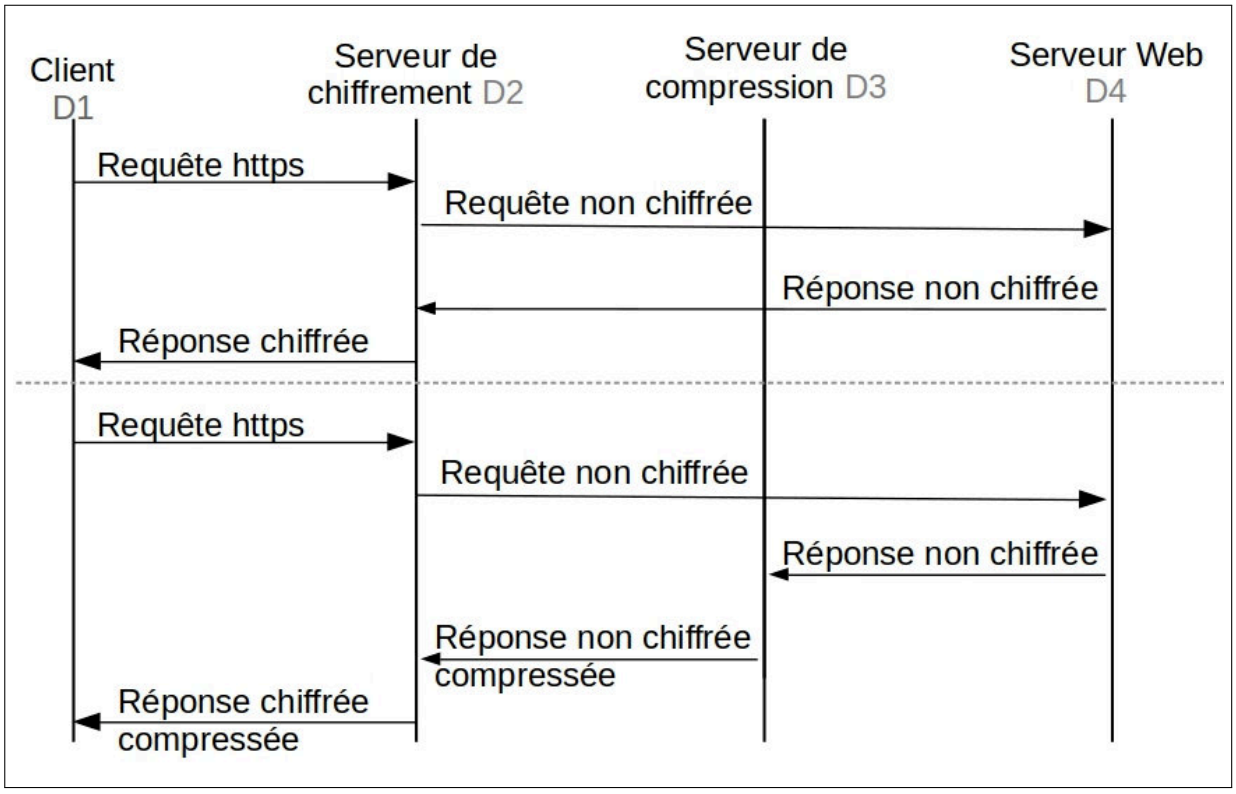

FIGURE 5.6 – Déroulement des échanges dans notre environnement de démonstration.

Ce call flow nous permet de créer le contrat de comportement réseau suivant : $Req\_compression\_activée = \{Req\_https, Req\_déchiffrée, Rep\_non\_chiffrée, Rep\_non\_chiffrée\_comp, Rep\_chiffrée\_comp\}$, avec :
$Req\_https = \{Client, S\_Chiffrement, 1\}$,
$Req\_déchiffrée = \{S\_Chiffrement, S\_Web, 2\}$,
$Rep\_non\_chiffrée = \{S\_Web, S\_Compression, 3\}$,
$Rep\_non\_chiffrée\_comp = \{S\_Compression, S\_Chiffrement, 4\}$,
$Rep\_chiffrée\_comp = \{S\_Chiffrement, Client, 5\}$.

Test du bon déroulement du scénario A

Une architecture similaire à celle que l'on souhaite vérifier est déployée grâce à un programme s'appuyant sur Contrainernet. Le commutateur S2 à reçu les même règles que celles présentes sur le commutateur d'origine :

```
root@ff0be0409842:~# ovs-ofctl dump-flows s2
NXST_FLOW reply (xid=0x4):
 in_port=4 actions=output:5
 priority=101,in_port=2 actions=output:5
 priority=98,in_port=1 actions=output:5
 priority=1000,arp actions=FLOOD
 priority=102,tcp,in_port=2,tp_src=49153 actions=output:1
 priority=100,tcp,tp_dst=443 actions=output:2
 priority=99,tcp,tp_src=443 actions=output:3
```

Chaque conteneur docker reçoit les actions qu'il doit effectuer afin de valider le déploiement. Ces actions viennent du contrat du comportement réseau (comme celui détaillé en 5.5.2), voici les informations qu'il contient :

```
{
    "D1-start": {
#envoi de la requete au serveur web
        "dip": "10.0.0.254",
        "dport": "443"
```

```
    },
   "D2-1": {
#reception requete chiffree
        "sip": "10.0.0.251",
        "sport": "443",
#transmission requete dechiffree
        "dip": "10.0.0.254",
        "dport": "443"
    },
    "D4-1": {
#reception de la requete
        "sip": "10.0.0.252",
        "sport": "443",
#reponse du serveur web
        "dip": "10.0.0.251",
        "dport": "49153"
    }
    "D2-2": {
#reception reponse non chiffree
        "sip": "10.0.0.254",
        "sport": "49153",
#transmission reponse chiffree
        "dip": "10.0.0.251",
        "dport": "49153"
    }
    "D1-1": {
#reception de la reponse : fin
        "sip": "10.0.0.252",
        "sport": "49153"
    }
}
```

Dans notre cas nous avons ajouté des contraintes sur les ports réseau en plus de celles sur les adresses IP. Il est possible d'ajouter des contraintes sur tout les champs des paquets en fonction des besoins, nos conteneurs vérifient et modifient uniquement

ces champs sur les paquets qu'ils reçoivent. Lorsque toutes les étapes de chaque serveur sont traitées, le call flow s'est déroulé correctement.

Il est possible d'activer un journal des opérations pour contrôler le détail du déroulement et savoir jusqu'où il s'est exécuté. Voici ce que cela donne dans notre exemple :

```
d1 starts the callflow
d2 received a pkt from 10.0.0.251:443 and
   sent a pkt to 10.0.0.254:443
d4 received a pkt from 10.0.0.252:443 and
   sent a pkt to 10.0.0.251:49153
d4's callflow finished
d2 received a pkt from 10.0.0.254:49153 and
   sent a pkt to 10.0.0.251:49153
d2's callflow finished
d1 received a pkt from 10.0.0.252:49153 and
   sent a pkt to 127.0.0.1:80
d1's callflow finished
```

Test du bon déroulement du scénario B

Le scénario B introduit S3 et D3 pour gérer la compression des paquets sortant (voir 5.5), ainsi qu'une règle supplémentaire dans S2. Les commutateurs sont maintenant configurés comme ceci :

```
root@deef8d6b6968:~# ovs-ofctl dump-flows s2
NXST_FLOW reply (xid=0x4):
 priority=95,in_port=4 actions=output:3
 in_port=4 actions=output:5
 priority=101,in_port=2 actions=output:5
 priority=98,in_port=1 actions=output:5
 priority=1000,arp actions=FLOOD
 priority=102,tcp,in_port=2,tp_src=49153 actions=output:1
 priority=100,tcp,tp_dst=443 actions=output:2
 priority=99,tcp,tp_src=443 actions=output:3
```

```
root@deef8d6b6968:~# ovs-ofctl dump-flows s3
NXST_FLOW reply (xid=0x4):
 actions=normal
 arp actions=FLOOD
 priority=10,tcp,tp_src=443 actions=output:1
```

La procédure de test permet de faire ressortir un problème, car le call flow ne se déroule pas jusqu'au bout. Le détail nous permet de comprendre que le problème a lieu après l'envoi de la réponse par D4 :

```
d1 starts the callflow
d2 received a pkt from 10.0.0.251:443 and
   sent a pkt to 10.0.0.254:443
d4 received a pkt from 10.0.0.252:443 and
   sent a pkt to 10.0.0.251:49153
d4's callflows finished
```

Pour des raisons de concision, le problème et les critères de vérification étaient simples. Mais l'erreur de configuration du scénario B ne serait pas apparue avec un simple test de connectivité entre les serveurs ou un test qui ne combinait pas à la fois des paquets chiffrés et une délégation de la compression.

5.6 Conclusion

Nous avons abordé dans ce chapitre l'exactitude du déploiement de VNFs lorsqu'elles sont distribuées sur des centres de données distants. Nous avons argumenté pour le développement d'une vision unifiée de toutes les ressources impliquées dans la mise en œuvre et le déploiement des VNFs, notamment la façon dont elles sont interconnectées. Sur la base de ce point de vue, nous avons introduit le concept de topologie étendue où les micro-services apparaissent comme des commutateurs. Grâce à des paquets virtuels associés aux call flows de la VNF, il est possible d'utiliser le formalisme NetKAT pour vérifier que l'historique des paquets virtuels est conforme aux call flow d'une VNF et donc que la VNF sera correctement déployée.

Nous pensons en outre qu'une telle approche est la plus pertinente dans le cadre d'ONAP (Open Network Automation Project) [83], qui vise à automatiser la création et l'instanciation des VNFs. En adoptant la solution proposée dans ce document, ONAP sera en mesure d'instancier en toute sécurité les VNFs dans un réseau configuré au moyen de SDN.

ONAP a identifié un certain nombre de caractéristiques nécessaires à la création et à l'instanciation automatique des VNFs. En particulier, ONAP vise à développer une vision globale des ressources du réseau, y compris les éléments traditionnels du réseau (types de connectivité et de bande passante) gérés avec OpenDayLight ainsi que les ressources informatiques (stockage et calcul) gérées par OpenStack. Ainsi, ONAP peut développer une vue unifiée de toutes les ressources afin de les optimiser et de les configurer.

L'outil de test présenté nous a permis d'avoir une solution légère et simple à mettre œuvre pour appliquer nos contrats de comportement réseau. Nous avons détecté une erreur de configuration réseau dans notre exemple et les traces nous ont montré à quel endroit les paquets ne suivaient plus le contrat.

Cette méthode de vérification plus légère n'apporte pas les même garanties que les vérifications formelles, le résultat des tests ne valide le contrat que pour les conditions exactes dans lesquels ils ont eu lieu. Mais elle nous permet de mettre facilement la conception de nos contrats à l'épreuve de cas réels.

L'utilisation de gestionnaire de configuration couplée à une vérification formelle du contrat de comportement réseau nous semblent être de bons outils pour développer des VNFs et pour des déploiements à petite échelle. Pour la gestion des déploiements dans de grands centres de données l'utilisation d'un orchestrateur devient indispensable. La vérification dans ce cadre-là est l'objet des travaux du chapitre suivant.

CHAPITRE 6

PERMETTRE LA VÉRIFICATION AU SEIN DES ARCHITECTURES NFV MANO

6.1 Introduction

La promesse de NFV de faire fonctionner un service réseau en quelques minutes au lieu de plusieurs mois, permet l'agilité, mais cela peut aussi créer le chaos. La nécessité d'une bonne gestion a été mise en évidence à un stade précoce. L'utilisation du modèle a été perçue par la communauté des réseaux comme une solution commune pour permettre une intégration agile, améliorer l'interopérabilité et prévenir ce chaos. L'ETSI a définit une norme (NFV MANO) pour améliorer l'interopérabilité dans ce domaine. NFV MANO relève ce défi en offrant un ensemble de modèles pour formaliser la configuration du réseau et le déploiement de VNFs.

Nous proposons une étude du déploiement de Network Services en suivant la norme NFV MANO pour comprendre comment nous pourrions appliquer la technique de model@runtime pour vérifier la cohérence de la configuration actuelle du réseau. Cette étude met en évidence deux lacunes. Premièrement, les opérations de vérification et de validation ne sont pas explicitement définies dans la boucle de reconfiguration. Deuxièmement, les informations sur le déploiement physique des différents composants du Network Service ne sont pas modélisées dans la norme NFV MANO. Nous concluons cette étude en discutant d'une extension initiale pour soutenir l'audit de configuration de réseau.

6.2 NFV MANO

Les techniques de virtualisation révolutionnent l'architecture des réseaux de télécommunications. Le déploiement de nouveaux services peut être beaucoup plus ra-

pide, ils peuvent être instanciés à la demande pour répondre aux besoins des clients (par exemple : de scalabilité ou de proximité géographique).

Plusieurs fonctions sont actuellement repensées afin d'être virtualisées et instanciées de manière flexible sur des infrastructures virtuelles (voir par exemple [84]). Mais la décomposition d'un service global en plusieurs composants (ou micro-services [67]), qui peuvent être instanciés sur des serveurs distants, pose le problème de leur interconnexion.

La technologie SDN est souvent utilisée de manière complémentaire à la virtualisation des fonctions réseaux, car elle permet de configurer via le contrôleur la façon dont les différentes entités communiquent entre elles.

Ce domaine a souvent utilisé des techniques de modélisation à la fois pour assurer l'interopérabilité au sein du réseau et pour permettre la configuration et la communication entre tous les équipements SDN, quel que soit le fabricant.

Ces modèles décrivent différents aspects du comportement des VNFs et s'accompagnent de leur propre point de vue et objectif. L'un des principaux défis est de pouvoir vérifier la cohérence entre ces modèles.

La configuration réseau de ces infrastructures doit être dynamique pour répondre aux besoins des clients (nouveau déploiement, modification d'un Network Service) et du gestionnaire d'infrastructure (y compris : changement de matériel, optimisation de la charge par le déplacement de machines virtuelles pour réduire le nombre de machines physiques nécessaires à un moment donné).

En outre, il est important de comprendre qu'un micro-service est une brique logicielle développée par des entités indépendantes (ce problème a été détaillé dans la section 2.4.2 et avec la figure 2.4).

L'écart entre ces mondes peut provoquer des malentendus ou des erreurs de configuration entre l'architecture logique [73] et la configuration réelle.

Afin de coordonner les différentes infrastructures et les différents logiciels utilisés pour mettre en place un nouveau service, des orchestrateurs de virtualisation des fonctions réseau ont été développés. Ils communiquent avec les gestionnaires d'infrastructure pour créer des instances virtuelles des différentes fonctionnalités formant un service réseau. Leur différents rôles sont définis par l'ETSI dans la norme : NFV MANO. Elle sert de modèle pour les différentes solutions d'orchestration des VNFs

(dont Open Source MANO [1], ONAP [2], Cloudify [3]).

Nous nous sommes intéressés au cas où les infrastructures orchestrées gèrent des réseaux de type SDN. Le SDN et le NFV vont rapidement combiner leurs forces et poser le problème de la cohérence entre le service qui doit être offert par une VNF et la manière dont la VNF est déployée au sein du réseau.

6.2.1 Architecture NFV MANO

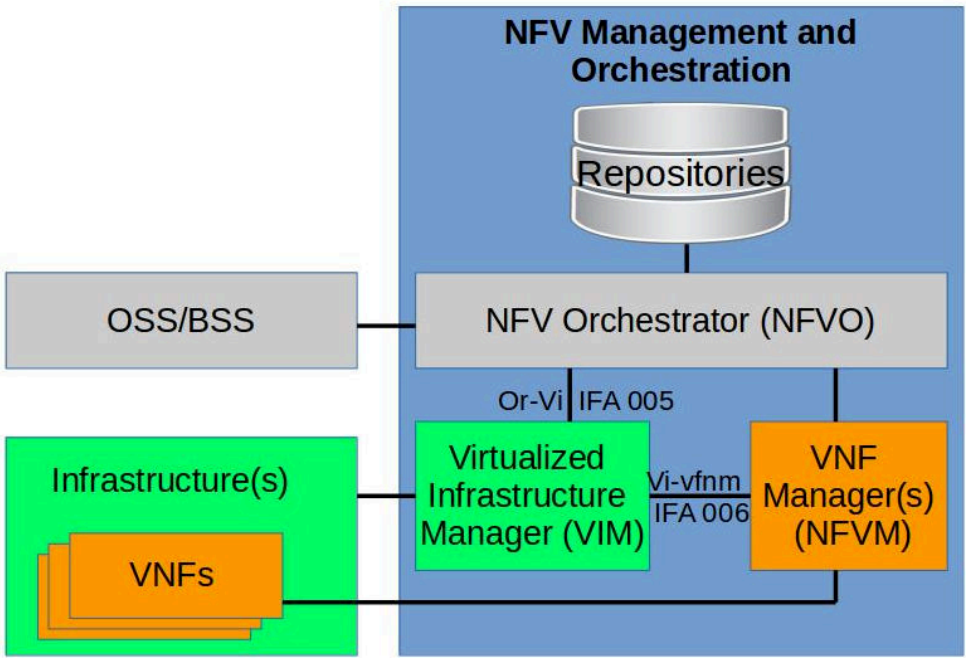

FIGURE 6.1 – L'architecture NFV MANO

La figure 6.1 donne un aperçu des différents rôles et de leurs connexions. Nous allons en donner une brève description.

— **Virtualized Infrastructure Manager** (VIM) : il gère les ressources physiques (stockage, calcul, réseau) d'une infrastructure. Les machines virtuelles utilisées pour héberger les VNFs sont instanciées par le VIM du domaine. Il est chargé de la gestion du cycle de vie des VMs et de leur suivi. L'orchestrateur peut gérer

1. https://osm.etsi.org/
2. https://www.onap.org
3. https://cloudify.co/nfv/

plusieurs VIM (souvent un par centre de données), chaque VIM liste ses VMs et ses ressources physiques. Le VIM masque la complexité due aux différentes technologies utilisées dans les infrastructures.

— **Virtual Network Function Manager** (VNFM) : il fournit et adapte les modèles de configuration des différents services réseau pour la gestion des VNFs. Il est en charge de la gestion du cycle de vie des VNFs : il démarre, surveille, s'adapte en focntion la charge et termine les instances des VNFs. Il peut y avoir plusieurs VNFMs gérant des VNFs séparées.

— **NFV Orchestrator** (NFVO) : il est en charge de l'orchestration des ressources et des services. Il a une vision globale de l'ensemble des ressources disponibles à travers l'ensemble des VIMs et coordonne leur utilisation. Pour l'orchestration des services, il coordonne les VNFMs pour créer un service de réseau de bout en bout entre leurs VNFs. Il gère également le catalogue de services et est responsable de la gestion du cycle de vie des services réseau.

— **Repositories** : ils listent 4 types d'informations : un catalogue de VNFs, un catalogue de services réseau (Network Services Catalog), une liste des ressources utilisées et une liste de toutes les instances de VNFs. Le catalogue de VNFs contient tous les modèles de déploiement disponibles. Ils décrivent les exigences en matière de déploiement et de comportement opérationnel. Le catalogue des services réseau répertorie les modèles de déploiement d'un service réseau avec les VNFs utilisées, leurs paramètres et la façon dont ils sont inter-connectés.

— **Business/Operations Support System (OSS/BSS)** : c'est une collection d'outils commerciaux utilisés par les fournisseurs de services. Il reçoit les demandes de déploiement ou de gestion des clients et les traduit en demandes techniques pour le NFV Orchestrator.

6.2.2 Modèle NSD

Le standard définit une liste d'éléments pour modéliser un service réseau. Pour construire le diagramme 6.3 nous nous sommes basé sur les données décrites dans "NFV-MAN 001: Management and Orchestration" [4] et sur ses dépendances.

4. `http://www.etsi.org/deliver/etsi_gs/NFV-MAN/001_099/001/01.01.01_60/gs_nfv-man001v010101p.pdf`

Dans ce document, chaque élément est décrit par un nom, un type (feuille ou élément) et une cardinalité. Nous n'avons pas inclus tous les détails des options de configuration matérielle dans notre diagramme car ils ne sont pas au centre de notre réflexion et cela en facilite la lecture.

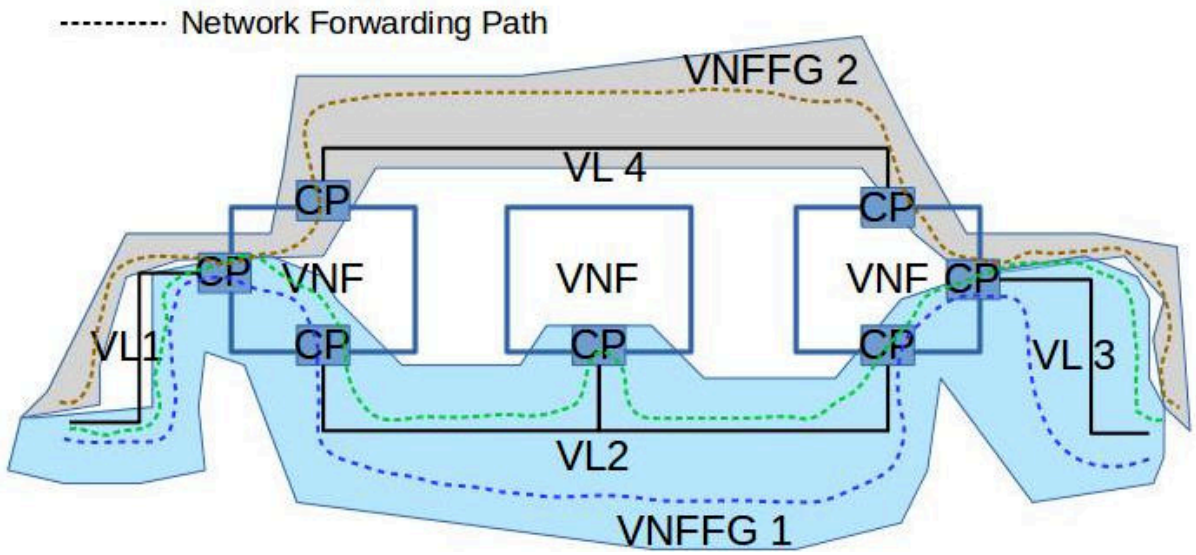

FIGURE 6.2 – Les communications entre VNFs

Le Network Service Descriptor (NSD) est l'élément central qui rassemble tous les paramètres nécessaires à l'orchestrateur pour son déploiement. Le descripteur de VNF (VNFD) décrit les différentes fonctions qui composent le service réseau. Ils sont reliés entre eux par l'intermédiaire de leurs *Connection Points* (CP dans Fig 6.2) par des *Virtual Links* (VL dans la figure 6.2). Les *Connection Points* et les *Virtual Links* servent à définir tous les liens entre deux éléments (physique, virtuel ou logique) et leurs paramètres. Le chemin suivi par le trafic est décrit dans le *Network Forwarding Path Descriptor* (NFPD) et regroupé dans le *VNF Forwarding Graph Descriptor* (VNFFGD).

Chaque VNF peut être instanciée différemment. C'est le *Virtual Deployment Unit* (VDU) qui décrit comment la VM, qui sera utilisée pour répondre aux besoins de la fonctionnalité, sera déployée. L'image utilisée et les capacités matérielles y sont détaillées (calcul, stockage, réseau). Ces différents paramètres et le nombre d'instances constituent une *flavor*, elle définit la taille/puissance et le nombre de serveurs virtuels. Différentes *flavors* sont définies pour s'adapter à la charge et l'orchestrateur choisira celle qui correspond à la charge actuelle en fonction des paramètres définis.

Les NS, les VNF et les VDU ont des politiques de surveillance et de mise à l'échelle

qui leur permettent de déclencher des actions comme arrêter ou lancer des instances supplémentaires, ou augmenter la puissance de calcul. Elles sont exécutées en fonction de critères observés comme le nombre de connexions par seconde, la charge CPU ou la bande passante.

Les *Physical Network Functions* (PNF) permettent d'inclure aux services réseau des fonctionnalités réseau non virtualisées (par exemple pour profiter d'une optimisation matérielle importante ou pour utiliser des équipements pré-existants). Par définition, les PNFs ne sont situées qu'à un seul endroit et ne peuvent être déplacées, mises à l'échelle sur demande ou avoir plusieurs instances. Cela explique les différences dans leurs descripteurs.

Sur la base de ce modèle, différents outils sont utilisés pour orchestrer et gérer le trafic à travers les VNFs. La couche de virtualisation va abstraire les contraintes matérielles permettant aux VNFs de connecter leurs ports (appelés Connection Point) via des liens virtuels (VL). L'autre outil important est le Network Forwarding Path. Il montre le chemin logique que le trafic doit prendre ; il peut différer du chemin créé avec les liens virtuels : par exemple en raison de politiques de sécurité qui peuvent interdire certaines communications directes. Le dernier outil est le VNF Forwarding Graph (VNFFG), ils sont constitués des différents chemins d'acheminement réseau appartenant à une même fonctionnalité (comme montré dans l'exemple de la Fig 6.2).

Lorsque les VNFs sont implémentées sur différentes machines virtuelles instanciées sur différents sites physiques, elles doivent communiquer entre elles à travers le réseau pour fournir un service de bout en bout. Leurs messages passent par les différentes VNFs du service réseau, mais ils sont également traités par les différents équipements réseau qu'ils traversent. Le point clé est de vérifier que leurs messages sont correctement échangés et qu'ils pourraient atteindre leur but.

6.3 Vérification de la cohérence d'un déploiement de VNFs

Face à la complexité d'adapter constamment le réseau et les logiciels pendant l'exécution, nous nous sommes intéressés à la manière dont les modèles peuvent être utilisés pour valider, surveiller et adapter le comportement à l'exécution. L'utilisation des techniques de model@runtime étend l'utilisation des techniques de modélisation au-

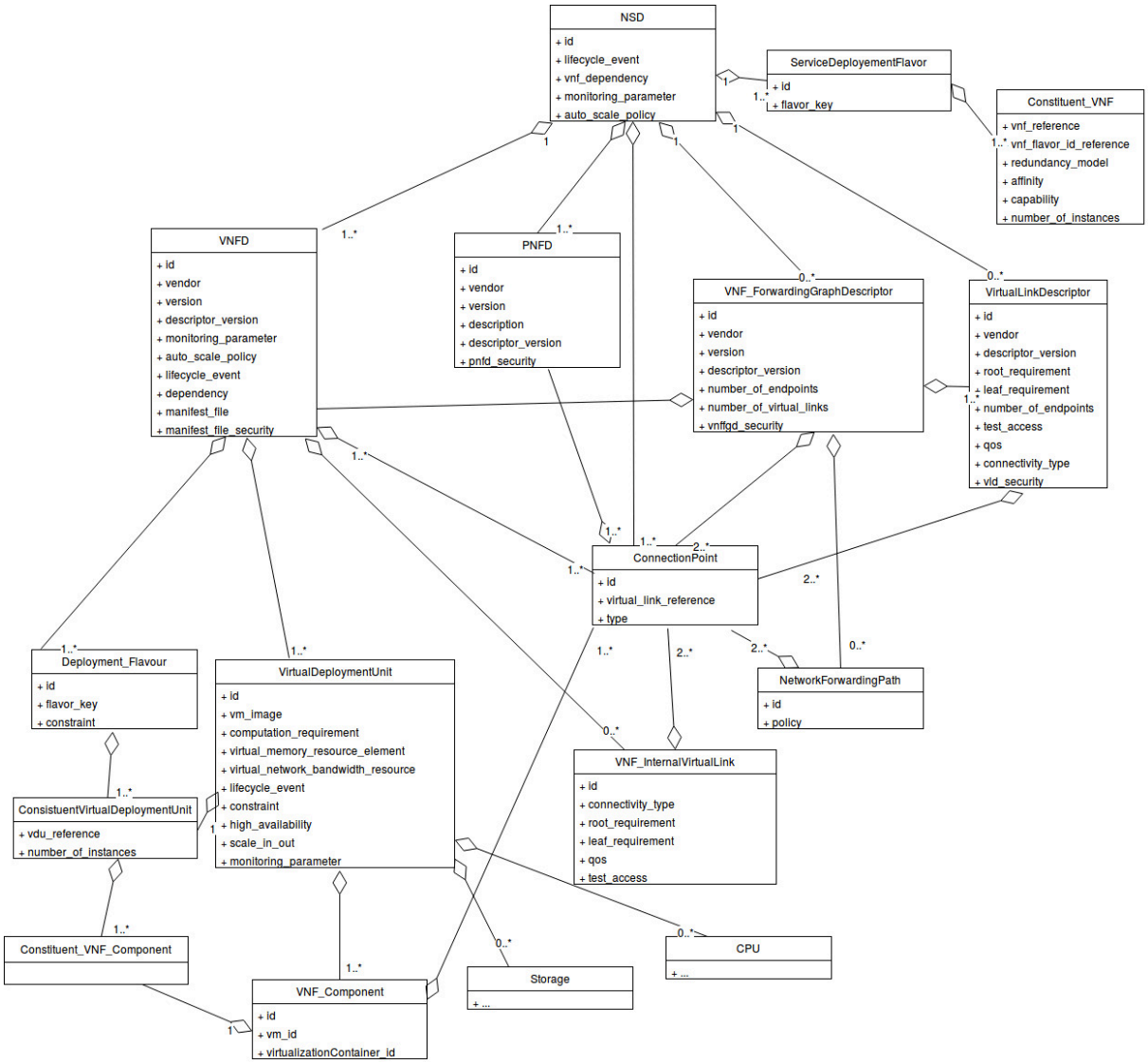

FIGURE 6.3 – Metamodel du Network Service Descriptor

delà des phases de conception et de mise en œuvre pour permettre la vérification à l'exécution [85, 86, 87].

Dans le cas d'un réseau, la principale difficulté vient de l'absence d'un état global observable. Si à la conception, il est possible de simuler un état du réseau, lors de l'exécution il est complexe d'observer précisément l'état du réseau. Par exemple, des informations plus précises sur le flux, le débit des liaisons pourraient permettre d'ajouter des contrôles de congestion. Mais surveiller avec précision et tenir à jour tous les flux est trop coûteux. De plus, certains équipements prennent en compte le contexte (les paquets déjà traités) pour décider quelles règles s'appliquent au trafic, par exemple, les commutateurs à apprentissage (learning switch) ou des pare-feu à état (stateful firewall). La configuration ne peut donc être validée que pour un état donné à un instant donné.

Toutefois, cela n'empêche pas d'examiner certains types d'erreurs et de se demander si l'état de la normalisation peut déjà permettre la mise en œuvre d'une telle vérification au moment de l'exécution. Le premier risque d'erreur dans le déploiement de VNFs dans un réseau est l'incapacité de deux VNFs à communiquer ou l'incapacité de plusieurs micro-services composant une seule VNF à communiquer (voir le paragraphe 2.4.2). Cela nous mène à la question de recherche suivante :

Pourrions-nous facilement vérifier la cohérence du déploiement d'un Network Service au sein de l'architecture NFV MANO ?

6.3.1 Protocole d'analyse de la norme

Il est beaucoup plus compliqué de prouver l'absence de certains critères que leur présence, c'est pourquoi nous allons expliquer notre méthodologie d'analyse pour montrer certaines lacunes de la norme actuelle.

L'ETSI a publié 77 documents [5] pour NFV MANO, ce qui représente un total de 4625 pages, nous ne pouvons donc pas étudier chaque page mais nous pouvons focaliser nos recherches sur la partie qui se concentre sur le déploiement. Les interfaces et les éléments d'information échangés via ces interfaces sont décrits dans différents documents de la norme correspondant aux différents liens entre les entités. Nous nous sommes concentrés sur les informations topologiques qui pourraient être envoyées par le gestionnaire de l'infrastructure virtuelle. Puisqu'il établit la topologie, il est le seul à

5. `https://docbox.etsi.org/ISG/NFV/Open/Publications_pdf/Specs-Reports`

disposer de cette information (au moins jusqu'à ce qu'il ne la transmette).

Le VIM communique avec l'orchestrateur (via Or-Vi) et avec le VNFM (via Vi-Vnfm), les informations échangées sont décrites respectivement dans la section 8 de la spécification de leur modèle IFA 005 [6] et IFA 006 [7] (voir figure 6.1).

Le VIM peut communiquer des informations sur les options choisies pour le déploiement ou la gestion de la configuration (par exemple les adresses IP des différents ports) mais ne donne aucune information sur la topologie par laquelle le trafic réseau passe et ni sur les règles appliquées à ce trafic.

Nous avons également doublé cette lecture ciblée avec une recherche par mot-clé dans tous les documents publiés. Le mot-clé était *topology*. Une recherche sur ce mot-clé donne 249 résultats dans tous ces documents. Une lecture de chaque section contenant ce mot-clé nous a permis de vérifier qu'aucune section de la norme ne propose un modèle ou une interface de programmation pour récupérer les informations précises de déploiement d'une VNF.

6.3.2 Discussion

L'architecture de NFV MANO est conçue pour répondre aux besoins opérationnels mais ne prend pas en compte les besoins analytiques. Pour s'assurer du bon déploiement des VNFs, le VNFM doit faire des hypothèses sur l'état actuel de la configuration du réseau. L'étude de la norme NFV MANO nous a permis de synthétiser les informations relatives à la gestion du déploiement de VNFs au sein d'un méta-modèle unique présenté dans la figure 6.3. Cette synthèse montre que rien n'est prévu dans NFV MANO pour interroger un choix de déploiement ni pour connaître la configuration finale. De même rien n'a été pensé pour introduire des outils de simulation réseau avant l'exécution d'un changement de configuration.

6.3.3 Validité

Notre première analyse met en évidence quelques lacunes dans le standard NFV MANO pour permettre à un outillage externe de vérifier le déploiement de VNFs. Ce-

6. http://www.etsi.org/deliver/etsi_gs/NFV-IFA/001_099/005/02.01.01.01_60/gs_NFV-IFA005v0201p.pdf

7. http://www.etsi.org/deliver/etsi_gs/NFV-IFA/001_099/006/02.01.01.01_60/gs_NFV-IFA006v0201p.pdf

pendant, comme dans tout protocole expérimental, notre évaluation initiale présente un certain biais. Notre lecture attentive peut faire l'objet d'un oubli mais surtout d'une mauvaise interprétation de la norme de notre part. Pour atténuer ce biais, nous avons constamment échangé avec plusieurs experts d'un grand opérateur télécom français afin de comparer nos connaissances et les leurs.

La section suivante traite d'une première extension pour permettre cette vérification.

6.4 Une extension de NFV MANO

Pour combler les lacunes de la norme NFV MANO à ce sujet, nous proposons un ajout au modèle existant en respectant le raisonnement global comme le montre la figure 6.4. Chaque infrastructure possède une ou plusieurs topologies regroupant les équipements de son réseau à travers lesquels passe le trafic des VNFs qu'elle héberge. Cet équipement peut être, physique ou virtuel, un commutateur ou une middlebox (voir la section 3.5). Ils sont définis par les règles qu'ils appliquent au trafic. Ces appareils ont des ports réseau qui sont reliés entre eux par des liens.

La machine virtuelle hôte d'un composant d'une VNF est connectée à l'un de ces équipements et communique par le biais de sa topologie. C'est là que se fait le lien avec le modèle NFV MANO. La topologie globale du Network Service est la somme des différentes topologies auxquelles sont liées les machines hébergeant ses VNFs.

6.4.1 Échange d'information de topologie

Nous croyons que cette information pourrait être retournée par le VIM lors de la réservation ou de l'allocation des ressources. Actuellement, le NFVO, après validation du VNFM, demandera au VIM un certain nombre de ressources avec des paramètres spécifiques (la procédure est détaillée dans le document MAN 001 [8] et les paramètres sont détaillés dans IFA 005 [9]). Une fois que ceux-ci ont été attribués ou réservés, le VIM enverra un accusé de réception au NFVO avec les détails concernant les ressources sélectionnées (par exemple, valider les critères demandés ou la zone

8. https://www.etsi.org/deliver/etsi_gs/NFV-MAN/001_099/001/01.01.01_60/gs_nfv-man001v010101p.pdf

9. http://www.etsi.org/deliver/etsi_gs/NFV-IFA/001_099/005/02.01.01.01_60/gs_NFV-IFA005v020101p.pdf

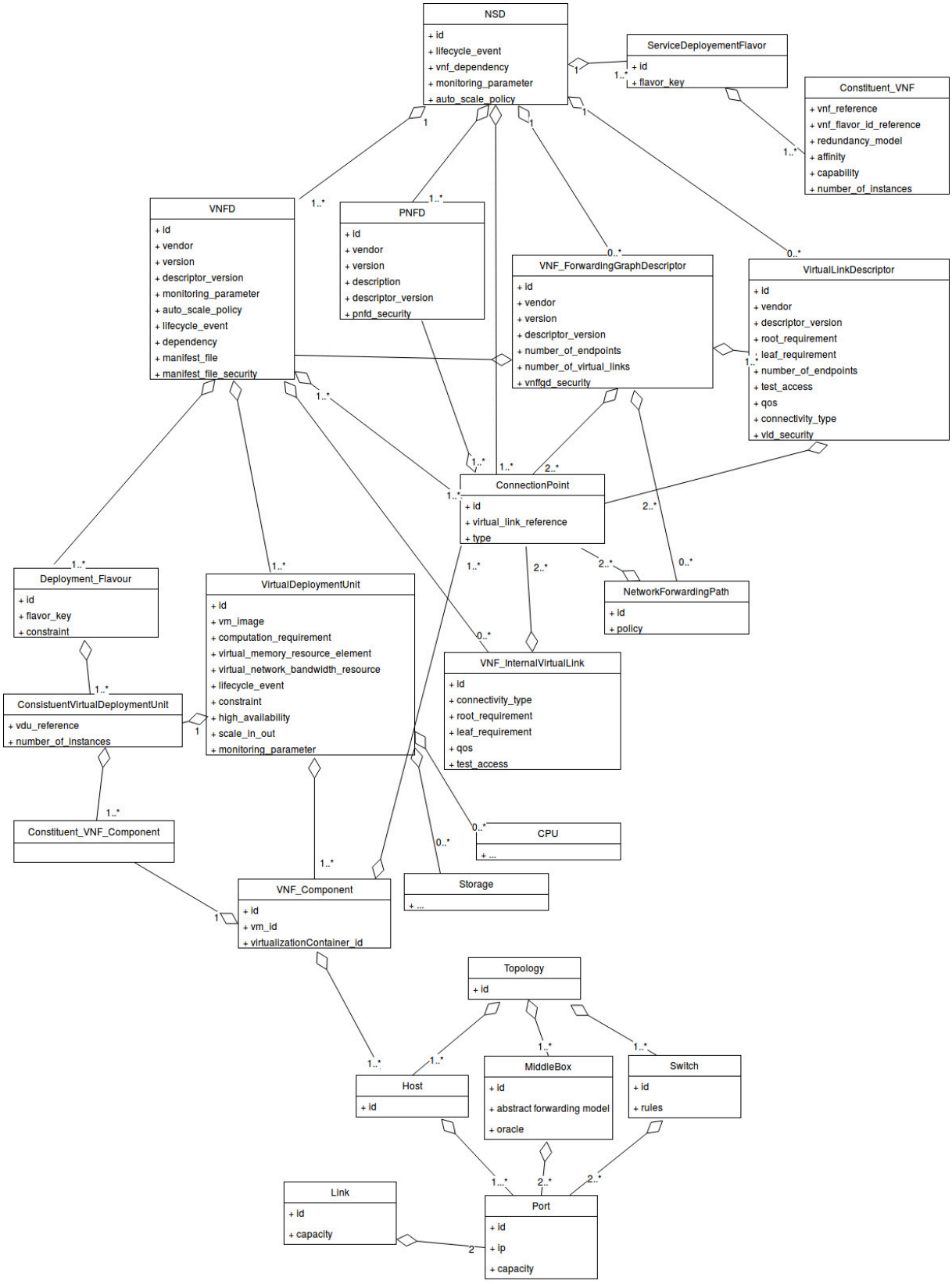

FIGURE 6.4 – Metamodel comprenant l'extension du NSD

de déploiement). Les informations concernant la topologie pourraient être partagées à ce moment-là pour compléter les informations globales sur les réseaux avec lesquels travaille l'orchestrateur.

6.4.2 Vérification au niveau de l'orchestrateur

Nous avons un temps envisagé d'effectuer la vérification dans les infrastructures, mais cette solution nous a paru moins intéressante. Déplacer le contrôle de la logique du déploiement nous semble s'éloigner de l'architecture où le NFVO commande les autres composants pour assurer un déploiement conforme aux *descriptors*. Il nous a aussi paru plus complexe de diviser la vérification entre les infrastructures où sont répartis les VNFs du NS sans perdre en fiabilité. Les changements à effectuer seraient aussi plus importants, il faudrait que tous les gestionnaires d'infrastructure prennent en compte les modifications, alors que des modifications au niveau des drivers et de l'orchestrateur lui permettrait de rester compatible avec tous les gestionnaires d'infrastructure dont les API permettent déjà d'obtenir les informations souhaitées.

6.5 Implémentation

6.5.1 Récupération des informations sur la topologie

Nous avons étudié les relations deux orchestrateurs vis à vis de leurs réseaux SDN. OpenBaton [10] fait partie des frameworks compatibles avec NFV MANO et mais l'implémentation de ses drivers n'exploite pas les possibilités offertes par les contrôleurs SDN. D'autres orchestrateur comme OpenSource MANO avec OpenVIM [11] implémentent déjà des mécanismes permettant de remonter des informations sur les réseaux gérés par les contrôleurs SDN. Il peut s'interfacer avec les contrôleurs Floodlight [12], OpenDaylight [13] ou ONOS [14] et via leur API récupérer :

— La version du contrôleur (et donc les versions OpenFlow compatibles).

— La liste des équipements connectés au contrôleur (adresse IP et DPID).

10. `https://openbaton.github.io/features.html`
11. `https://osm.etsi.org/wikipub/index.php/OpenVIM_installation`
12. `http://www.projectfloodlight.org/floodlight/`
13. `https://www.opendaylight.org/`
14. `https://onosproject.org/`

— Le détail des règles installées sur le contrôleur et les commutateurs dans lesquels elles s'appliquent.

— Les ports physiques et leurs ports OF équivalents.

Il possède aussi des fonctions lui permettant d'ajouter ou supprimer des règles dans les contrôleurs.

Le code de ces drivers interroge l'API des contrôleurs pour interagir avec eux (voir l'extrait 6.1). Il est possible d'étendre ces informations en utilisant toutes celles disponibles via l'API[15].

```
def get_of_switches(self):
    try:
        of_response = requests.get(
        self.url + "/wm/core/controller/switches/json",
        headers=self.headers)

        [...]

        switch_list = []
        for switch in info:
            switch_list.append((switch[self.ver_names["dpid"]],
            switch['inetAddress']))
        return switch_list
```

Listing 6.1 – Crée la liste des commutateurs dans OSM

Ces informations sont stockées dans différentes tables de la base de données qui reprennent pour certaines la structure des repositories détaillée plus haut (voir la section 6.2.2). À partir de ces informations et de celles des hôtes des VMs, il est possible de reconstruire une topologie et de raisonner sur les règles réseau qui s'y appliquent.

Ces ajouts pourraient être intégrés dans la norme pour que ces informations rejoignent celles déjà présentes dans le NSD et soient ainsi disponible de la même manière pour tous les orchestrateurs.

15. https://floodlight.atlassian.net/wiki/spaces/floodlightcontroller/pages/1343539/Floodlight+REST+API

Les critères qui devront être vérifiés peuvent provenir du VNF Forwarding Graph Descriptor qui nous renseigne sur quelles VNFs doivent communiquer entre elles et dans quel ordre. Les informations de déploiement de ces VNFs peuvent être obtenues grâce aux NSD et le détail des règles réseau qui s'appliquent à leurs communications peut être construit à partir des informations récupérées aux prés des différents VIMs par l'orchestrateur.

La possibilité de spécifier d'autres critères pourrait s'avérer utile mais dépendra des capacités du vérificateur choisi.

6.5.2 Vérification interne ou externe

Une fois que l'orchestrateur dispose des informations nécessaires pour vérifier la cohérence entre le déploiement d'un NS et ses besoins réseau, il faut effectuer cette vérification. Cela pourrait être fait entre la réservation de ressources prévu par la norme (voir le document MAN 001 [16]) et l'instanciation des VMs.

La vérification peut être faite par le client à l'extérieur de l'orchestrateur, cela impliquerait de mettre à disposition les informations de topologie via une API, comme MANO le prévoit déjà pour certaines informations. Le client vérifierait ainsi la configuration utilisée, mais pour que la vérification se fasse avant le déploiement et surtout avant chaque modification de la topologie (par exemple pour adapter le NS à la charge ou pour une raison liée à l'infrastructure dans laquelle il est déployé) il faudrait que ce soit l'orchestrateur qui interroge l'outil externe du client.

Cette solution nous semble plus complexe à être mise en œuvre d'autant que pour des raisons de confidentialité, il est possible que les gestionnaires d'infrastructure préfèrent ne pas partager ces informations sur leur réseau. La vérification peut se faire au sein de l'orchestrateur qui ne partagerait avec les clients que les résultats. Le NFVO interrogerait son module de vérification avant chaque déploiement et avant chaque modification de la configuration du réseau.

6.5.3 Choix du vérificateur

De nombreux outils permettant de faire des vérifications dans des réseaux de type SDN existent, ils ont été détaillés dans le chapitre 3. Si la vérification est faite au

16. `https://www.etsi.org/deliver/etsi_gs/NFV-MAN/001_099/001/01.01.01_60/gs_nfv-man001v010101p.pdf`

sein des orchestrateurs, la taille des réseaux et le nombre élevé de vérifications impose de choisir les outils qui ont les meilleurs temps de réponse. NetPlumber [30] et Delta-net [43] nous semble être les plus appropriés. Il faudra modifier leur mécanisme d'introspection du réseau afin qu'ils ne récupèrent plus les règles et la topologie via les contrôleurs SDN mais en interrogeant la base de données de l'orchestrateur. Enfin il faudra traduire les critères à vérifier dans le langage utilisé par ces outils (ici Flowexp ou Flow-based Management Language [32] pour NetPlumber et directement en C++ pour Delta-net).

6.6 Conclusion

Les outils de gestion de configuration vue au chapitre 5 peuvent être mis en place facilement mais ne sont pas conçus pour gérer des déploiements répartis sur plusieurs infrastructures.

La gestion de grandes infrastructures dispersées géographiquement permet une meilleure répartition de la charge et une meilleure optimisation des ressources. Cela évite par exemple d'avoir un centre de données qui refuse de nouveaux déploiements fautes de capacités et d'autres centres de données presque vides. Avoir des serveurs à différents endroits permet aussi de proposer des déploiements au plus proche de la demande. Cela permet de réduire fortement le temps de réponse des services déployés aux requêtes des clients.

La vérification de configuration réseau au sein de NFV MANO est un défi intéressant. Cet environnement met en exergue des points déjà abordés dans nos travaux précédents. La topologie y est hautement dynamique à cause des nouveaux déploiements mais surtout de l'adaptation constante des instances à la charge du service. De plus NFV MANO réunit des acteurs différents avec des préoccupations différentes et utilise déjà des formalismes pour certaines d'entre elles. L'ajout d'outils de vérification peut s'inscrire dans la continuité de la démarche de NFV MANO.

Il est essentiel pour le bon fonctionnement des services réseau de permettre la vérification afin d'éviter autant que possible les erreurs de configuration. SDN et NFV offrent des outils pour faciliter les vérifications mais la modélisation proposée dans le cadre de référence NFV MANO ne donne pas accès aux données analytiques permettant ce type de vérification. L'ajout de ces données n'entraînerait que peu de changements, car la procédure est similaire à celle utilisée pour l'échange d'information dans

le reste du modèle et certains orchestrateurs le font déjà.

De plus les informations analytiques peuvent aussi être utilisé dans d'autres cadres, comme le benchmarking qui permettrait de comparer différentes solutions de déploiement pour un même Network Service. C'est pourquoi nous pensons que cette problématique devrait être prise en compte dans le standard.

CHAPITRE 7

CONCLUSION

La thèse a abordé la problématique de la vérification de configuration réseau dans le contexte de l'arrivée des technologies Software Defined Networking et Network Functions Virtualization. Ce contexte amène de nouveaux outils, de nouveaux usages, de nouvelles façons d'aborder des problématiques préexistantes mais aussi de nouvelles problématiques auxquelles nous avons essayé de répondre.

Dans le chapitre 1, nous avons présenté le contexte dans lequel sont arrivées ces technologies et nous avons introduit les notions nécessaires à la compréhension de la thèse. Le chapitre 2 a présenté l'état de l'art des outils et méthodes de vérification au sein de réseaux de type SDN. Nous avons concentré notre analyse sur la façon dont sont spécifiées les contraintes à vérifier et sur les techniques utilisées pour limiter le temps de vérification.

7.1 Contributions de la thèse

Les erreurs de configurations réseau restent fréquentes et désastreuses pour les systèmes qu'elles impactent. Nous pensons que les technologies SDN et NFV apportent des facilités pour raisonner sur les comportements du réseau qui pourraient aider à éviter ces erreurs. Dans les travaux que nous avions étudiés au début de la thèse nous avions remarqué que l'aspect dynamique des configurations réseau n'était pas pris en compte. Nous pensons que ce manque est un frein majeur à l'utilisation de la vérification dans les réseaux.

Nous avons d'abord pensé la vérification au sein des contrôleurs puis nous avons travaillé à standardiser les critères qui devaient être vérifiés. Après ces premiers travaux nous avons réfléchi à comment vérifier ces critères dans des environnements regroupant différents contrôleurs gérant des réseaux différents mais ayant des contraintes communes.

7.1.1 Vérification de configuration au sein de WAN

Nous proposons, au chapitre 4, les bases d'un langage de programmation de contrôleur SDN permettant de vérifier l'absence de certaines erreurs. Il a été conçu pour être déployé uniquement dans les Policy Enforcement Point (PEP) afin de limiter son impact sur les performances.

Ce langage utilise une algèbre de Kleene avec test (KAT) pour fournir des preuves formelles du respect de certaines contraintes. Des efforts ont été faits lors de sa conception pour permettre aux opérateurs réseau d'exprimer des politiques dynamiques de manière concise et intuitive afin de limiter les incohérences entre les souhaits de l'opérateur et la configuration écrite.

Nous avons préconisé de raisonner sur les flux de paquets, car il nous semble important de prendre en compte la logique qui lie des paquets entre eux pour en faire des flux. Notamment pour des questions de gestion des ressources et de qualité de service (par exemple en assurant la même route et les même conditions à tous les paquets d'un même flux).

Nous avons proposé de conserver certaines composantes de base d'Internet, notamment le routage distribué et le contrôle de bout en bout, et d'utiliser le concept de Policy Enforcement Point (PEP) agissant sur les flux de paquets. Cela permet de limiter les inconvénients liés à la centralisation du routage dans SDN (vulnérabilité, temps de réponse, etc.) en profitant de protocoles de routage IP distribués, plus résistants aux défaillances, plus évolutifs et nécessitant moins d'échanges d'informations (comme OSPF, ISIS ou BGP). Ainsi tout en conservant la grande vitesse des routeurs de cœur de réseau et du routage IP distribué, nous pouvons configurer grâce à SDN les passerelles d'accès et les éléments de peering, garantissant que le trafic délivré par d'autres réseaux est conforme à certaines règles.

Cette solution encourage la mise œuvre du SDN au sein de WAN en permettant aux deux technologies de cohabiter et en facilitant l'écriture de configurations pouvant être vérifiées.

7.1.2 Contrats de comportement réseau

Nous proposons un système de contrat permettant de définir le comportement attendu par le réseau où est déployé un Network Service.

Cette approche est complémentaire de la vérification dans les contrôleurs, elle dé-

finit les critères qui devront être vérifiés. Grâce à une vision unifiée de toutes les ressources impliquées dans la mise en œuvre et le déploiement des VNFs, notamment la façon dont elles sont interconnectées, elle est une réponse à la vérification des NS amenés par le NFV.

Le contrat de comportement réseau est défini par un ensemble de messages ordonné. Il prend en compte les middlebox et certains équipements à états, en acceptant l'ajout de règles réseau qui modifient son comportement avant ou après le traitement du message.

Ces contrats peuvent être écrits lors de la conception du service, cela évite les incohérences dues aux nombreux corps de métiers différents qui interviennent de la conception au déploiement d'un NS.

Une fois définis ces contrats peuvent être déployés sur différentes architectures en adaptant seulement le modèle de déploiement de VNF. Ainsi il n'y a qu'un modèle à écrire par Network Service, celui-ci est réutilisable sur tous les déploiements et quels que soient les logiciels utilisés tant qu'ils suivent le même call flow.

Le respect de ces contrats peut être contrôlé pas différents outils de vérification. Par exemple, nous avons montré qu'il était possible d'utiliser NetKAT en décrivant une topologie étendue qui prend en compte les micros-services et leur traitement des paquets.

En ne définissant plus la manière dont les équipements devaient agir mais le résultat attendu, les contrats de comportement réseau évitent des erreurs dans la configuration réseau du NS ou des conflits avec les besoins réseau d'autres NS déjà déployés.

Nous avons développé un outil de test qui simule le comportement réseau des micro-services composant le NS et qui forge des paquets pour vérifier le respect du contrat. Cette méthode simple n'offre pas les même garanties que les vérifications formelles, mais elle nous à permis de mettre la logique de ces contrats à l'épreuve de cas réels.

7.1.3 MANO

Dans cette dernière contribution nous avons cherché comment effectuer des vérifications dans des systèmes qui orchestrent des NS pouvant être déployés sur différents centre de données utilisant plusieurs contrôleurs avec leurs propres configurations. Cette question nous permet de nous appuyer des sujets abordés dans les contribu-

tions précédentes : les techniques de vérification réseau avec un seul contrôleur et la définition des comportements réseau attendus par un NS.

Nous avons étudié le standard des orchestrateurs NFV dans le but d'intégrer la vérification de configuration dans leur architecture. Cette étude nous a permis de mettre en avant les informations manquantes pour vérifier les configurations réseau des NS. Afin de combler ces lacunes, nous avons proposé un ajout au modèle de description existant, tout en respectant son raisonnement global.

Ainsi une fois que les informations nécessaires sont remontées à l'orchestrateur, il devient possible de vérifier la cohérence de la configuration réseau sur lequel sont déployés les NS. Nous avons défini les étapes pour effectuer cette vérification. L'échange d'information que nous préconisons se base sur les méthodes utilisées par le NFVO pour récupérer d'autres informations. Ces informations peuvent être conservées grâce à l'extension au modèle NSD que nous avons proposée.

Après avoir étudié la possibilité d'effectuer la vérification via un outils externe au NFVO et un outils interne, nous avons plaidé pour l'utilisation d'un outil interne effectuant les vérifications avant chaque modification de configuration.

7.2 Perspectives

La démocratisation du SDN dans les WAN devrait se poursuivre. Elle permet d'adapter dynamiquement le routage en fonction des performances réelles, tant au sein de son propre réseau que sur les autres réseaux par lesquels passe le trafic. Tous les acteurs qui possèdent différents centres de données et font transiter des données par des réseaux qu'ils ne contrôlent pas devraient s'y intéresser.

Récemment des outils de vérification ayant de bonnes performances sur les réseaux de grande taille ont été publié (par exemple Delta-net [43]). Nous envisageons d'adapter FlowKAT pour pouvoir les utiliser.

Afin de faciliter l'emploi des contrats de comportement réseau, nous travaillons sur l'introspection automatique du système en cours d'exécution. Nous visons à obtenir les informations sur la topologie et ses règles mais aussi sur les VNFs déployése sans l'utilisation de contrats. La collecte des informations nous amènerait à construire automatiquement une topologie de l'infrastructure complète (informatique et réseau).

Cette introspection serait aussi intéressante pour utiliser les contrats dans les outils

de vérification existants, comme nous l'avons fait avec NetKAT en utilisant une topologie étendue prenant en compte l'action des micro-services.

Pour faciliter l'écriture de ces contrats et favoriser leur adapotion, il est possible de créer des bibliothèques de contrats pour les services couramment déployés, comme le fait par exemple Verificare [37] dans un domaine proche. Le call flow est lié au service et aux protocoles utilisés mais il n'est pas impacté par le choix des logiciels servant à l'implémenter. Pour la majorité des déploiements, il n'y aurait que le modèle de déploiement à écrire. Pour l'instant notre bibliothèque ne contient que les contrats que nous avons utilisés, mais nous devrions étudier les call flow de NS populaires pour en écrire les contrats.

Nous réfléchissons à d'autres mesures pouvant aider à l'adoption des contrats, car pour être efficace et éviter les incohérences, ils doivent être pris en compte par toute la chaîne des métiers qui conçoivent, assemblent, configurent et déploient les NS.

Nous étudions aussi l'ajout de nouvelles contraintes prises en charge par les contrats (comme les protocoles utilisés) et les impacts que cela aura sur la vérification.

Notre proposition d'extension à NFV MANO a pour but d'encourager l'organisme de standardisation à réfléchir à la question de la vérification. Nous travaillons sur l'implémentation d'un outil de vérification au sein d'OpenSource MANO. Ce développement s'appuie sur les réflexions menées dans le chapitre 6. Les trois grandes étapes de ce développement seront :

— l'ajout d'une étape facultative de vérification dans la procédure de déploiement d'un NS,

— la mise en forme des informations détenues par le NFVO afin qu'elles correspondent à celles attendues par l'outil,

— la prise en compte de la réponse dans le déploiement fait par l'orchestrateur.

Ce développement nous permettra de mettre à l'épreuve certains choix de conception et d'effectuer des vérifications au sein d'un orchestrateur même en dehors du standard. Cependant l'inclusion de la vérification dans le standard de l'architecture NFV MANO nous semble importante pour faciliter son adoption et le développement d'outils dédiés.

Après l'inclusion de la vérification nous pourrions réfléchir à quelles autres analyses pourraient être faites avec ces données. Il est par exemple possible d'envisager un benchmarking des différentes topologies avec un choix des critères propres aux besoins du client.

Notre modèle de VNF pourrait être étendu à d'autres contraintes réseau (comme les protocoles utilisés) et nous travaillons actuellement à l'introspection automatique du système en cours d'exécution pour construire le modèle de configuration réseau et pour découvrir :

1. comment les commutateurs des centres de données sont connectés aux commutateurs/routeurs du réseau.
2. et si certaines VNF sont déjà déployées sans utiliser notre modèle VNF :
   (a) les numéros de port utilisés par leurs micro-services,
   (b) comment les micro-services sont déployés dans les différents centres de données,
   (c) comment ils sont reliés aux commutateurs internes des centres de données.

La collecte des informations ci-dessus nous amènerait à construire automatiquement une topologie de l'infrastructure complète (informatique et réseau).

La solution développée est valable pour un environnement statique, où les micro-services ne migrent pas d'un centre de données à un autre. La construction d'un tel modèle, en particulier le modèle de configuration du réseau, qui peut être très dynamique, pourrait être sujet aux erreurs. Mais nous pouvons l'obtenir par l'introspection du système.

BIBLIOGRAPHIE

[1] V.-G. Nguyen, A. Brunstrom, K.-J. Grinnemo, and J. Taheri, "Sdn/nfv-based mobile packet core network architectures : a survey," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 3, pp. 1567–1602, 2017.

[2] M. Chiosi, D. Clarke, P. Willis, A. Reid, J. Feger, M. Bugenhagen, W. Khan, M. Fargano, C. Cui, H. Deng *et al.*, "Network functions virtualisation : An introduction, benefits, enablers, challenges and call for action," in *SDN and OpenFlow World Congress*, 2012, pp. 22–24.

[3] I. Alawe, Y. Hadjadj-Aoul, A. Ksentini, P. Bertin, and D. Darche, "On the scalability of 5G Core network : the AMF case," in *CCNC 2018 - IEEE Consumer Communications and Networking Conference*, Las Vegas, United States, Jan. 2018, pp. 1–6. [Online]. Available : https://hal.inria.fr/hal-01657667

[4] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese *et al.*, "P4 : Programming protocol-independent packet processors," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 87–95, 2014.

[5] P. Wu, Y. Cui, J. Wu, J. Liu, and C. Metz, "Transition from ipv4 to ipv6 : A state-of-the-art survey," *IEEE Communications Surveys & Tutorials*, vol. 15, no. 3, pp. 1407–1424, 2013.

[6] K. Nichols, S. Blake, F. Baker, and D. Black, "Definition of the differentiated services field (ds field) in the ipv4 and ipv6 headers," Tech. Rep., 1998.

[7] E. Hernandez-Valencia, S. Izzo, and B. Polonsky, "How will nfv/sdn transform service provider opex ?" *IEEE Network*, vol. 29, no. 3, pp. 60–67, 2015.

[8] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese *et al.*, "P4 : Programming protocol-independent packet processors," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 87–95, 2014.

[9] D. Durham, J. Boyle, R. Cohen, S. Herzog, R. Rajan, and A. Sastry, "The cops (common open policy service) protocol," Tech. Rep., 1999.

[10] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow : enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[11] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu *et al.*, "B4 : Experience with a globally-deployed software defined wan," in *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4. ACM, 2013, pp. 3–14.

[12] A. Singh, J. Ong, A. Agarwal, G. Anderson, A. Armistead, R. Bannon, S. Boving, G. Desai, B. Felderman, P. Germano, A. Kanagala, J. Provost, J. Simmons, E. Tanda, J. Wanderer, U. Hölzle, S. Stuart, and A. Vahdat, "Jupiter rising : A decade of clos topologies and centralized control in google's datacenter network," in *Sigcomm '15*, 2015.

[13] J. Pelay, F. Guillemin, and O. Barais, "Verifying the configuration of virtualized network functions in software defined networks," in *Network Function Virtualization and Software Defined Networks (NFV-SDN), 2017 IEEE Conference on*. IEEE, 2017, pp. 223–228.

[14] N. Paladi, "Towards secure sdn policy management," in *Utility and Cloud Computing (UCC), 2015 IEEE/ACM 8th International Conference on*. IEEE, 2015, pp. 607–611.

[15] E. Rojas, R. Doriguzzi-Corin, S. Tamurejo, A. Beato, A. Schwabe, K. Phemius, and C. Guerrero, "Are we ready to drive software-defined networks ? a comprehensive survey on management tools and techniques," *ACM Computing Surveys (CSUR)*, vol. 51, no. 2, p. 27, 2018.

[16] N. Paladi, "Towards secure sdn policy management," in *Utility and Cloud Computing (UCC), 2015 IEEE/ACM 8th International Conference on*. IEEE, 2015, pp. 607–611.

[17] C. Metter, M. Seufert, F. Wamser, T. Zinner, and P. Tran-Gia, "Analytical model for sdn signaling traffic and flow table occupancy and its application for various types of traffic," *IEEE Transactions on Network and Service Management*, vol. 14, no. 3, pp. 603–615, 2017.

[18] T. Sato, S. Ata, I. Oka, and Y. Sato, “Abstract model of sdn architectures enabling comprehensive performance comparisons,” in *Network and Service Management (CNSM), 2015 11th International Conference on*. IEEE, 2015, pp. 99–107.

[19] D. Sethi, S. Narayana, and S. Malik, “Abstractions for model checking sdn controllers.” in *FMCAD*. Citeseer, 2013, pp. 145–148.

[20] T. Soenen, S. Sahhaf, W. Tavernier, P. Sköldström, D. Colle, and M. Pickavet, “A model to select the right infrastructure abstraction for service function chaining,” in *Network Function Virtualization and Software Defined Networks (NFV-SDN), IEEE Conference on*. IEEE, 2016, pp. 233–239.

[21] E. Haleplidis, J. H. Salim, S. Denazis, and O. Koufopavlou, “Towards a network abstraction model for sdn,” *Journal of Network and Systems Management*, vol. 23, no. 2, pp. 309–327, 2015.

[22] K. Giotis, Y. Kryftis, and V. Maglaris, “Policy-based orchestration of NFV services in Software-Defined Networks.” IEEE, Apr. 2015, pp. 1–5. [Online]. Available : http://ieeexplore.ieee.org/document/7116145/

[23] A. Panda, K. Argyraki, M. Sagiv, M. Schapira, and S. Shenker, “New directions for network verification,” in *LIPIcs-Leibniz International Proceedings in Informatics*, vol. 32. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2015.

[24] G. G. Xie, J. Zhan, D. A. Maltz, H. Zhang, A. Greenberg, G. Hjalmtysson, and J. Rexford, “On static reachability analysis of ip networks,” in *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, vol. 3. IEEE, 2005, pp. 2170–2183.

[25] E. Al-Shaer and S. Al-Haj, “Flowchecker : Configuration analysis and verification of federated openflow infrastructures,” in *Proceedings of the 3rd ACM workshop on Assurable and usable security configuration*. ACM, 2010, pp. 37–44.

[26] E. Al-Shaer, W. Marrero, A. El-Atawy, and K. ElBadawi, “Network configuration in a box : towards end-to-end verification of network reachability and security.” IEEE, Oct. 2009, pp. 123–132. [Online]. Available : http://ieeexplore.ieee.org/document/5339690/

[27] E. Al-Shaer and M. N. Alsaleh, “ConfigChecker : A tool for comprehensive security configuration analytics.” IEEE, Oct. 2011, pp. 1–2. [Online]. Available : http://ieeexplore.ieee.org/document/6111667/

[28] P. Kazemian, “Header space analysis : Static checking for networks.” 2012.

[29] Y. Yang, X. Huang, S. Cheng, S. Chen, and P. Cong, “Shsa : A method of network verification with stateful header space analysis,” in *2016 IEEE 22nd International Conference on Parallel and Distributed Systems (ICPADS)*. IEEE, 2016, pp. 232–238.

[30] P. Kazemian, M. Chan, H. Zeng, G. Varghese, N. McKeown, and S. Whyte, “Real time network policy checking using header space analysis.” in *NSDI*, 2013, pp. 99–111.

[31] H. Hu, W. Han, G.-J. Ahn, and Z. Zhao, “Flowguard : building robust firewalls for software-defined networks,” in *Proceedings of the third workshop on Hot topics in software defined networking*. ACM, 2014, pp. 97–102.

[32] T. L. Hinrichs, N. S. Gude, M. Casado, J. C. Mitchell, and S. Shenker, “Practical declarative network management,” in *Proceedings of the 1st ACM workshop on Research on enterprise networking*. ACM, 2009, pp. 1–10.

[33] C. Basile, A. Cappadonia, and A. Lioy, “Network-level access control policy analysis and transformation,” *IEEE/ACM Transactions on Networking (TON)*, vol. 20, no. 4, pp. 985–998, 2012.

[34] C. Basile, D. Canavese, A. Lioy, and F. Valenza, “Inter-technology conflict analysis for communication protection policies,” in *International Conference on Risks and Security of Internet and Systems*. Springer, 2014, pp. 148–163.

[35] C. Basile, D. Canavese, A. Lioy, C. Pitscheider, and F. Valenza, “Inter-function anomaly analysis for correct sdn/nfv deployment,” *International Journal of Network Management*, vol. 26, no. 1, pp. 25–43, 2016.

[36] M. Canini, D. Venzano, P. Peresini, D. Kostic, and J. Rexford, “A nice way to test openflow applications,” in *Proceedings of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, no. EPFL-CONF-170618, 2012.

[37] R. Skowyra, A. Lapets, A. Bestavros, and A. Kfoury, “A verification platform for sdn-enabled applications,” in *Cloud Engineering (IC2E), 2014 IEEE International Conference on*. IEEE, 2014, pp. 337–342.

[38] A. Khurshid, W. Zhou, M. Caesar, and P. Godfrey, “Veriflow : Verifying network-wide invariants in real time,” in *Proceedings of the first workshop on Hot topics in software defined networks*. ACM, 2012, pp. 49–54.

[39] T. Ball, N. Bjørner, A. Gember, S. Itzhaky, A. Karbyshev, M. Sagiv, M. Schapira, and A. Valadarsky, "Vericon : Towards verifying controller programs in software-defined networks," in *ACM Sigplan Notices*, vol. 49, no. 6. ACM, 2014, pp. 282–293.

[40] L. De Moura and N. Bjørner, "Z3 : An efficient smt solver," in *International conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2008, pp. 337–340.

[41] H. Zeng, S. Zhang, F. Ye, V. Jeyakumar, M. Ju, J. Liu, N. McKeown, and A. Vahdat, "Libra : Divide and conquer to verify forwarding tables in huge networks." in *NSDI*, vol. 14, 2014, pp. 87–99.

[42] J. Dean and S. Ghemawat, "Mapreduce : simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[43] A. Horn, A. Kheradmand, and M. R. Prasad, "Delta-net : Real-time network verification using atoms." in *NSDI*, 2017, pp. 735–749.

[44] M. Kang, J.-Y. Choi, H. H. Kwak, I. Kang, M.-K. Shin, and J.-H. Yi, "Formal modeling and verification for sdn firewall application using pacsr," in *Electronics, Communications and Networks IV : Proceedings of the 4th International Conference on Electronics, Communications and Networks (CECNET IV), Beijing, China, 12–15 December 2014*. CRC Press, 2015, p. 155.

[45] M.-K. Shin, Y. Choi, H. H. Kwak, S. Pack, M. Kang, and J.-Y. Choi, "Verification for nfv-enabled network services," in *Information and Communication Technology Convergence (ICTC), 2015 International Conference on*. IEEE, 2015, pp. 810–815.

[46] P. Brémond-Grégoire, I. Lee, and R. Gerber, "Acsr : An algebra of communicating shared resources with dense time and priorities," in *International Conference on Concurrency Theory*. Springer, 1993, pp. 417–431.

[47] C. J. Anderson, N. Foster, A. Guha, J.-B. Jeannin, D. Kozen, C. Schlesinger, and D. Walker, "NetKAT : Semantic foundations for networks," ser. POPL '14. New York, NY, USA : ACM, 2014, pp. 113–126. [Online]. Available : http://doi.acm.org/10.1145/2535838.2535862

[48] S. Smolka, S. Eliopoulos, N. Foster, and A. Guha, "A fast compiler for netkat," *ACM SIGPLAN Notices*, vol. 50, no. 9, pp. 328–341, 2015.

[49] D. Kozen, “A completeness theorem for kleene algebras and the algebra of regular events,” *Inf. Comput.*, vol. 110, no. 2, pp. 366–390, May 1994.

[50] ——, “Kleene algebra with tests,” *ACM Trans. Program. Lang. Syst.*, vol. 19, no. 3, pp. 427–443, May 1997.

[51] K. G. Larsen, S. Schmid, and B. Xue, “Wnetkat : A weighted sdn programming and verification language,” *arXiv preprint arXiv :1608.08483*, 2016.

[52] H. Kim, A. Gupta, M. Shahbaz, J. Reich, N. Feamster, and R. Clark, “Kinetic : Verifiable dynamic network control,” in *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, Oakland, CA, May 2015.

[53] J. Reich, C. Monsanto, N. Foster, J. Rexford, and D. Walker, “Modular sdn programming with pyretic,” *Technical Reprot of USENIX*, 2013.

[54] A. Cimatti, E. Clarke, F. Giunchiglia, and M. Roveri, “Nusmv : a new symbolic model checker,” *International Journal on Software Tools for Technology Transfer*, vol. 2, no. 4, pp. 410–425, 2000.

[55] R. Beckett, X. K. Zou, S. Zhang, S. Malik, J. Rexford, and D. Walker, “An assertion language for debugging sdn applications,” in *Proceedings of the third workshop on Hot topics in software defined networking*. ACM, 2014, pp. 91–96.

[56] B. Meyer, “Eiffel : programming for reusability and extendibility,” *ACM Sigplan Notices*, vol. 22, no. 2, pp. 85–94, 1987.

[57] ——, “Applying’design by contract’,” *Computer*, vol. 25, no. 10, pp. 40–51, 1992.

[58] S. Spinoso, M. Virgilio, W. John, A. Manzalini, G. Marchetto, and R. Sisto, “Formal Verification of Virtual Network Function Graphs in an SP-DevOps Context,” in *Service Oriented and Cloud Computing*, S. Dustdar, F. Leymann, and M. Villari, Eds. Cham : Springer International Publishing, 2015, vol. 9306, pp. 253–262. [Online]. Available : http://link.springer.com/10.1007/978-3-319-24072-5_18

[59] R. Majumdar, S. D. Tetali, and Z. Wang, “Kuai : A model checker for software-defined networks,” in *Formal Methods in Computer-Aided Design (FMCAD), 2014*. IEEE, 2014, pp. 163–170.

[60] D. Lebrun, S. Vissicchio, and O. Bonaventure, “Towards test-driven software defined networking,” in *Network Operations and Management Symposium (NOMS), 2014 IEEE*. IEEE, 2014, pp. 1–9.

[61] B. Heller, R. Sherwood, and N. McKeown, “The controller placement problem,” in *Proceedings of the first workshop on Hot topics in software defined networks*. ACM, 2012, pp. 7–12.

[62] J. K. Ousterhout, A. R. Cherenson, F. Douglis, M. N. Nelson, and B. B. Welch, “The sprite network operating system,” *Computer*, vol. 21, no. 2, pp. 23–36, 1988.

[63] D. Awduche, J. Malcolm, J. Agogbua, M. O’Dell, and J. McManus, “Requirements for traffic engineering over mpls,” Tech. Rep., 1999.

[64] K. Yap, M. Motiwala, J. Rahe, S. Padgett, M. Holliman, G. Baldus, M. Hines, T. Kim, A. Narayanan, A. Jain, V. Lin, C. Rice, B. Rogan, A. Singh, B. Tanaka, M. Verma, P. Sood, M. Tariq, M. Tierney, D. Trumic, V. Valancius, C. Ying, M. Kallahalla, B. Koley, and A. Vahdat, “Taking the edge off with espresso : Scale, reliability and programmability for global internet peering,” 2017. [Online]. Available : http://dl.acm.org/ft_gateway.cfm?id=3098854&ftid=1898911&dwn=1&CFID=776908660&CFTOKEN=29525737

[65] J. Roberts and S. Oueslati, “Quality of service by flow aware networking,” *Philosophical Transactions of The Royal Society of London, Series A*, vol. 358, no. 1773, 2000.

[66] N. Foster, D. Kozen, M. Milano, A. Silva, and L. Thompson, “A coalgebraic decision procedure for NetKAT,” in *Proc. POPL 2015*, 2015.

[67] D. Namiot and M. Sneps-Sneppe, “On micro-services architecture,” *International Journal of Open Information Technologies*, vol. 2, no. 9, pp. 24–27, 2014.

[68] D. Jaramillo, D. V. Nguyen, and R. Smart, “Leveraging microservices architecture by using docker technology,” in *SoutheastCon, 2016*. IEEE, 2016, pp. 1–5.

[69] C. Dumez, M. Bakhouya, J. Gaber, M. Wack, and P. Lorenz, “Model-driven approach supporting formal verification for web service composition protocols,” *Journal of network and computer applications*, vol. 36, no. 4, pp. 1102–1115, 2013.

[70] K. Katsalis, N. Nikaein, E. Schiller, R. Favraud, and T. I. Braun, “5g architectural design patterns,” in *Communications Workshops (ICC), 2016 IEEE International Conference on*. IEEE, 2016, pp. 32–37.

[71] W. John, F. Moradi, B. Pechenot, and P. Sköldström, “Meeting the observability challenges for vnfs in 5g systems.”

[72] A. Sheoran, X. Bu, L. Cao, P. Sharma, and S. Fahmy, “An empirical case for container-driven fine-grained vnf resource flexing,” in *Network Function Virtualization and Software Defined Networks (NFV-SDN), IEEE Conference on*. IEEE, 2016, pp. 121–127.

[73] P. B. Kruchten, “The 4+ 1 view model of architecture,” *IEEE software*, vol. 12, no. 6, pp. 42–50, 1995.

[74] D. Giannakopoulou and J. Magee, “Fluent model checking for event-based systems,” in *ACM SIGSOFT Software Engineering Notes*, vol. 28, no. 5. ACM, 2003, pp. 257–266.

[75] A. Beugnard, J.-M. Jézéquel, N. Plouzeau, and D. Watkins, “Making components contract aware,” *Computer*, vol. 32, no. 7, pp. 38–45, 1999.

[76] K. Gierlowsky, “Ubiquity of client access in heterogeneous access environment,” *Journal of Telecommunications and Information technology*, 2014.

[77] P. Ravali, S. K. Vasudevan, and R. Sundaram, “Open air interface–adaptability perspective,” *Indian Journal of Science and Technology*, vol. 9, no. 6, 2016.

[78] R. van Glabbeek and U. Goltz, “Equivalence notions for concurrent systems and refinement of actions,” in *Mathematical Foundations of Computer Science 1989*. Springer, 1989, pp. 237–248.

[79] R. Milner, “A calculus of communicating systems,” 1980.

[80] J. Magree, “Behavioral analysis of software architectures using ltsa,” in *Software Engineering, 1999*. IEEE, 1999, pp. 634–637.

[81] B. Pfaff, J. Pettit, K. Amidon, M. Casado, T. Koponen, and S. Shenker, “Extending networking into the virtualization layer.” in *Hotnets*, 2009.

[82] M. Peuster, H. Karl, and S. Van Rossem, “Medicine : Rapid prototyping of production-ready network services in multi-pop environments,” *arXiv preprint arXiv :1606.05995*, 2016.

[83] “Open network automation project,” https://www.onap.org/, accessed : 2017-07-09.

[84] B. Han, V. Gopalakrishnan, L. Ji, and S. Lee, “Network function virtualization : Challenges and opportunities for innovations,” *IEEE Communications Magazine*, vol. 53, no. 2, pp. 90–97, 2015.

[85] H. J. Goldsby, B. H. Cheng, and J. Zhang, “Amoeba-rt : Run-time verification of adaptive software,” in *International Conference on Model Driven Engineering Languages and Systems*. Springer, 2007, pp. 212–224.

[86] B. Morin, O. Barais, G. Nain, and J.-M. Jezequel, “Taming dynamically adaptive systems using models and aspects,” in *Software Engineering, 2009. ICSE 2009. IEEE 31st International Conference on*. IEEE, 2009, pp. 122–132.

[87] A. Filieri and G. Tamburrelli, *Probabilistic Verification at Runtime for Self-Adaptive Systems*. Berlin, Heidelberg : Springer Berlin Heidelberg, 2013, pp. 30–59.

UNIVERSITE BRETAGNE LOIRE / MATHSTIC

UNIVERSITÉ DE RENNES 1

Titre : Garantir la cohérence applicative dans un environnement SDN

Mot clés : SDN, NFV, vérification, VNF

Resumé : Des solutions de vérification efficaces nous semblent indispensables afin d'assurer la continuité des services en place et le déploiement de nouveaux services dans les réseaux 5G. Cette problématique ne se limite pas aux techniques de vérification, nos travaux portent aussi sur la formalisation des besoins d'un service réseau complet. Nous avons travaillé sur un langage de programmation facilitant la vérification dans les contrôleurs SDN. Puis nous avons défini des contrats de comportement réseau afin de clarifier les propriétés à vérifier pour assurer le bon fonctionnement d'un service. Enfin nous avons étudié MANO pour proposer une extension du standard permettant de vérifier les configurations réseau au niveau de l'orchestrateur.

Title : Ensure application consistency in SDN environment

Keywords : SDN, NFV, verification, VNF

Abstract : We believe that effective verification solutions are essential to ensure the continuity of existing services and the deployment of new services in 5G networks. This problem is not limited to verification techniques, our work also concerns the formalization of needs of a complete network service. We worked on a programming language to facilitate verification in SDN controllers. Then we defined network behavior contracts to clarify the properties to be verified to ensure the proper functioning of a service. Finally, we studied MANO to propose an extension of the standard to check the network configurations at the orchestrator level.