UNIVERSITÉ DU QUÉBEC À MONTRÉAL

SYSTÈME DE GESTION DES RÉSEAUX VIRTUELS DANS LE CONTEXTE DE L'INFORMATIQUE EN NUAGE

MÉMOIRE

PRÉSENTÉ

COMME EXIGENCE PARTIELLE

DE LA MAÎTRISE EN INFORMATIQUE

PAR

AYMEN FRIKHA

NOVEMBRE 2015

REMERCIEMENTS

Je tiens à exprimer mes plus profonds remerciements au Prof. Omar Cherkaoui qui m'a fait profiter de sa vaste culture, de ses conseils judicieux et de ses directives qui m'ont servi de meilleures aides durant la période d'encadrement.

Je remercie également Prof. Sylvain Hallé de m'avoir aidé à acquérir des connaissances nouvelles en matière de formalisation et de modélisation. Il m'a aussi donné son support pour avoir des méthodes scientifiques et une discipline de travail. Je le remercie également pour tout le temps et l'énergie qu'il a investi pour suivre mon avancement et me faire part de ses remarques pertinentes.

Je présente aussi ma profonde gratitude à tous les membres de l'équipe LTIR pour leur esprit de collaboration et en particulier Madame Amina Mseddi, Monsieur Khalil Blaiech, Monsieur Salaheddine Hamadi, Monsieur Ilyas Snaiki et Monsieur Omar Mounaouar.

Enfin, je tiens à remercier tous ceux qui m'ont aidé de près ou de loin dans cette maîtrise et que j'ai oublié de mentionner. À tous un très grand merci.

TABLE DES MATIÈRES

LISTE DES FIGURES

LISTE DES TABLEAUX

LEXIQUE

- ***Alloy***: Langage et outil de formalisation et de validation des modèles
- ***API (Application Programming Interface)***: Une interface de programmation
- ***Blade***: Serveur lame
- ***Broadcasting***: Domaine de diffusion
- ***CIDR (Classless Inter-Domain Routing)***: Permet la classification des adresses IP en classe A, B et C
- ***Cisco et Juniper***: Célèbres entreprises de commercialisation d'équipements réseau
- ***Cloud Stack***: Plate-forme libre de l'informatique en nuage
- ***CPU (Central Processing Unit)***: Unité centrale de traitement
- ***DMTF (Distributed Management Task Force)***: Une organisation qui développe et maintient des standards pour l'administration de systèmes informatiques
- ***EoMPLS (Ethernet over MPLS)***: Protocole permet d'étendre le domaine de la couche 2
- ***EVB (Edge Virtual Bridging)***: Une norme IEEE pour les systèmes virtualisés
- ***Fibre Channel over Ethernet (FCoE)***: Un protocole qui encapsule les trames *Fibre Channel*
- ***GRE (Generic Routing Encapsulation)***: Un protocole de tunneling développé par Cisco
- ***IDS***: Système de détection des intrusions
- ***IPS***: Système de protection contre les intrusions
- ***IPv4***: La version 4 du protocole IP

- ***IPv6***: La version 6 du protocole IP
- ***KVM (Kernel-based Virtual Machine)***: Hyperviseur du noyau Linux
- ***KSP (K Shortest Path)***: Algorithme de calcule du plus court chemin
- ***Line Card***: Carte de ligne, carte électronique
- ***LTIR***: Laboratoire de téléinformatique et de réseau de l'UQAM
- ***MPLS (Multiprotocol Label Switching)***: Un protocole de commutation indépendant des autres protocoles de transport
- ***MySQL***: Est un système de gestion de bases de données relationnelles
- ***NAS (Network Area Storage)***: Réseau reliant des équipements de stockage
- ***NETCONF***: Protocole de configuration des équipements réseau
- ***Neutron***: Projet de *OpenStack* pour la gestion du réseau virtuel
- ***Nexus1000V***: Commutateur virtuel conçu par *Cisco*
- ***OpenFlow***: Protocole conçu pour le concept *SDN*
- ***OpenStack***: Plate-forme libre de l'informatique en nuage
- ***OTV (Overlay Transport Virtualization)***: Protocole conçu par Cisco pour l'extension du domaine de la couche 2
- ***Overlay network***: Réseau superposé
- ***OVS***: Commutateur virtuel conçu par *Nicira*
- ***RAM (Random Access Memory)***: Mémoire à accès aléatoire
- ***Router***: Routeur
- ***SDN (Software Defined Network)***: Réseaux définis de manière logicielle
- ***SQLAlchemy***: Est un toolkit *open source SQL*
- ***Switch***: Commutateur
- ***VDC (Virtual Data Center)***: Centre de données virtuel

- ***VLAN***: Réseau local virtuel
- ***VM***: Machine virtuelle
- ***VNIC (Virtual NIC)***: Interface virtuelle
- ***VPN***: Réseau virtuel privé
- ***VRF (Virtual Router Functions)***: Fonctions d'un routeur virtuel
- ***Vsphere***: Plate-forme de l'informatique en nuage conçu par *VMware*
- ***VxLAN***: Réseau local virtuel extensible
- ***XEN***: Célèbre hyperviseur de Linux initié par l'université de Cambridge
- ***XML (Extensible Markup Language)***: langage de balisage extensible

RÉSUMÉ

La venue de l'informatique en nuage a suscité un grand besoin d'une isolation efficace et automatique des locataires dans les centres de données. Cette isolation ne peut être garantie qu'avec la considération simultanée des infrastructures physiques et virtuelles. Cependant, la plupart des systèmes de gestion des réseaux virtuels pour les plate-formes de l'informatique en nuage ne prennent pas en considération la dépendance entre ces deux types d'infrastructures lors de la gestion des réseaux virtuels.

Dans ce mémoire, nous avons proposé une solution pour l'anticipation des conflits lors de la création des réseaux *Overlay* pour les plates-formes de l'informatique en nuage. Dans notre approche, nous avons considéré comme cruciale la dépendance entre l'infrastructure physique et l'infrastructure virtuelle. Le modèle *CMnet* que nous avons élaboré décrit cette dépendance. Ce modèle fournit une abstraction des différents paramètres de l'infrastructure d'un centre de données multi-locataire tel que les topologies et les services du réseau. En effet, nous avons utilisé le langage d'abstraction de la configuration *Meta-Cli* pour la représentation des services et des configurations des équipements réseau. Par la suite, ce modèle a été validé grâce au langage de validation *Alloy*.

Notre approche a été mise en œuvre dans la plate-forme *Openstack*, à travers la conception et le développement d'un plug-in pour Neutron appelé *MAP* basé sur *CMnet* et sur *Meta-Cli*. *MAP* nous a permis d'avoir une maîtrise presque complète de l'infrastructure du centre de données et d'effectuer des opérations complexes telles que la validation des réseaux *Overlay* et leurs adaptations aux caractéristiques de l'infrastructure physique. En effet, les tests réalisés sur *MAP* à travers trois scénarios différents ont permis d'affirmer la détection des conflits sur les services et la topologie du réseau.

INTRODUCTION

L'informatique en nuage est un modèle relativement nouveau dans le monde de l'informatique. Selon la définition du *NIST (National Institue of Standard and technologie)* [32], l'informatique en nuage est un modèle permettant l'accès omniprésent et pratique aux réseaux contenant un ensemble de ressources informatiques (réseau, serveurs, disques de stockage, applications, services...) configurables, partagées et pouvent être rapidement approvisionnées ou libérées avec un effort minimal de gestion et d'interaction avec le fournisseur.

De nos jours, la culture de l'informatique en nuage s'est relativement propagée, plusieurs personnes utilisent les services de stockage en ligne tels que *Dropbox* ou *Google Drive* ainsi que d'autres services relatifs au modèle *Iaas (Infrastructure as a service)* tels que la location des machines virtuelles. L'augmentation rapide des demandes pour les services de l'informatique en nuage engendre une évolution exponentielle des centres de données. Cette évolution est caractérisée essentiellement par l'ajout de ressources physiques et matérielles telles que les serveurs, les routeurs et les équilibreurs de charge.

L'augmentation des demandes pour l'informatique en nuage nécessite aussi une infrastructure multi-locataires plus robuste. Il est essentiel de partager la même infrastructure physique, que ce soit pour les nuages publics qui hébergent plusieurs clients avec des exigences et des niveaux de services différents, ou pour les nuages privés partagés par plusieurs organismes et départements. Cependant, l'infrastructure multi-locataires rend la gestion d'un centre de données plus difficile à réaliser. Les locataires d'un centre de données doivent être isolés à tous les niveaux et surtout au niveau du réseau. L'isolation au niveau du réseau s'effectue grâce aux technologies de virtualisation des réseaux qu'on appelle les réseaux *Overlay* [7]. Chaque locataire doit avoir son propre réseau *Overlay* qui lui garantit l'isolation par rapport aux autres locataires du nuage. Chaque réseau *Overlay* est en réalité un service tel que le *VLAN* [30], *VxLAN* [28], *VPN* [39], etc . Pour garantir le bon fonctionnement des réseaux des locataires, il faut que les propriétés de ces services soient valides par rapport aux propriétés déjà présentes dans les configurations des équipements réseau de l'infrastructure physique.

Comment s'assurer que les services d'isolation du trafic fonctionnent correctement ?

C'est un défi de taille surtout en raison de la complexité des infrastructures et du nombre des équipements réseau présents dans les centres de données. De plus, chaque service réseau nécessite une logique de configuration et des paramètres totalement différents des autres services. La majorité des tâches pour la configuration des équipements est réalisée manuellement [18], ce qui les rend très lents à mettre en place et sujets à des erreurs humaines.

Pour implémenter ces services, il faut que l'ingénieur réseau s'assure que la configuration des équipements physiques et virtuels soit correcte. Cela implique que les nouvelles configurations qui vont s'ajouter au fur et à mesure de l'ajout des services dans le réseau ne doivent pas entrer en conflit avec la configuration existante [16]. Cela limite considérablement l'évolutivité et la flexibilité des centres de données et augmente la surcharge administrative.

L'un des plus grands défis pour l'automatisation des configurations de services dans un centre de données est l'adaptation à une topologie très instable et sujette à des modifications fréquentes. Tous les réseaux *Overlay* dans un centre de données ainsi que les autres services tels que les services de sécurité sont très dépendants de la topologie. Les administrateurs réseau doivent avoir une vue globale sur la localisation des équipements physiques et virtuels pour pouvoir configurer les services et s'assurer de leur bon fonctionnement.

Plusieurs services peuvent être utilisés pour les réseaux *Overlay*. La dépendance entre ces services est nécessaire pour assurer l'isolation du réseau du locataire. L'utilisation de plusieurs services pour un même réseau *Overlay* est exigée parfois par la topologie du réseau tel que l'existence de plusieurs domaines de *Broadcasting* dans un même réseau physique ou pour d'autres raisons telles que la sécurité et le chiffrement des données.

Objectifs et Contribution

Dans notre travail, nous proposons une solution permettant l'anticipation des conflits et erreurs lors de la création des réseaux *Overlay* pour les plates-formes de l'informatique en nuage. Cette solution permet de s'assurer du bon fonctionnement des services réseau en effectuant des opérations de validation sur l'infrastructure physique et en détectant les problèmes et les conflits avec le nouveau réseau virtuel aussi bien au niveau de la configuration des équipements qu'au niveau de la topologie du réseau. De plus, notre solution est capable d'adapter le réseau *Overlay* aux caractéristiques de l'infrastructure déjà en place.

L'aboutissement de ces objectifs est réalisé à l'aide du langage d'abstraction de la con-

figuration (Meta-Cli) [8] et du modèle d'abstraction de l'infrastructure physique et virtuelle. À travers le langage *Meta-Cli* et des règles de validation, le système est capable de valider le bon fonctionnement des services du réseau *Overlay* avec les services et les configurations déjà existants dans les équipements réseau. Le modèle d'abstraction, quant à lui, nous permet de prendre en considération d'autres propriétés intéressantes de l'infrastructure physique telle que la topologie du réseau, c'est à dire l'extraction des chemins entre les différents nœuds du réseau ainsi que l'extraction des différents domaines de *Broadcasting*.

OpenStack [38], est de nos jours la plate-forme *OpenSource* la plus utilisée et la plus performante dans le domaine de l'informatique en nuage. C'est pour cette raison qu'on a décidé de contribuer à ce grand projet à travers un nouveau plug-in appelé *MAP*. Ce plug-in représente la concrétisation de notre solution et s'intègre au projet *Neutron* [23] de *OpenStack* qui est responsable de la gestion du réseau pour la plate-forme.

Division du document

Le chapitre 1 présente un aperçu du problème de la configuration des équipements réseau dans un centre de données et illustre ce problème par quelques exemples de services et d'applications que propose l'informatique en nuage.

Dans le chapitre 2, nous décrivons plusieurs concepts et technologies qui ont été mis en œuvre pour tenter de remédier aux problèmes déjà évoqués. De plus, nous allons présenter les forces et faiblesses de chacun d'entre eux et leurs différences avec notre solution.

Le chapitre 3 présente une description détaillée du modèle *CMnet* et de ses différents composants. La validation de ce modèle sera aussi évoquée en présentant sa formalisation avec le langage *Alloy* [20] ainsi que la formalisation des différentes contraintes et règles qui sont appliquées sur la topologie du réseau et sur les configurations des équipements.

Dans le chapitre 4, nous présentons la conception et l'implémentation du nouveau plug-in *MAP* de la plate-forme de l'informatique en nuage *OpenStack*. Nous allons aussi détailler dans ce chapitre le rôle du modèle que nous avons conçu ainsi que celui du langage *Meta-Cli* dans l'aboutissement de nos objectifs.

Le chapitre 5 décrit les résultats des expérimentations de notre plug-in *MAP* sur des infrastructures simples et complexes d'un centre de données.

La conclusion fait le point sur la contribution du travail, les points forts et faibles de notre solution ainsi que les perspectives vers lesquelles notre nouvelle méthode peut nous amener.

CHAPITRE I

GESTION D'UN CENTRE DE DONNÉES

De nos jours, il n'existe pas de solution nous permettant de gérer l'infrastructure physique et virtuelle d'un centre de données. L'existence de telles solutions pourrait alléger considérablement les lourdes tâches des administrateurs réseau et remédier aux problèmes et aux défaillances que rencontrent souvent les systèmes informatiques. La difficulté de la conception d'une telle solution réside dans la complexité des infrastructures des centres de données et surtout dans l'intégration de la virtualisation dans la majorité des équipements de ces centres. Dans ce chapitre, nous allons détailler cette problématique et comprendre la composition d'un centre de données comportant des plates-formes d'informatique en nuage tel que *OpenStack*.

1.1 Infrastructure d'un centre de données

Les centres de données modernes possèdent une très large gamme d'équipements physiques et virtuels. Ces derniers peuvent provenir de fournisseurs et de technologies différents tels que **Cisco** [19] et **Juniper** [9] pour les commutateurs et les routeurs ou **KVM** [22] et **XEN** [5] pour les hyperviseurs. Bien que ces produits soient technologiquement différents, ils peuvent toutefois interagir pour offrir des services d'informatique en nuage. Cette hétérogénéité complexifie considérablement la gestion des infrastructures. Dans cette section, nous allons détailler les différents composantes des centres de données et le rôle de chaque type d'équipement pour la garantie des services de l'informatique en nuage.

1.1.1 Infrastructure physique

Les infrastructures modernes des centres de données contiennent plusieurs types d'équipements tels que :

- **Les serveurs de calcul:** Ce sont de grands serveurs qui possèdent une grande capacité de calcul et de mémoire. Ces serveurs sont dédiés à l'hébergement des machines virtuelles.
- **Les serveurs de stockages ou *NAS* (*Network Attached Storage*):** Ce sont des serveurs de fichiers autonomes, reliés à un réseau dont la principale fonction est le stockage de données dans un volume centralisé pour des clients de réseaux hétérogènes. Les serveurs de stockage peuvent utiliser des services réseau spécifiques tels que *FCoE* (*Fibre Channel over Ethernet*).
- **Les commutateurs de la couche d'accès:** La couche d'accès contient des dispositifs qui permettent aux groupes de travail et aux utilisateurs d'utiliser les services fournis par les couches de distribution et de base. Dans la couche d'accès, il est possible d'étendre ou de contracter des domaines de collision en utilisant un répéteur, un concentrateur ou un commutateur standard.
- **Les commutateurs de la couche de distribution:** C'est à partir de cette couche que nous commençons à exercer un contrôle sur les transmissions réseau. Cette couche permet également de limiter et créer des domaines de diffusion et de créer des réseaux locaux virtuels.
- **Les commutateurs de la couche de base:** La couche de base est responsable du transport rapide et fiable de données sur un réseau. La couche centrale est souvent contenue dans les réseaux de fédérateurs, car toutes les autres couches reposent sur elle. Son but est de réduire le temps de latence de la transmission de paquets. Par ailleurs, les facteurs à prendre en considération lors de la conception des appareils à utiliser dans la couche de base sont:
  - Un débit du réseau élevé.
  - Une faible latence du réseau.
- **Les équipements de sécurité:** Les équipements de sécurité dans un centre de données permettent de protéger l'infrastructure du centre de données et ses clients contre des

attaques qui peuvent survenir dans le réseau. Plusieurs types d'équipements peuvent être utilisés tels que les pare-feu, les systèmes de détection d'intrusions (*IDS*), les systèmes de protection d'intrusion (*IPS*) et les équilibreurs de charges. L'utilisation de ces équipements peut garantir une meilleure qualité de service pour les clients.

Cette liste non exhaustive des types d'équipements dans un centre de données basique nous montre la complexité de l'infrastructure ainsi que son hétérogénéité. Ceci rend la gestion automatique et centrale du centre de données très difficile.

1.1.2 Infrastructure virtuelle

Afin de réaliser des économies en termes de coûts sur la consolidation des centres de données tout en garantissant la performance et la flexibilité, les entreprises ont recouru à la virtualisation et aux services de l'informatique en nuage. Pour supporter la virtualisation dans les centres de données, il faut utiliser plusieurs types de technologies et de protocoles.

- **La machine virtuelle:** Est une émulation des ordinateurs ou des serveurs. Elle a le même comportement qu'un ordinateur physique réel. Elle permet d'exécuter des programmes exactement comme une machine physique. D'autre part, pour accéder aux ressources physiques de la machine hôte, la machine virtuelle doit interagir avec un hyperviseur.

- **L'hyperviseur:** Est un logiciel nécessaire pour l'émulation de plusieurs machines virtuelles dans une même machine physique. La tâche principale d'un hyperviseur est de partager les ressources matérielles de la machine hôte entre toutes les machines virtuelles. Plusieurs types d'hyperviseurs existent sur le marché tels que *KVM*, *Xen*, *VSphere* [26].

- **Le commutateur virtuel:** Le commutateur virtuel est généralement situé à l'intérieur de l'hyperviseur. Il permet aux machines virtuelles de communiquer entre elles et d'accéder au réseau externe. Plusieurs types de commutateurs virtuels existent sur le marché, tels que le commutateur *Nexus1000v* [35] de *Cisco* et le commutateur *OVS* [37] qui utilise la technologie *OpenFlow* [31] pour assurer l'acheminement des paquets.

- **Les réseaux virtuels:** Les réseaux virtuels sont des réseaux superposés qui sont implémentés au-dessus du réseau physique existant. Les réseaux virtuels permettent l'isolation des réseaux des locataires des centres de données. Généralement, les réseaux virtuels

utilisent des services tels que *VLAN* (*Virtual LAN*) ou *VPN* (*Virtual Private Network*) ou encore de nouveaux services tels que *VxLAN* (*Virtual Extensible LAN*).

Les multiples concepts et technologies de virtualisation s'ajoutent à l'infrastructure physique traditionnelle d'un centre de données, complexifiant la gestion de l'infrastructure de ce dernier.

1.2 Validation des infrastructures réseaux

Le concept de la configuration et de la gestion de l'infrastructure réseau repose sur un principe paradoxal. D'un côté, tous les nœuds de l'infrastructure doivent interagir entre eux et se comporter en groupe, mais de l'autre côté leurs configurations sont effectuées individuellement. De plus, les nœuds de l'infrastructure physique doivent supporter les services des réseaux virtuels dont la configuration ne doit pas être contradictoire et doit assurer l'isolation entre les différents locataires.

1.2.1 Dépendance entre l'infrastructure physique et l'infrastructure virtuelle

Le concept de la virtualisation est apparu pour nous détacher de tout ce qui est réel et nous apporter plus de flexibilité. Cependant, il est très difficile d'atteindre cet objectif, étant donné l'étroite liaison qu'il y a entre ces deux concepts (par exemple, une machine virtuelle ne peut pas être plus puissante que la machine physique qui l'héberge). Plusieurs cas d'utilisation dans un centre de données multi-locataires peuvent nous expliquer cette dépendance.

Exemple de migration des machines virtuelles

La migration d'une machine virtuelle d'un serveur à un autre ou d'un centre de données à un autre est l'une des principales fonctionnalités que proposent aujourd'hui la majorité des fournisseurs de l'informatique en nuage. La figure 1.1 est un exemple basique de la migration d'une machine virtuelle. La raison de cette migration est d'augmenter les performances du réseau entre les deux machines virtuelles, ce qui amène à leur rapprochement et à la diminution du nombre de nœuds entre eux. La migration de la machine virtuelle du *Server3* vers le *Server2* engendre plusieurs changements dans le réseau. Cette migration ne peut être effectuée sans une connaissance complète de la topologie du réseau. Les machines virtuelles situées dans le *Server1*

et le *Server3* communiquent ensemble à travers le service *VLAN* avec l'identifiant 22, ce qui engendre la configuration de ce service dans tous les équipements du chemin les séparant. Avant la migration, il n'y a que l'interface *Fe2/0* du *Switch1* qui n'implémente pas le service *VLAN22*. Cependant lors de la migration, il faut que cette interface *Fe2/0: Switch1* implémente le service *VLAN22* pour que la machine virtuelle puisse être transférée vers le serveur *Server2*. Après l'opération de migration, l'interface *Fe2/0:Switch2* ne doit plus contenir le service *VLAN22* pour maintenir l'isolation du locataire et l'interface *Fe2/0: Switch1* doit garder le service *VLAN22* pour assurer la communication entre les deux machines virtuelles.

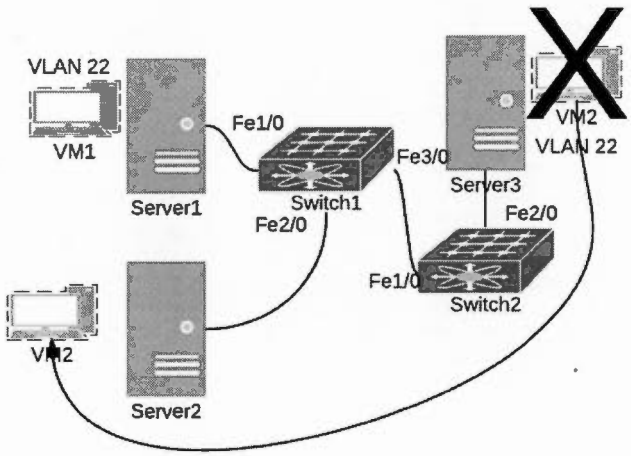


Figure 1.1: Migration d'une machine virtuelle

La configuration du réseau virtuel connectant des machines virtuelles affecte la configuration des équipements physiques.

Exemple de la localisation des machines virtuelles

L'une des plus grands objectifs dans les centres de données est de pouvoir assurer la bande passante et la fiabilité du réseau pour une machine virtuelle. De nos jours, un client utilisant une plate-forme d'informatique en nuage ne peut ni choisir les performances du réseau de ses machines virtuelles ni de son infrastructure virtuelle. Assurer un tel service dans une plate-forme d'informatique en nuage nécessite un contrôle total de l'infrastructure physique et une connaissance totale de la topologie physique et virtuelle.

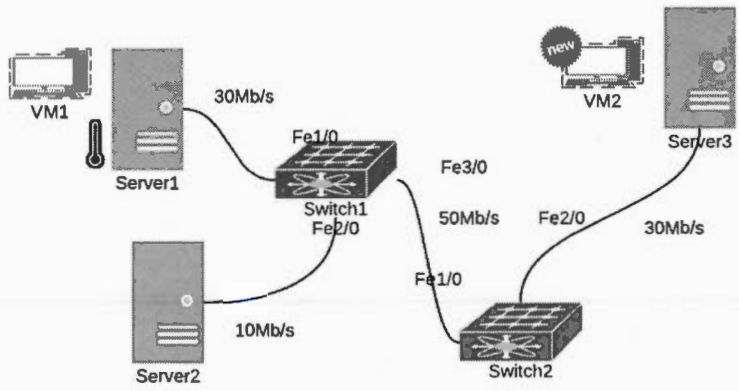


Figure 1.2: Considération des performances réseau pour le choix de la localisation d'une machine virtuelle

La figure 1.2 représente un exemple de cette problématique. Supposons qu'un client disposant d'une machine virtuelle dans un centre de données veut créer une nouvelle machine virtuelle et veut obligatoirement que la bande passante entre ses deux machines virtuelles soit supérieure à 20Mb/s. Puisque que la bande passante entre le *Server1* et le *Server2* ne peut pas dépasser les 10Mb/s et que le *Server1* a atteint la totalité de ses capacités, la machine virtuelle dans notre cas ne peut être créée que sur le *Server3*. Ce service que peut offrir la plate-forme d'informatique en nuage ne peut être réalisé que si, et seulement si, on a une vue sur la topologie du réseau et qu'on dispose de toutes les informations nécessaires concernant les performances des liens. Dans ce contexte, le client n'a pas une vue de l'infrastructure sur laquelle repose son système. Cependant, le fournisseur de ce type de service doit impérativement avoir une connaissance détaillée de l'infrastructure physique et avoir un modèle de description qui lui permet d'offrir des services de ce genre d'une manière automatique.

Exemple du choix de la technologie d'isolation entre les locataires

Comme mentionné précédemment, les technologies d'isolation dans un centre de données sont variées. Par ailleurs, chaque technologie présente des avantages et des inconvénients. Par exemple, le service *VLAN* est utilisé pour l'isolation entre les locataires d'un même centre de données. D'autre part, étant donné que le protocole *VLAN* est présent dans la couche 2 du réseau, les performances du réseau ne sont presque pas détériorées. Tandis que le protocole *VxLAN* est utilisé pour permettre l'isolation entre les locataires et l'extension du réseau local

virtuel à deux ou plusieurs centres de données séparés par des domaines de la couche 3. Par contre, le service *VxLAN* peut supporter beaucoup plus de locataires que le service *VLAN*. Il existe aussi plusieurs autres protocoles tels que *VPN*, *OTV* [36], *FCoE* [21]. Cependant pour chaque protocole et technologie, il faut configurer adéquatement les différents nœuds présents dans l'infrastructure du centre de données.

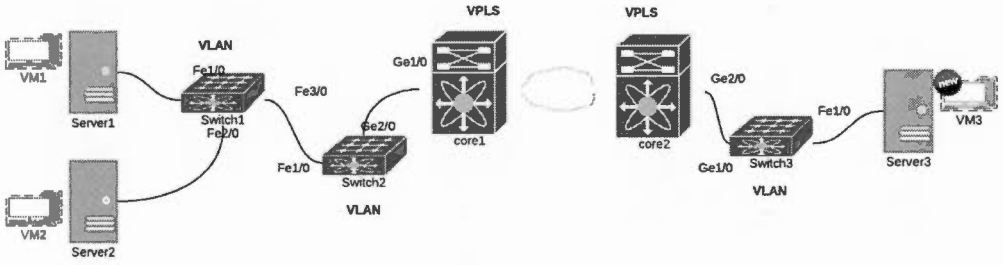


Figure 1.3: Changement de la technologie d'isolation

L'exemple présent dans la figure 1.3 illustre l'impact de la création d'une machine virtuelle sur l'ensemble du réseau. Dans cet exemple, le client a décidé de créer une machine virtuelle dans un centre de données différent de celui qui héberge déjà ses anciennes machines. Cependant, il veut que toutes ses machines puissent communiquer et se comporter comme si elles étaient dans le même réseau local. Le fournisseur de service peut alors utiliser le service *VxLAN* comme technologie qui permet de virtualiser son réseau local. Pour subvenir aux besoins de son client, le fournisseur de *Cloud* doit adapter les configurations de ses différents nœuds physiques et virtuels. Par exemple, les commutateurs virtuels présents dans les serveurs 1, 2 et 3 doivent être configurés de manière à ce qu'ils puissent ajouter le champ *VxLAN* aux paquets qui sortent de la machine virtuelle concernée. De plus, la configuration des commutateurs *Core1* et *Core2* est obligatoire pour pouvoir acheminer les paquets avec le champ *VxLAN* vers le centre de données adéquat. Finalement, les configurations des commutateurs *Switch1* et *Switch2* concernant le *VLAN* ne sont plus nécessaires puisque les machines virtuelles communiquent maintenant avec *VxLAN*.

1.3 Vers une solution

Malgré l'évolution rapide des réseaux de nos jours, et l'apparition de plusieurs outils de gestion de l'infrastructure et des équipements réseau, il n'existe pour l'instant aucun mécanisme,

ni outil permettant de détecter automatiquement les dépendances entre l'infrastructure physique et virtuelle. De nos jours, le bon fonctionnement des services dans les centres de données multi-locataires est souvent dévolu à l'intuition des administrateurs, qui dans le meilleur des cas, ne peuvent soumettre leurs configurations qu'à une batterie de tests. Les cas d'utilisation présentés précédemment nous montrent qu'on ne pourra jamais gérer le réseau virtuel sans prendre en considération l'infrastructure physique qui l'abrite. C'est pour cette raison que nous avons opté pour une description et une modélisation de ces deux infrastructures. La modélisation de l'infrastructure d'un centre de données et des configurations des différents équipements pourra assurer la génération automatique d'une configuration valide et la détection des différents problèmes qui peuvent survenir dans le réseau. Quatre étapes doivent être réalisées pour atteindre cet objectif:

1. Description de l'infrastructure physique et virtuelle et création d'un modèle complet et formel. Ce modèle doit décrire aussi bien la topologie du réseau que la configuration et les paramètres des équipements.

2. Élaboration de règles de validation pour chaque service. Ces règles vont permettre de gérer l'interdépendance entre l'infrastructure virtuelle et physique. Elles doivent s'appliquer sur le modèle déjà élaboré et prendre en considération la topologie du réseau pour le choix des équipements à configurer.

3. Formalisation du modèle et des règles de validation avec le langage *Alloy* pour la formalisation et la validation du modèle.

4. Implémentation du modèle, des règles de validation et des algorithmes nécessaires dans un système de gestion d'un centre de données virtualisé tel que OpenStack.

CHAPITRE II

REVUE DE LA LITTÉRATURE

Les administrateurs des centres de données de nos jours sont dans une impasse. D'une part, les demandes des services de l'informatique en nuage augmentent quotidiennement, d'autre part la tâche de la gestion des centres de données devient de plus en plus lourde et complexe. Plusieurs solutions ont vu le jour, essayant de résoudre les problèmes de gestion des infrastructures virtuelles et physiques ainsi que la validation des configurations des équipements réseau. Dans ce chapitre, nous étudierons les concepts de l'informatique en nuage en prenant comme exemple la célèbre plate-forme OpenStack. Nous allons ensuite étudier les différentes approches proposées par la communauté scientifique pour résoudre les problèmes mentionnés dans le chapitre précédent.

2.1 Plate-formes de l'informatique en nuage

Ce sont des plates-formes qui permettent de contrôler un grand nombre de ressources matérielles. Ces ressources peuvent être différentes: ressources de calcul, ressources de stockage ou ressources de réseaux. Ces plates-formes sont généralement gérées par des tableaux de bord sous forme d'applications web.

2.1.1 Types de plate-formes de l'informatique en nuage

Les plates-formes de nos jours proposent plusieurs types de modèles [24]. Dans notre étude, nous allons nous intéresser au modèle *Iaas*. Ce type offre plus de flexibilité aux entreprises par rapport aux autres modèles tel que le Paas et le Saas. De plus, il leur permet de réduire le coût de gestion de leurs centres de données en déployant toute leur infrastructure dans l'informatique en nuage. C'est dans ce modèle qu'il y a le plus de défis et le plus de problématiques. Les fournisseurs utilisent plusieurs types de technologies tels que les hyperviseurs

et les commutateurs virtuels. Comme mentionné dans 1.1.2, les hyperviseurs sont utilisés pour exécuter plusieurs machines virtuelles sur les serveurs hôtes et les commutateurs virtuels sont responsables de connecter les machines virtuelles au réseau. Plusieurs compagnies offrent ce genre de service de nos jours telles que *Amazon* ou *Rackspace*. Il existe aussi des plates-formes qui permettent de créer ce modèle *Iaas* tel que OpenStack et *CloudStack*. Dans notre projet, nous allons utiliser la plate-forme OpenStack puisque c'est la plate-forme *Open source* la plus connue et la plus utilisée de nos jours. Nous allons nous intéresser spécialement à la partie réseau de OpenStack.

2.1.2 La plate-forme OpenStack

OpenStack est composée d'une multitude de projets. *Nova* [2] est un projet qui permet aux administrateurs de gérer les hôtes physiques pour qu'ils puissent créer les machines virtuelles et partager les ressources entre elles. Keystone est le projet qui assure la gestion des rôles, des utilisateurs et des projets. Horizon est le projet qui offre une interface graphique pour la gestion de la plate-forme. Nous nous intéresserons dans notre travail au projet Neutron qui assure le côté réseau de la plate-forme.

Présentation de Neutron

Neutron qui permet aux administrateurs de créer des infrastructures réseau et d'y connecter les machines virtuelles. Ce dernier est aussi responsable de la configuration des équipements nécessaires dans la gestion de ces réseaux. Neutron décompose le réseau virtuel en trois parties [1]:

- **Le réseau:** représente un domaine virtuel de la couche 2 du réseau. Il peut être considéré comme un commutateur virtuel.

- **Le sous-réseau:** représente des blocs d'adresses IPv4 ou IPv6. C'est à partir de ces blocs qu'on assigne une adresse IP à une machine virtuelle.

- **Le port:** représente un port d'un commutateur virtuel sur lequel la machine virtuelle est connectée.

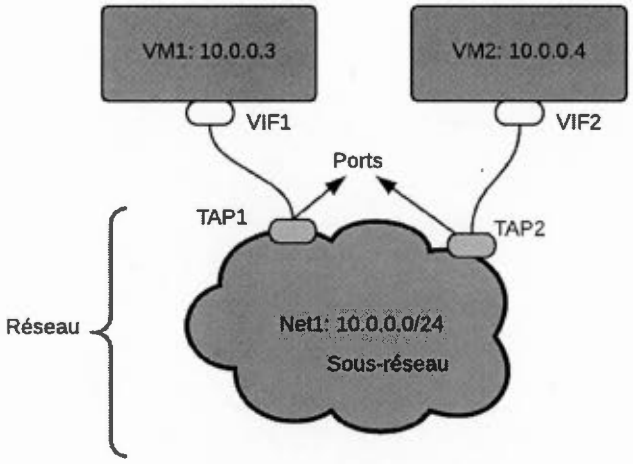


Figure 2.1: Concepts de Neutron

La figure 2.1 illustre un exemple d'architecture réseau dans Neutron et OpenStack. Dans cet exemple, on a deux machines virtuelles *VM1* et *VM2* qui ont les adresses IPs respectives 10.0.0.3 et 10.0.0.4. Ces deux machines virtuelles appartiennent au même sous-réseau 10.0.0.0/24 qui lui appartient au réseau virtuel *Net1*. Les deux machines virtuelles sont connectées au réseau virtuel à travers les ports virtuels *TAP1* et *TAP2*. Par ailleurs, le réseau virtuel *Net1* peut être configuré sur les équipements physiques comme étant soit un réseau *VLAN* ou *VxLAN* ou un tunnel *OTV* ou *VPN*. La configuration de ces équipements réseau pour supporter ces services se fait à travers les plug-ins de Neutron.

Allons plus en détail sur les étapes pour la configuration d'un commutateur virtuel lors de la création d'une machine virtuelle dans un serveur physique. Nous allons prendre comme exemple le plug-in *OVS* [4] de Neutron.

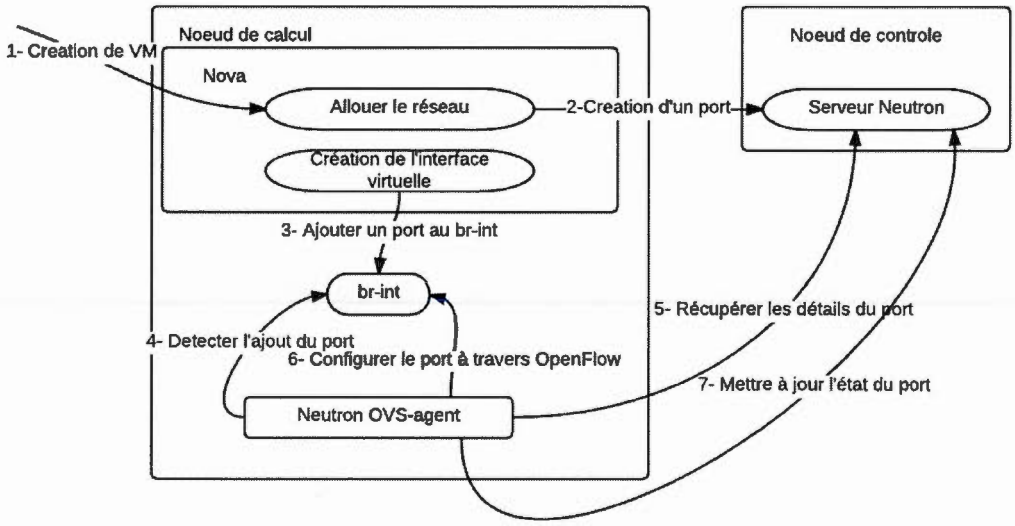


Figure 2.2: Fonctionnement du plug-in *OVS*

Comme présenté dans la figure 2.2, *Nova* est responsable de la création de la machine virtuelle dans les serveurs physiques ou les nœuds de calcul. Ce dernier va ensuite demander à Neutron de mettre à jour sa base de données en lui donnant les informations nécessaires sur le nouveau port créé. Puis il va ajouter un port virtuel sur le pont *br-int* du commutateur *OVS*. Ensuite, l'agent qui est déjà installé dans le serveur physique s'aperçoit qu'un nouveau port a été ajouté à son commutateur virtuel. Pour connaître les paramètres de configuration de ce port, l'agent consulte le serveur de Neutron puis configure le commutateur à l'aide du protocole *OpenFlow*. Après avoir terminé la configuration du commutateur, l'agent met à jour l'état du port auprès du serveur Neutron. On constate que, d'après le fonctionnement de OpenStack et des autres plates-formes d'informatique en nuage, la gestion du réseau virtuel est effectuée essentiellement sur l'infrastructure virtuelle. La dépendance entre l'infrastructure virtuelle et l'infrastructure physique est mal gérée ce qui limite la diversification des services que proposent ces plates-formes.

2.2 Modèle d'abstraction de l'infrastructure: CIM

Plusieurs modèles d'abstraction de l'infrastructure ont vu le jour pour réduire la complexité et le coût de la gestion des infrastructures virtuels tel que le modèle *CIM* [40] (*Common*

Information Model). L'objectif du modèle *CIM* est de fournir une représentation uniforme d'information de gestion pour des systèmes, des applications et des réseaux. Développé par le *DMTF* (*Distributed Management Task Force*) le modèle *CIM* de la gestion de l'infrastructure virtuelle s'intéresse essentiellement à l'infrastructure virtuelle. Dans le travail [14], les chercheurs ont présenté un ensemble de concepts et une décomposition des composants de l'infrastructure virtuelle.

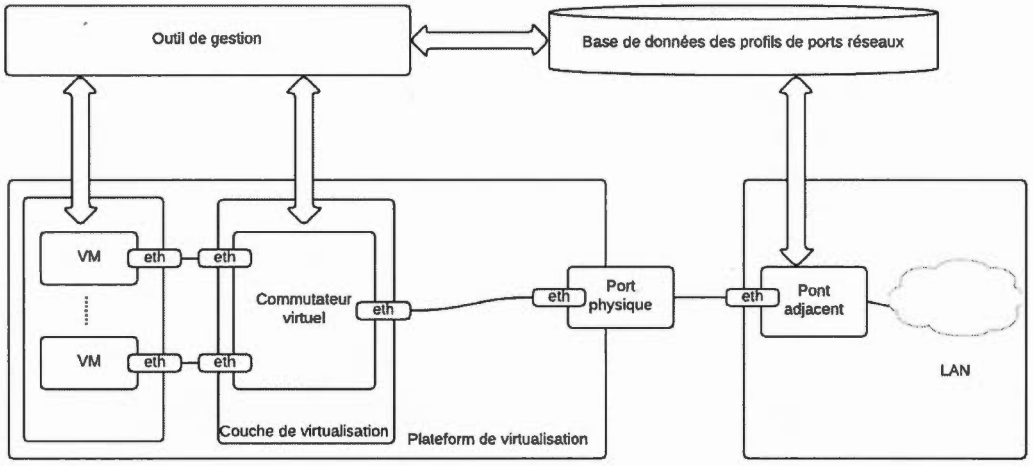


Figure 2.3: Modèle *CIM* de gestion du réseau virtuel

La figure 2.3 illustre le modèle que propose le groupe de travail de DMTF pour la gestion de l'infrastructure virtuelle. Ce modèle présente quatre composantes essentielles qui sont:

- **Plate-forme de virtualisation:** Elle est fournie par un système hôte et permet le déploiement de systèmes et de machines virtuelles. D'autre part, elle comprend le service de gestion de la virtualisation.
- **Pont adjacent:** C'est un pont IEEE 802.1Q qui est connecté à la station physique à travers un ou plusieurs ports Ethernet. La principale différence entre ce pont et un pont générique 802.1D est le support des protocoles de détection et de contrôle EVB de IEEE.
- **Outil de gestion:** Il est responsable de la gestion de la plate-forme de virtualisation ainsi que de la mise en place de machines virtuelles (VM) sur cette plate-forme.
- **Base de données de profils de ports réseau:** Est un répertoire contenant un ensem-

ble d'attributs qui peuvent être appliqués aux ports de l'infrastructure réseau lors de la création de machines virtuelles ou de leurs migrations.

Le modèle *CIM* est un modèle qui semble être intéressant pour la gestion de l'infrastructure virtuelle. Il permet la gestion de la plate-forme de virtualisation et des équipements qui sont adjacents à cette plate-forme. Cependant, il n'offre pas une vue globale sur l'ensemble des composants de l'infrastructure tels que les équipements entre les nœuds de l'infrastructure virtuelle. De plus, il n'offre pas une vue sur la topologie physique de l'infrastructure, ce qui limite considérablement les cas d'utilisation de ce modèle.

2.3 Langage d'abstraction des configurations réseau: Meta-Cli

L'interface de la ligne de commande représente la principale interface pour la configuration des équipements réseau. Elle se base sur un ensemble de séquences de commandes que l'administrateur envoie à l'équipement pour configurer les différents services réseau. Cependant, la configuration à travers cette interface est très complexe et nécessite plusieurs heures de lecture de manuels pour chaque type d'équipement et pour chaque fournisseur. La logique de la configuration et des fichiers de configurations sont très différents d'un système à un autre. C'est pour cette raison que le langage *Meta-CLI* [8] a été créé. Ce langage offre une abstraction intelligente des configurations des équipements réseau. Il permet aussi d'abstraire la configuration des services pour automatiser leur déploiement sur chaque type d'équipement. Le modèle *Meta-CLI* traduit essentiellement les contextes, les fichiers de configuration et les autres informations de configuration en des structures qui peuvent être manipulées d'une manière simple et qui garantissent la validité et la cohérence du processus de configuration.

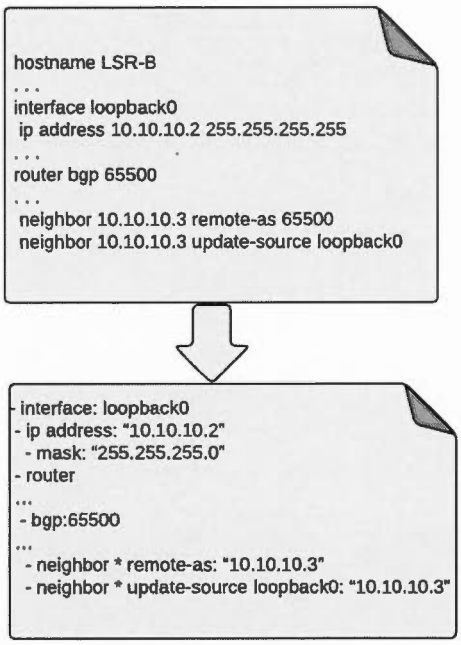


Figure 2.4: Exemple de traduction d'une configuration en *Meta-CLI*

La figure 2.4 illustre un exemple de traduction de la configuration *IOS* d'un équipement *Cisco* en langage *Meta-CLI*. On voit bien que l'information est devenue plus structurée et sous forme (clé:valeur) telle que (ip address: 10.10.10.2). Ainsi, la valeur du masque de sous réseau est devenue un fils au paramètre adresse IP. Le langage *Meta-CLI* structure l'information sous forme hiérarchique selon l'appartenance de chaque paramètre à son noeud parent adéquat.

Ce formalisme a facilité considérablement la gestion de la configuration des équipements et nous a permis de gérer les configurations de différents types d'équipements de la même façon. La structuration de la *Meta-Cli* a permis aussi de créer des règles et des contraintes pour automatiser la validation de certains types de services. C'est ce qui a donné naissance à l'outil *ValidMaker*.

2.4 Outil de validation des configurations réseau: ValidMaker

Conformément aux exigences de configuration du réseau, des outils ont été proposés pour la gestion de la configuration du réseau, tels que *ValidMaker* [17]. Cet outil a été développé par

l'équipe du laboratoire de téléinformatique et réseau (*LTIR*) à l'Université du Québec à Montréal. Ce logiciel sert à deux fins principales. D'abord, il abstrait l'hétérogénéité des équipements et des multiples plates-formes en fournissant une représentation commune des informations des configurations sous forme *Meta-CLI*. Deuxièmement, *ValidMaker* permet d'exprimer des contraintes formelles sur des structures *Meta-CLI*. L'outil implémente aussi un moteur de validation pour vérifier automatiquement la conformité d'une configuration donnée. Les contraintes peuvent imposer des dépendances. Pour répondre à ces deux objectifs, *ValidMaker* est composé de deux modules: le moteur de validation et le gestionnaire de configuration, comme représenté dans la figure 2.5.

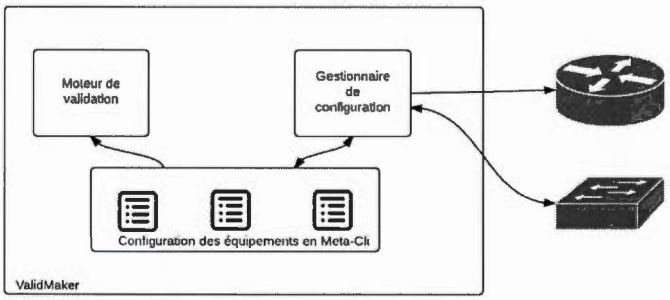


Figure 2.5: Architecture de l'outil *ValidMaker*

Nous allons maintenant détailler le fonctionnement du gestionnaire de configuration, car c'est la partie la plus importante pour notre travail. En effet, c'est cette composante qui est responsable de la formalisation des paramètres et des configurations réseau. Le gestionnaire de configuration de l'appareil est la partie du système responsable de la communication avec les dispositifs, la récupération de la configuration et la transformation en structures *Meta-CLI*. À l'inverse, les configurations *Meta-CLI* internes à *ValidMaker* peuvent être traduites en une structure prise en charge par l'équipement cible en fonction du système d'exploitation du fournisseur et du numéro de sa version. Le gestionnaire de configuration permet aussi la détection des services présents dans chaque équipement. L'ajout de services dans la configuration est aussi possible grâce aux différents concepts qu'utilise *ValidMaker*. Principalement, quatre structures sont utilisées par l'outil pour la gestion et la validation des services, comme illustré dans la figure 2.6:

- **Service générique:** Durant la phase de la conception, l'administrateur doit étudier le service et extraire ses principaux paramètres. Les paramètres doivent appartenir au service et pas à l'équipement. Ensuite, l'administrateur doit formaliser ces paramètres et les structurer à l'aide du langage *Meta-Cli.*

- **Instance de service:** C'est l'instance du service générique déjà construite. L'instance de service comprend les valeurs des paramètres du service générique. On prend comme exemple le service *VLAN.* L'instance du service *VLAN* doit comprendre l'identifiant du service c'est-à-dire le *VLAN_ID.*

- **Équipement:** La configuration de l'équipement en *Meta-CLI* peut contenir des références à des instances de services. Si un équipement implémente un service, les paramètres du service possèdent une référence aux paramètres de l'instance de service.

- **Règle:** La règle dans *ValidMaker* représente une opération logique pour la validation d'un service. Chaque opérande d'une règle est une référence au nœud du service générique. Grâce aux règles définies dans l'outil, le moteur de validation valide les équipements deux par deux.

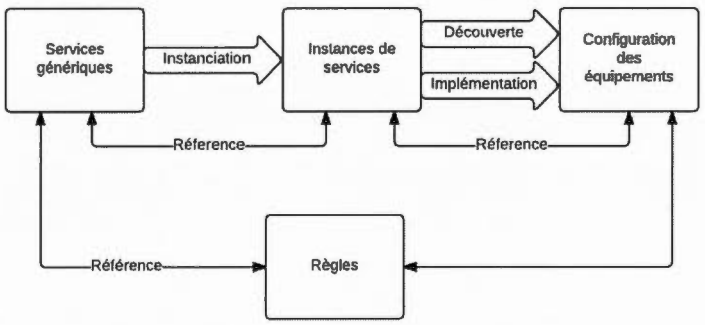


Figure 2.6: Les concepts de *ValidMaker*

L'outil *ValidMaker* est très puissant dans la gestion et la validation des configurations des équipements réseau. Cependant, l'opération de validation qu'il réalise ne s'effectue que sur les équipements et ne prend pas en considération la topologie globale du réseau. De plus, les structures de données de *ValidMaker* ne gèrent pour l'instant que les équipements physiques et traditionnels. Les équipements *OpenFlow* et les commutateurs virtuels ne sont pas encore

supportés.

2.5 Langage de description et de validation des modèles: Alloy

Alloy [20] est un langage de description des structures et un outil permettant de les explorer. Il a été utilisé dans une large gamme d'applications de recherche pour trouver les failles de sécurité dans les mécanismes de conception des réseaux de commutation téléphoniques [3].

Le modèle *Alloy* représente une collection de contraintes qui décrivent (implicitement) un ensemble de structures, par exemple: toutes les configurations possibles de sécurité d'une application web, ou toutes les topologies possibles d'un réseau de commutation. L'outil *Alloy* est un solveur qui prend les contraintes d'un modèle et trouve des structures qui les satisfont. Il peut être utilisé pour l'exploration du modèle en générant des structures valides, ou pour la vérification des propriétés du modèle en générant des contre-exemples.

Alloy [10] propose une syntaxe de déclaration compatible avec les modèles d'objets, et une autre syntaxe pour l'expression de formules très complexes.

2.5.1 Syntaxe et Sémantique du langage

Alloy présente une syntaxe riche et très accessible. Il est inspiré des langages orientés objet : il permet de spécifier le type des objets, leurs attributs et le type de ces attributs. Il permet également de spécifier des contraintes sur ces objets et de spécifier un «cadre» qui définit un nombre fini d'instances de chaque objet. Pour ce qui concerne sa syntaxe, elle est composée de deux groupes différents qui sont les structures et les contraintes.

Déclaration des structures

Pour la déclaration d'un objet ou d'une structure (appelé aussi signature), il faut utiliser le mot clé «*sig*» suivi du nom de l'objet comme le montre le code 2.1.

Code 2.1: Déclaration d'une structure dans *Alloy*

```
sig TargetNetwork {
}
```

Les objets dans *Alloy* peuvent hériter des caractéristiques et des paramètres d'autres objets. L'héritage peut être réalisé à l'aide du mot clé «*extends*». D'autre part, un ou plusieurs liens entre les différents objets ou avec l'objet lui-même peuvent être créés. Pour cela, il suffit de créer un élément à l'intérieur des accolades qui suivent la déclaration de l'objet. Le code 2.2 un exemple de création de liens entre les objets.

Code 2.2: Création des liens entre les objets dans *Alloy*

```
sig PhysicalNetwork {
    tg_net: one TargetNetwork
    }
```

Dans l'exemple 2.2 l'élément à l'intérieur de la déclaration de l'objet «*PhysicalNetwork*» représente une relation entre «*PhysicalNetwork*» et «*TargetNetwork*». Nous avons appelé cette relation «*tg_net*». «*one*» est le mot clé qui représente la multiplicité de la relation. Il indique qu'un «*PhysicalNetwork*» ne peut avoir une relation «*tg_net*» qu'avec un seul objet de type «*TargetNetwork*». Plusieurs autres mots clés décrivant la multiplicité peuvent être utilisés dans Alloy comme présenté dans les deux tableaux 2.1 et 2.2.

e peut être une variable ou une expression	
x: **set** e	x est un ensemble de e
x: **lone** e	x peut être vide ou un seul élément de type e
x: **some** e	x est un ensemble non vide de e
x: **one** e	x ne peut être qu'un élément de type e
x: e	x peut être vide ou un ensemble de e

Tableau 2.1: Ensemble de déclarations dans *Alloy* avec les multiplicités

A et **B** sont des expressions qui produisent des relations	
m et **n** sont **some**, **lone**, **one** ou **set**	
r: $Am \rightarrow nB$	m éléments de A corresponds à chaque élément de B
	Chaque élément de A correspond à n éléments de B

Tableau 2.2: Déclaration de la relation «→»

Déclaration des contraintes

Les contraintes dans le langage *Alloy* peuvent être représentées suivant plusieurs manières:

- On peut ajouter des contraintes juste après la déclaration d'un objet. Cette contrainte

Pour chaque variable **V**, relation **R**, ensemble **Set** et formule ϕ		
	Quantificateur Q V: Set | ϕ	prédicats dans les relations Q R
all	Quantificateur universel **all** v: Set | ϕ est vrai si ϕ est vrai pour chaque valeur de v dans Set	
some	Quantificateur existentiel **Some** v: Set | ϕ est vrai si ϕ est vrai pour un ou plusieurs valeurs de v dans Set.	Taille de R est 1 ou supérieur
no	Négation du quantificateur existentiel **no** v: Set | ϕ est vrai si ϕ est vrai pour aucune valeur de v dans Set.	Taille de R est zéro
lone	Zéro ou un existe **lone** v: Set | ϕ est vrai si ϕ est vrai pour pas plus qu'une valeur de v dans Set.	Taille de R est zéro ou un
one	exactement un seul existe **one** v: Set | ϕ est vrai si ϕ est vrai pour exactement une valeur de v dans Set.	Taille de R est un

Tableau 2.3: Quantificateurs et prédicats dans *Alloy*

ne sera appliquée que sur l'objet et ses paramètres. Il s'agit d'un raccourci pour une contrainte.

- On peut aussi utiliser le mot clé «*fact*». Un «*fact*» ou un fait définit une formule que nous pouvons assumer comme toujours valide pour n'importe quelle situation. L'analyseur de *Alloy* utilise des faits comme des axiomes dans la construction des exemples et des contre-exemples.

- On peut aussi utiliser les prédicats avec le mot clé «*pred*». Un prédicat définit une formule (vrai ou faux). Il peut prendre des paramètres qui sont utilisés dans l'obtention de son résultat.

Dans ces types de contraintes, plusieurs quantificateurs peuvent être utilisés. Le tableau 2.3 présente une partie de ces quantificateurs.

Alloy permet aussi la déclaration de fonctions. Une fonction dans *Alloy* est précédée par le mot clé «*fun*». Une fonction peut avoir comme résultat de retour une relation, un ensemble ou un atome (instance d'une signature). Elle peut prendre des paramètres qui seront utilisés

dans le calcul du résultat. Elle peut définir une relation (généralement à l'aide de «→») et en faire usage pour produire son résultat.

Après la déclaration de tout le modèle, ses signatures, ses contraintes et ses fonctions, nous pouvons en générer une ou plusieurs instances valides. La génération des instances se fait à l'aide du mot clé «*run*». Ce dernier nous permet de spécifier l'étendue des instances générées du modèle. Par exemple si on fait «*run for 4 TargetNetwork and 2 PhysicalNetwork*», l'analyseur nous génère une instance du modèle dont les atomes ne dépassent pas 4 pour la signature «*TargetNetwork*» et 2 pour la signature «*PhysicalNetwork*».

Alloy est un outil efficace pour la formalisation et la validation des modèles. Plusieurs projets de recherche se sont basés sur le langage et outil *Alloy* pour la modélisation des équipements réseau et de leurs configurations tel que le travail de *S. Narain* [34].

2.6 Solution de la résolution des exigences: Requirement Solver

Le principe du travail de *S. Narain* présenté dans son article [34] repose sur la modélisation de la configuration du réseau sous forme d'un ensemble de variables, la fixation de la taille des domaines, l'écriture des contraintes sur ces variables en termes de formules du premier ordre puis la conversion du tout avec *Alloy* sous forme d'une formule booléenne et puis la transmission du problème à un solveur *SAT*. Si une solution existe, l'outil *Alloy* la reconvertit en valeurs du problème initial et l'affiche. Pour faciliter l'écriture, *Narain* utilise un modèle «*objet*». On peut par exemple définir une interface qui possède deux champs: l'adresse et le routeur auquel elle appartient. On peut ensuite référer à ces champs d'une manière traditionnelle: «interface.routeur». Cette notation est celle employée par l'outil *Alloy*. L'auteur propose 6 façons différentes d'utiliser le solveur (toujours pour des domaines fixés):

1. **Synthèse de configuration (*Configuration Synthesis*):** pour produire une configuration à partir d'un ensemble de formules R, soumettre R au solveur puis récupérer le résultat.

2. **Renforcement des contraintes (*Requirement Strengthening*):** pour reconfigurer un réseau avec un ensemble additionnel S de contraintes, soumettre R & S au solveur et prendre le résultat.

3. **Ajout de composants (*Component Addition*):** modifier les domaines en conséquence,

resoumettre R au solveur et récupérer le résultat.

4. **Vérification des contraintes (*Requirement vérification*):** pour vérifier qu'un réseau respectant R ne peut causer la situation U, soumettre R & U au solveur et constater l'absence de solutions.

5. **Détection des erreurs de configuration (*Configuration Error Detection*):** pour modéliser la configuration actuelle du réseau par une conjonction d'affectations C, puis vérifier si R & C a une solution (si pas de solution, il y a une erreur)

6. **Correction des erreurs de configuration (*Configuration Error Fixing*):** si une configuration ne respecte pas R, soumettre R au solveur et choisir la solution la plus proche de la configuration actuelle

Puis, jusqu'à la fin de l'article, l'auteur s'est aussi intéressé au problème de la modélisation d'un *VPN* avec un certain nombre de contraintes de sécurité. Le travail proposé par le chercheur *S. Narain* montre l'efficacité de *Alloy* dans la modélisation et la validation des équipements et des services réseau. À travers une modélisation de la configuration des équipements, des services et de leurs contraintes, *S. Narain* a réussi à générer une configuration globale du réseau. Les services ont été validés à travers des règles qui prennent en charge deux équipements adjacents. Cependant la modélisation globale de l'infrastructure et de la topologie du réseau est absente dans le travail de *S. Narain* ce qui limite considérablement l'étendue des contraintes et affecte le fonctionnement global du réseau.

2.7 Résumé

Dans ce chapitre, nous avons évoqué les différents travaux dont nous nous sommes inspirés pour résoudre notre problématique. Nous avons présenté la plate-forme OpenStack et les concepts qu'elle utilise pour la gestion du réseau virtuel afin de nous familiariser avec les architectures virtuelles utilisées actuellement dans les centres de données. Par la suite, nous avons étudié le langage *Meta-Cli* et *ValidMaker* pour l'abstraction et la validation des services et des configurations réseau. Le travail réalisé par *S. Narain* nous a aussi inspiré sur la manière de valider et de formaliser notre modèle d'abstraction du centre de données en utilisant le langage et l'outil *Alloy*. Pour la suite de ce mémoire, nous allons présenter la manière avec laquelle nous avons combiné tous ces travaux pour résoudre notre problématique et gérer l'infrastructure du centre de données d'une manière efficace.

CHAPITRE III

MODÈLE D'ABSTRACTION DU CENTRE DE DONNÉES MULTI-LOCATAIRES

Les différents outils et travaux présentés dans le chapitre précédent nous ont permis de mieux comprendre les différentes composantes d'une infrastructure d'un centre de données. Plusieurs concepts présentés dans ces outils et travaux nous ont permis de créer un modèle d'abstraction de l'infrastructure virtuelle et physique d'un centre de données. Nous avons appelé notre modèle *CMnet Model* ou (*Cloud Management Network Model*). Dans ce chapitre, nous allons détailler *CMnet model* et présenter son application pour un exemple d'une infrastructure réelle. Nous allons par la suite détailler les différentes opérations que nous pouvons réaliser sur le modèle et présenter quelques exemples de contraintes sur les services applicables sur le modèle.

3.1 Description du modèle CMnet:

Afin d'aboutir à nos objectifs et de gérer l'infrastructure physique et virtuelle, nous avons élaboré un modèle permettant de décrire la correspondance entre ces deux infrastructures. Ce modèle nous permet d'avoir une vue globale sur la topologie du réseau et de localiser les infrastructures virtuelles. Ce modèle nous permet aussi d'avoir les caractéristiques et la configuration de chaque équipement physique grâce à l'abstraction fournie par le langage *Meta-CLI*. Nous allons maintenant détailler chaque élément de l'arbre de notre modèle présenté dans la figure 3.3:

- **Target Network:** représente le réseau global ou l'infrastructure globale du centre de données.
- **Generic service:** est un service réseau représenté à travers le langage *Meta-CLI*. L'utilisateur doit analyser le service réseau et bien le représenter sous une forme *Meta-CLI*.

- **Physical network:** Représente un réseau physique du réseau global. Cet élément regroupe zéro ou plusieurs nœuds physiques (*physical nodes*) et des liens entre ces nœuds. Le réseau physique est composé des éléments suivants:
  - **Instance service:** Cet élément représente une instance d'un service générique. L'instance de service est représentée en langage *Meta-CLI*. Il contient la même structure arborescente que celle d'un service générique en attribuant des valeurs aux nœuds spécifiques au service. Cependant, les nœuds spécifiques à la topologie ou à l'équipement seront insérés lors du déploiement de l'instance de service dans la configuration d'un équipement.
  - **Physical Links:** Un lien physique représente un câble réseau ou une connexion embarquée qui lie deux interfaces réseaux entre elles. Il peut représenter un câble *Ethernet* ou une fibre optique, etc. Il peut aussi représenter un lien interne entre un serveur «*Blade*» et un commutateur interne.
  - **Physical Node Group:** Cet élément représente un ensemble de nœuds physiques. Cet objet peut être le châssis d'un serveur «*blade*», qui comprend plusieurs serveurs et plusieurs équipements réseau.
  - **Physical Node:** Le nœud physique peut être un équipement de réseau, de stockage ou de traitement. Ce nœud est associé à un réseau physique et il peut être associé à un groupe de nœuds physiques (*Physical Node Group*) à travers son identifiant. Un nœud physique est composé des éléments suivants:
    - **Physical Interface:** Un nœud physique regroupe zéro ou plusieurs interfaces. Chaque interface peut aussi être associée à un lien physique.
    - **Physical Interface Group:** Représente un ensemble d'interfaces d'un même nœud physique. Cet ensemble peut être créé selon un ou plusieurs critères, tels que l'appartenance à une seule «*Linecard*», ou l'appartenance à un seul *VDC* (*Virtual Data Center*), etc.
    - **Configuration:** Représente la configuration du nœud physique en langage *Meta-CLI*.
- **Virtual Network:** Cet élément représente un des réseaux virtuels du réseau global du centre de données. Le réseau virtuel est composé des éléments suivants:
  - **Virtual Node Group:** le groupe de nœuds virtuels représente l'ensemble des nœuds

virtuels qui appartiennent au même nœud physique. Il contient aussi une association entre l'élément du nœud physique (*Physical Node*) et les interfaces correspondantes.

- **Virtual Links:** Un lien virtuel est utilisé pour lier deux interfaces virtuelles. Il peut représenter une liaison entre une machine virtuelle et un commutateur virtuel. Le lien virtuel peut aussi relier une interface virtuelle et une interface physique.

- **Instance service:** Cet objet est similaire à l'objet «*instance service*» pour les réseaux physiques. Ces deux objets héritent de l'élément «*generic service*».

- **Virtual Node:** Un nœud virtuel représente un équipement virtuel dans le réseau. Il peut être un commutateur virtuel «*OVS*», un bridge virtuel ou une machine virtuelle. Il faut que les nœuds virtuels appartiennent au même réseau virtuel. Ce nœud peut être par exemple un *VRF* (*Virtual Routing and Forwarding*) configuré dans un router. Un nœud virtuel est comporte plusieurs autres éléments tel que:

  - **Virtual Interface:** Cet objet représente une interface virtuelle d'une machine virtuelle ou d'un commutateur virtuel tel un «*VNIC*». Une interface virtuelle est connectée à une autre interface virtuelle à travers un lien virtuel.

  - **Configuration:** C'est la configuration de l'équipement virtuel.

La liaison entre les groupes de nœuds virtuels et le nœud physique est nécessaire pour la correspondance entre le réseau virtuel et le réseau physique.

Un centre de données multi-locataires doit supporter un ou plusieurs services réseau. Ces services sont généralement utilisés pour l'isolation des locataires. Ces services peuvent être soit des réseaux locaux virtuels (*VLANs*), des réseaux (*LANs*) virtuels et extensibles (*VxLANs*) ou d'autres services tels que *FCoE* utilisés pour le stockage des données. Une erreur de configuration de l'un de ces services peut engendrer des problèmes de sécurité ou de disponibilité. Prenons comme exemple, un centre de données multi-locataire qui utilise le service VLAN pour l'isolation des réseaux virtuels. Ce dernier ne doit en aucun cas avoir le même identifiant de VLAN pour deux locataires différents. Ceci doit être pris en compte dans la gestion des services dans un centre de données. Le modèle *CMnet* nous permet d'associer ces services au réseau dans lequel ils sont implémentés comme illustre la figure 3.1.

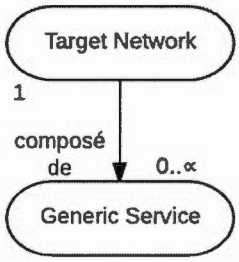


Figure 3.1: Les services génériques dans *CMnet model*

L'association présentée à la de la figure 3.1 indique que le réseau global peut supporter un ou plusieurs services génériques. À travers cette association, nous pouvons effectuer plusieurs opérations sur les services comme l'ajout, la suppression ou la modification des services. Nous pouvons en plus découvrir des services dans une infrastructure virtuelle ou physique. Cependant, les services ont des paramètres reliés à l'équipement comme le nom des interfaces et les adresses IP. Par exemple, le service *VLAN* avec l'identifiant «1001» est appliqué sur l'interface «Fe1/0» de l'équipement «AccessSwitch». C'est pour cette raison qu'on a utilisé le concept de l'instance de service de l'outil ValidMaker. L'élément de l'instance de service doit être relié à l'élément du service générique et ceux des nœuds physiques et virtuels comme illustré dans la figure 3.2.

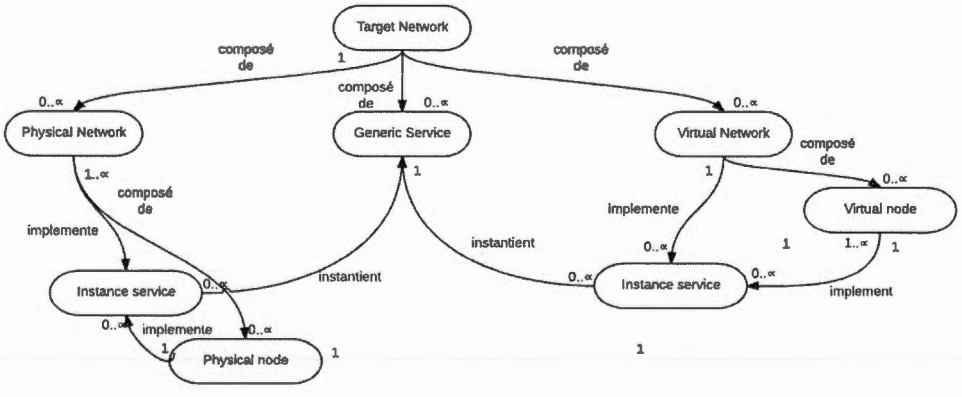


Figure 3.2: Les instances de services dans *CMnet model*

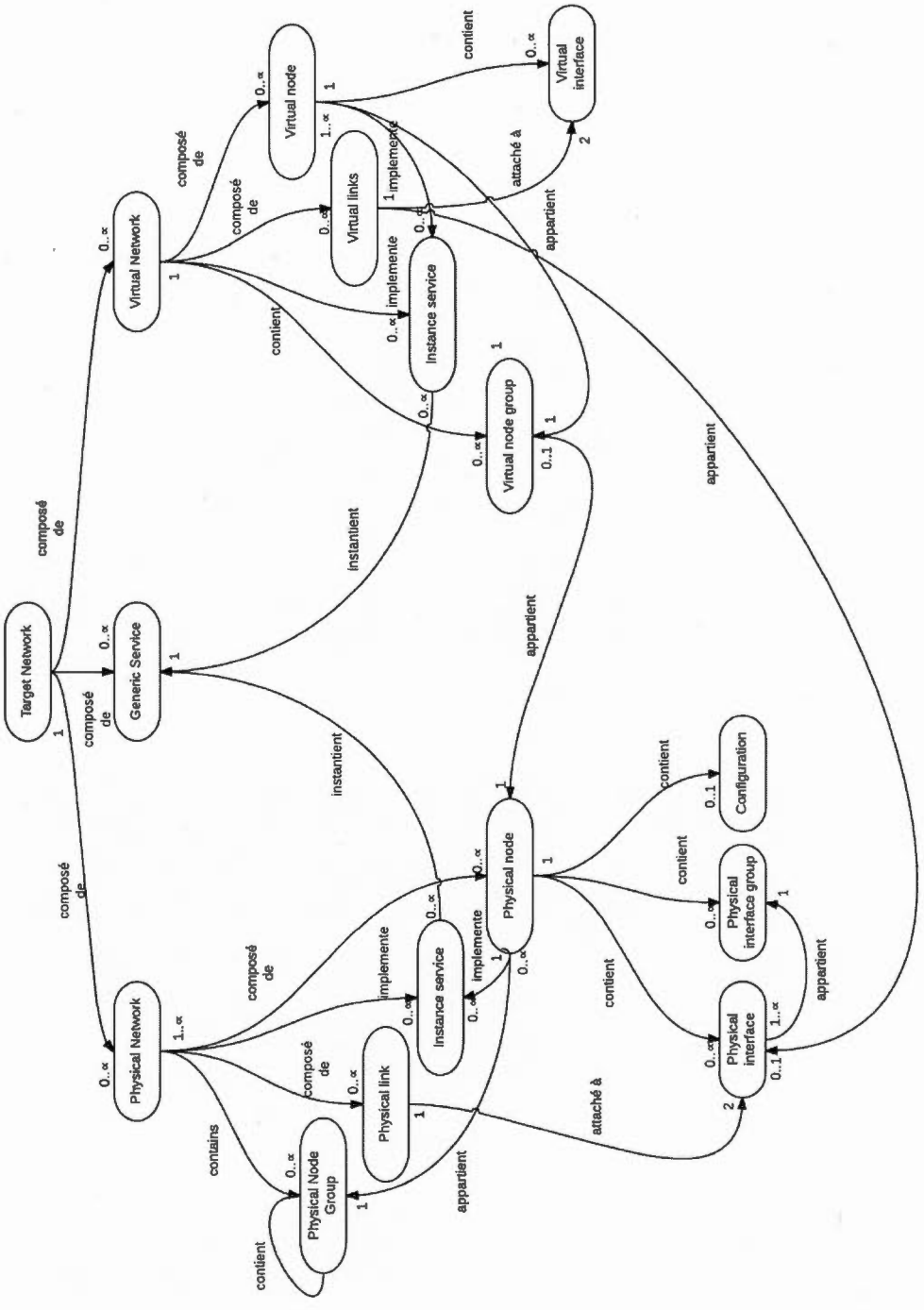


Figure 3.3: *CMnet model*

3.2 Exemple d'application du modèle CMnet

En se basant sur ce modèle, plusieurs cas d'utilisations peuvent être couverts. La figure 3.4 présente un exemple d'une infrastructure basique composée d'un serveur de calcul qui contient deux machines virtuelles pour le locataire 1 et un autre serveur de calcul qui ne contient aucune machine virtuelle. Ces deux serveurs sont connectés à l'aide de deux commutateurs *Cisco* de type *Nexus*.

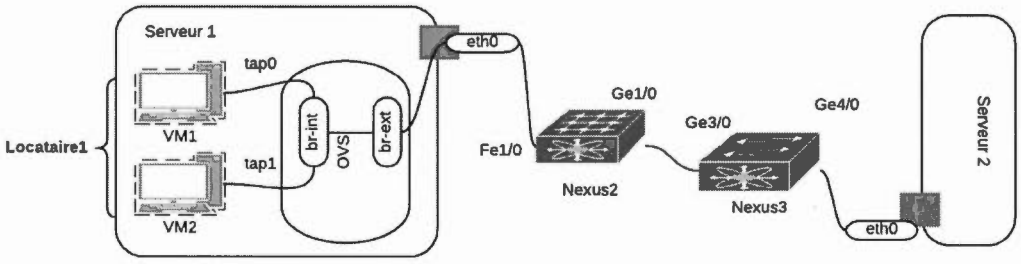


Figure 3.4: Exemple d'une infrastructure basique dans un centre de données

La figure 3.5 présente une instance de notre modèle d'abstraction qui décrit l'infrastructure basique décrite dans la figure 3.4. L'élément « *Target Network1* » représente le réseau global de l'infrastructure incluant le réseau physique qui est « *Physical Network1* » et le réseau virtuel « *Virtual Network 1* ». Le réseau physique est composé des nœuds physiques suivants (*nexus1*, *nexus2*, *serveur1* et *serveur2*). Il est aussi composé des différents liens physiques (*lien1*, *lien2*, *lien3*) qui relient les interfaces des différents nœuds physiques. Quant au réseau virtuel, il est composé de 2 bridges (*br-int* et *br-ext*). Ces deux bridges appartiennent au groupe de nœuds virtuels « *OVS* ». La correspondance entre le réseau virtuel et le réseau physique se manifeste avec deux liaisons principales. La première liaison est entre le groupe de nœuds virtuels « *OVS* » et le nœud physique « *serveur1* ». La deuxième liaison relie l'interface du nœud virtuel « *br-ext* » à l'interface du nœud physique « *serveur1* ».

À partir de cet exemple, nous avons montré que notre modèle peut supporter une infrastructure basique d'un centre de données multi-locataires. L'instance de donnée décrit correctement les différentes composantes de l'infrastructure, la topologie du réseau virtuel et physique ainsi que la correspondance entre les deux types de réseau.

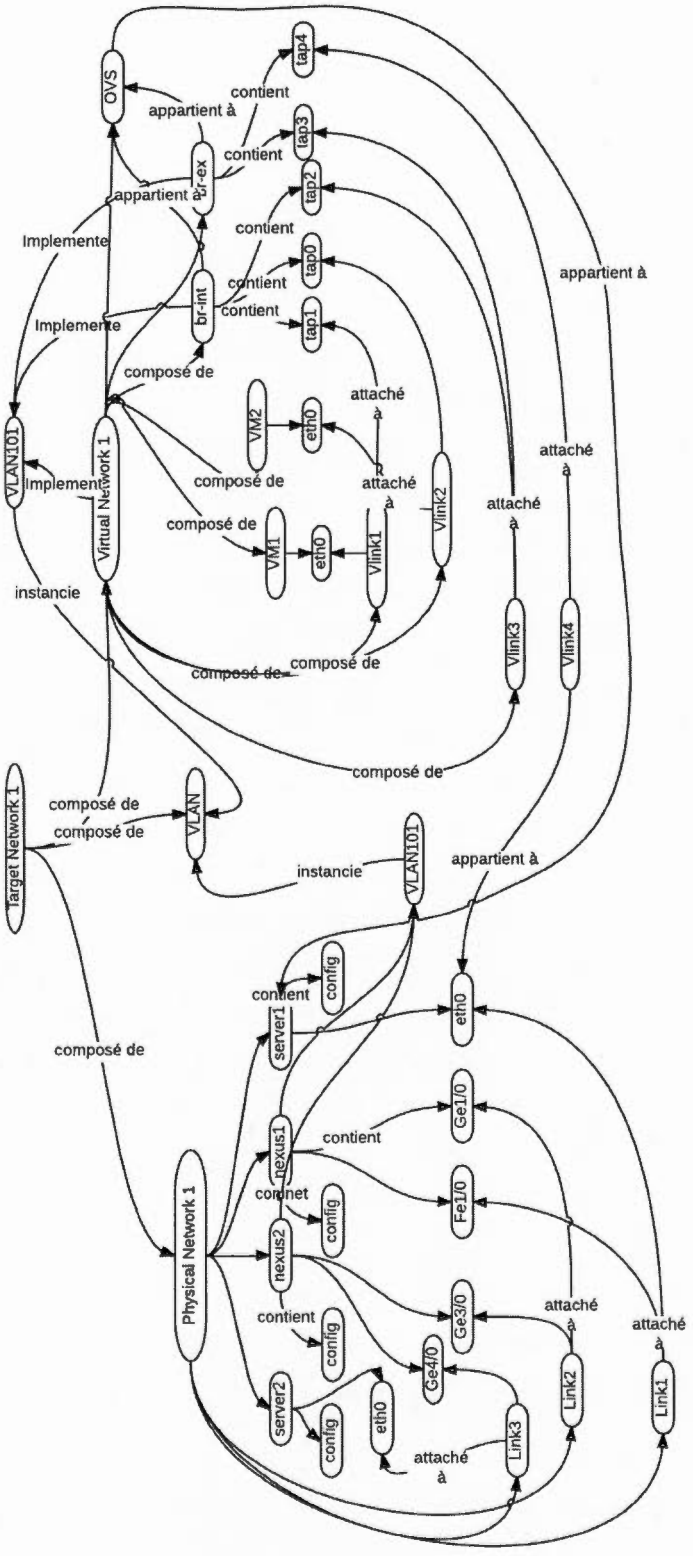


Figure 3.5: Instance du modèle *CMnet* pour l'exemple de l'infrastructure du centre de données

3.3 Les opérations supportées par CMnet

Le cycle de vie d'une machine virtuelle ou d'autres composantes virtuelles à l'intérieur d'un centre de données engendre plusieurs modifications dans la configuration de nombreux équipements. Le cycle de vie d'un composant virtuel est généralement constitué de plusieurs opérations. Nous allons décrire dans cette section un ensemble d'opérations qui peuvent intervenir dans le cycle de vie d'une machine virtuelle. Nous allons par la suite démontrer l'efficacité de notre modèle en décrivant l'implication de chacun de ses éléments dans les opérations.

- **Création de machines virtuelles:** La création d'une machine virtuelle représente une opération principale sur son cycle de vie. Lors de la création d'une *VM*, l'application de gestion du centre de données doit manipuler plusieurs éléments de notre modèle. Tout d'abord, elle doit spécifier l'emplacement de cette *VM* dépendamment de ses caractéristiques. L'application doit choisir le serveur physique adéquat où sera hébergé la *VM* en prenant en considération non seulement les paramètres de la mémoire et de la vitesse du processeur, mais aussi les performances réseau telles que la bande passante et la latence. Ces dernières ne peuvent être prises en compte qu'en ayant une vue globale sur la topologie du réseau et sur la performance de chaque lien. Après avoir choisi l'emplacement, l'application peut créer la machine virtuelle en tant qu'élément « *VirtualNode*» et l'associer au « *VirtualNetwork*» correspondant. Ensuite elle crée les « *VirtualInterface*» avec lesquels la *VM* peut se connecter au réseau puis les relier à travers les « *VirtualLink*». L'application peut alors associer le groupe de nœuds virtuels auxquels est associée la *VM* « *VirtualNodeGroupe*» au nœud physique «*PhysicalNode*» correspondant et relier l'interface virtuelle « *VirtualInterface*» à l'interface physique «*PhysicalInterface*» adéquate. Enfin, l'application se charge de la configuration des équipements virtuels et physiques. Elle doit d'abord identifier tous les équipements qui vont être affectés par la création de la *VM*. La découverte de la configuration « *Config*» et des services «*InstanceService*» sur les équipements doit être effectuée pour la validation des nouveaux services. Cette découverte peut être effectuée grâce au langage *Meta-Cli* qui décrit les configurations des équipements réseau. Après la découverte et la validation des services, l'application applique les nouveaux changements sur les équipements réseau en ajoutant de nouveaux services.

- **Création de réseau virtuel:** Dès sa création, une machine virtuelle doit toujours être

connecté à un réseau virtuel. Ainsi, la création de réseau virtuel représente une opération essentielle dans le cycle de vie d'une *VM*. Lors de la création du réseau virtuel, l'application de gestion du centre de données met à jour sa base de données. Cette opération nécessite la création d'un nouvel élément de type « *VirtualNetwork*» avec tous les paramètres correspondants et son association à l'élément « *TargetNetwork*» correspondant. D'autre part, la création de réseaux virtuels n'affecte pas la configuration de l'infrastructure physique du centre de données. Les paramètres d'un réseau virtuel ne s'appliquent sur la configuration de l'infrastructure physique que lors de la création d'un nœud virtuel associé à ce réseau virtuel.

- **Modification du service d'isolation:** L'opération de modification du service d'isolation n'est pas fréquente dans le cycle de vie d'une machine virtuelle. Généralement, cette opération n'est demandée par le locataire que s'il y a un changement de la stratégie tel que la migration d'un tunnel *GRE* vers un tunnel *VPN* pour des raisons de sécurité. La modification peut être aussi réalisé pour changer l'identifiant d'un service tel que le service *VLAN* ou *VxLAN*. Pour réaliser cette opération, l'application de gestion du centre de données doit identifier les nœuds physiques «*PhysicalNodes*» dont dépend le réseau virtuel « *VirtualNetwork*» du locataire. Ensuite, les instances de services «*InstanceService*» doivent être soit modifiées ou supprimées et instanciées d'un autre service générique « *GenericService*». Toutefois, la modification des instances de services ne peut être appliquée qu'après une validation des nouveaux paramètres pour qu'elle ne perturbe pas les autres services implémentés dans les autres équipements du réseau. Finalement, après la mise à jour de la base de données, l'application de gestion du centre de données applique les nouveaux changements sur les équipements réels de l'infrastructure physique et virtuelle.

- **Migration de la machine virtuelle:** La migration d'une machine virtuelle est une des opérations des plus délicates et les plus difficiles à réaliser. En effet, il n'existe pas beaucoup de plates-formes ou d'applications de gestion de centres de données capables d'effectuer une telle opération. Par ailleurs, cette opération comporte plusieurs sous-tâches et nécessite une gestion complète de toute l'infrastructure. Tout d'abord l'application doit identifier le nouveau serveur physique «*PhysicalNode*» sur lequel la machine virtuelle va être hébergée. Ce nouveau serveur physique doit non seulement supporter les caractéristiques de la machine virtuelle qu'il va héberger, telles que la *RAM* et le *CPU*, mais doit aussi se situer dans un emplacement où il peut assurer les performances réseau pour

Opérations	Méthodologie	
	Élément impliqué	**Moyen de configuration**
Création d'une VM	Virtual Node	Application de Gestion / BD
	Virtual Interface	Application de Gestion / BD
	Virtual Links	BD
	Configuration	OF/Netconf/CLI/Puppet ...
Création d'un réseau virtuel	Virtual Network	BD
Modification du service d'isolation	Instance Service	BD
	Vnode / Vswitch	OF/ Application de gestion
	Configuration	OF/Netconf/ CLI/Puppet ....
Lecture de vNIC	Virtual Interface	Application de gestion
Migration de la machine virtuelle	PhysicalNode	Application de gestion
	PhysicalLink	BD/ Application de gestion
	Configuration	OF/Netconf/CLI/Puppet ....
	Virtual Node	Application de gestion/BD
	Virtual Interface	Application de gestion/BD
	Virtual Links	BD

Tableau 3.1: Exemples d'opérations dans le cycle de vie d'une machine virtuelle.

la machine virtuelle. D'autre part, pour calculer des performances réseau pour la machine virtuelle, il faut calculer les performances réseau de chaque lien «*PhysicalLink*» dont dépend le réseau virtuel «*VirtualNetwork*». Après l'identification du serveur physique, l'application doit configurer tous les équipements réseau qui séparent l'ancien et le nouveau serveur afin de faciliter la migration de la machine virtuelle. Pour cela, elle doit calculer le plus court et fiable chemin entre ces deux serveurs d'après la topologie fournie par le modèle. Ensuite, l'application doit configurer tous ces équipements avec le même service «*InstanceService*» que celui du réseau virtuel auquel la *VM* appartient. Dès que l'application termine l'opération de la migration, elle met à jour la base de données avec les nouveaux paramètres de l'élément «*VirtualNode*» de la *VM* qui devient membre d'un nouvel élément de groupe de nœuds virtuels «*VirtualNodeGroup*». Enfin, l'application supprime les services ajoutés à l'équipement concernant la migration de la *VM* et met à jour la configuration des équipements dont dépend le réseau virtuel «*VirtualNetwork*» de la *VM*.

Le tableau 3.1 résume les différentes opérations qu'on vient de détailler.

3.4 Validation de CMnet avec Alloy

Nous avons utilisé *Alloy* pour la validation de notre modèle. Cet outil nous permet de trouver des problèmes liés à la conception. Il nous permet également de spécifier des contraintes sur ses objets et un cadre qui définit un nombre fini d'instances de ces objets. Tout d'abord, *Alloy* compile la spécification dans une formule propositionnelle et il utilise un solveur comme *Berkmin* [15] ou *zChaff* [27] afin de vérifier si la formule est satisfaisable. Si tel est le cas, il convertit les valeurs de variables propositionnelles, les enregistre et les affiche.

3.4.1 Formalisation du modèle CMnet

Notre modèle contient un ensemble de structures et de contraintes qu'on peut formaliser avec *Alloy*. Les contraintes qu'on va traiter dans cette sous-section sont liées aux structures du modèle.

Formalisation des nœuds:

Nous avons déclaré, pour la formalisation de notre modèle à l'aide de *Alloy*, une structure qui s'appelle «*Node*». Cette structure va être la classe parente des nœuds physiques et virtuels. En effet, cette structure englobe les paramètres communs entre les nœuds physiques et virtuels.

Code 3.1: Formalisation de *Node*

```
abstract sig Node {
    w: Node -> lone W
    }
```

Dans notre déclaration de la structure «*Node*» dans le code 3.1, nous avons spécifié une relation «*w*» qui indique le poids du lien entre chaque deux nœuds. Le poids peut caractériser un lien entre deux nœuds virtuels, deux nœuds physiques ou un nœud physique et un autre virtuel. Nous avons déclaré la structure «*Node*» pour faciliter la formalisation des nœuds physiques et virtuels.

La déclaration des nœuds physiques est illustrée par le code 3.2.

Code 3.2: Formalisation de *PhysicalNode*

```
abstract sig PhysicalNode extends Node {
      phys_Group: PhysicalNodeGroup,
      phys_net: one PhysicalNetwork
}
sig Switch, Storage, Server, Firewall
     extends PhysicalNode{}
```

Comme le montre la formalisation 3.2, on remarque que les nœuds physiques héritent de la structure «*Node*» déclarée précédemment. La structure «*PhysicalNode*» possède deux liaisons avec deux autres structures du modèle: une liaison «*phys_Group*» qui lie la structure «*PhysicalNode*» avec la structure «*PhysicalNodeGroup*» et qui indique qu'un nœud physique peut appartenir à zéro ou à plusieurs groupes de nœuds physiques. L'autre liaison est appelée «*phys_net*» et indique qu'un nœud physique doit appartenir à un seul réseau physique. Vu que les différents types de nœuds physiques n'ont pas tous le même comportement dans une infrastructure, nous avons déclaré des structures pour chacun d'eux. Dans notre implémentation nous avons spécifié cinq types de nœuds physiques qui sont les commutateurs d'accès «*AccessSwitch*», le commutateur de base «*CoreSwitch*», le serveur de stockage «*Storage*», le serveur de calcul «*Server*» et le pare-feu «*Firewall*».

Code 3.3: Formalisation de *VirtualNode*

```
abstract sig VirtualNode extends Node {
  v_group: VirtualNodeGroup,
  v_net: some VirtualNetwork
}
sig VM, Vswitch, Vrouter, Bridge
    extends VirtualNode {}
```

Les nœuds virtuels 3.3 héritent aussi de la structure «*Node*». La structure «*VirtualNode*» possède deux liaisons:

- Une liaison «*v_group*» qui lie la structure «*VirtualNode*» avec celle de «*VirtualNodeGroup*» et qui indique qu'un nœud virtuel peut appartenir à zéro ou à plusieurs groupes de nœuds virtuels.
- Une liaison appelée «*v_net*» et qui indique qu'un nœud virtuel peut appartenir à un ou à plusieurs réseaux virtuels.

Par ailleurs, les différents types de nœuds virtuels n'ont pas le même comportement dans une infrastructure, ce qui nous a amenés à déclarer dans notre implémentation quatre types de nœuds virtuels qui sont les machines virtuelles «*VM*», les commutateurs virtuels «*Vswitch*», les routeurs virtuels «*Vrouter*» et les ponts virtuels «*Bridge*».

La formalisation de la configuration pour les nœuds physiques est implémentée dans *Alloy* comme illustré dans le code 3.4:

Code 3.4: Formalisation de *Configuration*

```
sig Configuration {
  services:  InstanceService,
  phys_node: one PhysicalNode
}
```

La structure de la configuration 3.4 possède deux liaisons distinctes avec deux autres structures: une liaison «*services*» qui indique qu'une configuration peut contenir zéro ou plusieurs instances de services et une autre liaison «*phys_node*» qui indique qu'une configuration ne doit appartenir qu'à un et un seul nœud physique.

Formalisation des liens

La formalisation des liens physiques 3.10 est composée de deux blocs. Le premier représente la déclaration de l'élément du modèle «*PhysicalLink*» et le deuxième représente les contraintes à appliquer sur la déclaration. Plusieurs paramètres sont présents dans la déclaration. Les paramètres «*phys_int_one*» et «*phys_int_two*» indique qu'un lien physique ne peut être relié qu'à deux interfaces physiques. De plus le paramètre «*weight*» indique que chaque lien physique doit avoir un poids et le paramètre «*phys_net*» indique que le lien physique appartient à un seul réseau physique.

Code 3.5: Formalisation de *PhysicalLink*

```
sig PhysicalLink {
  weight: one W,
  phys_net: one PhysicalNetwork ,
  phys_int_one: one PhysicalInterface,
  phys_int_two: one PhysicalInterface
}{ disj[phys_int_one, phys_int_two ] and
  w[phys_int_one.phys_node, phys_int_two.phys_node] = weight }
```

Le deuxième bloc représente une contrainte sur la liaison «*weight*». Il indique que pour chaque lien physique qui relie deux interfaces physiques disjointes, le poids de ce lien physique est égal au poids entre les deux nœuds physiques des deux interfaces.

Code 3.6: Formalisation de *VirtualLink*

```
sig VirtualLink {
  weight: one W,
  v_int_one: one VirtualInterface,
  v_int_two: lone VirtualInterface,
  phys_int: lone PhysicalInterface,
  v_net: some VirtualNetwork
}{ w[v_int_one.v_node, v_int_two.v_node] = weight
  or w[v_int_one.v_node, phys_int.phys_node] = weight }
```

Cependant, le lien virtuel 3.6 peut relier deux interfaces virtuelles ou une interface virtuelle et une interface physique. À travers les paramètres de la déclaration des liens virtuels « *VirtualLink*» on remarque qu'il n'y a qu'un seul paramètre qui relie le lien virtuel à l'interface virtuelle et qu'il est considéré comme obligatoire en utilisant le mot clé «*one*». D'autre part, l'autre liaison avec l'interface virtuelle est considérée comme optionnelle de même pour la liaison avec l'interface physique.

Formalisation des services

Notre modèle décompose les services en deux groupes très dépendants qui sont les services génériques et les instances de services. Comme mentionné dans le chapitre 1, le service générique doit avoir seulement les paramètres les plus essentiels. Pour faciliter la formalisation des services dans *Alloy*, nous avons déclaré la structure «*GenericService*» 3.7. Cette structure va être la structure parente de tous les services génériques.

Code 3.7: Formalisation de *GenericService*

```
abstract sig GenericService {
    tg_net: one TargetNetwork
}
```

L'élément du service générique doit appartenir à l'élément du réseau global selon notre modèle. C'est pour cette raison qu'on a ajouté le paramètre «*tg_net*» qui représente une liaison avec la structure « *TargetNetwork*» en indiquant qu'un service générique doit appartenir à un et un seul réseau global.

Code 3.8: Formalisation de *Vlan*

```
abstract sig Mode{}
sig Access, Trunk extends Mode{}
sig Vlan extends GenericService{
  val : one Int
  mode: one Mode
}
```

Le service générique *VLAN* présenté dans le code 3.8 contient deux principaux paramètres qui sont «*val*» et «*mode*». Le paramètre «*val*» correspond à l'identifiant du *VLAN* tandis que le paramètre «*mode*» correspond au mode du service *VLAN*. Dans notre formalisation, le service générique ne peut être lié à travers sa liaison «*mode*» qu'a une structure de type «*Access*» ou « *Trunk*».

Après la formalisation des services génériques, la formalisation des instances de services est nécessaire pour leurs insertion dans les configurations des nœuds. Les instances de services doivent avoir une liaison d'instanciation avec les services génériques. Leur formalisation est réalisée comme illustré dans le code 3.9.

Code 3.9: Formalisation de *InstanceService*

```
abstract sig InstanceService{}
sig InstanceVlan extends InstanceService{
  instanceOf: one Vlan,
  interface: one PhysicalInterface
}
```

Toutes les structures des instances de services doivent hériter de la structure «*InstanceService*». La structure de l'instance de service VLAN contient deux principaux paramètres qui sont «*instanceOf*» et «*interface*». En effet, «*instanceOf*» représente une liaison d'instanciation entre l'instance de service et le service générique VLAN tandis que le paramètre «*interface*» indique l'interface du nœud physique sur lequel l'instance du service VLAN est appliqué.

La formalisation du modèle est essentielle cependant elle n'est pas suffisante pour générer une configuration valide. Nous allons maintenant détailler notre formalisation des contraintes du modèle et des services ainsi que les fonctions qui vont être utilisées dans ces contraintes.

3.4.2 Formalisation de l'algorithme de calcul de chemin

Pour implémenter l'algorithme de calcul de chemin, nous devons implémenter d'autres fonctions qui vont nous aider à aboutir à notre objectif. Pour calculer le plus court chemin dans une topologie, il faut formaliser la relation que les liens ont avec la distance:

- La distance entre deux nœuds physiques est égale au poids du lien physique qui relie ces deux nœuds.

Code 3.10: Formalisation de la contrainte sur les liens physiques

```
fact { all x,y: PhysicalNode | one w[x,y] <=>
       { some z: PhysicalLink |
         { z.phys_int_one.phys_node = x and z.phys_int_two.phys_node = y }
           or {z.phys_int_two.phys_node = x and z.phys_int_one.phys_node = y }
       }
}
```

- La distance entre deux nœuds virtuels est égale au poids du lien virtuel qui relie ces deux nœuds.

Code 3.11: Formalisation de la contrainte sur les liens virtuels

```
fact { all x,y: VirtualNode | one w[x,y] <=>
       { some z: VirtualLink |
          { {z.v_int_one.v_node = x and z.v_int_two.v_node = y }
            or {z.v_int_two.v_node = x and z.v_int_one.v_node = y }
          }
       }
}
```

- La distance entre un nœud virtuel et un nœud physique est égale au poids du lien virtuel qui relie ces deux nœuds.

Code 3.12: Formalisation de la contrainte des liens virtuels qui relient les nœuds virtuels et les nœuds physique

```
fact { all x: VirtualNode,y: PhysicalNode |
       { one w[x,y] <=> { some z: VirtualLink |
                          {z.v_int_one.v_node = x and
                           z.phys_int.phys_node = y }
                        }
     }
     and
     { one w[y,x] <=> { some z: VirtualLink |
                        { z.v_int_one.v_node = x and
                        z.phys_int.phys_node = y }
                         }
       }
}
```

Après la formalisation de la distance, le calcul de chemin nécessite la définition de la notion du voisin. Dans notre cas, deux voisins doivent absolument avoir une distance entre eux. Donc la fonction pour le calcul des voisins est défini dans *Alloy* comme illustré dans le code 3.13.

Code 3.13: Formalisation de la fonction de la recherche des voisins

```
fun edge: Node -> set Node {
  x,y: Node | some w[x,y]
}
```

Cette fonction 3.13 retourne tous les nœuds voisins d'un nœud passé en paramètre. Après la définition de la distance et des voisins, nous pouvons formaliser notre algorithme pour le calcul des chemins à travers notre modèle. Pour formaliser le calcul des chemins dans *Alloy*, nous avons utilisé le module "*util / ordering*" sur la structure "*State*" définit dans le code 3.14.

Code 3.14: Formalisation des états

```
sig State {
  path: set Node,
  node: one Node
}
```

Le module «*util/ordering*» crée un ordre linéaire sur les atomes de la structure «*State*». En effet, la structure «*State*» représente l'état de l'algorithme de calcul des chemins. Par ailleurs, le changement de l'état ou «*State*» signifie la découverte d'un voisin d'un nœud avec le minimum de distance. Le paramètre «*Path*» doit contenir l'ensemble des nœuds du chemin qui ont été découverts pour un état donné. Le paramètre «*node*» représente le nœud actuel pour un état donné. L'algorithme de calcul de chemin est formalisé avec *Alloy* comme illustré dans le code 3.15.

Code 3.15: Formalisation de l'algorithme de calcul de chemins

```
pred calculate_path [pre, post: State , dst:Node] {
{ pre.node != dst and #pre.node.edge >= 1 } =>
   { some y: Node |
     y in pre.node.edge and y ! in pre.path and all y':Node-y |
       { y' in pre.node.edge => lte[w[pre.node,y], w[pre.node,y']] }
        =>{ post.node = y and post.path = pre.path + y }
   }
   else { post.path = post.path + pre.path and pre.node = post.node }
}
```

La fonction de calcul de chemins prend comme paramètre d'entrée le nœud source et le nœud destination. Tout d'abord l'algorithme vérifie si le nœud source est différent du nœud destination et que le nœud source possède des voisins. Si tel est le cas, il choisit le nœud voisin qui a le minimum de distance et l'ajoute au paramètre «*Path*» de la structure «*State*». Aucun changement d'état n'est autorisé si on a atteint le nœud de destination.

3.4.3 Formalisation des contraintes

Les contraintes sont des formalismes qui décrivent les structures du modèle. *Alloy* nous permet d'avoir deux types de contraintes qui sont les faits «*Facts*» et les prédicats «*Predicates*». Lorsque *Alloy* cherche des instances du modèle, il rejette tous ceux qui sont en contradiction avec les faits. Cependant les prédicats sont des contraintes paramétrées qui ne sont pas automatiquement prises en compte par *Alloy*.

Les contraintes sur le modèle

Les contraintes qui décrivent les structures du modèle doivent toujours être vraies. C'est pour cette raison que ces contraintes sont toujours des faits. Voici quelques contraintes du modèle formalisé avec *Alloy*.

- Une configuration ne peut pas appartenir à deux nœuds différents et un nœud physique ne peut avoir qu'un seul fichier de configuration. Le code 3.16 illustre cette contrainte.

Code 3.16: Formalisation de la contrainte sur les configurations

```
fact {
  all y: Configuration,
  x: Configuration - y |
     disj [y.phys_node, x.phys_node]
}
```

- Un lien virtuel peut connecter deux interfaces virtuelles ou une interface virtuelle et une interface physique (pour connecter une machine virtuelle avec l'interface de l'hôte par exemple). Le code 3.17 illustre cette contrainte.

Code 3.17: Formalisation de la contrainte entre les interfaces et les liens virtuels

```
fact { all x: VirtualLink |
       { (#x.v_int_one = 1 and #x.v_int_two = 1 and  #x.phys_int = 0) or
         (#x.v_int = 1 and #x.v_int_two = 0 and #x.phys_int = 1)
       }
       and all y: (VirtualLink - x), z:x.v_int | not z in y.v_int
}
```

Formalisation des contraintes sur les services

Les contraintes sur les services sont des règles qui peuvent être appliquées pour s'assurer du bon fonctionnement des services. Nous avons pris comme exemple une règle (3.18) qui permet de valider le service *VLAN* pour assurer l'accès internet aux machines virtuelles. En effet, pour chaque réseau virtuel utilisant le service *VLAN* et contenant des machines virtuelles, il existe un chemin entre chaque machine virtuelle et le routeur de base «*Core Switch*» du centre de données et il faut que chaque équipement physique qui appartient à ce chemin implémente la même instance du service *VLAN*.

Code 3.18: Formalisation de la contrainte «Accès internet aux machines virtuelles»

```
pred rule_vlan_path {
    all z: VirtualNetwork, y: InstanceVlan | some sw: CoreSwitch, v: VM |
       { v.v_net = z and y.InstanceOf in z.service } <=>
         { createPath[v,sw] and all p: so/last.path |
         some phys_int: PhysicalInterface |
         p = phys_int.phys_node and y.interface = phys_int
         }
}
```

3.4.4 Génération d'une instance de configuration

L'outil *Alloy* nous permet d'effectuer des vérifications sur notre formalisation des structures et des contraintes. Pour la vérification de l'inconsistance de notre formalisation, on peut générer des instances de configuration en utilisant le mot clé «*run*». Si notre modèle est consistant, alors *Alloy* nous génère l'instance d'une configuration ainsi qu'une visualisation de cette instance.

Exemple d'une infrastructure simple:

Pour vérifier la cohérence de notre modèle, nous avons généré une instance de configuration pour une infrastructure basique d'un centre de données similaire à celle de la figure 3.4. Pour générer une configuration pour une telle infrastructure, nous avons exécuté la commande présenté dans le code 3.19:

Code 3.19: Formalisation de l'infrastructure de la figure 3.4

```
run rule_vlan_path for exactly 1 TargetNetwork,
exactly 1 PhysicalNetwork,
exactly 3 PhysicalLink,
exactly 1 PhysicalNodeGroup,
exactly 4 PhysicalNode,
exactly 1 AccessSwitch,
exactly 1 CoreSwitch,
exactly 2 Server,
exactly 0 Storage,
exactly 1 PhysicalInterfaceGroup,
6 PhysicalInterface,
exactly 0 InstanceService,
exactly 2 Configuration,
exactly 1 Vlan,
exactly 1 GenericService,
exactly 1 Tenant,
exactly 2 VirtualNetwork,
exactly 2 Bridge,
exactly 2 VM,
exactly 4 VirtualNode,
exactly 4 VirtualLink,
exactly 1 VirtualNodeGroup,
exactly 7 VirtualInterface
```

Nous avons choisi pour notre environnement le nombre de chaque structure de notre modèle. La commande indique que nous voulons générer une configuration valide selon la contrainte «*rule_vlan_path*». Vu que notre modèle est complexe et met en jeu plusieurs structures, nous allons vérifier la configuration générée de quelques éléments tels que les liens physiques et virtuels ou l'appartenance des nœuds physiques au réseau physique.

- **Vérification des liens physiques:** Après le raffinement du résultat de la génération de la configuration pour une meilleure visibilité de l'interaction des liens physiques avec les nœuds physiques, on obtient le résultat représenté dans la figure 3.6:

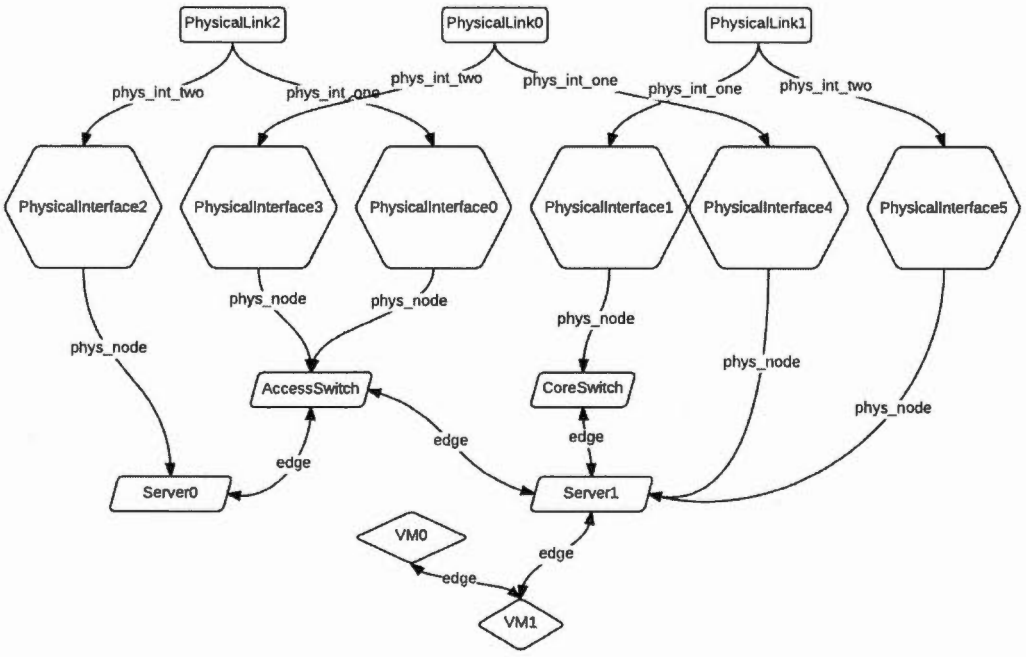


Figure 3.6: Résultat de la génération de la configuration par *Alloy* raffiné pour les liens physiques

À travers cette instance de configuration 3.6, on remarque que toutes nos contraintes ont été prises en compte et que les différentes structures interagissent entre elles comme convenu. En effet les liens physiques relient toujours deux interfaces distinctes: le lien physique «*PhysicalLink1*» relie les nœuds physiques «*server1*» et «*CoreSwitch*» à travers les interfaces «*PhysicalInterface1*» et «*PhysicalInterface5*». Le lien «*PhysicalLink0*» relie les deux nœuds physiques «*AccessSwitch*» et «*Server1*» à travers les interfaces «*PhysicalInterface3*» et «*PhysicalInterface4*».

Le troisième lien «*PhysicalLink2*» relie les nœuds physiques «*server0*» et «*AccessSwitch*» à travers les interfaces «*PhysicalInterface2*» et «*PhysicalInterface0*».

- **Vérification des liens virtuels:** Après le raffinement du résultat de la génération de la configuration pour une meilleure visibilité de l'interaction des liens virtuels avec les nœuds virtuels et physiques, on obtient le résultat représenté dans la figure 3.7 :

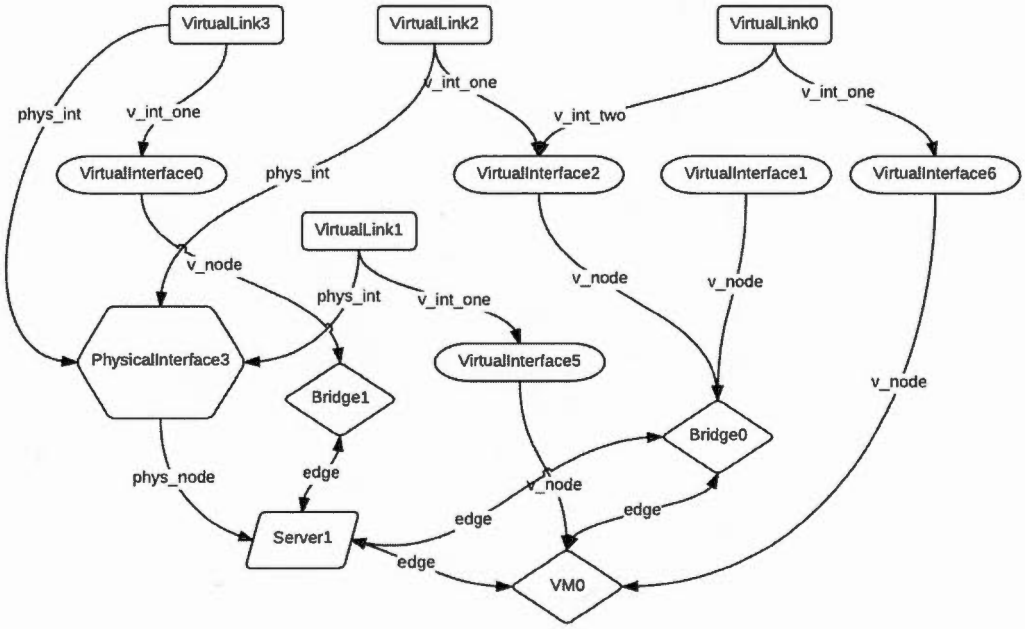


Figure 3.7: Résultat partiel de la génération de la configuration par *Alloy* raffiné pour les liens virtuels

À travers l'instance de configuration partielle présenté dans la figure 3.7, on remarque que toutes nos contraintes ont été prises en compte et que les différentes structures interagissent entre elles comme convenu. En effet les liens virtuels relient toujours deux interfaces virtuelles distinctes ou une interface virtuelle et une interface physique. Par exemple, le lien virtuel « *VirtualLink0*» relie les deux nœuds virtuels « *VM0*» et «*bridge0*» à travers les deux interfaces virtuelles « *VirtualInterface6*» et « *VirtualInterface2*». Le lien virtuel « *VirtualLink1*» relie le nœud virtuel « *VM0*» avec le nœud physique «*server1*» à travers une interface virtuelle «*virtualInterface5*» et une interface physique «*PhysicalInterface3*».

3.5 Résumé

CMnet est un modèle d'abstraction de l'infrastructure physique et virtuelle. L'un des avantages les plus importants que nous fournit ce modèle est la correspondance entre ces deux types d'infrastructures. *CMnet* est aussi capable de décrire et abstraire les services réseau pour chaque équipement grâce au langage *Meta-Cli*. Ce modèle est destiné à la gestion des centres de données multi-locataires capables de supporter la virtualisation. Nous avons conçu ce modèle afin de nous permettre l'anticipation de la création des réseaux *Overlay* pour les plates-formes de l'informatique en nuage.

Dans ce chapitre, nous avons présenté notre modèle et décrit toutes ses composantes. Nous avons par la suite proposé un exemple d'application de *CMnet*. Grâce à la puissance de l'outil et du langage *Alloy*, nous avons validé notre modèle et généré une configuration complète et valide d'une infrastructure d'un centre de données multi-locataire. Dans les chapitres suivants, nous allons détailler l'implémentation de *CMnet* et décrire des scénarios d'utilisation.

CHAPITRE IV

CONCEPTION ET IMPLÉMENTATION D'UN SYSTÈME DE GESTION DU RÉSEAU DANS UN CENTRE DE DONNÉES MULTI-LOCATAIRES

MAP (*Modular Adaptative Plugin*) est un plug-in dédié au projet réseau de OpenStack Neutron. Ce plug-in est plus riche que les autres plug-ins de Neutron en termes de fonctionnalités puisqu'il se base sur le puissant modèle d'abstraction *CMnet*. Dans ce chapitre, nous allons présenter la spécification du plug-in *MAP*, son architecture et les principales fonctionnalités qu'il réalise pour gérer l'infrastructure virtuelle de Neutron.

4.1 Spécification de MAP

4.1.1 Description

MAP, un plug-in pour le projet Neutron d'OpenStack, est conçu pour gérer efficacement et automatiquement les réseaux *Overlays*. Sur la base du modèle *CMnet*, *MAP* est capable de choisir un ou plusieurs services réseaux (*VLAN / MPLS / OTV / VXlan*, etc.) pour mettre en œuvre les réseaux virtuels et configurer les réseaux physiques afin qu'ils puissent les supporter. En effet, grâce à diverses technologies de configurations telles que *NETCONF* et *CLI* et à la représentation abstraite des configurations fournies par la *Meta-CLI*, *MAP* est capable de gérer différents types d'équipements.

4.1.2 Interaction de MAP avec Neutron

MAP est fortement couplé avec Neutron en lui fournissant une implémentation de son *API*. La figure 4.1 illustre cette interaction.

Comme montré dans la figure 4.1, le plug-in est capable de gérer à la fois les équipements physiques et les équipements virtuels. D'autre part, la gestion de ces équipements est effectuée

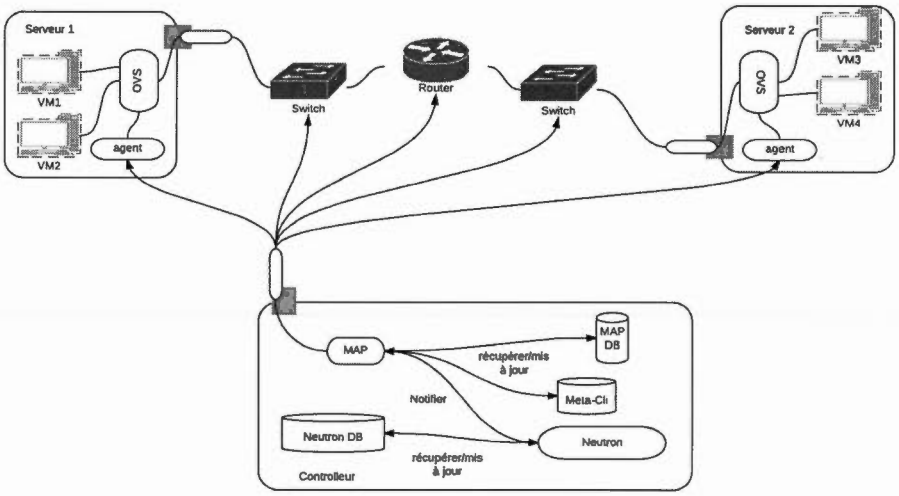


Figure 4.1: Intégration du plug-in *MAP* dans Neutron

à travers plusieurs types de *drivers* tels que le driver *CLI*, *Netconf* et *OpenFlow* ... . De plus, les paramètres et les configurations des équipements gérés sont fournis par deux bases de données de types différents. La première base de données comporte les tables et les paramètres du modèle d'abstraction *CMnet* vus dans le chapitre 3. Cette dernière est gérée par le système de gestion de bases de données MySQL [33]. Par ailleurs, l'interaction du plug-in avec cette base est effectuée à travers la boite à outils libre de Python appelée SQLAlchemy [6]. La seconde base de données est composée de fichiers XML qui représentent les configurations des équipements, les services et les règles de validation sous forme *Meta-Cli*. L'interaction entre le plug-in *MAP* et Neutron est réalisée à travers l'*API* de Neutron. Cet *API* représente l'interface des services du réseau virtuel pour les utilisateurs et pour les autres services et doit en effet être implémenté par chaque plug-in de Neutron. L'*API* de Neutron comporte:

- Pour le concept ***Network***:
  - ***List Networks***: Cette opération renvoie la liste de tous les réseaux dont le locataire a accès.
  - ***Show Network***: Cette opération renvoie la liste complète des paramètres du réseau. Cette opération ne peut être effectuée que si le réseau existe et que l'utilisateur qui fait la requête est autorisé à accéder à ces informations.

- ***Create Network***: Cette opération permet la création d'un réseau virtuel pour Neutron.
- ***Update Network***: Cette opération permet aux utilisateurs de mettre à jour certains attributs du réseau virtuel, tels que le nom du réseau. Par ailleurs certains attributs comme les identifiants ne peuvent pas être mis à jour.
- ***Delete Network***: Cette opération permet la suppression du réseau virtuel dont l'identifiant est spécifié dans la requête.

- Pour le concept ***Subnet***:
  - ***List Subnets***: Cette opération renvoie la liste des sous-réseaux auxquels le locataire a accès.
  - ***Show Subnet***: Cette opération donne des informations sur le sous-réseau spécifié dans la demande.
  - ***Create Subnet***: Cette opération crée un nouveau sous-réseau sur un réseau virtuel spécifique.
  - ***Update Subnet***: Cette opération met à jour les informations concernant un sous-réseau, tels que la version IP, et le masque du sous réseau. Cependant, l'ensemble des allocations IP ne peut pas être mis à jour.
  - ***Delete Subnet***: Cette opération permet la suppression d'un sous-réseau appartenant au réseau virtuel spécifié dans la requête.

- Pour le concept ***Port***:
  - ***List Port***: Cette opération renvoie la liste des ports virtuels auxquels le locataire a accès.
  - ***Show Port***: Cette opération fournit des informations sur le port virtuel spécifié dans la demande.
  - ***Create Port***: Cette opération crée un nouveau port virtuel sur un réseau virtuel spécifique.
  - ***Update Port***: Cette opération peut être utilisée pour mettre à jour les informations sur un port virtuel, telles que le nom symbolique et les adresses IP associées.
  - ***Delete Port***: Cette opération permet la suppression d'un port virtuel d'un réseau virtuel donnée.

4.2 Conception du plug-in

4.2.1 Architecture de MAP

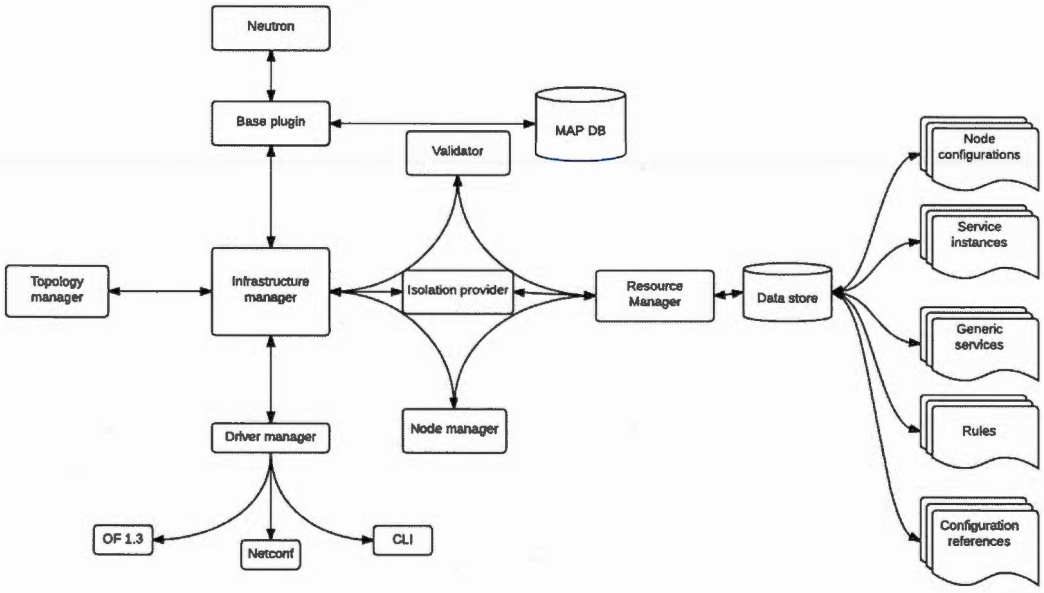


Figure 4.2: Architecture du plug-in *MAP*

Le plug-in *MAP* est constitué de plusieurs modules et composantes qui interagissent entre eux pour la réalisation de plusieurs opérations telles que la validation des services réseau et la gestion de plusieurs types d'équipements. Le plug-in *MAP* est réalisé d'une manière modulaire sous forme de blocs. De plus, il est très facile d'y ajouter ou d'en supprimer des fonctionnalités. Comme mentionné dans la section 4.1.2, le plug-in *MAP* communique avec Neutron à travers l'*API* de Neutron. Cette API est implémentée dans le module **Base Plugin**. Ce dernier communique avec l'**Infrastructure Manager** pour effectuer des opérations sur l'infrastructure du centre de données. Parmi ces opérations, on note la connexion entre deux ou plusieurs machines virtuelles distantes, la migration d'une machine virtuelle d'un serveur à un autre ou bien l'établissement de l'accès d'une machine virtuelle au réseau externe tel qu'Internet ou autre. Pour réaliser ce type d'opérations élémentaires dans les infrastructures des centres de données virtualisés, l'**Infrastructure Manager** interagit avec plusieurs autres modules qui sont le **Topology Manager**, le **Node Manager**, l'**Isolation Provider** et le **Validator**. Le module **Topology Manager** permet d'effectuer plusieurs opérations liées à la topologie à savoir le calcul

du plus court chemin et la recherche de nœuds voisins pour un nœud donné. Ce module utilise des algorithmes pour la recherche du plus court chemin tel que l'algorithme *KSP* (*K-shortest path*) [43].

Le module **Isolation Provider** est le module le plus critique, car il est étroitement lié aux services supportés par le plug-in. Ce module doit être mis à jour pour chaque nouveau service. De plus, il implémente la logique de configuration pour chaque service réseau. La principale mission de ce module est de créer des instances de services à partir des services génériques qui sont fournis par le **Resource Manager**. Le **Node Manager** quant à lui, est responsable de l'implémentation des instances de services sur les configurations des différents équipements. Ce dernier est aussi capable de découvrir les services présents dans une configuration donnée. Après l'ajout des instances de services dans les configurations, le module **Validator** permet la détection et l'extraction des conflits présents dans ces services. Grâce à son algorithme de validation qui est basé sur le langage *Meta-Cli*, ce module est capable de détecter l'équipement responsable du conflit et la partie de la règle à laquelle il ne répond pas. La configuration des équipements réseau est effectuée par l'intermédiaire des *drivers*. Par ailleurs, le choix du *driver* correspondant à chaque équipement est effectué par le module **Driver Manager**. Ce module est capable de détecter les caractéristiques de chaque équipement et par la suite utiliser le meilleur *driver* afin de configurer l'équipement. En effet, plusieurs *drivers* peuvent être utilisés avec notre plug-in. Cependant, nous n'avons considéré que deux drivers dans notre implémentation qui sont *Netconf* et *CLI*. *Netconf* est utilisé pour communiquer avec les équipements réseau à travers le protocole *Netconf* [11]. Le driver *CLI* permet la récupération et l'envoi de fichiers de configurations vers les équipements.

4.2.2 Concepts de MAP

MAP gère plusieurs notions présentes dans le modèle d'abstraction *CMnet* décrit dans le chapitre 3. Ces notions sont très importantes pour le plug-in, car elles lui permettent de disposer d'une abstraction complète de l'infrastructure physique et virtuelle afin d'avoir une correspondance entre elles. Dans cette section nous n'allons détailler que trois principales notions qui sont le *Virtual Node*, le *Physical Node* et le *Virtual Node Group*. Les autres notions seront détaillées dans l'annexe A de cette thèse. Nous notons que la signification **CRUD** pour chaque attribut d'une notion correspond à:

C - *Create*: L'attribut peut être créé;

R - *Read*: L'attribut peut être retourné en réponse à une opération «show» ou «list»;

U - *Update*: La valeur de l'attribut peut être modifiée;

D - *Delete*: La valeur de l'attribut peut être supprimée;

Notion *Physical Node*

Attribut	Type	Obligatoire	CRUD	Valeur par défaut	Contraintes de validation
id	uuid-str	Oui	CR	Généré	UUID_Pattern
name	String	Non	CRU	None	N/A
description	String	Non	CRUD	N/A	N/A
conf_file	URI	Oui	CRU	N/A	Fichier de configuration existant
type	String	Oui	CR	Switch	Switch | Storge | Server | PC | Firewall | IDS| IPS
status	String	Oui	CRU	Active	Active | Inactive
equipment Model	String	Non	CRU	N/A	N/A
system _version	String	Non	CRU	N/A	N/A
services	List(uuid-str)	Non	CRUD	Liste vide	Liste d'instances de services existants dans le réseau physique
physical_network_id	uuid-str	Oui	CR	Généré	UUID_Pattern
physical _node _group _id	uuid-str	Non	CR	Généré	UUID_Pattern

Tableau 4.1: Liste des attributs de «Physical Node».

Le nœud physique est représenté dans MAP par la notion *Physical Node*. Le tableau 4.1 présente les attributs de l'élément *Physical Node* du modèle décrit dans le chapitre 3. Un nœud physique dans *MAP* doit absolument avoir un identifiant et un fichier de configuration «*conf_file*». Un nœud physique peut avoir plusieurs types tels qu'un commutateur, un routeur, un serveur de stockage, un serveur de traitement ou un équipement de sécurité. Les nœuds physiques peuvent appartenir à un *Physical Node Group* comme présenté dans le tableau A.8, mais doivent obligatoirement appartenir à un *Physical Network* comme présenté dans le tableau A.2. Pour chaque nœud la spécification du modèle de l'équipement et de la version du système avec laquelle il fonctionne est nécessaire. Ces deux paramètres permettent au **Driver Manager** de choisir le driver approprié pour communiquer avec ces équipements.

Attribut	Type	Obligatoire	CRUD	Valeur par défaut	Contraintes de validation
id	uuid-str	Oui	CR	Généré	UUID_Pattern
name	String	Non	CRU	None	N/A
description	String	Non	CRUD	N/A	N/A
conf_file	URI	Oui	CRU	N/A	Fichier de configuration existant
type	String	Oui	CR	VM	VM | switch | router | bridge
status	String	Oui	CRU	Active	Active | Inactive
equipment Model	String	Non	CRU	N/A	N/A
system_version	String	Non	CRU	N/A	N/A
services	liste(uuid-str)	Non	CRUD	liste vide	Liste d'instances de services présents dans le réseau virtuel
virtual_network_id	list(uuid-str)	Oui	CR	Généré	UUID_Pattern
virtual _node _group _id	uuid-str	Non	CR	Généré	UUID_Pattern

Tableau 4.2: Liste des attributs de «Virtual Node».

Notion *Virtual Node*

Le nœud virtuel est représenté dans MAP par la notion *Virtual Node*. Le tableau 4.2 présente les attributs de l'élément *Virtual Node* du modèle décrit dans le chapitre 3. Comme pour le nœud physique, le nœud virtuel doit obligatoirement avoir un identifiant et un fichier de configuration décrit en langage *Meta-Cli*. Le paramètre type indique si le nœud virtuel est une machine virtuelle, un commutateur, un routeur ou un pont. Dans le choix du driver pour la configuration de l'équipement virtuel, le **Driver Manager** doit connaître le modèle de l'équipement et la version du système avec lequel il fonctionne.

Notion *Virtual Node Group*:

Attribut	Type	Obligatoire	CRUD	Valeur par défaut	Contraintes de validation
id	uuid-str	Oui	CR	Généré	UUID_Pattern.
name	String	Non	CRU	None	N/A
description	String	Non	CRUD	N/A	N/A
type	String	Non	CRUD	None	OVS | Vrouter | Bridge
physical_node *	uuid-str	Non	CRUD	None	identifiant d'un nœud physique existant
virtual_network _id	list(uuid-str)	Oui	CR	Généré	UUID_Pattern

Tableau 4.3: Liste des attributs de «Virtual Node Group».

Le groupe de nœuds virtuels est représenté dans MAP par la notion *Virtual Node Group* et

c'est l'une des notions les plus importantes de *MAP*. Le groupe de nœuds virtuels permet à *MAP* de faire la correspondance entre l'infrastructure virtuelle et l'infrastructure physique. Chaque nœud virtuel doit appartenir à un groupe de nœuds virtuels pour qu'il puisse être associé à un nœud physique. Les groupes de nœuds virtuels doivent appartenir aux mêmes réseaux virtuels que les nœuds qu'ils regroupent.

4.2.3 Fonctionnalités de MAP

MAP est conçu pour supporter plusieurs opérations et effectuer plusieurs fonctionnalités pour la plateforme OpenStack. Le principal objectif de ce plug-in est l'anticipation des conflits liés aux configurations des services et à la topologie. Dans cette section, nous allons détailler les principales opérations qui nous permettent d'atteindre cet objectif. Ces opérations sont la création des réseaux virtuels et la connexion des machines virtuelles à ces réseaux virtuels.

Mais d'abord, et avant qu'un utilisateur ne puisse effectuer ces opérations et lors de l'initialisation du plug-in, les informations sur l'infrastructure physique sont mises à jour dans la base de données de *MAP*. Ensuite, les infrastructures virtuelles qui vont être créées seront mappées sur cette infrastructure physique.

Création d'un réseau virtuel

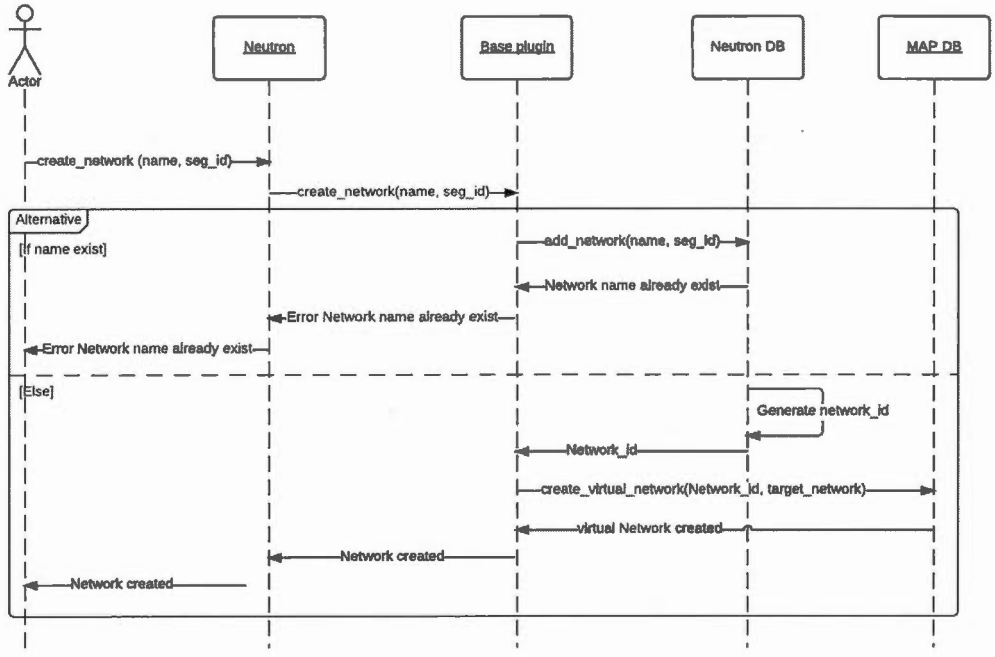


Figure 4.3: Opérations pour la création d'un réseau virtuel dans *MAP*

Avant la création des machines virtuelles, un client de OpenStack doit obligatoirement créer le réseau virtuel auquel elle va être connectée. La création d'un réseau virtuel est l'une des opérations les plus essentielles pour la création d'une infrastructure virtuelle, car toutes les autres opérations reposent sur celle-ci. Avec le plug-in *MAP*, lorsqu'un utilisateur veut créer son réseau virtuel, il doit utiliser l'*API* de Neutron. La méthode «*create_network*» est alors exécutée dans le module **Base pulgin**. L'exécution de cette méthode engendre l'ajout du réseau virtuel dans la base de données de Neutron, puis dans celle de *MAP*. Cependant d'après la spécification de l'*API* de Neutron, les réseaux virtuels doivent avoir des noms différents. D'autre part, l'objet **Network** dans les deux bases de données doit avoir le même identifiant pour pouvoir déployer l'infrastructure virtuelle sur l'infrastructure physique. La figure 4.3 illustre les opérations nécessaires pour la création d'un réseau virtuel

Connexion d'une machine virtuelle au réseau virtuel

La connexion d'une machine virtuelle à un réseau virtuel est l'une des fonctions les plus importantes dans la plateforme OpenStack et dans toutes les plateformes de l'informatique en nuage de type *Iaas*. Par ailleurs, l'implémentation de cette fonctionnalité dans *MAP* est très différente de celles des autres plug-ins de la plateforme. En effet, ce qui distingue notre plug-in par rapport aux autres c'est qu'il est très flexible et prend en considération presque toutes les caractéristiques de l'infrastructure physique et virtuelle. La figure 4.4, illustre les opérations nécessaires pour la connexion d'une machine virtuelle à un réseau virtuel.

Chaque noeud de traitement possède un agent qui communique avec le plug-in de Neutron et qui le notifie de chaque changement dans l'infrastructure virtuelle. Lorsque *Nova* crée la machine virtuelle dans un nœud de traitement, il crée avec cette machine un port virtuel et l'attache au commutateur virtuel de l'hyperviseur. L'agent de Neutron détecte l'ajout d'un port virtuel attaché au commutateur virtuel et avertit le plug-in de cet ajout. À travers ses drivers, *Map* récupère la notification de l'agent et vérifie les informations de ce nouveau port dans la base de donnée de Neutron. Après la récupération de ces informations, le processus de configuration peut commencer. Le module **Infrastructure Manager** recherche toutes les machines qui appartiennent au même réseau virtuel de la nouvelle machine virtuelle. Pour chaque machine virtuelle trouvée, l'**Infrastructure Manager** fait appel au module **Topology Manager** pour rechercher tous les chemins entre ces machines virtuelles. Le **Topology Manager** quant à lui utilise l'algorithme *KSP* pour rechercher tous les plus courts chemins entre chaque deux nœuds puis envoi tous les chemins à l'**Infrastructure Manager**. Cependant, si aucun chemin trouvé, le **Topology Manager** envoi une liste vide à l'**Infrastructure Manager**. Après l'identification des équipements et leurs interfaces qui constituent les chemins trouvés, l'**Infrastructure Manager** les envoie à l'**Isolation provider** pour la création des instances de services. **L'Isolation Provider** crée, à partir des services génériques disponibles, les instances de services pour chaque équipement du chemin. La configuration de ces instances de services est réalisée en considérant le type des équipements et sa position dans la topologie du réseau. Les instances de services crée sont alors acheminées vers le **Node Manager**. Ce dernier implémente ces instances de services sur les configurations des équipements et retourne les nouvelles configurations à l'**Infrastructure Manager** pour une validation complète des services. Le Validator utilise les règles de validation présente dans la base de données *Meta-Cli* pour effectuer la validation des services. L'algorithme de validation utilisé dans le **Validator**

ne récupère que les règles correspondant aux services qui ont été utilisés pour la configuration des équipements. Cet algorithme génère le booléen « *True* » si la validation s'effectue avec succès sinon il retourne la cause du conflit dans le cas contraire. Si la validation a réussi, les configurations sont envoyées au **Driver Manager** pour choisir le driver qui va configurer l'équipement.

4.2.4 Description de l'environnement de développement

Nous avons implémenté *MAP* sur un environnement de développement OpenStack nommé *Devstack* avec le langage *Python* dans sa version 3.

Devstack

DevStack est un script capable de créer rapidement un environnement de développement d'OpenStack. La mission de *DevStack* est de fournir et entretenir les outils utilisés pour l'installation des services de OpenStack à partir du dépôt *Git* [25]. Il peut également être utilisé pour faire des démonstrations sur l'activation et l'exploitation des services de OpenStack et fournir des exemples de leur utilisation à partir de la ligne de commande. Le script de *Devstack* peut être exécuté sur plusieurs types de plateformes telles qu'*Ubuntu* [13] ou *Fedora* [12]. Dans notre implémentation, nous avons travaillé sur la plate-forme Ubuntu server 12.4.

Devstack est aussi capable d'installer et de configurer toutes les composantes de OpenStack sur un même nœud (c'est à dire une seule machine de traitement), mais aussi sur plusieurs nœuds distants.

Python

Python [41] est un langage de programmation orienté objet. C'est le langage principal avec lequel sont écrits la majorité des projets de OpenStack. Les développeurs de OpenStack ont récemment migré les projets de la plateforme de la version 2 de Python à la version 3, car ce dernier présente plusieurs nouvelles fonctionnalités. Les principales fonctionnalités qui nous intéressent dans la version 3 sont la gestion des adresses IPv4 et la possibilité de création d'environnements virtuels pour les tests de scénarios. C'est pour cette raison qu'on opté pour l'utilisation de Python 3.

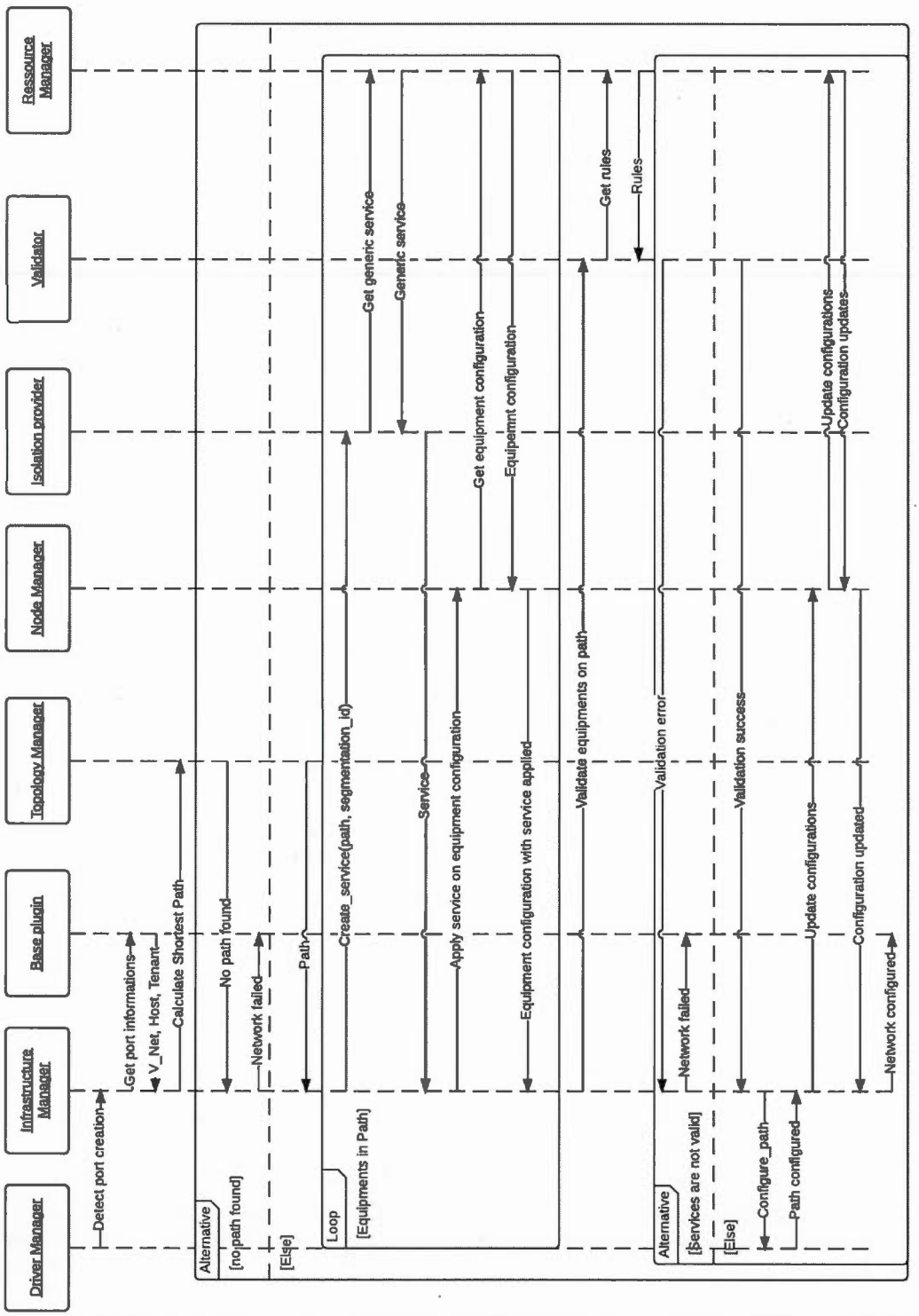


Figure 4.4: Opérations pour la connexion d'une machine virtuelle à un réseau virtuel dans *MAP*

4.3 Résumé

MAP est le plug-in réseau de Neutron que nous proposons pour la gestion et la configuration du réseau dans un environnement OpenStack. *MAP* possède des outils et des concepts qui lui permettent d'effectuer plusieurs opérations qu'on ne trouve pas chez les autres plug-ins de Neutron. Ces opérations lui permettent d'atteindre ses objectifs qui sont la détection des conflits liés à la configuration des équipements et de la topologie du réseau et l'adaptation des réseaux *Overlay* à l'infrastructure existante. Nous allons par la suite tester ces nouvelles fonctionnalités dans des infrastructures issues du monde réel des centres de données.

CHAPITRE V

ÉVALUATION DE L'IMPLÉMENTATION

Dans ce chapitre, nous allons présenter une évaluation du modèle et du plug-in proposés dans les chapitres 3 et 4. En effet, nous déploierons sous OpenStack, un ensemble d'infrastructures simples et complexes, que nous contrôlerons par l'intermédiaire de notre plug-in. Nous observerons par la suite les comportements de ces dernières suivant différents scénarios nécessitant un changement de la configuration. Nous voudrions aussi déceler les limites de notre solution en terme de nombre de nœuds et de nombre de chemins à configurer dans chaque exemple d'infrastructure. A la fin de ce chapitre, nous allons établir une comparaison entre *MAP* et d'autres plug-ins disponibles pour Neutron.

5.1 Environnement d'évaluation

Afin d'évaluer le plug-in *MAP* et le modèle *CMnet*, nous avons déployé un environnement OpenStack comme illustré dans la figure 5.1 composé de deux nœuds de traitements (*Compute Nodes*) et d'un nœud de contrôle (*Control Node*). Les nœuds de traitement vont héberger les machines virtuelles tandis que le nœud de contrôle sera responsable du contrôle de toute l'infrastructure OpenStack. Pour cela, nous avons utilisé 3 machines virtuelles avec *Ubuntu* 12.04. Par ailleurs, chaque nœud de traitement possède deux interfaces réseau: l'une reliée au nœud de contrôle et l'autre reliée aux autres nœuds de traitement par le biais d'un réseau physique émulé par *GNS3* [42] et que nous décrirons dans la suite de ce chapitre. Cet outil représente un environnement pour la simulation des réseaux informatiques et peut être utilisé pour la vérification et l'expérimentation des configurations des équipements réseau. Afin de mettre en œuvre les différents réseaux pour nos scénarios, nous avons utilisé deux types d'équipements:

- Un émulateur du routeur Cisco 7200 représenté dans la figure 5.2a: Ce routeur est utilisé

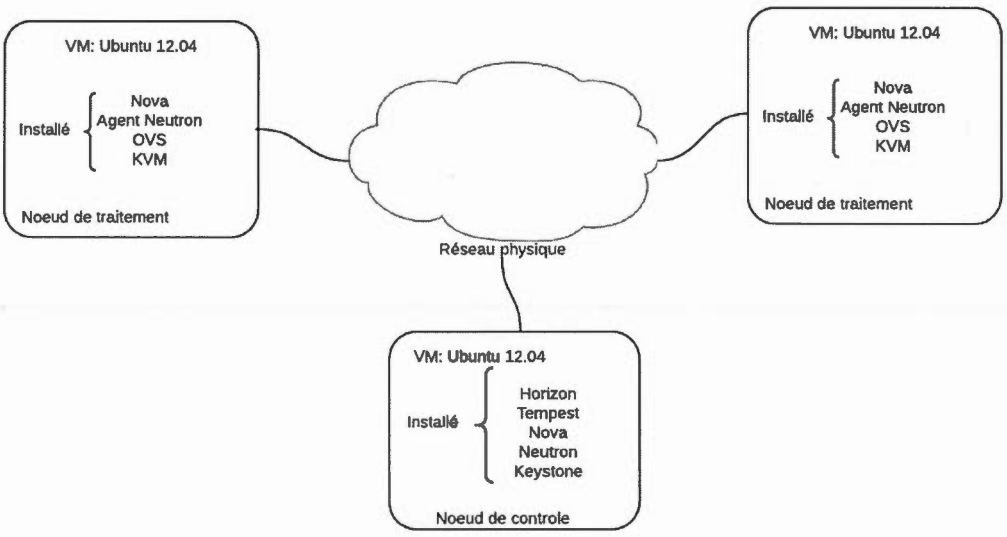
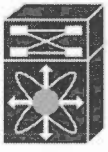


Figure 5.1: Environnement d'évaluation du plug-in *MAP*

normalement pour les infrastructures des prestataires de services en tant que routeur *edge*.

- Un émulateur du routeur Cisco 3725 représenté dans la figure 5.2b: Ce routeur peut effectuer des fonctionnalités de commutations et créer des réseaux virtuels basés sur la technologie VLAN.



(a) Émulateur du routeur Cisco 7200

(b) Émulateur du routeur Cisco 3725

Figure 5.2: Équipements utilisés dans *GNS3*

Le déploiement de OpenStack a nécessité l'installation de quelques composantes essentielles pour assurer les fonctions de base de cet environnement:

- *KVM*: Il représente l'hyperviseur déployé dans le nœud de traitement qui hébergera nos

machines virtuelles.

- *Nova*: C'est le projet d'OpenStack responsable de la gestion du cycle de vie des instances de machines virtuelles.

- *Keystone*: C'est le projet d'OpenStack responsable de la gestion des rôles, des utilisateurs, des projets et des tenants.

- *Neutron*: C'est le projet d'OpenStack fournissant la plateforme *SDN* et responsable de la communication des machines virtuelles.

- *OVS*: Il représente le commutateur virtuel de notre infrastructure permettant aux machines virtuelles d'accéder au réseau physique. L'*OVS* sera configuré à l'aide d'un agent de Neutron

5.2 Scénario 1: Détection des conflits sur la topologie du réseau:

5.2.1 Description du scénario

Ce scénario met en évidence la possibilité d'anticipation du bon fonctionnement d'un réseau *Overlay*. En effet, grâce au modèle d'abstraction de l'infrastructure physique et virtuelle, le plug-in est capable de calculer les chemins entre les différents nœuds physiques et virtuels. De plus, le plug-in est aussi capable de générer une erreur si aucun chemin n'a été trouvé entre deux nœuds distants. Dans ce scénario, nous avons simulé la création d'une machine virtuelle dans le «*serveur2*» pour un locataire donnée. Ce locataire possède déjà une machine virtuelle dans un «*serveur1*» et voudrait que ces 2 machines soient dans le même réseau virtuel. Nous avons aussi simulé la panne d'un lien connectant deux équipements physiques, ce qui engendrera en effet l'élimination de tout chemin pouvant connecter ces deux machines virtuelles. Dans ce scénario, le plug-in *MAP* est configuré de sorte à utiliser le service *VLAN* pour le réseau *Overlay* permettant la communication des deux machines virtuelles comme illustré dans la figure 5.10.

5.2.2 Évaluation de l'algorithme de calcul des chemins

La détection des conflits liés à la topologie du réseau est basée essentiellement sur l'algorithme de calcul de chemins *KSP*. Nous cherchons à évaluer notre implémentation de cet algorithme à travers des tests unitaires. Ces tests varient selon deux principaux facteurs qui sont le nombre de nœuds physiques et le nombre de plus courts chemins que l'algorithme *KSP* doit calculer. Nous

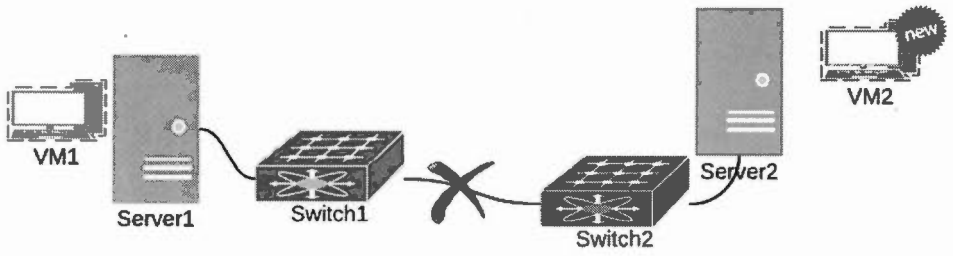


Figure 5.3: Infrastructure adoptée pour le premier scénario

tentons aussi d'élaborer l'équation pour calculer le temps de recherche du plus court chemin en fonction de ces deux facteurs. Cette équation est la suivante:

$$T_{KSP} = k + k_N N + k_C C \tag{5.1}$$

La variable T_{KSP} représente le temps d'exécution de l'algorithme de recherche du plus court chemin *KSP*. Les variables N et C représentent respectivement le nombre de nœuds du centre de données et le nombre de chemins qui doit être généré par l'algorithme *KSP*. La constante k représente une constante d'initialisation. Les principales valeurs que nous devons chercher sont la constante k_N qui représente le facteur d'impact du nombre de nœuds sur le temps d'exécution de l'algorithme *KSP* et la constante k_C qui représente le facteur d'impact du nombre de chemins sur le temps d'exécution de l'algorithme *KSP*.

La figure 5.4 présente le temps d'exécution de l'algorithme *KSP* en fonction du nombre de nœuds N présents dans l'infrastructure du centre de données. Pour ce test, nous avons fixé le nombre de chemins C égale à 1. Nous observons qu'avec l'augmentation du nombre de nœuds, l'algorithme *KSP* prend plus de temps pour le calcul et la génération du plus court chemin T_{KSP}.

La figure 5.5 représente le temps d'exécution de l'algorithme *KSP* en fonction du nombre de chemins C tout en fixant le nombre de nœuds présents dans l'infrastructure physique à $N = 29$. Nous observons qu'avec l'augmentation du nombre de chemins C, l'algorithme *KSP* prend plus de temps pour le calcul et la génération des plus courts chemins.

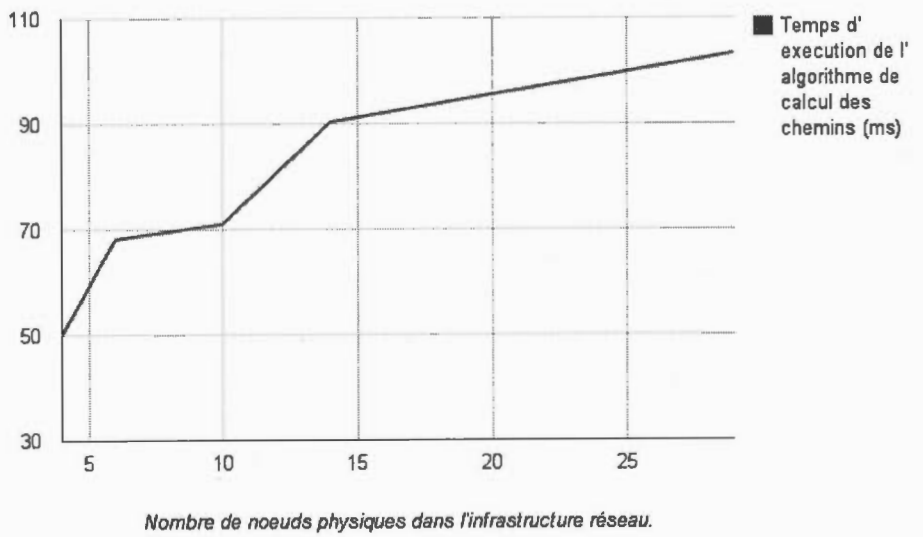


Figure 5.4: Temps d'exécution de l'algorithme *KSP* en fonction du nombre de nœuds présent dans l'infrastructure

À travers ces deux courbes 5.4 et 5.5, nous avons élaboré l'équation 5.1 de calcule du temps de recherche du plus court chemin en calculant les deux facteurs d'impact k_N et k_C:

$$k_N \simeq 1,4$$

$$k_C \simeq 115$$

Nos résultats démontrent que le facteur d'impact du nombre de chemins k_C est supérieur à celui du nombre de nœuds k_N. Ceci est expliqué par le fait qu'ajouter un chemin de plus à calculer pour l'algorithme *KSP* implique l'ajout d'un autre cycle d'exécution de l'algorithme tout en éliminant les chemins calculés précédemment.

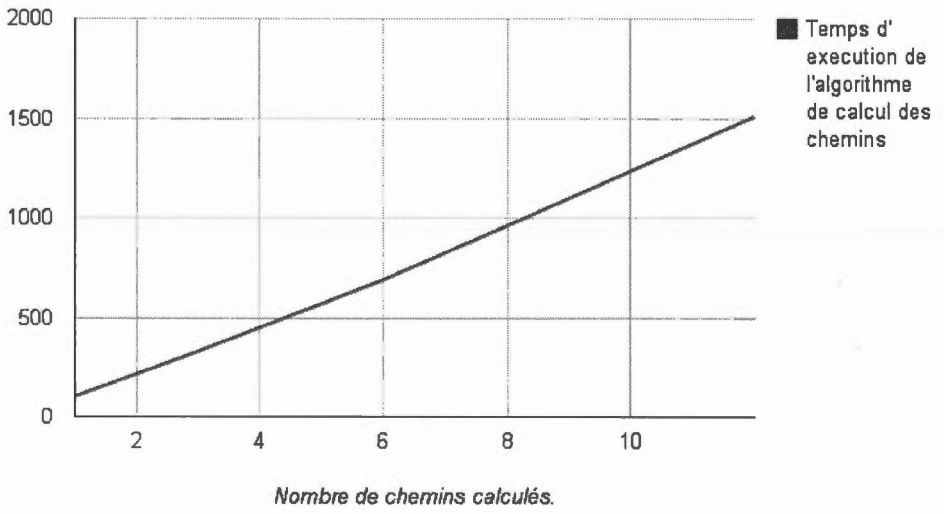


Figure 5.5: Temps d'exécution de l'algorithme *KSP* en fonction du nombre de chemins calculés

5.3 Scénario 2: Détection des conflits sur la configuration

5.3.1 Description du scénario

Ce second scénario nous permet aussi d'anticiper le bon fonctionnement d'un réseau *Overlay*. En effet, grâce à l'abstraction fournie par la *Meta-Cli* et les règles de validation qui sont adaptées à ce langage, le plug-in est capable de détecter les conflits et les problèmes qui peuvent survenir dans la configuration des équipements réseau. Sur l'infrastructure illustrée par la figure 5.6, nous avons simulé la création d'une machine virtuelle dans le «*serveur2*» pour un locataire donnée. Ce locataire possède déjà une machine virtuelle dans le «*serveur1*» et voudrait que ces deux machines virtuelles soient dans le même réseau virtuel. Pour ce fait, le plug-in *MAP* doit créer la configuration des équipements physiques reliant les deux serveurs. Par ailleurs, le plug-in *MAP* est configuré pour utiliser le service *EoMPLS* [29] pour mettre en œuvre un réseau *Overlay* entre deux serveurs physiques séparés par un domaine *L3*. D'autre part, la configuration des équipements réseau avec le service *EoMPLS*, nécessite la configuration préalable des interfaces «*Loopback0*» des équipements concernés en leur attribuant une adresse IP. Cette dernière configuration n'est en effet pas mise en œuvre dans notre scénario.

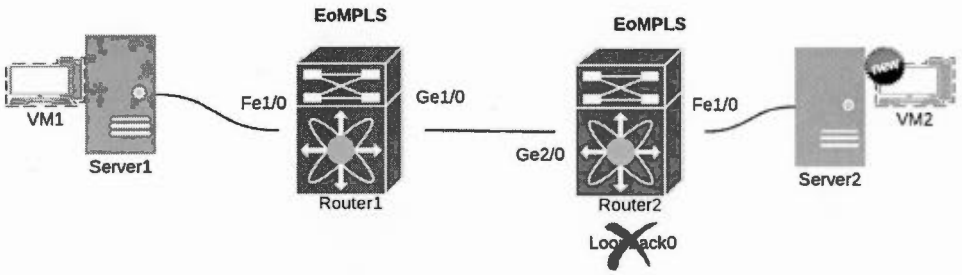


Figure 5.6: Infrastructure adoptée pour le $2^{ème}$ scénario

L'une des règles que nous avons implémentées dans *MAP* nous permet de détecter l'absence de la configuration de l'interface «*Loopback0*». Cette règle énonce que pour chaque routeur appartenant au chemin reliant les deux machines virtuelles, il faut que l'interface «*Loopback0*» existe et qu'elle ait une adresse ip valide. La description de la règle est comme suit: Si, au minimum, deux machines virtuelles appartiennent à un réseau virtuel basé sur le service VLAN, il existe un chemin entre ces deux machines tel que les adresses ip des interfaces «*Loopback0*» des routeurs qui appartiennent à ce chemin sont différent de «Null». La formule logique suivante décrit cette règle:

$$\forall Virtualnetwork : vn/\{vn.service =' vlan'\}$$
$$\exists VirtualNode : vm1, vm2/\{vm1 \neq vm2 \wedge vm1 \in vn \wedge vm2 \in vn\}$$
$$\exists paths : y/\{y = vm1 \rightarrow vm2 \wedge |y| = 1\}$$
$$\forall PhysicalNode : PhysNode/\{PhysNode \subset y \wedge PhysNode.type = Router\}$$
$$\Rightarrow (\exists PhysNode : Loopback0/\{Loopback0.ipaddress \neq null\}$$

5.3.2 Évaluation de l'algorithme de validation de la configuration

La détection des conflits liés aux services réseaux implémentés sur les configurations des équipements est basée essentiellement sur l'algorithme de validation des configurations *Meta-Cli*. Nous cherchons à évaluer l'implémentation de notre algorithme à travers des tests unitaires. Ces

tests varient selon trois principaux facteurs qui sont le nombre de nœuds du centre de données, le nombre de chemins générés par l'algorithme *KSP* et le nombre de règles de validation utilisées. Nous tentons aussi d'élaborer l'équation pour le calcul du temps de validation des services réseaux qui est sous la forme:

$$T_{val} = k\prime + k_{N}\prime N + k_{C}\prime C + k_{R}\prime R \tag{5.2}$$

La variable T_{val} représente le temps d'exécution de l'algorithme de validation des services réseaux. Les variables N, C et R représentent respectivement le nombre de nœuds du centre de données, le nombre de chemins générés par l'algorithme *KSP* et le nombre de règles de validation. La constante $k\prime$ représente une constante d'initialisation. Les principales valeurs que nous devons chercher sont:

- La constante $k_{N}\prime$ qui représente le facteur d'impact du nombre de nœuds sur le temps d'exécution de l'algorithme de validation de la configuration.
- La constante $k_{C}\prime$ qui représente le facteur d'impact du nombre de chemins générés par l'algorithme *KSP* sur l'algorithme de validation de la configuration.
- La constante $k_{R}\prime$ qui représente le facteur d'impact du nombre des règles sur l'algorithme de validation de la configuration.

La figure 5.7 présente le temps d'exécution de l'algorithme de validation en fonction du nombre de nœuds N présents dans l'infrastructure du centre de données. Nous avons aussi fixé le nombre de chemins générés par l'algorithme *KSP* à $C = 1$ et le nombre de règles de validation à $R = 2$. Nous observons qu'avec l'augmentation du nombre de nœuds l'algorithme de validation prend plus de temps pour valider les services réseaux présents dans les configurations des équipements.

La figure 5.8 représente le temps d'exécution de l'algorithme de validation en fonction du nombre de chemins C générés par l'algorithme *KSP* tout en fixant le nombre de nœuds à $N = 29$ et le nombre de règles de validation à $R = 2$. Nous observons qu'avec l'augmentation du nombre de chemins, l'algorithme de validation prends plus de temps pour valider les services réseaux présents dans la configuration des équipements.

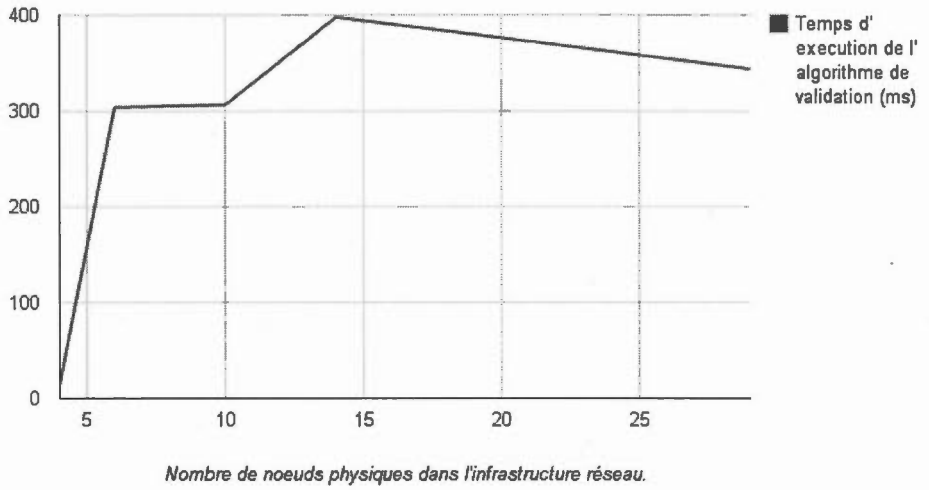


Figure 5.7: Temps d'exécution de l'algorithme de validation en fonction du nombre de nœuds présent dans l'infrastructure

La courbe de la figure 5.9 représente le temps d'exécution de l'algorithme de validation en fonction du nombre de règles de validation disponibles tout en fixant le nombre de nœuds à $N = 29$ et le nombre de chemins à $C = 2$. Nous observons qu'avec l'augmentation du nombre de règles, l'algorithme de validation prends plus de temps pour valider les services réseaux présents dans la configuration des équipements.

À travers ces trois courbes 5.8, 5.7 et 5.9, nous avons élaboré l'équation 5.2 de calcul du temps de validation des configurations des équipements réseaux en calculant les trois facteurs d'impact $k_{N'}$, $k_{C'}$ et $k_{R'}$:

$$k_{N'} \simeq 0.65$$

$$k_{C'} \simeq 488.65$$

$$k_{R'} \simeq 52.98$$

Nos résultats démontrent que le facteur d'impact du nombre de nœuds $k_{N'}$ est très inférieur aux deux autres facteurs $k_{C'}$ et $k_{R'}$. Ces résultats impliquent que le nombre de nœuds

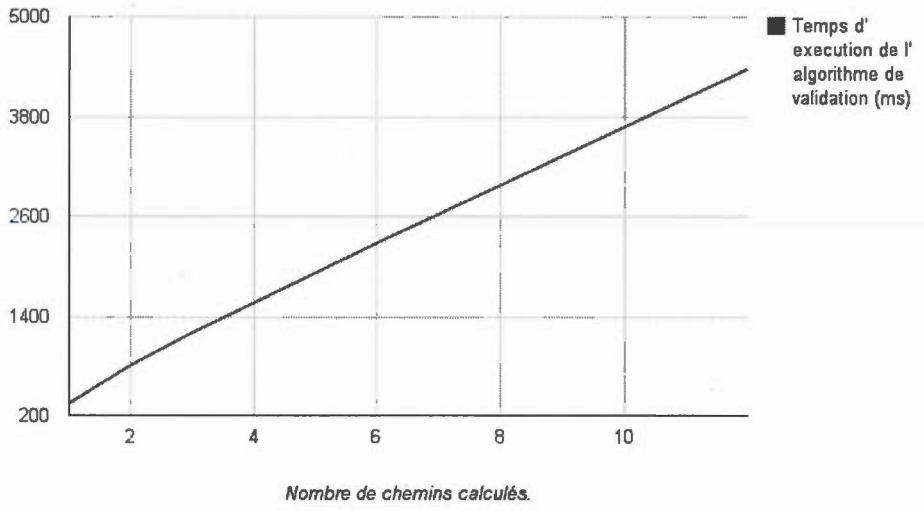


Figure 5.8: Temps d'exécution de l'algorithme de validation en fonction du nombre de chemins calculés

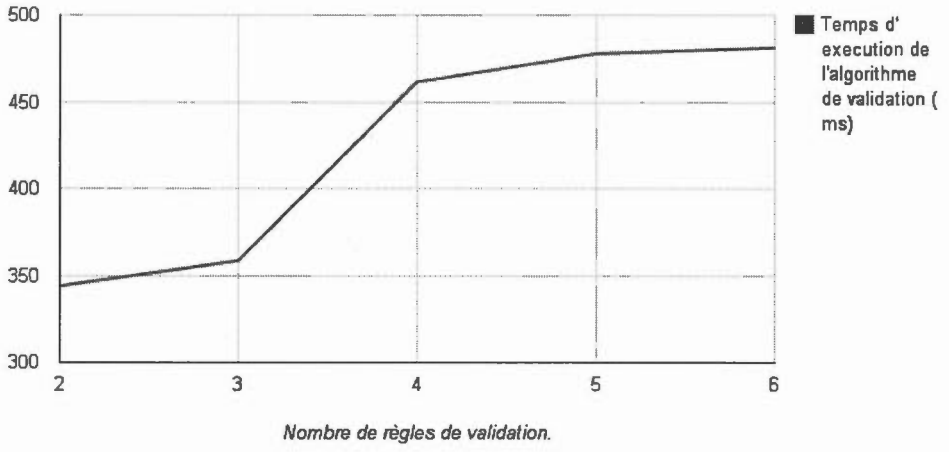


Figure 5.9: Temps d'exécution de l'algorithme de validation en fonction du nombre de règles

dans un centre de données n'a pas d'influence sur les performances de notre algorithme de validation puisqu'on ne valide que les chemins générés par l'algorithme *KSP*. Le facteur d'impact du

nombre de chemins sur l'algorithme de validation reste le plus élevé étant donné que le nombre de chemins représente le nombre de cycles d'exécution de cet algorithme.

5.4 Scénario 3: Adaptation des services à l'infrastructure physique

5.4.1 Description du scénario

Ce scénario met en évidence l'un de nos objectifs qui est le choix automatique du service réseau assurant le bon fonctionnement d'un réseau *Overlay*. En effet, ce choix doit prendre en considération le changement du domaine de diffusion dans l'infrastructure du réseau physique. Dans ce scénario, nous allons simuler la création d'une machine virtuelle dans le «*serveur2*» pour un locataire donné. Ce locataire possède déjà une machine virtuelle dans le «*serveur1*» et voudrait que ces 2 machines soient dans le même réseau virtuel. Au début du scénario, les deux nœuds de traitement sont séparés par un domaine *L3*. Après, suite à la création de la seconde machine virtuelle, le plug-in va étendre le domaine à *L2* aussi. Pour cela, le service *EoMPLS* sera utilisé, étant donné que nous avons configuré le plug-in de la sorte. L'isolation des locataires dans le domaine *L2* sera assurée par le service *VLAN*. Nous notons finalement que le locataire de ces machines virtuelles possède un réseau virtuel basé sur le protocole *VLAN*. L'identifiant de ce réseau virtuel est 10.

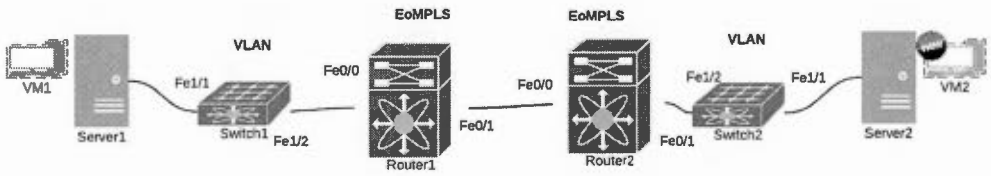


Figure 5.10: Infrastructure adoptée pour le troisième scénario

Lors de l'exécution de ce scénario, le plug-in *MAP* nous génère les configurations des différents équipements du réseau. Ces configurations sont sous format *Meta-Cli* et seront ensuite traduites sous la forme de configuration des systèmes *IOS* de *Cisco*.

Les figures 5.11 à 5.14 représentent les configurations générées pour les différents équipements *Cisco* reliant les deux nœuds de traitement. Les lignes en gras représentent les paramètres des services utilisés. Ces figurent montrent que le service *EoMPLS* a été correctement configuré

dans les routeurs pour l'extension du domaine de *Broadcasting*, tandis que les commutateurs ont été configurés à l'aide du service *VLAN*. Nous constatons aussi que seulement les interfaces qui font partie du chemin entre les deux machines virtuelles ont été configurées.

```
version 12.4
hostname Router1
ip cef
INTERFACE LOOPBACK0
 IP ADDRESS 9.9.9.9 255.255.255.0
 IP OSPF NETWORK POINT-TO-POINT
INTERFACE FASTETHERNET0/1
 IP ADDRESS 192.168.23.2 255.255.255.0
 MPLS IP
INTERFACE FASTETHERNET0/0
 no ip address
 duplex auto
 speed auto
 XCONNECT 5.5.5.5 15 ENCAPSULATION
MPLS
ROUTER OSPF 1
 LOG-ADJACENCY-CHANGES
 PASSIVE-INTERFACE FASTETHERNET0/0
 NETWORK 0.0.0.0 255.255.255.255 AREA 0
```

Figure 5.11: Configuration du routeur 1

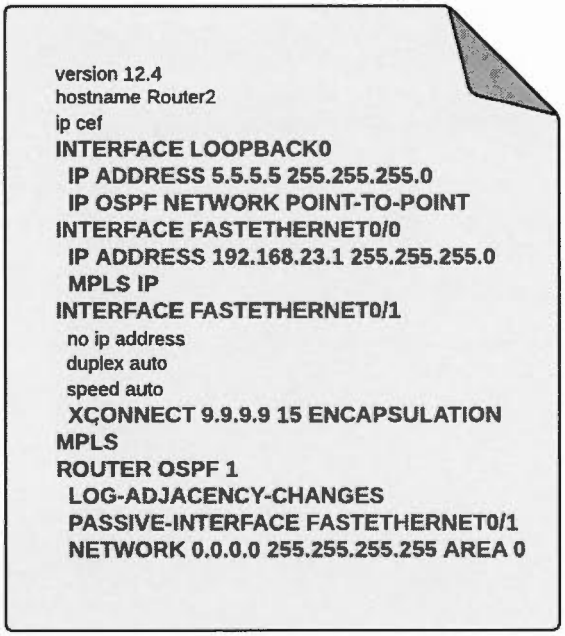


Figure 5.12: Configuration du routeur 2

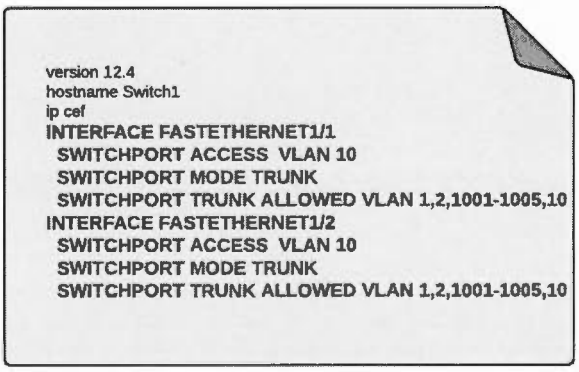


Figure 5.13: Configuration du commutateur 1

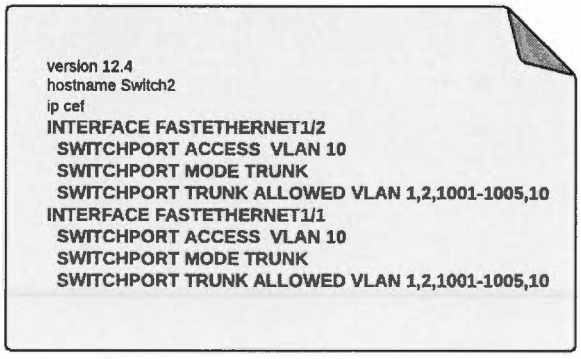


Figure 5.14: Configuration du commutateur 2

5.4.2 Évaluation de l'algorithme de génération de la configuration

Lors de l'absence des conflits liés à la topologies du réseaux et des conflits liés aux configurations des services réseaux, le processus de génération de la configuration est effectué. Ce processus dépend étroitement de celui du calcul des chemins et de la validation des configurations.Ces deux derniers sont responsables du choix des services et des équipements sur lequels ils sont appliqués. Dans cette section, nous cherchons à évaluer l'implémentation de notre algorithme à travers des tests unitaires. Ces tests varient selon deux principaux facteurs qui sont le nombre de nœuds du centre de données et le nombre de chemins générés par l'algorithme *KSP*. Nous tentons aussi d'élaborer l'équation pour le calcul du temps de validation des services réseaux qui est sous la forme:

$$T_{gen} = k// + k_N//N + k_C//C \tag{5.3}$$

La variable T_{gen} représente le temps d'exécution de l'algorithme de génération de la configuration. Les variables 'N' et 'C' représentent respectivement le nombre de nœuds du centre de donnée et le nombre de chemins générés par l'algorithme *KSP*. La constante $k//$ représente une constante d'initialisation. Les principales valeurs que nous devons chercher sont:

- La constante $k_N//$ qui représente le facteur d'impact du nombre de nœuds sur le temps d'exécution de l'algorithme de génération de la configuration.

- La constante $k_{C''}$ qui représente le facteur d'impact du nombre de chemins généré par l'algorithme *KSP* sur le temps d'exécution de l'algorithme de génération de la configuration.

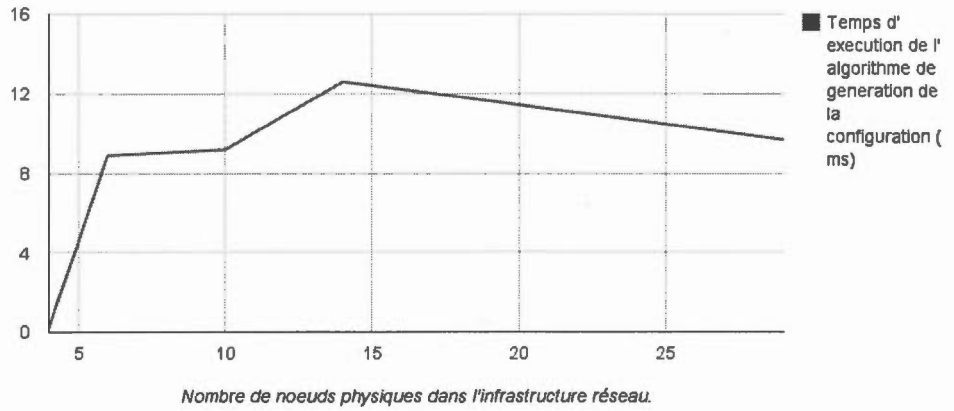


Figure 5.15: Temps d'exécution de l'algorithme de génération de la configuration en fonction du nombre de nœuds présent dans l'infrastructure

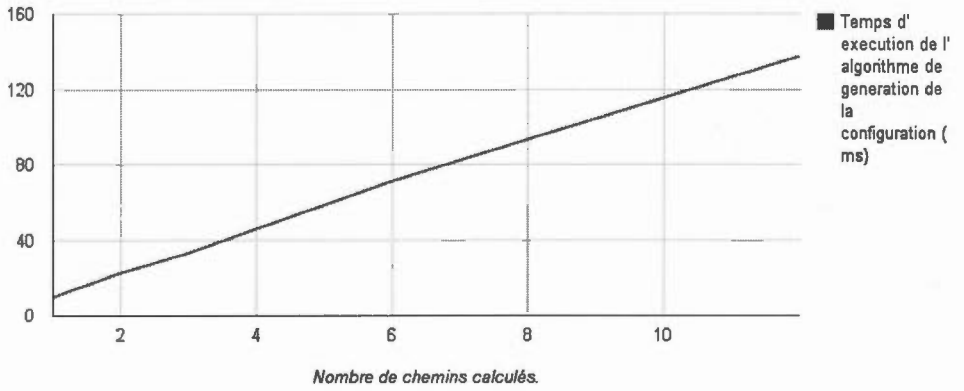


Figure 5.16: Temps d'exécution de l'algorithme de génération de la configuration en fonction du nombre de chemins calculés

La figure 5.15 présente le temps d'exécution de l'algorithme de génération de la configuration en fonction du nombre de nœuds N présents dans l'infrastructure du centre de données.

Nous avons aussi fixé le nombre de chemins générés par l'algorithme *KSP* à $C = 1$ et le nombre de règle de validation à $R = 2$. Nous observons qu'avec l'augmentation du nombre de nœuds l'algorithme de génération de la configuration prends plus de temps pour générer les configurations *IOS* des équipements.

La figure 5.16 représente le temps d'exécution de l'algorithme de génération de la configuration en fonction du nombre de chemins C générés par l'algorithme *KSP* tout en fixant le nombre de nœuds à $N = 29$ et le nombre de règles de validation à $R = 2$. Nous observons qu'avec l'augmentation du nombre de chemins, l'algorithme de génération de la configuration prend plus de temps pour générer les configurations *IOS* des équipements.

A travers ces deux courbes 5.16 et 5.15, nous avons élaboré l'équation de calcul du temps de génération des configurations des équipements réseaux en calculant les deux facteurs d'impact $k_N//$ et $k_C//$:

$$k_N// \simeq 0.04$$
$$k_C// \simeq 11.59$$

Ces résultats impliquent que le nombre de nœuds dans un centre de données n'a pas d'influence sur les performances de notre algorithme de génération de la configuration puisqu'on ne génère que les configurations des nœuds appartenant aux chemins générés par l'algorithme *KSP*. Le facteur d'impact du nombre de chemins sur l'algorithme de génération des configuration reste plus élevé que le facteur d'impact du nombre de nœuds, étant donné que le nombre de chemins représente le nombre de cycles d'exécution de cet algorithme.

5.5 Discussion des résultats des expérimentations

D'après les résultats des tests effectués sur les scénarios présentés précédemment, notre implémentation répond à toutes les spécifications énoncées, à savoir la détection des conflits causé par la configuration ou aux liaisons dans la topologie et l'adaptation des services réseaux à la topologie et aux types d'équipements présents dans l'infrastructure du centre de données.

Nous avons observé à travers les tests effectués que la configuration du réseau du centre de données lors d'une opération d'ajout d'une machine ou d'un nœud virtuel se compose de trois processus principaux qui sont:

- La recherche de chemins entre les nœuds à travers l'algorithme KSP
- La validation des services réseaux implémentés sur les configurations des équipements sous format Meta-Cli
- La génération des configurations valides en format correspondant au système de l'équipement de destination

Par ailleurs, le temps de configuration de l'infrastructure physique pour supporter les réseaux virtuels est composé en grande partie du temps d'exécution de ces trois processus. Par conséquent, les principaux facteurs qui influent sur les performances et le temps de mise en place des réseaux virtuels lors de la création des nœuds virtuels sont:

- Le nombre de nœuds présents dans l'infrastructure du centre de données
- Le nombre de chemins calculés entre les différents nœuds
- Le nombre de règles de validation pour les services réseaux

Nous tentons aussi d'élaborer l'équation pour le calcule du temps de mise en place des réseaux virtuels lors de la création des nœuds virtuels:

$$T_{total} = K + K_N N + K_C C + K_R R \tag{5.4}$$

La variable T_{total} représente le temps de gestion des réseaux virtuels. Les variables N, C et R représentent respectivement le nombre de nœuds du centre de données, le nombre de chemins générés par l'algorithme KSP et le nombre de règles de validation. La constante K représente une constante d'initialisation. Les principales valeurs que nous devons chercher:

- La constante K_N qui représente le facteur d'impact du nombre de nœuds sur le temps de mise en place des réseaux virtuels lors de la création des nœuds virtuels
- La constante K_C qui représente le facteur d'impact du nombre de chemins généré par l'algorithme KSP sur le temps de mise en place des réseaux virtuels lors de la création des nœuds virtuels
- La constante K_R qui représente le facteur d'impact du nombre des règles sur le temps de mise en place des réseaux virtuels lors de la création des nœuds virtuels

A travers les équations calculées du temps d'exécution des différents algorithmes utilisés, nous concluons que:

$$K_N = k_N + k_N\prime + k_N\prime\prime \tag{5.5}$$

$$K_C = k_C + k_C\prime + k_C\prime\prime \tag{5.6}$$

$$K_R = k_R\prime \tag{5.7}$$

D'après les résultats obtenus lors des tests de performance des différents algorithmes utilisés, nous constatons que le facteur d'impact du nombre de chemins K_C est largement supérieur aux autres facteurs à savoir K_N et K_R. En effet, un nombre de chemins important alourdit énormément le processus de mise en place des réseaux virtuels sur l'infrastructure physique. De plus, ajouter un chemin à configurer implique l'ajout d'un nouveau cycle pour l'algorithme *KSP*, l'algorithme de validation et celui de la génération des configurations. Par conséquent, le fournisseur de service qui utilise notre plug-in *MAP*, doit bien choisir le nombre de chemins qu'il veut configurer lors de chaque opération sur l'infrastructure virtuelle. En effet, un grand nombre de chemins implique une plus grande précision et une gestion plus complète de l'infrastructure mais aussi une diminution des performances et une augmentation du temps de gestion des réseaux virtuels.

5.6 Comparaison de MAP avec d'autres plug-ins de Neutron

Nous avons finalement comparé les fonctionnalités implémentées dans notre plug-in par rapport à celles d'autres plug-ins de Neutron. Notre comparaison s'est basée sur cinq critères qui sont:

1- La gestion des équipements OpenFlow

2- Le gestion des équipements traditionnels

3- Anticipation des conflits dans le réseau

Plugin	Configuration des équipements OpenFlow	Configuration des équipements traditionnels	Anticipation des conflits	Adaptation à l'infrastructure	Plusieurs types d'équipements simultanément
map	√	√	√	√	√
bigswitch	√				
cisco		√			
ml2	√	√			
openvswitch	√				
opencontrail	√	√			√
opendaylight	√	√			√
ryu	√				

Tableau 5.1: Charectéristiques des différents plug-ins de Neutron

4- Choix automatique des services à déployer

5- Gestion des équipements de différents constructeurs

MAP englobe l'ensemble de ces critères contrairement aux autres plug-ins qui vérifient quant à eux une moyenne de trois de ces propriétés.

5.7 Résumé

Notre implémentation est considérée comme satisfaisante vu qu'elle a répondu à tous nos objectifs. Cependant le plug-in *MAP*, n'est pas pour l'instant prêt pour opérer dans un environnement de production vu les performances modestes mesurées. Dans ce chapitre nous avons présenté différents scénarios de notre plug-in. Nous avons par la suite évalué les performances des algorithmes utilisés en calculant leurs temps d'exécution par rapport à différentes facteurs importants tel que le nombre de nœuds, le nombre de règles de validation et le nombre de chemins entre les nœuds. Nous avons par la suite discuté et analysé ces résultats en évoquant les points forts et les points faibles de notre implémentation. Enfin, nous avons fait une comparaison de *MAP* avec les principaux plug-in existants dans Neutron.

CONCLUSION

Les technologies de l'informatique en nuage témoignent d'une évolution importante aujourd'hui et ceci à tous les niveaux. En effet, cette évolution est due à une forte demande dans le marché mondiale pour les services de l'informatique en nuage vu leur apport considérable en matière de flexibilité et de disponibilité. Cependant, plusieurs défis restent encore à résoudre pour ces technologies émergentes et surtout en matière de gestion des réseaux virtuels pour les centres de données multi-locataires.

Dans ce mémoire, nous avons proposé une solution pour l'anticipation des conflits lors de la création des réseaux *Overlay* pour les plates-formes de l'informatique en nuage. Dans notre approche, nous avons considéré comme cruciale la dépendance entre l'infrastructure physique et l'infrastructure virtuelle. Le modèle *CMnet* que nous avons élaboré décrit bien cette dépendance. Ce modèle fournit une abstraction des différents paramètres de l'infrastructure d'un centre de données multi-locataires tel que les topologies et les services du réseau. Ce modèle a été par la suite validé grâce au langage de validation *Alloy*. De plus, nous avons utilisé le langage d'abstraction de la configuration *Meta-Cli* pour la représentation des services et des configurations des équipements réseau.

Les différents travaux réalisés pour la gestion des infrastructures virtuelles ne prennent pas en considération leurs dépendances avec l'infrastructure physique. En effet, dans le projet Neutron de la plate-forme OpenStack, les plug-ins implémentés pour la gestion de l'infrastructure virtuelle ne prennent pas en considération la topologie du réseau physique et les configurations de ses équipements. C'est pour cette raison que nous avons proposé *MAP*, un plug-in pour OpenStack proposant une nouvelle méthode pour la gestion du réseau dans un centre de données multi-locataires.

Se basant sur le modèle *CMnet* et sur le langage *Meta-Cli*, *MAP* nous permet d'avoir une maîtrise presque complète de l'infrastructure du centre de données et d'effectuer des opérations complexes tels que la validation des réseaux *Overlay* et leurs adaptations aux caractéristiques de l'infrastructure physique. L'un des principaux objectifs de ce nouveau plug-in est de permettre à la plate-forme d'OpenStack et à son projet de gestion des réseaux Neutron d'anticiper la création

des réseaux *Overlay* en détectant les conflits liés aux configurations des services réseaux et à la topologie. Ce plug-in est aussi capable d'adapter les configurations de ces services aux types d'équipements présents dans l'infrastructure du centre de données et à leurs emplacements.

Nous avons élaboré trois scénarios afin de tester le bon fonctionnement de notre plug-in. Les tests réalisés sur *MAP* ont permis d'affirmer la détection des conflits sur les services et la topologie du réseau. Nous avons par la suite testé les performances des différents algorithmes utilisé selon plusieurs facteurs qui sont le nombre de nœuds présents dans l'infrastructure, le nombre de chemins calculés et configurés entre les nœuds et le nombre de règles de validation des services réseaux. À travers ces tests, nous avons conclu que le facteur du nombre de chemins est le plus critique. En effet, un nombre important de chemins implique une plus grande précision et une gestion plus complète de l'infrastructure mais alourdit énormément le processus de mise en place des réseaux virtuels sur l'infrastructure physique.

Notre plug-in ne présente pas une solution complète pour la gestion d'une infrastructure entière d'un centre de données. En effet elle ne supporte pas pour l'instant les services réseau transactionnels. Ces services, tels que *VPN*, nécessitent un ordre logique lors de la configuration des équipements. Néanmoins, nous étions capables d'effectuer des opérations complexes sur le réseau tel que le calcul des plus courts chemins entre les nœuds et la validation des réseaux *Overlays*.

La gestion des réseaux est un domaine large et plein de défis non résolus. Dans de futurs travaux, nous pourrons explorer la partie de la visualisation du réseau en nous basant sur notre nouveau modèle pour permettre aux administrateurs réseau de mieux gérer leurs infrastructures physique et virtuelle.

APPENDICE A

CONCEPTS DE MAP

Dans cette annexe, nous allons présenter les différents concepts de *MAP* et leurs attributs.

A.1 La notion *Target Network*

Attribut	Type	Obligatoire	CRUD	Valeur par défaut	Contraintes de validation
id	uuid-str	Oui	CR	Généré	UUID_Pattern
name	String	Non	CRU	None	N/A
status	String	Oui	CRU	Active	Active | Inactive

Tableau A.1: Liste des attributs de «Target Network».

A.2 La notion *Physical Network*

Attribut	Type	Obligatoire	CRUD	Valeur par défaut	Contraintes de validation
id	uuid-str	Yes	CR	Généré	UUID_Pattern
name	String	Non	CRU	None	N/A
status	String	Oui	CRU	Active	Active | Inactive
target_netw-ork_id	uuid-str	Oui	CR	Généré	UUID_Pattern

Tableau A.2: Liste des attributs de «Physical Network».

A.3 La notion *Virtual Network*

Attribut	Type	Obligatoire	CRUD	Valeur par défaut	Contraintes de validation
id	uuid-str	Oui	CR	Généré	UUID_Pattern
name	String	Non	CRU	None	N/A
status	String	Oui	CRU	Active	Active | Inactive
tenant_id	uuid-str	Oui	CR	Généré	UUID_Pattern
target_netw-ork_id	uuid-str	Oui	CR	Généré	UUID_Pattern

Tableau A.3: Liste des attributs de «Virtual Network».

A.4 La notion *Tenant*

Attribut	Type	Obligatoire	CRUD	Valeur par défaut	Contraintes de validation
id	uuid-str	Oui	CR	Généré	UUID_Pattern
name	String	Non	CRU	None	N/A
status	String	Oui	CRU	Active	Active | Inactive

Tableau A.4: Liste des attributs de «Tenant».

A.5 La notion *Generic Service*

Attribut	Type	Obligatoire	CRUD	Valeur par défaut	Contraintes de validation
id	uuid-str	Oui	CR	Généré	UUID_Pattern
name	String	Non	CRU	None	N/A
description	String	Non	CRUD	N/A	N/A
conf_file	URI	Oui	CRU	N/A	N/A
target_netw-ork_id	uuid-str	Oui	CR	Généré	UUID_Pattern

Tableau A.5: Liste des attributs de «Generic Service».

A.6 La notion *Physical Link*

Attribut	Type	Obligatoire	CRUD	Valeur par défaut	Contraintes de validation
id	uuid-str	Oui	CR	Généré	UUID_Pattern
name	String	Non	CRU	None	N/A
description	String	Non	CRUD	N/A	N/A
cost	Integer	Oui	CRU	1	N/A
status	String	Oui	CRU	Active	Active | Inactive
physical_interfaces	tuple(uuid-str)	Oui	CRU	N/A	Identifiants des interfaces physiques existants
physical_netw-ork_id	uuid-str	Oui	CR	Généré	UUID_Pattern

Tableau A.6: Liste des attributs de «Physical Link».

A.7 La notion *Instance Service*

Attribut	Type	Obligatoire	CRUD	Valeur par défaut	Contraintes de validation
id	uuid-str	Oui	CR	Généré	UUID_Pattern
name	String	Non	CRU	None	N/A
description	String	Non	CRUD	N/A	N/A
conf_file	URI	Yes	CRU	N/A	N/A
generic_service_id	uuid-str	Oui	CR	Généré	UUID_Pattern
physical_network_id (optional)	uuid-str	Oui	CR	Généré	UUID_Pattern
virtual_network_id (optional)	uuid-str	Oui	CR	Généré	UUID_Pattern

Tableau A.7: Liste des attributs de «Instance Service».

A.8 La notion *Physical Node Group*

Attribut	Type	Obligatoire	CRUD	Valeur par défaut	Contraintes de validation
id	uuid-str	Oui	CR	Généré	UUID_Pattern
name	String	Non	CRU	None	N/A
description	String	Non	CRUD	N/A	N/A
type	String or Null	Non	CRUD	None	Chassis | None
physical_network_id	uuid-str	Oui	CR	Généré	UUID_Pattern

Tableau A.8: Liste des attributs de «Physical Node Group».

A.9 La notion *Physical Interface Group*

Attribut	Type	Obligatoire	CRUD	Valeur par défaut	Contraintes de validation
id	uuid-str	Oui	CR	Généré	UUID_Pattern
name	String	Non	CRU	None	N/A
description	String	Non	CRUD	N/A	N/A
type	String or Null	Non	CRUD	None	Linecard
physical_node_id	uuid-str	Oui	CR	Généré	UUID_Pattern

Tableau A.9: Liste des attributs de «Physical Interface Group».

A.10 La notion *Physical interface*

Attribut	Type	Obligatoire	CRUD	Valeur par défaut	Contraintes de validation
id	uuid-str	Oui	CR	Généré	UUID_Pattern
name	String	Non	CRU	None	N/A
description	String	Non	CRUD	N/A	N/A
type	String	Oui	CR	Fast Ethernt	Ethernet | Fast Ethernet | GigabitEthernet | 10 Gigabit Ethernet
status	String	Oui	CRU	Active	Active | Inactive
link_id	uuid-str	Non	CRUD	N/A	N/A
physical_nod-e_id	uuid-str	Oui	CR	Généré	UUID_Pattern
phisical_interf-ace_group_id	uuid-str	Non	CRUD	N/A	N/A.

Tableau A.10: Liste des attributs de «Physical interface».

A.11 La notion *Virtual links*

Attribut	Type	Obligatoire	CRUD	Valeur par défaut	Contraintes de validation
id	uuid-str	Oui	CR	Généré	UUID_Pattern
name	String	Non	CRU	None	N/A
description	String	Non	CRUD	N/A	N/A
status	String	Oui	CRU	Active	Active | Inactive
cost	Integer	Oui	CRU	1	N/A
virtual_interfaces	tuple(uuid-str)	Oui	CRU	N/A	UUID_Pattern
virtual_network_id	list(uuid-str)	Oui	CR	Généré	UUID_Pattern

Tableau A.11: Liste des attributs de «Virtual links».

A.12 La notion *Virtual interface*

Attribut	Type	Obligatoire	CRUD	Valeur par défaut	Contraintes de validation
id	uuid-str	Oui	CR	Généré	UUID_Pattern
name	String	Non	CRU	None	N/A
description	String	Non	CRUD	N/A	N/A
type	String	Non	CR	virtio	virtio | e1000 |ne2k_pci | pcnet | rtl8139 | tun | tap
status	String	Oui	CRU	Active	Active | Inactive
link_id	uuid-str	.Non	CRUD	N/A	N/A
virtual_nod- e_id	uuid-str	Oui	CR	Généré	UUID_Pattern

Tableau A.12: Liste des attributs de «Virtual interface».

BIBLIOGRAPHIE

[1] Neutron. `https://wiki.openstack.org/wiki/Neutron`, 2011. [En ligne le 20-Feb-2015].

[2] OpenStack Nova. `http://docs.openstack.org/developer/nova/`, 2011. [En ligne le 20-Feb-2015].

[3] alloy: a language and tool for relational models. `http://alloy.mit.edu/alloy/`, 2012. [En ligne le 10-oct-2015].

[4] Neutron Installation with OVS and VLANs (Grizzly). `https://openstack.redhat.com/Neutron_with_OVS_and_VLANs`, 2012. [En ligne le 20-Feb-2015].

[5] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt et Andrew Warfield : Xen and the art of virtualization. *ACM SIGOPS Operating Systems Review*, 37(5):164–177, 2003.

[6] Michael Bayer : Sqlalchemy-the database toolkit for python. *URL http://www.sqlalchemy. org/*, 2012. [En ligne le 20-Feb-2015].

[7] N. M. Mosharaf Kabir Chowdhury et Raouf Boutaba : Network virtualization: state of the art and research challenges. *Communications Magazine, IEEE*, 47(7):20–26, 2009.

[8] Rudy Deca : *Meta-cli configuration model for network device management.* Thèse de doctorat, Concordia University, 2003.

[9] Jeff Doyle et Matthew C Kolon : *Juniper Networks Routers: The Complete Reference.* McGraw-Hill, Inc., 2002.

[10] Omar Tayeb Edward Yue Shung Wong, Michael Herrmann : A guide to alloy. *URL http://www.doc.ic.ac.uk/project/examples/2007/271j/suprema_on_alloy/Final%20Report/LaTeX/report.pdf*, 2007.

[11] Rob Enns, Martin Bjorklund et Juergen Schoenwaelder : Netconf configuration protocol. *URL https://tools.ietf.org/html/rfc4741*, 2011.

[12] OS family Unix-like : Fedora (operating system).

[13] OS family Unix-like et GNU Userland : Ubuntu (operating system).

[14] Walter Fuertes, JE Lopez de Vergara, F Meneses et F Galan : A generic model for the management of virtual network environments. *In Network Operations and Management Symposium (NOMS), 2010 IEEE*, pages 813–816. IEEE, 2010.

[15] Eugene Goldberg et Yakov Novikov : Berkmin: A fast and robust sat-solver. *Discrete Applied Mathematics*, 155(12):1549–1561, 2007.

[16] Sylvain Hallé, Rudy Deca, Omar Cherkaoui et Roger Villemaire : Automated validation of service configuration on network devices. *In Management of Multimedia Networks and Services*, pages 176–188. Springer, 2004.

[17] Sylvain Hallé, Éric Lunaud Ngoupé, Gaëtan Nijdam, Omar Cherkaoui, Petko Valtchev et Roger Villemaire : Validmaker: A tool for managing device configurations using logical constraints. *In Network Operations and Management Symposium (NOMS), 2012 IEEE*, pages 1111–1118. IEEE, 2012.

[18] Sylvain Hallé, Éric Wenaas, Roger Villemaire et Omar Cherkaoui : Self-configuration of network devices with configuration logic. *In Autonomic Networking*, pages 36–49. Springer, 2006.

[19] Brian Hill : *Cisco: The Complete Reference.* Osborne-McGraw-Hill, 2002.

[20] Daniel Jackson : Alloy: a lightweight object modelling notation. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 11(2):256–290, 2002.

[21] Robert W Kembel : *Fibre Channel Over Ethernet (FCoE).* Northwest Learning Assoc, 2010.

[22] Avi Kivity, Yaniv Kamay, Dor Laor, Uri Lublin et Anthony Liguori : kvm: the linux virtual machine monitor. *In Proceedings of the Linux Symposium*, volume 1, pages 225–230, 2007.

[23] Rakesh Kumar, Neha Gupta, Shilpi Charu, Kanishk Jain et Sunil Kumar Jangir : Open source solution for cloud computing platform using openstack. *International Journal of Computer Science and Mobile Computing*, 3(5):89–98, 2014.

[24] Alexander Lenk, Markus Klems, Jens Nimis, Stefan Tai et Thomas Sandholm : What's inside the cloud? an architectural map of the cloud landscape. *In Proceedings of the 2009 ICSE Workshop on Software Engineering Challenges of Cloud Computing*, pages 23–31. IEEE Computer Society, 2009.

[25] Jon Loeliger et Matthew McCullough : *Version Control with Git: Powerful tools and techniques for collaborative software development.* " O'Reilly Media, Inc.", 2012.

[26] Scott Lowe : Introducing vmware vsphere 4. 2009.

[27] Yogesh S Mahajan, Zhaohui Fu et Sharad Malik : Zchaff2004: An efficient sat solver. *In Theory and Applications of Satisfiability Testing*, pages 360–375. Springer, 2005.

[28] Mallik Mahalingam, D Dutt, Kenneth Duda, Puneet Agarwal, Lawrence Kreeger, T Sridhar, Mike Bursell et Chris Wright : Vxlan: A framework for overlaying virtualized layer 2 networks over layer 3 networks. *draft-mahalingam-dutt-dcops-vxlan-08*, 2014.

[29] Luca Martini, E Rosen, N El-Aawar et G Heron : Encapsulation methods for transport of ethernet over mpls networks. *RFC4448, April*, 2006.

[30] Keith McCloghrie, Bernard R James, Christopher Young et Norman W Finn : Multiple vlan architecture system. Google Patents, 2000. US Patent 6,035,105.

[31] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker et Jonathan Turner : Openflow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, 38(2):69–74, 2008.

[32] Peter Mell et Timothy Grance : The nist definition of cloud computing. *National Institute of Standards and Technology*, 2011.

[33] AB MySQL : *MySQL: the world's most popular open source database.* MySQL AB, 1995.

[34] Sanjai Narain *et al.* : Network configuration management via model finding. *In LISA*, volume 5, pages 15–15, 2005.

[35] Cisco Nexus : 1000v series switch.

[36] Jaimin Pankajkumar Patel, Tejas Kokje et Ankur Goyal : Otv scaling: site virtual mac address, octobre 31 2012. US Patent App. 13/664,905.

[37] Ben Pfaff, Justin Pettit, Keith Amidon, Martin Casado, Teemu Koponen et Scott Shenker : Extending networking into the virtualization layer. *In Hotnets*, 2009.

[38] Omar Sefraoui, Mohammed Aissaoui et Mohsine Eleuldj : Openstack: toward an open-source solution for cloud computing. *International Journal of Computer Applications*, 55(3):38–42, 2012.

[39] Howard A Seid et Albert Lespagnol : Virtual private network. Google Patents, juin 16 1998. US Patent 5,768,271.

[40] Mathias Uslar, Michael Specht, Sebastian Rohjans, Jörn Trefke et José M González : *The Common Information Model CIM: IEC 61968/61970 and 62325-A practical introduction to the CIM*, volume 66. Springer Science & Business Media, 2012.

[41] Guido Van Rossum et Fred L Drake : *Python language reference manual*. Network Theory, 2003.

[42] Yulin WANG et Jinheng WANG : Use gns3 to simulate network laboratory. *Computer Programming Skills & Maintenance*, 12:046, 2010.

[43] Jin Y Yen : Finding the k shortest loopless paths in a network. *management Science*, 17(11):712–716, 1971.