الجمهورية الجزائرية الديمقراطية الشعبية
République Algérienne démocratique et populaire

وزارة التعليم العالي والبحث العلمي
Ministère de l'enseignement supérieur et de la recherche scientifique

جامعة سعد دحلب البليدة
Université SAAD DAHLAB de BLIDA

كلية التكنولوجيا
Faculté de Technologie

قسم الإلكترونيك
Département d'Électronique

MÉMOIRE DE MASTER

MENTION ÉLECTRONIQUE

SPÉCIALITÉ RÉSEAU & TÉLÉCOMMUNICATION

PRESENTÉ PAR
SAFSAF AHMED NAZIM
&
CHAIERE REDHOUANE

COMPARAISON DES PERFORMANCES DES CONTROLEURS OPENFLOW

PROPOSÉ PAR :

Promotrice : Docteur AMIROUCHE Nesrine

Co-promoteur : OULD TERKI Mohamed Amine

2016 - 2017

Acknowledgements

And I would like to acknowledge ...

Abstract

Le plan de control est une partie essentielle de l'architecture SDN, ou il faut donner une attention particulière à toute proposition ou conception d'un contrôleur Openflow qui sont le cœur de cette technologie qui instaure un nouveau paradigme dans le monde du réseau. Pendant les quelques années qui ont suivi l'apparition des réseaux SDN de nombreux contrôleurs Openflow ont vu le jour, ce qui offre un grand choix au (operateurs, fournisseurs, entreprise...) et pousse leur utilisateur à se poser des questions quant au choix d'un tel équipement, surtout que depuis l'explosion de l'informatique en nuage (cloud), de la virtualisation et des données massives (Big data) qui pousseront ses derniers à migrer vers ce type de réseau afin de mieux répondre aux besoin du consommateur ou à ce moment le choix du contrôleur Openflow adéquats aura un enjeux stratégique. Dans ce mémoire nous allons justement procéder à une évaluation de certains de ses contrôleurs (Beacon, Floodlight et Opendaylight) ou nous avons constatées que les contrôleurs XXXXXXX présentent les meilleures caractéristiques concernant les tests XXXXXXXX effectuer avec l'outil de benchmarking OFNet ; cependant, la détermination du contrôleur le plus performant devrait être basée sur plusieurs autres critères selon les exigences de l'utilisateur.

Table des matières

Table des figures

Liste des tableaux

Nomenclature

Acronyms / Abbreviations

ACL Access Control List

API Application Programming Interface

ARP Address Resolution Protocol

ATM Asynchronous Transfer Mode

CRC Cyclic Redundancy Check

DoS Deny of Service

FAI Fournisseur d'Accés Internet

ICMP Internet Control Message Protocol

IEEE Institute of Electrical and Electronics Engineers

MAC Media Access Control

MPLS Multi Protocol Label Switching

NAT Network Address Translation

NFV Network Function Virtualisation

NOS Network Operating System

ONF Open Network Foundation

OSI Open Systems Interconnection

OSPF Open Shortest Path First

VLAN Virtual LAN

QoS Quality of Service

SCTP Stream Control Transmission Protocol

SDN Software Defined Network

SSL Secure Sockets Layer

TCP Transmission Control Protocol

TLS Transport Layer Security

ToS Type of Service

UDP User Datagram Protocol

VM Virtual Machine

VPN Virtual Private Network

Chapitre 1

Introduction générale

Un réseau de communication est par définition un ensemble de ressources matérielles et logicielles mis en place pour offrir aux usagers un ensemble de services. Avec l'évolution des services de télécommunications et des trafics de données multimédia, les opérateurs ont déployé plusieurs technologies dans le but d'augmenter la capacité et les fonctionnalités des réseaux.

La virtualisation des serveurs et l'informatique en nuage (cloud computing) modifient la façon d'utiliser les centres de données. La virtualisation permet une utilisation plus efficace des ressources informatiques avec des niveaux élevés d'agilité et de contrôle. L'informatique en nuage (l'infonuagique) quant à lui étend ces avantages aux entreprises en leurs permettant de répondre à leurs besoins informatiques en utilisant des modèles flexibles à la demande et rapidement évolutifs.

Suite à ce développement rapide les infrastructures réseau actuels ont atteint un stade critique en raison de leur ossification [1]. Cette ossification est principalement causée par l'absence de changement dans le réseau et par la rigidité des équipements déployés. Elle rend la mise en place et le déploiement de nouveaux services réseaux difficiles et coûteux.

C'est dans ce contexte qu'est apparu le paradigme des réseaux définis par logiciels (SDN) qui permet principalement de s'adapter à la nature dynamique des applications susmentionnées. En effet, SDN permet principalement de centraliser la logique déterminant les politiques de gestion d'un réseau dans une ou plusieurs unités appelées contrôleurs. Ces contrôleurs communiquent avec le reste des équipements du réseau via des interfaces ouvertes. Elle offre ainsi la possibilité de mieux contrôler les différentes composantes du réseau.

OpenFlow est un protocole standard utilisé par le contrôleur pour transmettre au commutateur des instructions qui permettent de programmer leur plan de données et d'obtenir des informations de ces commutateurs. Il existe actuellements un assez grand nombre de controleurs compatible avec Openflow qui sont cités dans [2]

L'architecture SDN a été adoptée par plusieurs grandes entreprises et universités à savoir, Google et Stanford. D'autre part, les fabricants des équipements pour les réseaux tels que Cisco, HP et Juniper offrent désormais des solutions SDN qui permettent de gérer les centres de données.

C'est dans ce soucis de gain d'agilité de flexibilité et de contrôle qui engendre au niveau d'une entreprise tel que Djezzy une valeur sûre qui lui permettra de mieux répondre aux besoins grandissant de ses clients et ainsi enregistrer des bénéfices certain. Nous avons eu la possibilité d'y faire un stage pratique en participant à une partie de la longue période d'étude qui se fait en ce moment chez Djezzy dédiée à la migration vers cette nouvelle technologie.

1.1 Travaux précédents

Ce mémoire est basée sur des etudes passées [3], où une évaluation des contrôleurs NOX [4], Beacon [5], et Maestro [6] a été faite. D'autres travaux [7]et [8] ont fait l'évaluation de performance aussi bien pour un plus grand nombre de contrôleurs, y compris Floodlight [9], Ryu [10], OpenMul [11].
L'études faite dans [2] nous a le plus inspiré et a été cité a plusieur reprise. Toutes ces études faisaient des testes sur le débit et latence utilisant l'outil Cbench et la corrélation de ces traveaux montre que e le contrôleur Beacon et Openmul ont le débit le plus élevé et que le contrôleur OpenMul a la latence la plus faible.

1.2 Questionnement

Les questions à se poser pour ce mémoire sont les suivantes :

Qu'est-ce que l'architecture SDN, OpenFlow ? et quelles alternatives offrent ils pour l'amélioration des centres données comparativement aux architectures réseau traditionnelles ?

Quelle est la performance des contrôleurs OpenFlow à la pointe de la technologie actuellement disponibles ?
que prendre en considération pour faire un choix de controleur ?

1.3 Structure du mémoire

La structure de ce mémoire sera la suivante :
Ce mémoire est diviser en deux parties .
La premiere pratie théorique,
Après un premier chapitre introductif on abordera le chapitre 2 ou il y aura une présentation du réseau SDN detaillée et une comparaison avec les réseaux actuels. Le chapitre 3 décrira l'architecture du protocole OpenFlow et son fonctionnement.
La partie partique où dans le chapitre 4 sera expliquées la méthodologie et les expériences effectuées sur les plates-formes des contrôleurs : Beacon, Floodlight . Ce mémoire se termine par une conclusion indiquant lequel de ces contrôleurs étudiés indique les caractéristiques et peformances les plus interessantes, ainsi que les travaux futurs possibles.

Première partie

Partie théorique

Chapitre 2

Réseau défini par logiciel SDN

Selon la définition officielle de la Open Networking Foundation, le Réseau Défini par Logiciel (SDN) est une architecture réseau émergente où le plan de contrôle est découplé du plan de transfert et est directement programmable. Cette migration de contrôle, autrefois étroitement liée dans des dispositifs de réseau individuels vers des dispositifs informatiques accessibles, permet de soustraire l'infrastructure sous-jacente aux applications et aux services de réseau, qui peuvent traiter le réseau comme une entité logique ou virtuelle. [12].

2.1 Réseaux traditionnels

L'explosion des appareils mobiles et de leurs contenus, la virtualisation des serveurs, L'avènement des services clouding sont parmi les tendances qui ont poussé l'industrie à réexaminer les architectures de réseau traditionnelles. De nombreux réseaux classiques sont hiérarchisés, construits avec des niveaux de commutateurs Ethernet disposés dans une structure arborescente ou le plan de contrôle et le plan de transfert de données sont sur le même dispositif, comme vous pouvez le voir sur la figure 2.1.
Cette conception était logique lorsque l'architecture client-serveur était dominante, mais une telle architecture statique est mal adaptée à la dynamique Informatique de stockage des centres de données d'entreprise, des campus et des environnements d'opérateurs d'aujourd'hui.

2.2 Limites des technologies actuelles de mise en réseau

Répondre aux exigences actuelles du marché est pratiquement impossible avec les architectures de réseau traditionnelles. Confrontés à des budgets limités ou réduits, les départements informatiques des entreprises essaient de tirer le meilleur parti de leurs réseaux en

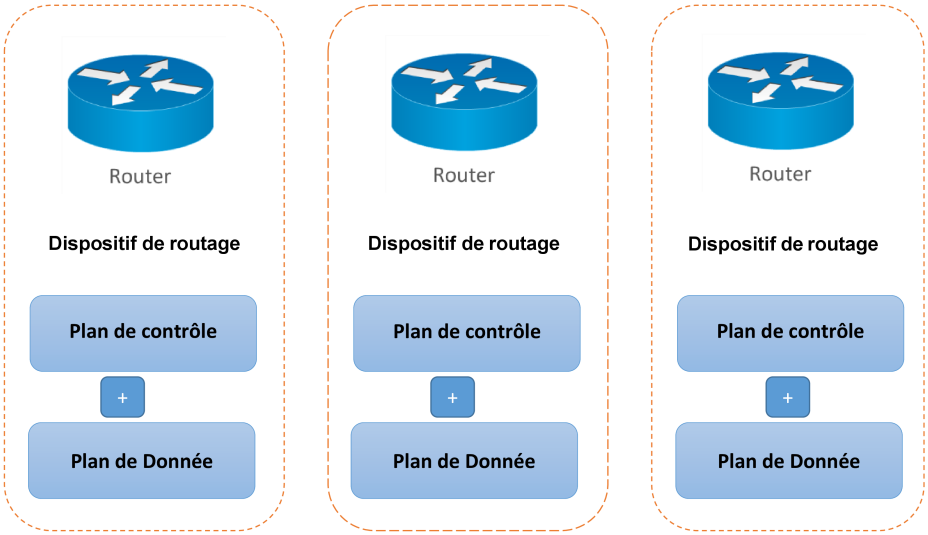

FIGURE 2.1 Réseaux traditionnels

utilisant des outils de gestion au niveau des périphériques et des processus manuels. Les IT font face à des défis similaires à ceux de la demande de mobilité et de bande passante ; Les bénéfices sont érodés par l'escalade des coûts d'équipement et les recettes stables ou en baisse. Les architectures réseau existantes n'étaient pas conçues pour répondre aux besoins des utilisateurs, des entreprises et des transporteurs d'aujourd'hui ; En vérité les concepteurs de réseau sont contraints par les limites des réseaux actuels qui sont [13] :

- **Complexité qui conduit à la stagnation :** La technologie de mise en réseau à ce jour a consisté principalement en des ensembles discrets de protocoles conçus pour connecter des hôtes de façon fiable sur des distances arbitraires, des vitesses de liaison limitées et des topologies diverses. Pour répondre aux besoins techniques et commerciaux des dernières décennies, l'industrie a développé des protocoles de mise en réseau pour offrir des performances et une fiabilité plus élevées, une connectivité plus large et une sécurité plus stricte.
  Les protocoles ont tendance à être définis de façon indépendante les uns des autres, cependant, chacun résolvant un problème spécifique et ne bénéficiant d'aucune abstraction fondamentale. Cela a eu pour résultat une des principales limitations des réseaux d'aujourd'hui : la complexité. Par exemple, pour ajouter ou déplacer un périphérique, on doit toucher plusieurs commutateurs, routeurs, pare-feux, portails d'authentification Web,etc... et mettre à jour les listes de contrôle d'accès, les ré-

seaux locaux virtuels, la qualité des services (QoS) et d'autres mécanismes basés sur des protocoles utilisant des outils de gestion au niveau périphérique. En outre, la topologie du réseau, le modèle de commutateur fournisseur et la version logicielle doivent tous être pris en compte. En raison de cette complexité, les réseaux actuels sont relativement statiques car ils cherchent à minimiser le risque de perturbation du service.

La nature statique des réseaux contraste fortement avec la nature dynamique de l'environnement serveur actuel, où la virtualisation des serveurs a considérablement augmenté le nombre d'hôtes nécessitant une connectivité réseau et des hypothèses fondamentalement modifiées sur l'emplacement physique des hôtes. Avant la virtualisation, les applications résidaient sur un seul serveur et échangent principalement le trafic avec des clients sélectionnés. Aujourd'hui, les applications sont réparties sur plusieurs machines virtuelles (VM), qui échangent des flux de trafic entre eux. Les VM migrent pour optimiser et rééquilibrer les charges de travail des serveurs, ce qui amène les points d'extrémité physiques des flux existants à changer (parfois rapidement) au fil du temps. La migration des machines virtuelles défie de nombreux aspects de la mise en réseau traditionnel, depuis l'adressage des schémas et des espaces de noms jusqu'à la notion de base d'une conception segmentée basée sur le routage.

En plus d'adopter des technologies de virtualisation, de nombreuses entreprises exploitent aujourd'hui un réseau IP convergé pour la voix, les données et le trafic vidéo. Alors que les réseaux existants peuvent fournir de multiples niveaux de QoS pour différentes applications, l'approvisionnement de ces ressources reste trés manuel. On doit configurer l'équipement de chaque fournisseur séparément et ajuster les paramètres tels que la bande passante du réseau et la QoS sur une base par session et par application. En raison de sa nature statique, le réseau ne peut pas s'adapter dynamiquement à l'évolution du trafic, des applications et des demandes des utilisateurs.

- Politiques inconsistantes : Pour mettre en œuvre une politique à l'échelle du réseau, l'informatique peut avoir à configurer des milliers de dispositifs et de mécanismes. Par exemple, chaque fois qu'une nouvelle machine virtuelle est évoquée, on peut prendre des heures, parfois quelques jours, pour reconfigurer les ACL(Listes de Contrôle d'Accès) sur l'ensemble du réseau. La complexité des réseaux actuels rend très difficile l'application d'un ensemble cohérent d'accès, de sécurité, de QoS et d'autres politiques à des utilisateurs de plus en plus mobiles, ce qui laisse l'entreprise

vulnérable aux violations de la sécurité, au non-respect de la réglementation et à d'autres conséquences négatives.

- Incapacité d'échelle : À mesure que les demandes sur le centre de données augmentent rapidement, le réseau doit également croître. Cependant, le réseau devient beaucoup plus complexe avec l'ajout de centaines ou de milliers de périphériques réseau qui doivent être configurés et gérés. Les IT ont également compté sur la sursouscription des liens pour étendre le réseau, en se fondant sur des modèles de trafic prévisibles ; Cependant, dans les centres de données virtualisés d'aujourd'hui, les modèles de trafic sont incroyablement dynamiques et donc imprévisibles.

Les mega-opérateurs, tels que Google, Yahoo !, et Facebook, font face à des défis d'évolutivité encore plus redoutables. Ces fournisseurs de services utilisent des algorithmes de traitement parallèle à grande échelle et des ensembles de données associés à la globalité de leur ensemble informatique. Au fur et à mesure que la portée des applications de l'utilisateur final augmente (par exemple, l'exploration et l'indexation de l'intégralité du Web pour renvoyer instantanément les résultats de recherche aux utilisateurs), le nombre d'éléments informatiques explose et les échanges de données entre les nœuds de calcul peuvent atteindre des petabytes. Ces entreprises ont besoin de réseaux dits «hyperscalaires» qui peuvent fournir une connectivité haute performance et à faible coût parmi des centaines de milliers - potentiellement des millions - de serveurs physiques. Cette mise à l'échelle ne peut pas être effectuée avec la configuration manuelle.

Pour demeurer concurrentiels, les opérateurs doivent offrir une valeur toujours plus élevée, des services mieux différenciés aux clients. Les fonctionnalités multiples compliquent davantage leur tâche, car le réseau doit desservir des groupes d'utilisateurs avec des applications différentes et des besoins de performance différents. Les opérations clés qui semblent relativement simples, telles que la gestion des flux de trafic d'un client pour fournir un contrôle de performance personnalisé ou une livraison à la demande, sont très complexes à mettre en œuvre avec les réseaux existants, en particulier à l'échelle de l'operateur. Ils ont besoin de dispositifs spécialisés au niveau du réseau, augmentant ainsi les dépenses en capital et les dépenses opérationnelles ainsi que le délai de mise sur le marché pour introduire de nouveaux services.

- Dépendance du fournisseur : Les operateurs et les entreprises cherchent à déployer de nouvelles capacités et services en réponse rapide aux besoins changeants des entreprises ou aux demandes des utilisateurs. Cependant, leur capacité de réaction est entravée par les cycles des produits d'équipement des vendeurs, qui peuvent aller jusqu'à trois ans ou plus. Le manque d'interfaces standard et ouvertes limite

la capacité des opérateurs de réseau à adapter le réseau à leurs environnements individuels. Cette inadéquation entre les exigences du marché et les capacités du réseau a amené l'industrie à un point de basculement. En réponse, l'industrie a créé l'architecture Software-Defined Networking (SDN) et développe des normes associées.

2.3 L'architecture SDN

Normalement, les routeurs, les commutateurs ou tout autre périphérique permettent de transférer les paquets d'un réseau à un autre. Sur la base de leur fonctionnement, on peut diviser ses équipements en deux parties ou plans : le plan de contrôle et le plan de données. Le plan de données est responsable du transport des données de la source à la destination. Il récupère les paquets de données au niveau de l'interface d'entrée afin d'exécuter des fonctions de commutation. Les tables de routage sont alors consultées pour déterminer l'interface de sortie des paquets. Tous les paquets qui ont la même destination suivent le même chemin. Le plan de contrôle est en charge de la construction et du maintien des tables de routage. Cette configuration est effectuée soit manuellement par les administrateurs du réseau soit à l'aide des informations distribuées collectées par des protocoles de routage. Actuellement, dans un routeur ou un commutateur classique les tables de routage sont programmées localement. Les nœuds du réseau choisissent librement la meilleure façon de traiter un flux. Le plan de données et le plan de contrôle sont co-localisés sur le même équipement. Les réseaux définis par logiciels (Software- Defined Network (SDN)) ont introduit une séparation entre le plan de données et le plan de contrôle pour rendre le réseau programmable. Le plan de contrôle est placé dans un contrôleur centralisé qui a une vision sur la topologie du réseau. Le plan de données réside encore sur le commutateur ou le routeur.
L'objectif de SDN est d'offrir une flexibilité et une programmabilité des réseaux afin de rendre sa gestion simple.La figure 2.2 illustre les différentes couches des réseaux SDN [14]. Le SDN est donc plus globalement reconnu aujourd'hui comme une architecture permettant d'ouvrir le réseau aux applications. Cela intègre les deux volets suivants :

- permettre aux applications de programmer le réseau afin d'en accélérer le déploiement
- permettre au réseau de mieux identifier les applications transportées pour mieux les gérer (qualité de service, sécurité, ingénierie de trafic...).

2.3.1 Modéles des réseaux SDN :

On peut citer trois différents modèles de programmabilité :

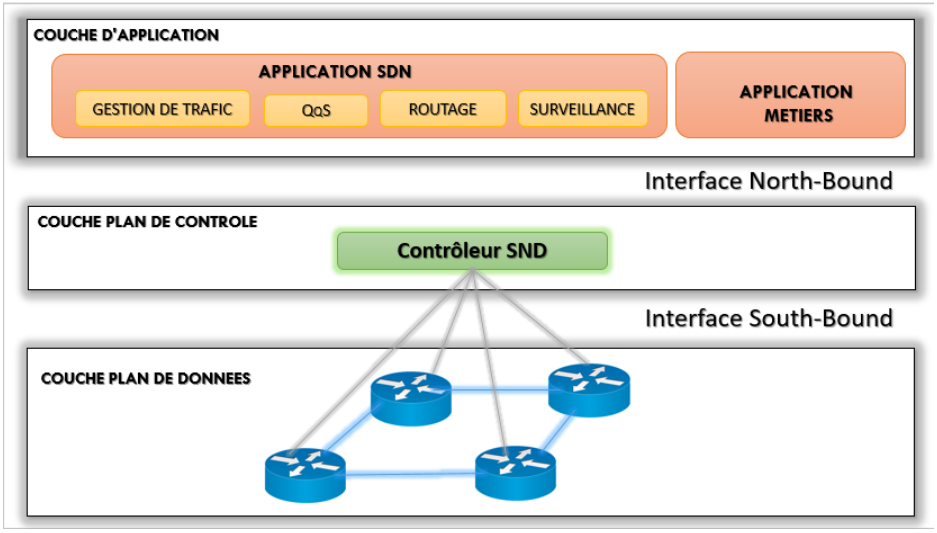

FIGURE 2.2 Architecture SDN

1. Programmabilité individuelle de chaque équipement.Dans ce modèle une application interagit directement avec chaque équipement via des API. L'application est centralisée ou peut être localisée directement sur l'équipement réseau pour réaliser des tâches spécifiques.
2. Programmabilité via un contrôleur. Dans ce modèle, une application donne un ordre abstrait et global à un contrôleur, qui à son tour traduit cette requête en une suite d'ordres auprès des équipements du réseau concerné. Ce modèle est certainement le plus populaire puisqu'il permet de simplifier le réseau. Le contrôleur masque la complexité du réseau. On peut distinguer plusieurs cas selon le type d'ordres échangés entre le contrôleur et les équipements. Si, au départ, il était question d'avoir une programmabilité du plan de données (via Openflow par exemple), des modèles plus récents implémentent des modèles dans lesquels des ordres plus abstraits sont donnés aux équipements, ces derniers restant libres de les implémenter au mieux. On parle dans ce cas d'un modèle (policyintent).
3. Création d'un réseau virtuel au dessus du réseau physique Dans ce modèle, les applications créent leur propre réseau (overlay), s'affranchissant des contraintes du réseau physique sous jacent. Ce dernier n'a pour mission que la simple connectivité entre les noeuds d'extrémité des tunnels, et le réseau d'overlay assure l'intégralité des services. On parle également de virtualisation des fonctions réseau (NFV Network Function Virtualization) quand les routeurs, commutateurs, firewalls, etc. sont des éléments virtualisés sur des serveurs. (ref ...)

2.3.2 Avantage et inconvinients des réseaux SDN

Le réseau défini par logiciel va changer la façon dont les ingénieurs réseau et les concepteurs ont de construire et d'exploiter leurs réseaux pour répondre aux besoins de l'entreprise. Avec la mise en place des SDN, les réseaux sont devenus des standards ouverts, non propriétaires et faciles à programmer et à gérer. SDN donnera aux entreprises et aux opérateurs un plus grand contrôle de leurs réseaux, leur permettra de les adapter et les optimiser pour réduire le coût global de la maintenance. Certains des principaux avantages de la SDN peuvent être résumés ci-dessous [15] :

- Simplicité de la gestion du réseau :
  Avec SDN, le réseau peut être visualisé et géré comme un nœud unique qui transférera des tâches de gestion de réseau par défaut compliquées à extraire dans une interface assez facile à gérer.
- Déploiement rapide des services :
  De nouvelles fonctionnalités et applications peuvent être déployées de manière rapide en quelques heures, au lieu de plusieurs jours.
- Configuration automatisée :
  Les tâches de configuration manuelle telles que l'attribution de VLAN et la configuration de QoS peuvent être approvisionnées automatiquement.
- Virtualisation réseau :
  Depuis la virtualisation des serveurs et du stockage est plus déployée qu'avant, les réseaux peuvent bénéficier de SDN pour être virtualisé aussi.
- Réduction des charges opérationnelles :
  Grâce à l'automatisation du déploiement du réseau, une modification du réseau n'a jamais été aussi simple, ce qui a permis de réduire le coût du fonctionnement du réseau.

Pour tous ces avantages les réseau SDN suscite un intérêt certain et permettrait des gains conséquents pour des operateur tel que Djezzy qui pourra représenter [16] :

- Un déploiement de 84% plus rapide des services et applications.
- Une reduction de 82% d'erreurs lors des déploiements
- Un approvisionnement de 81% plus rapide des nouveaux utilisateurs et clients.
- Jusqu'à 69% de réduction des dépenses d'exploitation
- Une plus grande indépendance vis-à-vis des fournisseurs équivalente à 68%.
- Une réduction des couts 53%
- Une amélioration des performances réseau 47%
- Un renforcement de la sécurité 45%.
- Une simplification des opérations réseau et maintenance 43%.

il faut bien l'avouer qu'a l'arrivée sur le marché de toute nouvelle technologie, des barrières se dressent néanmoins face aux clients tentés par un bouleversement en profondeur de leur infrastructure réseau.
Sur ce plan, ce sont les coûts associés qui arrivent en première position (47%), suivis des problématiques d'intégration et d'interopérabilité (42%). Viennent ensuite les questions de sécurité (37%) et de manque de modèle de déploiement. Là, le choix est trop restreint au regard des cas d'usages (35%), ce qui pèse sur la décision de passer de la phase de test à la mise en production réelle (34%).

2.3.3 Cas d'utilisation des réseaux SDN

— Recherche sur Internet :
Étant donné que l'Internet est un réseau vivant et est constamment utilisé, il sera difficile de faire des mises à jour ou des tests pour de nouvelles idées qui pourraient résoudre les problèmes auxquels l'infrastructure Internet actuelle fait face. Avec SDN nous avons plus de contrôle, puisque la partie contrôleur du réseau et le trafic de données est séparé, c'est-à-dire séparant la partie matérielle du logiciel. Cette séparation permet de tester de nouvelles idées sur l'architecture Internet future avant de l'implanter dans le réseau en direct [17].

— Équilibrage de charge pour les serveurs d'applications :
L'équilibrage de charge est une exigence nécessaire pour les réseaux d'entreprise afin qu'il puisse fournir une haute disponibilité et une évolutivité pour les demandes à un service particulier. Normalement, une telle fonctionnalité d'équilibrage des charges entre plusieurs serveurs est mise en oeuvre par un dispositif dédié implémenté dans le réseau. Bien qu'avec SDN, un commutateur OpenFlow puisse gérer cette fonctionnalité automatiquement et distribuer le trafic vers différents serveurs. Mais cela ne fonctionne pas convenablement a grande echelle ; Il est donc possible d'écrire une application qui peut fournir une fonction d'équilibrage de charge évolutive et efficace. Avec une telle application, la nécessité d'un milieu dédié dans le réseau sera éliminée [18].

— Mise à niveau des centres de données :
Les centres de données sont une partie interne essentielle de nombreuses entreprises à grande échelle. Par exemple, Google Facebook, Amazon et Yahoo ont un grand nombre de centres de données pour répondre à l'énorme nombre de demandes et cela le plus rapidement possible. De tels centres de données sont extrêmement coûteux et compliqués à entretenir et à exécuter, SDN et OpenFlow permettent aux entreprises

de réduire leurs coûts de configuration, car les données peuvent être gérées à partir d'un emplacement central [19].

2.4 La relation entre SDN et d'autres technologies

2.4.1 SDN et MPLS

Les réseaux MPLS(MultiProtocol Label Switching) ont évolué au cours des 10 à 15 dernières années et sont devenus d'une importance cruciale pour les fournisseur d'accés internet. MPLS est principalement utilisé de deux façons : concevoir l'ingénierie du trafic dans les réseaux IP, fournir un plus grand déterminisme et une utilisation plus efficace des ressources réseau ; Et pour l'activation des services VPN (Network Privé Virtuel) L2 ou L3 de MPLS, qui continue d'être l'un des services les plus rentables offerts par les fournisseur d'accés internet.
MPLS est la solution privilégiée pour de tels services, principalement parce que les réseaux IP sont incapables de fournir les mêmes services ; Et les anciennes façons de fournir ces services à l'aide de réseaux ATM ou Frame-Relay ne sont plus utilisées. Nous faisons observer que dans n'importe quel réseau MPLS, il existe des mécanismes d'étiquetage simples du plan de donnée MPLS (pushing-on, swapping et popping) Dans un chemin de commutation d'étiquette. Ces mécanismes sont contrôlés par un certain nombre de protocoles du plan de control qui aident à fournir les fonctionnalités et les services (Fig. 2.3.a). Cependant, toute modification de ces services ou la création d'un nouveau service, dans la plupart des cas, implique des modifications de ces protocoles ou la création d'un protocole entièrement nouveau, qui est un très longtemps Processus. Pourtant, les mécanismes du plan de données restent les mêmes simples opérations push, swap et pop.
Par conséquent il est possible d'adopter une approche différente des réseaux MPLS (Fig. 2.3.b). On utilise le plan de données MPLS standard avec un plan de commande simple et extensible basé sur OpenFlow et SDN. On constate que le plan de données MPLS a des similitudes avec l'extraction de flux, mais le plan de contrôle MPLS ne fait pas appel à l'abstraction de la carte. Ainsi, nous conservons le plan de données MPLS standard et introduisons la map-abstraction pour le plan de contrôle.
Il existe des avantages importants à utiliser l'abstraction de la carte. Le plan de contrôle est grandement simplifié et est découplé du plan de données. Nous pouvons toujours fournir tous les services offerts par les réseaux MPLS d'aujourd'hui ; Mais plus important encore, nous pouvons aller au-delà de ce que MPLS fournit aujourd'hui. Nous pouvons optimiser globalement les services ; Les rendre plus dynamiques ; Ou créer de nouveaux services en

programmant simplement des applications de réseau en plus de l'abstraction de la carte. Les nouvelles fonctionnalités ne sont plus liées à des couches de protocoles (qui sont éliminés). Et le switch-API (OpenFlow) n'a pas besoin de changer, car tout ce qu'il donne est un contrôle sur les opérations simples du plan de données push/pop/swap, qui restent les mêmes.

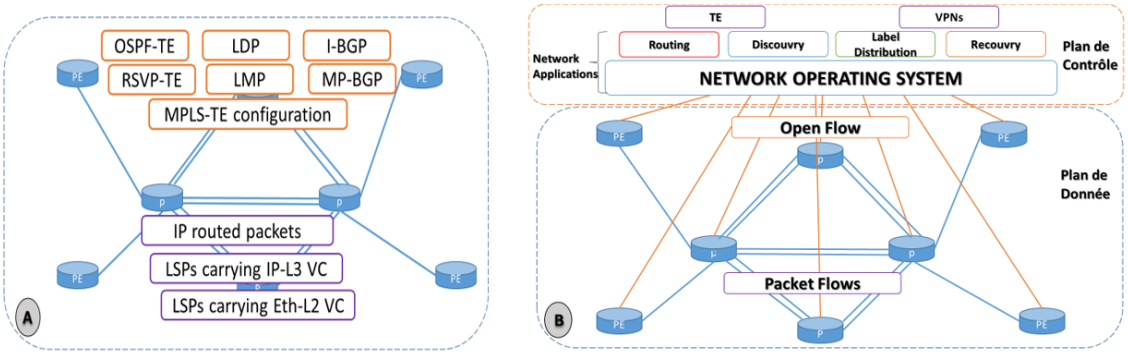

FIGURE 2.3 MPLS et SDN

2.4.2 Virtualisation réseau et SDN

Il n'y a peu de temps la virtualisation s'axée principalement sur le stockage et les centres de données mais vu la nécessité croissante en mobilité et l'augmentation de la taille des données cela a vite poussé à ce que le réseau lui-même soit virtualisé pour plus d'agilité et réponde au besoin en tout genre.

Lorsqu'un réseau est virtualisé, on assiste à une abstraction de ses composants physiques, c'est-à-dire que cela consiste à combiner des ressources réseau matérielles et logicielles dans une seule unité administrative ayant pour objectif de fournir aux systèmes et utilisateurs un partage efficace, contrôlé et sécurisé des ressources réseau. Ce qui fait que les utilisateurs n'ont plus besoin de considérer le réseau en termes de routeurs, de commutateurs, voire de ports spécifiques. Au lieu de cela, le réseau physique est partagé par différents réseaux virtuels.

L'idée n'est guère nouvelle, les réseaux virtuels 802.1Q associés à des tunnel Q-in-Q en sont un exemple. MPLS est une autre technologie éprouvée qui est fréquemment utilisée pour obtenir ce type de virtualisation de réseau. Mais s'il s'agit de technologies maîtrisées et matures, Q-in-Q et MPLS sont plutôt des technologies de fournisseurs de services et d'opérateurs et elles sont rarement déployées dans des environnements de centre de données. Dans les paradigmes SDN, la virtualisation de réseau s'effectue plutôt en utilisant des technologies d'overlay ou le trafic est associé à un réseau virtuel spécifique est encapsulé

dans une enveloppe d'identification spécifique, ce qui l'isole des autres réseaux virtuels qui partagent le même réseau physique sous-jacent. Même si cela n'est pas forcément obligatoire, un contrôleur SDN peut être utilisé pour identifier tous les points de terminaison d'un réseau virtuel et pour indiquer aux commutateurs où et comment encapsuler le trafic à l'intérieur d'une couche d'overlay, afin ainsi de maximiser l'efficacité des communications entre les différents points du réseau.

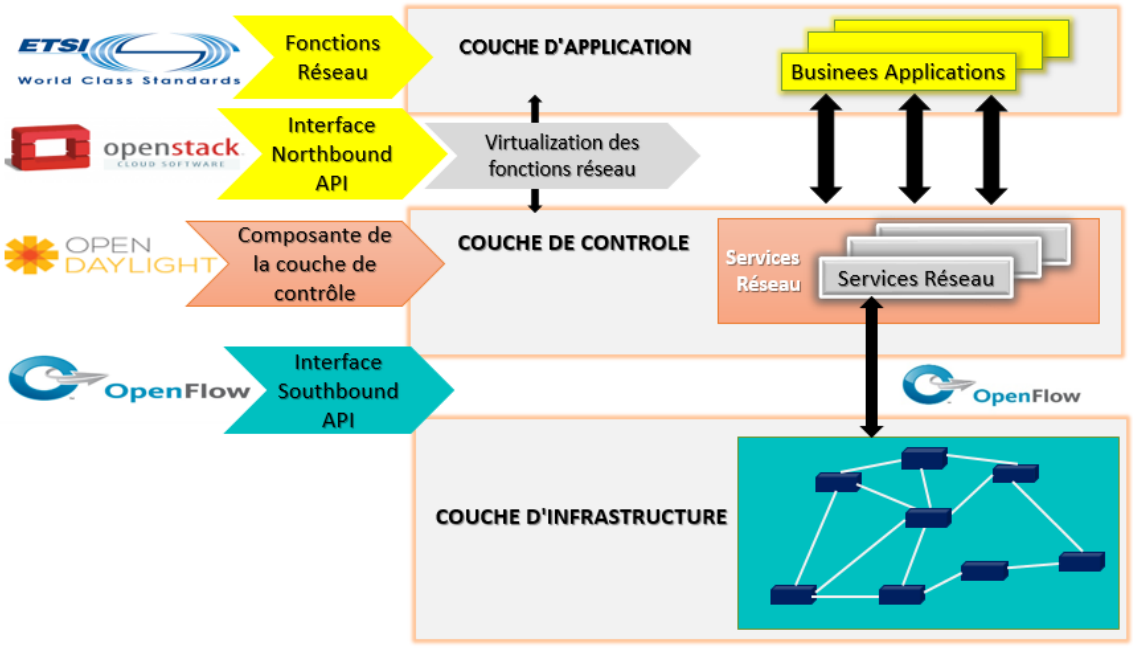

FIGURE 2.4 Virtualisation et SDN

Un concept qui est très souvent évoqué en parallèle à SDN est Network Functions Virtualization, Virtualisation des fonctions réseau (NFV).Alors que la technologie SDN a été créée par des chercheurs et des concepteurs de datacenter, la technologie NFV a été initialement créée par divers grands fournisseurs de services pour passer d'une infrastructure orientée matériel à une infrastructure orientée logiciel. Une fois les fonctions réseau sous le contrôle d'un hyperviseur, les services qui nécessitaient jusqu'alors un matériel dédié peuvent désormais être fournis à partir de serveurs x86 standard. Les fonctions logicielles accélèrent le déploiement de nouveaux services réseau et stimulent la croissance du chiffre d'affaires tout en réduisant les coûts opérationnels.
En résumé, la technologie SDN désigne une nouvelle façon de manipuler le réseau et la technologie NFV un nouveau type d'infrastructure à manipuler. En ce qui concerne la technologie SDN, il s'agit de contrôler le matériel réseau, alors qu'en ce qui concerne la technologie NFV, il s'agit de convertir du matériel spécialisé traditionnel en logiciel sur des

serveurs génériques.
NFV est une initiative fédérée à la base par les plus grands fournisseurs de services de télécommunication du monde avec l'European Telecommunications Standards Institute (ETSI) et qui a généré un grand intérêt chez les industriels. L'initiative a été créée pour aborder les principaux défis opérationnels et les coûts de gestion des applications réseau propriétaires et fermées qui sont actuellement déployées.
NFV pousse les technologies de virtualisation à consolider les applications réseau sur des serveurs, switches et baies de stockage physiques et standard de l'industrie dans une couche logique qui met à disposition des ressources réseau. Une ressource réseau virtualisée représente une Virtual Network Function, Fonction Réseau Virtuelle (VNF). Le principe des VNFs peut augmenter la flexibilité à partager ressources et réduire les coûts de configuration et de gestion. Un fournisseur de services peut mettre à disposition un ensemble d'applications et éléments d'infrastructure dans une plateforme qui fournit aux detenants les ressources dont ils ont besoin pour déployer leurs propres applications réseau adaptées à leur objectif professionel.
La virtualisation des fonctions réseau non seulement réduit les dépenses en équipements, mais aussi apporte d'autres bénéfices tels que l'extension agile d'applications, à une vitesse plus rapide, une plus haute disponibilité et une meilleure utilisation de ressources. Toutefois, pour bénéficier de ces apports, il est nécessaire que l'infrastructure réseau sous-jacente s'adapte rapidement et automatiquement. Par exemple, pour migrer une fonction réseau dans un nouveau matériel, les politiques et les configurations associées à ce service doivent être approvisionnées dans beaucoup d'autres équipements et fonctions. La complexité à configurer les réseaux dans un environnement dynamique augmente énormément avec l'introduction de nouveaux éléments réseaux, ce pourquoi des technologies supportant la programmation du réseau telles que SDN pourront habiliter la virtualisation des fonctions réseau. Bien que les développements de SDN et NFV puissent progresser indépendamment, l'association des deux principes est d'un fort intérêt pour le progrès des solutions cloud. SDN peut être employé en tant que technologie facilitatrice de la virtualisation des fonctions réseau, favorisant la consolidation des applications réseau dans des dispositifs industriels standard.

2.4.3 Évolutivité dans SDN

Malgré les nombreux avantages de SDN, puisque SDN présente un contrôleur centralisé pour le réseau, l'évolutivité est devenue un problème qui, à son tour, influe sur les performances du réseau. Dans les centres de données, la nécessité d'une performance élevée du contrôleur est cruciale. Par exemple, en supposant un petit centre de données avec 100K hôtes et 32 hôtes / rack, le débit d'arrivée maximum peut atteindre 300Mbits/s et le taux moyen est compris entre 1,5Mbits/s et 10Mbits/s [20]. En considérant les performances actuelles des contrôleurs OpenFlow, avec un débit moyen de 2Mbits/s , pour gérer le débit moyen entrant, il faut 1 à 5 contrôleurs, mais 150 pour le débit maximal. Dans les centres de données à grande échelle, le problème sera pire.
Pour résoudre ce problème, une approche consiste à améliorer le contrôleur lui-même, soit en mettant en œuvre des optimisations multi-thread plus avancées, soit en développant de nouveaux contrôleurs qui créent l'espace du noyau et éliminant ainsi la commutation entre l'utilisateur et l'espace du noyau qui consomme du temps supplémentaire [21]. Une autre approche consiste à classer les flux et les événements en fonction de leur durée et de leur priorité, où des flux de courte durée peuvent être gérés par les commutateurs et des flux de plus longue durée peuvent être envoyés à un contrôleur. Dans ce cas, la quantité de charge de traitement pour un contrôleur est réduite [22].
Outre les méthodes ci-dessus d'amélioration de la performance du contrôleur, l'utilisation d'équipes de contrôleurs distribués pour agir comme un seul plan de commande centralisé logique peut améliorer les performances de manière significative, comme illustré à la Figure 2.5. Le réseau dans cette approche est divisé en plusieurs segments qui pourraient être chevauchés et contrôlés par un certain contrôleur. Le cluster du contrôleur est connecté à un stockage de données distribué qui fournit tous les commutateurs et les informations des applications. Outre l'avantage de l'évolutivité du débit dans cette approche, la fiabilité est également gagnée si un seul contrôleur tombe en panne. Des applications telles que FlowVisor [23], HyperFlow [24] , Onix [25], sont des exemples de l'architecture du contrôleur distribué.

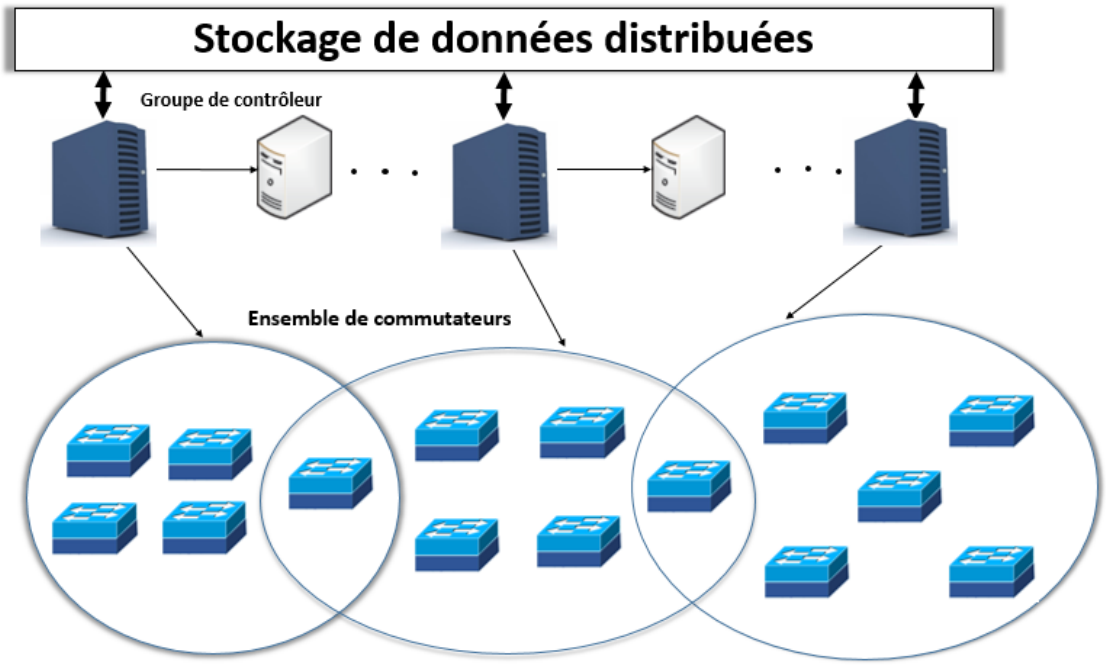

FIGURE 2.5 Controleur distribué

2.5 Mode de fonctionnement du contrôleur

Les contrôleurs de l'architecture SDN ont deux modes opérationnels, réactifs et proactifs. Dans l'approche réactive, les paquets de chaque nouveau flux venant au commutateur sont transmis au contrôleur pour décider comment gérer le flux. Cette approche prend un temps considérable lors de l'installation des règles. La quantité de latence peut être affectée par les ressources d'un contrôleur, leurs performances et la distance du contrôleur-commutateur. Dans l'approche proactive, des règles sont déjà installées dans les commutateurs ; Par conséquent, le nombre de paquets qui envoient au contrôleur est réduit. Dans cette approche, la performance devient meilleure et donc l'évolutivité. L'évaluation pour les deux approches a été réalisée dans [26], où une approche hybride a été présentée pour tirer profit des approches réactives et proactives. Dans ce mémoire, nous allons évalué le nombre de contrôleurs qui fonctionnent en mode réactif, car c'est le mode standard dans les contrôleurs OpenFlow.

Chapitre 3

Protocole Openflow

OpenFlow est la première interface de communication standard définie entre les couches de contrôle et de renvoi d'une architecture SDN. OpenFlow permet l'accès direct et la manipulation du plan d'acheminement des périphériques réseau tels que les commutateurs et les routeurs, physiques et virtuels[12].
Récemment, Openflow a progressé grâce aux transformations innovatrices qu'il a reçu et ainsi améliorer sa scalabilité, sa vitesse, sa fiabilité et sa sécurité. Au niveau physique, les périphériques réseau ont évolué en fournissant des liaisons de haute capacité, améliorant les puissances de calculs ainsi une multitude d'applications a émergé, offrant des outils pour inspecter les opérations facilement. Mais le réseau dans sa structure n'a pas vu beaucoup de changement depuis ses débuts.
Dans l'infrastructure existante, les tâches qui composent la fonctionnalité globale du réseau telles que le routage, la commutation ou les décisions d'accès au réseau sont prises en charge par des périphériques réseau provenant de différents fournisseurs, tous exécutant des microprogrammes différents. Cela ne fournit pas beaucoup d'espace pour de nouvelles idées de recherche telles que de nouveaux algorithmes de routage à tester à grande échelle. En outre, toute tentative d'idées expérimentales sur le réseau de production prioritaire critique peut aboutir à un échec du réseau à un moment donné, ce qui a conduit à une infrastructure de réseau statique et inflexible et n'a pas attiré d'innovations majeures dans cette direction [27].
OpenFlow est une approche pour résoudre ce problème. Permet aux opérateurs de réseau de mettre en œuvre et de contrôler les fonctionnalités qu'ils veulent dans le logiciel, plutôt que d'avoir à attendre qu'un fournisseur les mettes en plan dans leurs produits propriétaires. De plus, elle permet également aux fournisseurs de donner aux chercheurs l'accès à leurs équipements d'une manière unifiée sans ouvrir leurs produits, de sorte que les chercheurs peuvent effectuer des expériences avec de nouveaux protocoles dans un réseau réel sans

affecter le trafic de production.
OpenFlow utilise des tableaux de flux qui sont similaires aux tables de recherche dans les commutateurs et les routeurs Ethernet modernes. Ces tableaux peuvent implémenter des firewalls, NAT, QoS ou pour collecter des statistiques necessaire a la gestion du réseau sans se préoccuper du fournisseur avec lequel il traite. Ces tableaux contiennent des règles de correspondance / action qui peuvent être créées et modifiées par un contrôleur centralisé. Le contrôleur offre un contrôle programmable des flux pour que l'administrateur du réseau définisse un itinéraire spécifique de la source à la destination en utilisant le traitement basé par flux d'expédition de paquets. Il réduit ainsi, la consommation d'énergie et les coûts de gestion du réseau en éliminant le traitement des paquets du routeur en définissant des chemins via un contrôleur centralisé.

3.1 Architecture Openfolw

L'architecture réseau OpenFlow se compose de trois concepts de base :

1. Commutateurs compatibles OpenFlow qui composent le plan de données.
2. Le plan de commande se compose d'un ou plusieurs contrôleurs OpenFlow.
3. Un canal de contrôle sécurisé relie les commutateurs au plan de commande.

Les commutateurs communiquent avec les hôtes et les uns avec les autres à l'aide du chemin de données que le logiciel peut fournir, et le contrôleur communique avec les commutateurs en utilisant le chemin de contrôle comme illustré à la figure 3.1.

La connexion entre le contrôleur OpenFlow et le commutateur est sécurisée au moyen de protocoles cryptographiques SSL ou TLS, où le commutateur et le contrôleur sont mutuellement authentifiés par l'échange de certificats signés par les deux côtés de la clé privée. Bien qu'il s'agisse d'un algorithme de sécurité très puissant, le contrôleur peut être vulnérable à une attaque par déni de service (DoS) ou à une attaque l'homme au milieu (Man in the middle) [1] ; Par conséquent, des pratiques de sécurité appropriées doivent être mises en place pour prévenir de telles attaques.

1. attaque qui a pour but d'intercepter les communications entre deux parties.

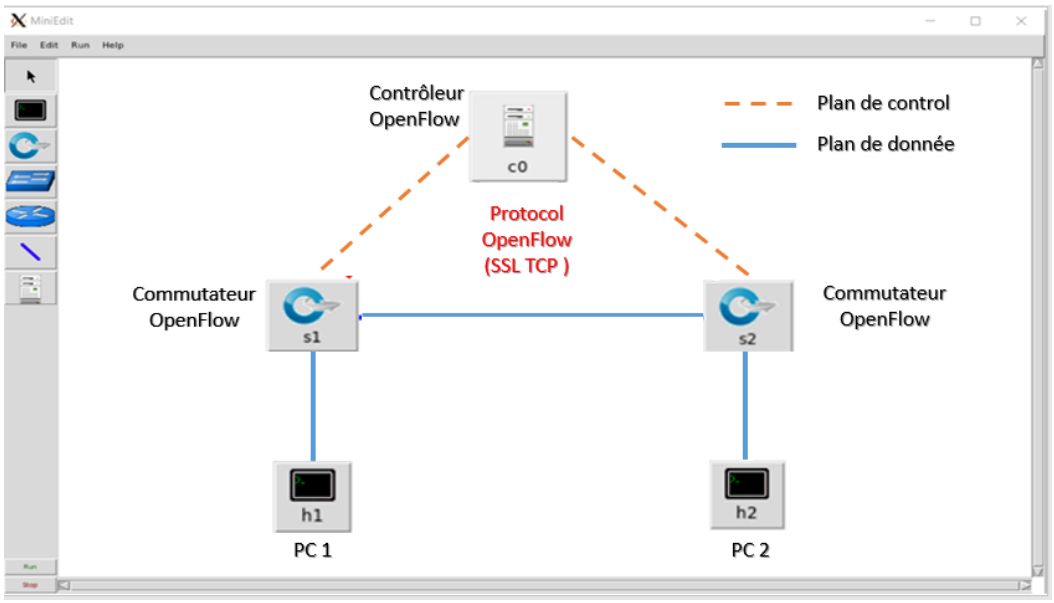

FIGURE 3.1 Architecture réseau Openflow

3.2 Commutateur OpenFlow

Les commutateurs Ethernet et les routeurs les plus modernes contiennent des tables de flux qui sont utilisées pour effectuer des fonctions de transfert selon les couches 2,3 et 4, indiquées dans les entêtes de paquets. Bien que chaque fournisseur ait des tables de flux différentes, il existe un ensemble commun de fonctions pour une large gamme de commutateurs et de routeurs. Cet ensemble commun de fonctions est mis à profit par OpenFlow, protocole entre un contrôleur central OpenFlow et un commutateur OpenFlow et qui, comme indiqué, peut être utilisé pour programmer la logique de transfert ou d'acheminement du commutateur.
La figure ci-dessus 3.2 décrit les fonctions réalisées par les équipements de réseau du plan de données (data plane) aussi appelés commutateurs au sens générique. Les principales fonctions des commutateurs sont les suivantes :

— Fonction de support du contrôle (Control support function) : Interagit avec la couche contrôle SDN afin de supporter la programmabilité via les interfaces ressource contrôle. Le commutateur communique avec le contrôleur et le contrôleur gère le commutateur avec le protocole OpenFlow. OpenFlow peut être utilisé aussi bien pour du contrôle que pour de la gestion.

— Fonction d'acheminement des données (Data forwarding function) : Accepte les flux de données entrants provenant d'autres équipements de réseau et des systèmes de terminaison et les relaie sur un chemin de commutation qui a été calculé et

établi à partir des règles définies par les applications SDN, passées au contrôleur et redescendues au commutateur.

Ces règles d'acheminement des données (data forarding rules) sont présentes dans les tables d'acheminement (forwarding tables). Ces règles indiquent pour des catégories de paquet données quel doit être le prochain saut sur la route. Le commutateur peut par ailleurs modifier l'en-tête du paquet avant son acheminement, ou rejeter le paquet. Comme montré à la figure 3.2, les paquets arrivant sont placés dans une file d'attente en entrée, attendant leur traitement par le commutateur ; les paquets acheminés sont placés dans une file d'attente en sortie, avant d'être transmis. Le commutateur à la figure 3.2 dispose de trois ports d'entrée/sortie : Un port fournissant la communication de contrôle avec un contrôleur SDN, et deux autres ports pour les entrées et sorties de paquets de données. Il s'agit d'un exemple simple. Le commutateur peut disposer de plusieurs ports pour communiquer avec plusieurs contrôleurs SDN, en plus de 2 ports pour les paquets de données entrant et sortant du commutateur.
Les flux de données consistent en des flux de paquets IP. Il peut être nécessaire pour la table d'acheminement (forwarding table) de définir des entrées sur la base de champs d 'entête de protocole de couche supérieure, tel que TCP, UDP, SCTP ou protocole d 'application. Le commutateur analyse l'en-tête IP et si nécessaire d'autre en-têtes dans chaque paquet et prend une décision pour son acheminement. L'autre flux de données important est via l'interface sud (southbound) consistant en le protocole OpenFlow ou tout protocole équivalent même si OpenFlow est le protocole de référence.
Un commutateur OpenFlow activé est le périphérique de transfert de base qui achemine les paquets en fonction de sa table de flux, qui est similaire aux tables d'acheminement traditionnelles mais qui n'est pas gérée et maintenue par le commutateur. Il est connecté au contrôleur via un canal sécurisé sur lequel des messages OpenFlow sont échangés entre le commutateur et le contrôleur, comme illustré à la figure 3.2.
Il existe différentes versions de spécifications de protocole OpenFlow disponibles sur OpenFlow Switch Specification ONF mais dans les sections suivantes, OpenFlow version 1.0 est expliquée car elle est considérée comme la base du protocole OpenFlow.

La figure 3.3 décrit les fonctions du commutateur OpenFlow V1.0 et sa relation au contrôleur. Comme attendu dans un commutateur de paquet, la fonction de base est de recevoir les paquets qui arrivent sur un port (Chemin X sur port 2 sur la figure 3.3) et les relayer via un autre port (Port N sur la figure 3.3) en réalisant toute modification nécessaire sur les paquets sur le chemin. La fonction de correspondance de paquet (packet-matching function) est très importante dans le commutateur OpenFlow. La table adjacente est une table de flux La flèche grisée sur le chemin commence dans la logique de décision, montre une

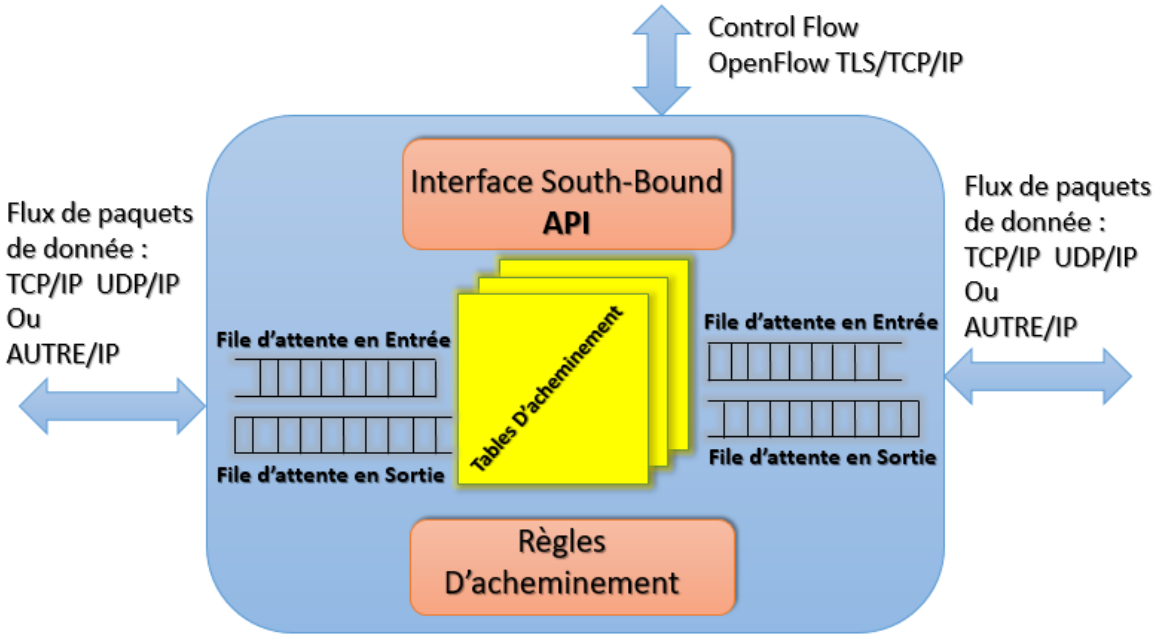

FIGURE 3.2 Commutateur Openflow

correspondance avec une entrée particulière dans la table de flux, et dirige le paquet pour lequel une correspondance a été trouvée à une case action sur la droite.
Cette case action a trois options de base pour le traitement du paquet arrivé :

A . Relayer le paquet sur un port de sortie, avec auparavant la possibilité de modifier certains champs d'en-tête du paquet.

B . Supprimer le paquet.

C .Passer le paquet au contrôleur. Le paquest est encapsulé dans un message OpenFlow PACKET_IN.

Ces trois chemins fondamentaux pour le paquet sont illustrés ci-dessus. Dans le cas du chemin C, le paquet est passé au contrôleur via un canal sécurisé montré à la figure 3.3. Si le contrôleur a soit un message de contrôle (e.g. mettre hors service un port du commutateur) ou un paquet de données à fournir au commutateur, le contrôleur utilise ce même canal sécurisé dans le sens inverse. Lorsque le contrôleur a un paquet de données à relayer via le commutateur, il utilise le message OpenFlow PACKET_OUT. Sur la figure 3.3, un paquet de données provenant du contrôleur peut suivre deux chemins, dénotés Y via la logique OpenFlow. Dans le cas du chemin Y de droite ; le contrôleur spécifie directement le port de sortie et le paquet est passé à ce port N. Dans le cas du chemin Y en bas, le contrôleur indique qu'il souhaite déléguer la décision de transfert du paquet à la logique de correspondance de paquets. Le contrôleur stipule alors le port de sortie TABLE pour le traitement du paquet

par la fonction de correspondance de paquets et l'identification par ce traitement du port de sortie.
Un commutateur OpenFlow peut être OpenFlow ou hybride. Dans ce dernier cas, le commutateur peut aussi commuter les paquets selon son mode traditionnel comme commutateur Ethernet ou routeur IP. Le cas hybride peut être vu comme un commutateur OpenFlow qui réside à côté d'un commutateur traditionnel et indépendant. Ce type de commutateur hybride requiert un mécanisme de classification qui soit dirige les paquets au contrôleur OpenFlow, soit les traite de manière traditionnelle.

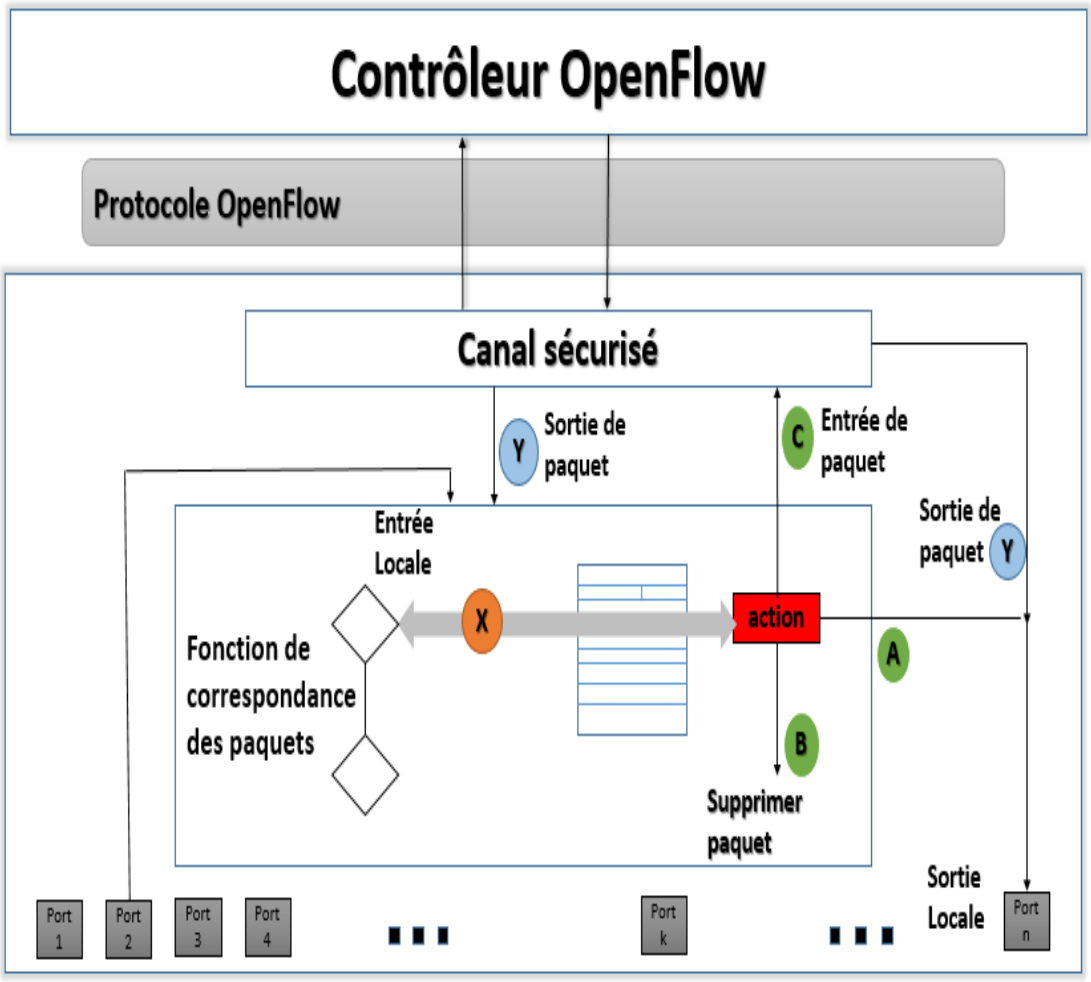

FIGURE 3.3 Fonctions du commutateur Openflow 1.0

3.2.1 Les tables de flux

Un commutateur dans le réseau OpenFlow comporte une (ou plusieurs) table (s) de flux qui contient un ensemble d'entrées, chacune constituée de champs de correspondance, d'action et de compteurs comme indiqué dans le tableau 3.1 :

TABLE 3.1 Table de flux pour Openflow v1.0

Champs de corréspondance	Actions	Conteurs

Tous les paquets traités par le commutateur sont comparés avec la table de flux. Si un en-tête de paquet correspond à une entrée de flux, une action pour cette entrée est effectuée sur le paquet (par exemple, l'action peut être de transférer un paquet sur un port spécifié). Si aucune correspondance n'est trouvée, le paquet est transmis au contrôleur via le canal sécurisé. Les compteurs sont réservés à la collecte de statistiques sur les flux. Ils stockent le nombre de paquets et octets reçus, ainsi que la durée du flux.

3.2.2 Champs de correspondance

Les Match fields pouvant être utilisés dans OpenFlow 1.0 sont :

- Ingress Port : port d'entrée sur lequel est reçu le paquet
- Ethernet source address (48bits) : adresse Ethernet source
- Ethernet destination address (48bits) : adresse Ethernet destination. Il est à noter que seule la couche Ethernet est considérée au niveau liaison de données avec OpenFlow 1.0.
- - Ethernet frame type (16bits) : Type de trame Ethernet : Ethernet II, LLC ou NSAP.
- VLAN id (12bits) : Identificateur de VLAN dans l'en-tête VLAN si présent.
- VLAN priority (3bits) : Priorité VLAN dans l'en-tête VLAN si présent.
- IPv4 source address (32bits) : adresse IPv4 source du paquet
- IPv4 destination address (32bits) : adresse IPv4 destination du paquet
- IP protocol (8bits) : Protocole contenu dans IP, e.g., OSPF, TCP, UDP, etc.
- IP ToS bits (6bits) : QoS spécifiée dans le paquet via le champ ToS (Type of Service)
- Transport source port /ICMP Type : Port source (couche transport)
- Transport destination port /ICMP code : Port destination

3.2.3 Compteurs

Les compteurs peuvent être maintenus pour chaque table de flux, entrée de flux, port, file d'attente. La liste ci-dessus décrit les compteurs.

Par table de flux

- Reference Count (active entries) : Nombre d'entrées actives dans la table.
- Packet Lookups : Nombre de paquets soumis à la table.
- Packet Matches : Nombre de paquets pour lesquels une des entrées de la table a pu s'appliquer.

Par entrée de flux

- Received Packets : Nombre de paquets reçus par l'entrée pour lesquels la recherche de correspondance a réussi.
- Received Bytes : Nombre d'octets de paquets reçus par l'entrée pour lesquels la recherche de correspondance a réussi.
- Duration (seconds) : Durée en secondes pendant laquelle l'entrée a été active.
- Duration (nanoseconds) : Durée en nanosecondes pendant laquelle l'entrée a été active au-delà de la durée en secondes.

Par port

- Received Packets : Nombre de paquets reçus sur ce port
- Transmitted Packets : Nombre de paquets transmis par ce port
- Received Bytes : Nombre d'octets reçus par ce port
- Transmitted Bytes : Nombre d'octets transmis par ce port.
- Receive Drops : Nombre de paquets reçus et rejetés par ce port
- Transmit Drops : Nombre de paquet rejetés en transmission
- Receive Errors : Nombre de paquets reçus en erreur
- Transmit Errors : Nombre de paquet transmis en erreur
- Receive Frame Alignment Errors : Nombre de paquet reçus avec erreur d'alignement
- Receive Overrun Errors : Nombre de paquets reçus et rejetés faute de mémoire dans les files d'attente en entrée du port
- Receive CRC Errors : Nombre de paquets reçus avec un CRC erroné
- Collisions : Nombre de paquets rejetés suite à une collision Par Queue (file d'attente)
- Transmit Packets : Nombre de paquets transmis sur cette file d'attente
- Transmit Bytes : Nombre d'octets transmis sur cette file d'attente
- Transmit Overrun Errors : Nombre de paquet transmis sur cette file d'attente et rejetés faute de mémoire sur la file. Lorsqu'un compteur arrive à sa valeur maximum, il repasse à 0 sans autre indication. Si un compteur n'est pas disponible, sa valeur doit être positionnée à -1.

3.2.4 Actions

Il existe deux types d'actions assurées par le commutateur Openflow des actions obligatoires et d'autres optionnelles.
Action (Obligatoire) : Forward. Les commutateurs OpenFlow doivent supporter l'acheminement de packet aux ports physiques ainsi qu'aux ports virtuels suivants :

- ALL : Est utilisé pour inonder un paquet sur tous les ports du commutateur à l'exception du port d'entrée ; ceci permet d'offrir une capacité broadcast rudimentaire au commutateur OpenFlow.

- CONTROLLER : Encapsuler le paquet et l'envoyer au contrôleur (Message PACKET_IN du commutateur au contrôleur). Cette action peut être indiquée dans l'entrée de flux. Aussi, si aucune entrée ne correspond au paquet à acheminer, le commutateur émet par défaut ce paquet au contrôleur.

- LOCAL : Envoyer le paquet à la CPU locale (contrôleur local dans le commutateur). Un commutateur dispose d'un contrôleur local vers lequel le paquet peut être envoyé pour la décision de son acheminement.

- TABLE : S'applique uniquement aux paquets que le contrôleur émet au commutateur. Ces paquets arrivent via le message PACKET_OUT du contrôleur, incluant la liste d 'actions. Cette liste d 'actions va généralement contenir une action de sortie, qui spécifie un numéro de port. Le contrôleur peut vouloir directement spécifier le port de sortie pour ce paquet de données, ou vouloir que le port soit déterminé par le traitement de paquet OpenFlow normal ; dans ce dernier cas, il stipule TABLE comme port de sortie.

- IN PORT Envoyer le paquet sur le port d'entrée. Cette action est nécessaire lorsque l'émetteur et le récepteur du paquet sont joignables via le même port du commutateur (e.g., émetteur et récepteurs présents sur le même hotspot WiFi).

-DROP :Une entrée de flux sans action indique que tous les paquets correspondants doivent être supprimés.

- Action(optionelle) **Forward** :Le commutateur peut en option supporter le port virtuel NORMAL qui revient à traiter le paquet selon la méthode de commutation traditionnelle. Le paquet doit donc être soumis à la logique d'acheminement traditionnelle du commutateur. Il faut distinguer le port virtuel NORMAL du port virtuel LOCAL, qui signifie que le paquet doit être passé au traitement du contrôle OpenFlow local. De même, les paquets pour lesquels l 'entrée correspondante indique NORMAL comme port de sortie doivent induire l 'interrogation des tables d'acheminement qui sont mises à jour par le plan de contrôle non OpenFlow local.

L'utilisation de NORMAL a du sens uniquement si le commutateur est hybride (commutateur à la fois OpenFlow et traditionnel).

- Action(optionelle) **Enqueue** : Cette action relaie un paquet via une file d'attente associée à un port. Le comportement d'acheminement est dicté par la configuration de la file d'attente et permet de fournir un support pour une qualité de service.
- Action(optionelle) **Modify Field** :Les actions indiquées ci-dessous apportent de la valeur à une implantation OpenFlow ; elles permettent de modifier des champs d'en-tête du paquet avant son acheminement sur un port de sortie, en particulier, les en-têtes VLAN, les adresses Ethernet source et destination, les adresses Pv4 source et destination, et les numéros de port TCP ou UDP.

- Set VLAN ID (12 bits) : Si aucun en-tête VLAN n'est présent, un nouvel en-tête est ajouté avec le VLAN ID et une priorité à 0. Si un en-tête VLAN est présent, le VLAN ID est remplacé avec la valeur spécifiée.

- Set VLAN priority (3 bits) : Si aucun en-tête VLAN n 'est présent, un nouvel en-tête est ajouté avec la priorité spécifiée et un VLAN ID positionné à 0. Si un en-tête VLAN est déjà présent, le champ priorité est remplacé par la priorité spécifiée.

- Strip VLAN header : Supprimer l 'en-tête VLAN si présent.

- Modify Ethernet source MAC address (48 bits) : Remplace l 'adresse MAC source Ethernet avec la nouvelle valeur.

- Modify IPv4 source address (32 bits) : Remplace l 'adresse source IPv4 avec la nouvelle valeur et recalculer le checksum IP.

- Modify IPv4 destination address (32 bits) : Remplace l 'adresse destination IPv4 avec la nouvelle valeur et recalculer le checksum IP.

- Modify IPv4 ToS bits (6 bits) : Remplace le champs IPv4 ToS avec la nouvelle valeur.

- Modify transport source port (16 bits) : Remplace le numéro de port source TCP ou UDP avec la nouvelle valeur et recalculer le checksum TCP ou UDP.

- Modify transport destination (16 bits) : Remplace le numéro de port destination TCP ou UDP avec la nouvelle valeur et recalculer le checksum TCP ou UDP.

3.3 Messages OpenFlow

Les messages OpenFlow entre le contrôleur et le commutateur sont transmis via un canal sécurisé, implanté via une connexion TLS sur TCP. Le commutateur initie la connexion TLS lorsqu'il connaît l'adresse IP du contrôleur. Chaque message entre le commutateur et le

contrôleur commence avec l'en-tête OpenFlow. Cet en-tête spécifie le numéro de version OpenFlow, le type de message, la longueur de message, et l'identificateur de transaction du message. Il existe trois catégories de message : symmetrique, controller-switch et asynchrone [2].

3.3.1 Messages symétriques

Les messages symétriques peuvent être émis indifféremment par le contrôleur ou le commutateur sans avoir été sollicité par l'autre entité. Les messages **HELLO** sont échangés une fois que le canal sécurisé a été établi afin de déterminer le numéro de version OpenFlow le plus élevé supporté par le commutateur et le contrôleur ; Le protocole spécifie que la plus petite des deux versions doit être utilisée pour la communication contrôleur –commutateur sur le canal sécurisé.
Les messages **ECHO** sont utilisés par n'importe quelle entité (contrôleur, commutateur) pendant le fonctionnement du canal pour s'assurer que la connexion est toujours en vie et afin de mesurer la latence et le débit au courants de la connexion. Chaque message **ECHO_REQUEST** doit être acquitté par un message **ECHO_REPLY**.
Les messages **VENDOR** sont disponibles pour des améliorations et pour une expérimentation spécifique au vendeur.

3.3.2 Messages asynchrones

Les messages asynchrones sont émis par le commutateur au contrôleur sans que le commutateur ait été sollicité par le contrôleur. Le message **PACKET_IN** est utilisé par le commutateur pour passer les paquets de données au contrôleur pour leur prise en charge (lorsque par exemple aucune entrée de flux ne correspond au paquet entrant ou lorsque l'action de l 'entrée correspondante spécifie que le paquet doit être relayé au contrôleur). Le trafic du plan de contrôle (e.g., paquet de routage OSPF) sera généralement relayé au contrôleur via ce message **PACKET_IN**. Si le commutateur dispose d'une mémoire suffisante pour mémoriser les paquets qui sont envoyés au contrôleur, les messages **PACKET_IN** contiennent une partie de l'en-tête (par défaut 128 octets) et un buffer ID à utiliser par le contrôleur. Les commutateurs ne supportant pas de mémorisation interne (ou ne disposant plus de mémoire) émettent le paquet entier au contrôleur dans le message **PACKET_IN**.
Le commutateur peut informer le contrôleur qu'une entrée de flux a été supprimée de la table de flux via le message **FLOW_REMOVED**. Cela survient lorsqu'aucun paquet entrant n'a de correspondance avec cette entrée pendant un temporisateur spécifié par le contrôleur lors de la création de cette entrée au niveau de la table de flux du commutateur.

Le message **PORT_STATUS** est utilisé afin de communiquer un changement de configuration du port (port désactivé par un usager) ou changement d'état du port (le lien est hors service). Finalement le commutateur utilise le message ERROR pour notifier des erreurs au contrôleur, e.g., ajout d'une entrée de flux par le contrôleur contenant des actions non supportées par le commutateur.

3.3.3 Messages contrôleur-commutateur

Les messages contrôleur-commutateur représentent la catégorie la plus importante de messages OpenFlow. Ils peuvent être représentés en cinq sous-catégories :*switch configuration, command from controller, statistics, queue configuration, et barrier*.

1. Les messages *Switch configuration* consistent en un message unidirectionnel et deux paires de messages requête-réponse. Le contrôleur émet le message unidirectionnel **SET_CONFIG** afin de positionner les paramètres de configuration du commutateur. Le contrôleur utilise la paire de message **FEATURES_REQUEST** et **FEATURES_REPLY** afin d'interroger le commutateur au sujet des fonctionnalités (notamment optionnelles) qu'il supporte. La paire de message **GET_CONFIG_REQUEST** et **GET_CONFIG_REPLY** est utilisée afin d'obtenir la configuration du commutateur.
2. Les messages *command from controller* sont au nomre de 3. **PACKET_OUT** est analogue à **PACKET_IN** mentionné précédemment. Le contrôleur utilise **PACKET_OUT** afin d'émettre des paquets de données au commutateur pour leur acheminement via le plan usager (Plan de données).
   Le contrôleur modifie les entrées de flux existantes dans le commutateur via le message **FLOW_MOD**.
   PORT_MOD est utilisé pour modifier l'état d'un port OpenFlow.
3. Des statistiques sont obtenues du commutateur par le contrôleur via la paire de message **STATS_REQUEST** et **STATS_REPLY** La configuration de files d'attente associées à un port n'est pas spécifiée par OpenFlow. Un autre protocole de configuration doit être utilisé pour ce faire.
4. Toutefois le contrôleur peut interroger le commutateur via **QUEUE_GET_CONFIG_REQUEST** acquitté par **QUEUE_GET_CONFIG_REPLY** pour apprendre quelle est la configuration des files d'attente associées à un port afin de pourvoir acheminer des paquets sur des file d'attente spécifiques et ainsi fournir à ces paquets un niveau de QoS désiré.

5. Le message **BARRIER_REQUEST** est utilisé par le contrôleur pour s 'assurer que tous les messages OpenFlow émis par le contrôleur et qui ont précédé cette requête ont été reçus et traités par le commutateur. La confirmation est retournée par le commutateur via la réponse **BARRRIER_REPLY**.

3.4 Démonstration des messages échangés dans le réseau OpenFlow

Afin d'illustrer les messages expliqués précédemment dans le réseau OpenFlow réel, nous avons utilisé l'émulateur de réseau Mininet [28] pour émuler deux hôtes connectés à un commutateur et un contrôleur, voir la figure 3.4 ci-dessous. Pour cette démonstration, nous expliquons d'abord l'établissement de connexion Switch-Controller, puis la communication hôte-hôte via le commutateur OpenFlow et le contrôleur.

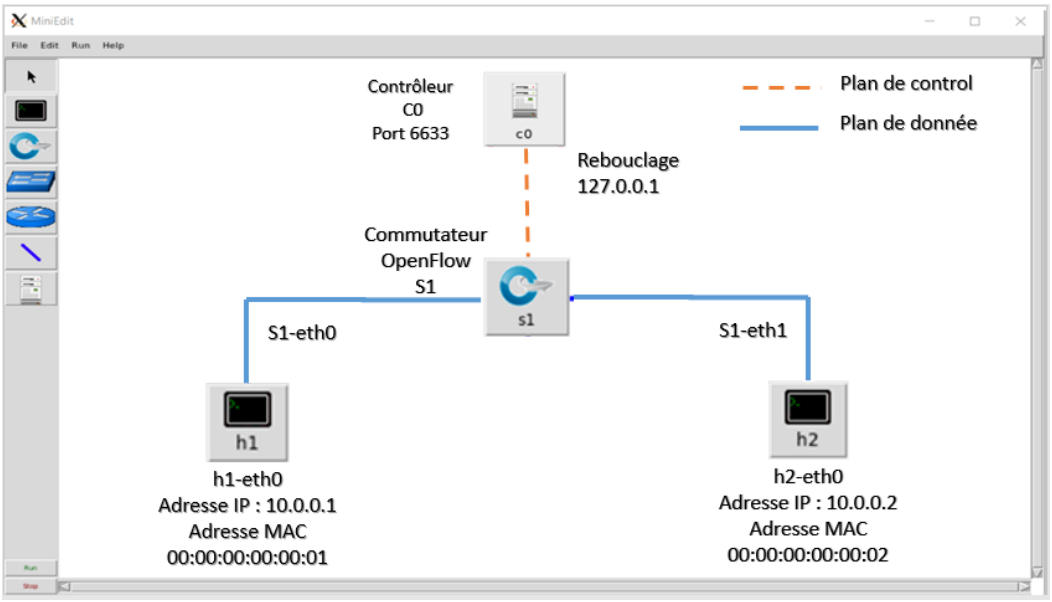

FIGURE 3.4 topologie de réseau en utilisant Mininet

3.4.1 Établissement de message entre un commutateur et un contrôleur

Lorsqu'un commutateur se connecte à un réseau OpenFlow, il établit une poignée de main (three-way handshake) TCP avec l'adresse IP du contrôleur (Loopback interface 127.0.0.1) et un port par défaut le 6633. Suite à ce processus, les deux parties commencent à échanger

des messages Hello incluant la version OpenFlow la plus élevée pris en charge. Après cela, le message de demande de fonctionnalité **FEATURES_REQUEST** est envoyé par le contrôleur pour voir quels ports sont disponibles dans le commutateur, qui à son tour répond avec un message de réponse de fonctionnalité **FEATURES_REPLY** qui contient une liste de ports, la vitesse des ports et les tables et actions prises en charge. Le message de configuration **SET_CONFIG** pour positionner les paramètres de configuration du commutateur.
Enfin, la demande d'écho, la réponse d'écho sont envoyées fréquemment entre le commutateur et le contrôleur pour échanger des informations relatives à la bande passante, la latence et la continuité de leur connexion. Les figures ci-dessous 3.5 illustrent la séquence de traitement et une capture de Wireshark pour les paquets, figure 3.5.

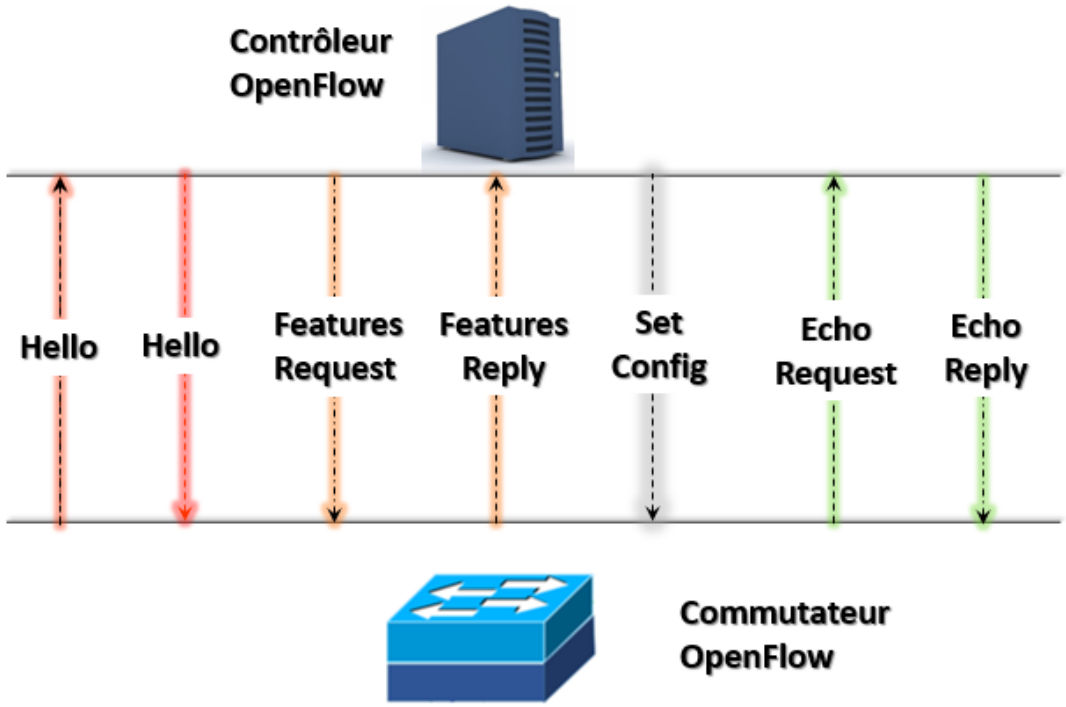

FIGURE 3.5 Messages de communications entre un commutateur Openflow et un contrôleur

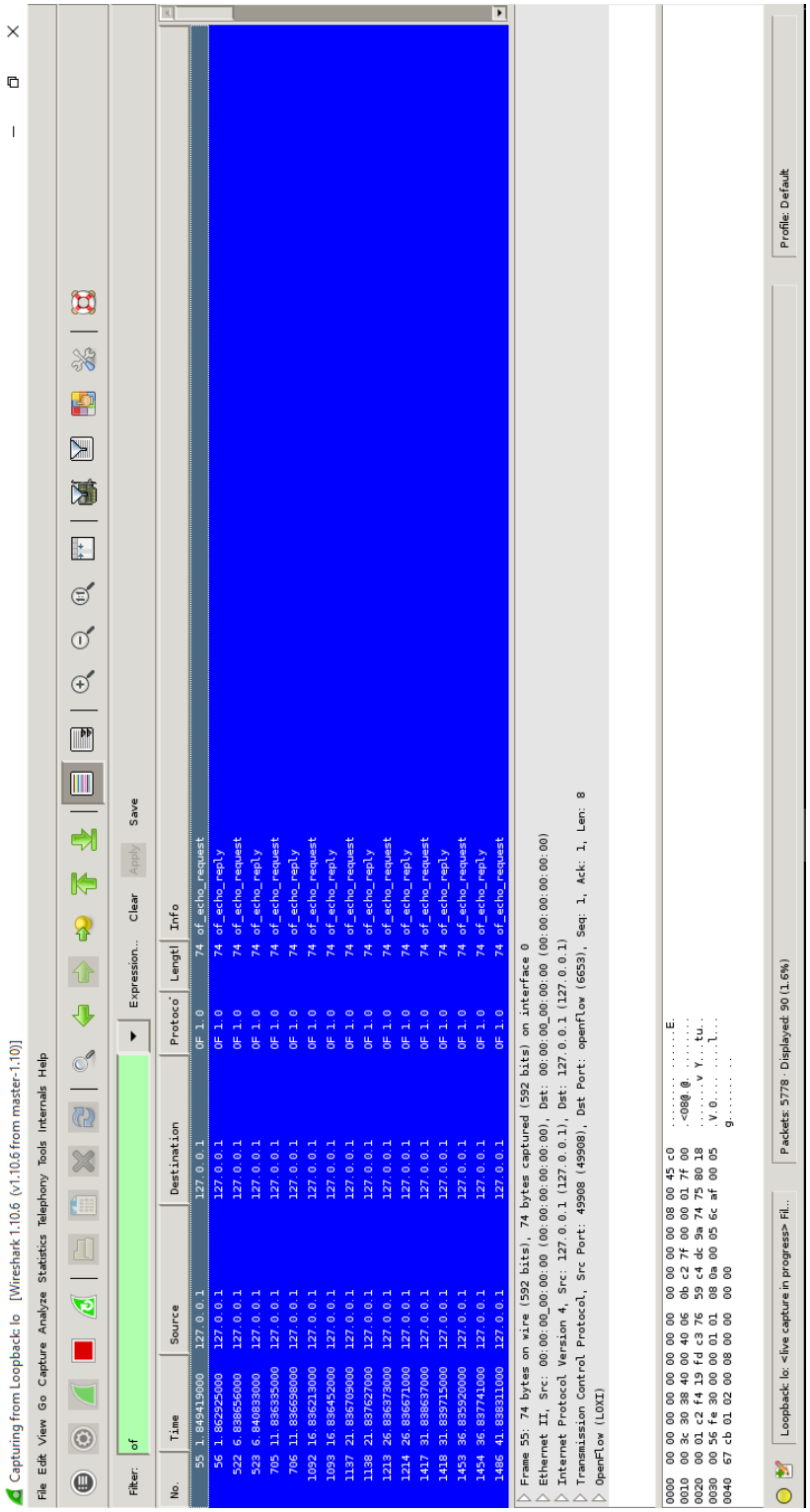

FIGURE 3.6 capture de Wireshark pour l'établissement de la connexion entre le Commutateur Openflow et le contrôleur

3.4.2 Messages échangés entre deux hôtes

Pour démontrer comment la connexion hôte à hôte est effectuée dans un réseau OpenFlow, nous avons utilisé l'outil Ping pour envoyer des paquets ICMP de h1 à h2 et vice versa. Le processus commence lorsque h1 envoie une demande ARP au commutateur, en demandant l'adresse MAC h2, le commutateur ne sait pas comment traiter le paquet, de sorte qu'il envoie le paquet en tant que message PACKET_IN au contrôleur. Le contrôleur répond avec un message PACKET_OUT qui a une action pour indiquer au commutateur d'envoyer le paquet à tous les ports, à l'exception du port entrant (dans notre cas, juste le port 2) et attendre la réponse à cette requête. Lorsque h2 répond pour cette requête, le commutateur envoie également cette réponse au contrôleur car il n'a aucune idée de l'endroit où transférer ce paquet. Lorsque le contrôleur reçoit la réponse ARP, il envoie un message FLOW-MOD pour installer une nouvelle entrée de flux pour les futures réponses ARP de h2 qui est éloigné de h1 pour être transmis directement par le commutateur sans avertir le contrôleur. Le même processus se produit lorsque h1 envoie la requête / réponse ICMP et lorsque h2 envoie une demande ARP pour demander l'adresse MAC h1 et la réponse ARP correspondante. À la fin, cinq nouvelles entrées de flux seront installées dans la table de flux du commutateur par le contrôleur OpenFlow, comme le montre les figures 3.7 et 3.8.

3.5 Les contrôleur Openflow

Le contrôleur est le noyau et la partie principale du système d'exploitation réseau (NOS) dans les réseaux SDN. Il est responsable de manipuler la table de flux du commutateur et des communications entre les applications et les périphériques réseau utilisant le protocole OpenFlow.
Les contrôleurs peuvent être classés en deux catégories principales [8] :

1. Contrôleur d'instance unique à source libre.
2. Contrôleur de distribution commercial à source fermée.

Les contrôleurs Open Source sont disponibles pour la recherche et le développement, donc il est représenté comme une instance de contrôleur unique avec la possibilité de développer diverses API sur leur plate-forme pour effectuer certaines tâches. Il existe de nombreux contrôleurs OpenFlow open source ; La principale distinction entre eux est le langage de programmation dans lequel ils sont écrits.
Voici une liste non exaustive pour le contrôleur Open Source en fonction de leur langage de programmation [29] :
C : Trema (aussi Ruby) et MUL[11].

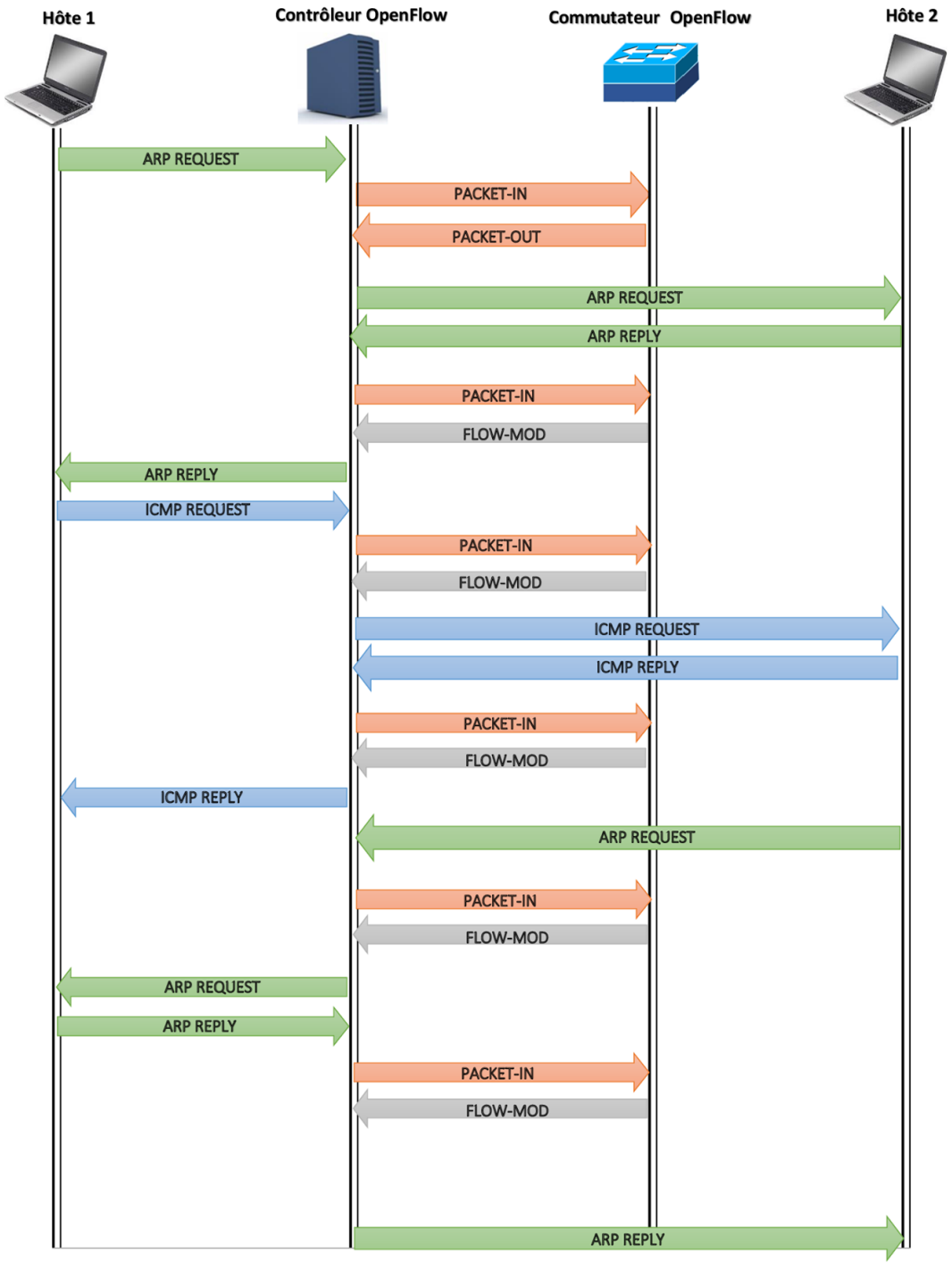

FIGURE 3.7 L'outil Ping entre deux hôtes h1 et h2

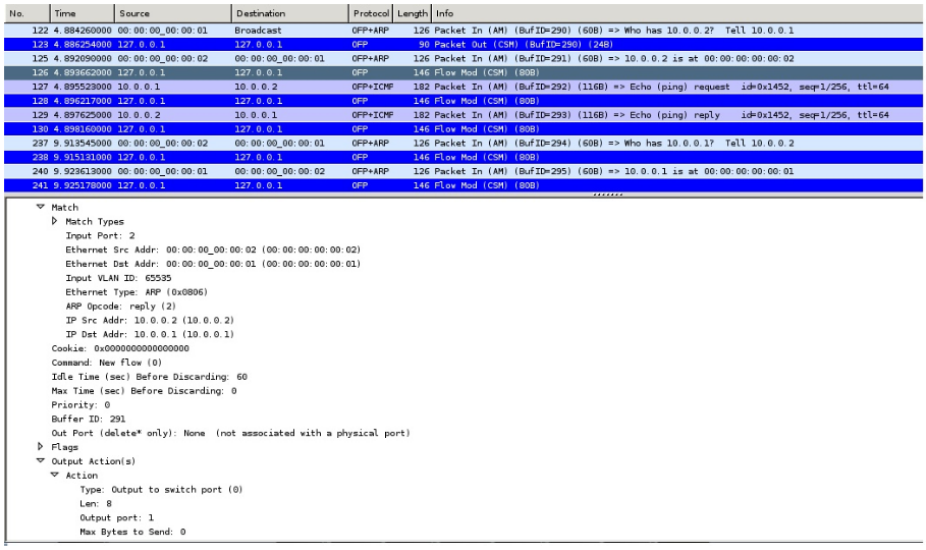

FIGURE 3.8 Paquets échangés entre h1 et h2

C ++ : NOX [4] (également Python).
Java : Beacon[5], Floodlight[9] , Open IRIS[30], Maestro[6] et OpenDaylight [31].
Python : POX [32], pyrétique [33] et RYU [10].

Les contrôleurs distribués sont capables de fonctionner et de contrôler le réseau via plusieurs instances de contrôleur. Avec une telle implémentation, les avantages sont d'avoir des couches d'abstraction supplémentaires pour le plan de contrôle et la tolérance aux pannes. Certains des exemples publics pour ces contrôleurs sont : Onix , de Nicira Networks, IRIS , par l'équipe de recherche d'ETRI, Big Network Controller de Big Switch Networks et Programmable Flow de NEC. Les contrôleurs Onix [25] IRIS[[34] ont la capacité supplémentaire de redimensionner les performances en ajoutant des contrôleurs supplémentaires dans le cluster du contrôleur.

Deuxième partie

Partie pratique

Chapitre 4

Methodologie et évaluation

4.1 Evaluation des contrôleurs OpenFlow

Dans ce chapitre, nous allons expliquer par quel moyen on peut evaluer un controleur Openflow et quelles sont les principaux parametres a prendre en compte. L'évaluation sera effectuée sur des commutateurs d'apprentissages. Les contrôleurs examinés lors de cette expérience sont Beacon et Floodlight, dans le but principal d'explorer quel contrôleur donnera les meilleurs performances.

4.2 Configuration de l'expérience

L'environnement de test a été effectué sur un logiciel d'émulation OpenFlow. Afin de surmonter les limitations de la vitesse de l'interface Ethernet, les deux contrôleurs et l'outil de benchmarking ont été installés sur le même hôte (CPU Intel Core i7-5500U 2.4Ghz / 3.0 GHz avec deux noyaux et 4 threads, RAM DDR3 de 8 Go), [35] s'éxécutant sous windows 10 professionnel 64bits. Au cours du test, nous avons organisé plusieurs instances en parallèles, l'outil de benchmarking (OFNet) ainsi que les contrôleurs qui utiliseront toutes les ressources disponibles.
Dans le test d'évaluation du contrôleur, nous allons tester chaque controleur sur douze paramétres qui seront afficher sur un tableau de bord qui comprendra plusieurs graphiques :

1. Le round-trip time (RTT) moyen par mili seconde. le le temps que met un signal pour parcourir l'ensemble d'un circuit fermé.
2. New flow generation rate/sec. correspondance au la quantité de flux génerées par rapport au temps(seconde).
3. Flow failure/sec. relative à la fiabilité de notre controleur.

4. Average flow setup latency. relatif a latence moyenne generer pour la configuration de flux.
5. OF msgs to controller/sec. le nombre de messages Openflow reçus par le controleur à un moment donnée t.
6. OF msgs from controller/sec. le nombre de messages Openflow emis par le controleur à un instant t en seconde.
7. CPU utilization (ovs-vswitch). l'utilisation du prosseceur par les commutateurs virtuels en %.
8. Flow misses to controller. les paquets qui ne sont pas arrivés au controleur par unité temporelle.
9. Flow_mods from controller/sec. Le nombre de Flow_mods envoyés par le controleur a chaque moment.
10. Max flow table entries. Le maximum d'entrées dans la table de flux a un moment donnée en sec.
11. Average flow table entries. La moyenne des entrées dans la table de flux.
12. Internet access latency. la latence d'accés a internet.

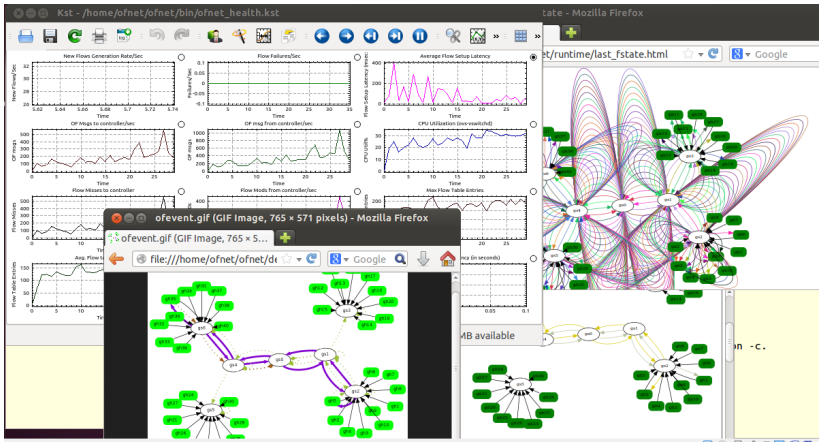

FIGURE 4.1 OFNet

Pour chaque contrôleur, nous avons exécuté le test avec septs Commutateurs connectés à 500 hotes le tout virtualisé ce qui aura pour effet d'exploiter considerablement les ressoureces du processeur. Nous avons choisi ce nombre aprés avoir effectué d'autres tests avec un plus grand nombre d'hotes mais cela a eu pour effet le plus souvent de pousser le PC à sa limite et planter, nous forçant à redémarrer l'application.

4.3 Méthodologie pour test de performances les contrôleurs Openflow

Nous allons effectuer une caractérisation des performances des controleurs Openflow émulés dans ce mémoire en prenant en compte les douzes fecteurs importants cité deja precedement dans 4.1
Ces douze graphes nous donnes aprés lecture des indication precieuse quant aux reactions du controleur testé, sa façon de gérer le flux, quelle politique ou quel mode utilise t-il, sa rapidité et sa fiabilité.

4.3.1 Contrôleur OpenFlow Paramètres d'analyse comparative

Pour la méthodologie d'analyse comparative des controleur OpenFlow, d'autres paramètres suivants ont été pris en considération :

- Nombre de commutateurs OpenFlow activés : le nombre de commutateurs OpenFlow qui établira une connexion avec le contrôleur.
- Nombre d'hôtes (Flux) : nombre d'hôtes connectés.
- Version OpenFlow : version de protocole que le contrôleur utilisera pour la configuration de la connexion, jusqu'à présent, cinq versions du protocole OpenFlow sont 1.1, 1.2, 1.3, 1.4 et 1.5.
- Durée de test : la durée du test, exprimée en mili secondes.

4.3.2 l'outil d'émulation de réseau SDN OFNet

OFNet : Est un outils qui fournit un environnement pour une émulation de réseau Openflow tout comme Mininet (defini dans A) et est livré sous Apache v2. Il fournit des fonctionnalités beaucoup plus intéressantes, comme le débugage visuel du trafic du plan de contrôle, la génération de trafic sur le réseau émulé et caractériser les performances d'un contrôleur SDN..

L'environement d'émulation OFNet permet d'aller au dela de la simple fourniture d'un réseau virtuel. il peut debuguer de maniere visuelle nous evitons ainsi certaine manoeuvre peinible comme faire une analyse des captures wireshark dans le cas ou on a un probléme dans la configuration des flux par exemple.

Grace a un generateur de trafic plus réaliste on peut obtenir plus de details concernant le comportemant du controleur et se poser des question comme : Quelles sont les performances caractéristiques des différents contrôleurs de mon choix ? Est-ce que cela répond à mes

besoins d'application ? Si votre application SDN est de nature réactive ? Combien de cycles CPU seront consommés sur votre commutateur OF ? Comment débuguer une défaillance de flux lorsque nous avons des milliers de flux actifs dans le réseau ?

Il existe de nombreuses raisons qui déclenchent l'utilisation d'un nouvel émulateur telque OFNet ; D'abord, il existe seulement quelques périphériques réseau disponibles pour la mise en œuvre de la norme SDN car ce n'est pas encore une technologie étendue du point de vue industriel. De plus, la mise en œuvre d'un réseau avec un grand nombre de périphériques réseau est très difficile et coûteuse. Par conséquent, pour surmonter ces problèmes, la stratégie en mode virtuel a été menée dans le but de prototypage et d'émulation de ce type de technologies de réseau. Il a donc la capacité d'imiter différents types d'éléments de réseau tels que ; Hôte, les commutateurs de couche 2, les routeurs de couche 3 et les liens. Il fonctionne sur un noyau Linux unique et il utilise la virtualisation dans le but d'émuler un réseau complet en utilisant un seul système. Cependant, l'hôte, les commutateurs, les routeurs et les liens créés sont des éléments du monde réel, bien qu'ils soient créés au moyen de logiciels.Donc OFNet offre une alternative à Mininet qui a souvent été le seul à offrir de telles prestations.

Afin d'exécuter nos images virtuelles nous avons utilisé le logiciel Virtualbox version 5.1.24 r117012, ou ne devons nous assuré que notre VM possède une interface réseau. elle doit être une interface NAT (eth0) qu'elle peut utiliser pour accéder à Internet,et est configurée en utilisant le DHCP. L'image virtuel de OFNet fonctionne sous Ubuntu 12.04 et posséde un fichier readme et un autre ficher démo qui nous montre un apérçu de ce dont est capable de faire OFNet.

nous avons commencé par installer certains package manquant et necessaire pour le bon fonctionnement de logiciel :
`sudo apt-get install openvswitch-switch` util pour virtualiser les commutateur openflow. `sudo apt-get install libpcap-dev`
`sudo apt-get install netperf`
`sudo apt-get install flex`
`sudo apt-get install bison`
`sudo apt-get install graphviz`
`sudo apt-get install kst`
`sudo apt-get install imagemagick`
`sudo apt-get install default-jre`(JAVAruntime).
installer wireshark :

```
sudo add-apt-repository ppa:wireshark-dev/stable
sudo apt-get update
sudo apt-get install wireshark
```

4.4 Création et exécution d'un scénario de simulation sur OFNet

Pour exécuter une émulation de réseau dans OFNet, On doit d'abord créer un fichier de topologie (fichier.topo) en utilisent un éditeur de texte. Ensuite, il faut le compiler pour obtenir ce qu'on appel un fichier réseau (fichier.net). On doit aprés démarrer et configurer le contrôleur SDN qui exécutera le scénario réseau. Enfin, OFNet utilisera le fichier réseau pour démarrer le réseau.Nous verrons ça en detail dans cette partie.

4.4.1 Création d'une topologie sur OFNet

On ne peut créer de topoligie sans avoir créer notre fichier.topo et pour cela OFNet utilise une notation très simple pour décrire la topologie en ce basant sur une grammaire simple mais qui prend également en charge des scénarios personnalisés complexes.

```
TOPOLOGY my_custom_topo {
     NODES SECTION :

     switches 7;
     hosts 40;

     LINKS SECTION :
     /*
        Lier les commutateurs.
        les noms des commutateur commence par gs0.
     */
     gs0 -- gs1;
     gs0 -- gs4;
     gs1 -- gs3;
     gs1 -- gs2;
```

```
    gs4 -- gs6;
    gs4 -- gs5;
    /*
       Relier les hotes aux commutateurs.
       Les noms des hotes commence par gh1.
    */
    (gh1-gh10) -- gs2;
    (gh11-gh20) -- gs3;
    (gh21-gh30) -- gs5;
    (gh31-gh40) -- gs6;

    CONTROLLER SECTION :

    contrl0(127.0.0.1);
    /* Relier les commutateurs au controleuur */

    ALL -- contrl0;
}
```

Comme on peut le voir, il y a 3 sections,

1. Section des noeuds

La section nœuds décrit le nombre de commutateurs et d'hôtes dans la topologie que L'on crée.Elle contient seulement 3 lignes :

"NODES SECTION :" indique le début de la section. Cela fait partie de la grammaire pour assurer la lisibilité du fichier de topologie.

"switches 7 ;" indique que l'on a 7 commutateurs dans le réseau. OFNet traite automatiquement les noms de commutateurs comme gs0 à gs6.

"hosts 40 ;" indique que l'on a 40 hôtes dans le réseau. OFNet traite automatiquement ces noms d'hôte comme gh1 à gs40.

2. Section des liens

Dans la section suivante, on peut écrire les liens entre les commutateurs et les liens entre les commutateurs et les hôtes. Une nouvelle section de liens commence par l'en-tête de section

"LINKS SECTION :"

Ensuite, nous décrivons les liens entre les commutateurs et les hôtes dans une notation simple. Si un commutateur gs0 est connecté à gs4, on dit simplement :
gs0 - gs4 ;

Cette syntaxe est dérivée de la spécification DOT pour décrire un graphique. De même, les hôtes sont connectés aux commutateurs.

gh1 - gs5 ; Indique que l'hôte gh1 est connecté au commutateur gs5. Plusieurs hôtes peuvent être configurés pour se connecter à un commutateur en donnant une portée :

(gh1-gh10) - gs5 ;

Cela signifie que les hôtes gh1 à gh10 sont connectés au commutateur gs5.
3. Section du contrôleur

Dans la dernière section, on décrit le contrôleur et la connectivité du contrôleur avec différents commutateurs et nœuds. Dans la première partie, on spécifie différents noeuds de contrôleur et leurs adresses IP.

nœud1 (192.168.1.51), nœud2 (192.168.1.50) ;

Cela décrit que l'on a deux noeuds avec les adresses IP 192.168.1.50 et 192.168.1.51 respectivement.

Dans la partie suivante, on décrit le passage à la connectivité du contrôleur. Si on utilise un seul nœud de contrôleur, il est très simple de décrire le même dans une seule ligne :

ALL - node1 ;

Cela signifie que TOUT ("ALL" est le mot-clé) les commutateurs sont connectés au noeud1 ayant l'adresse IP 192.168.1.51 dans ce cas. on peut également lier quelques interrupteurs sur node1 et rester dans le nœud2 :

(gs0, gs1) - noeud1 ; RESTANT - noeud2 ;

Ici "RESTANT" est le mot-clé. gs0 et gs1 sont connectés au noeud du contrôleur1 tandis que les commutateurs restants gs2 à gs6 (dans ce cas) sera connecté au noeud du contrôleur2.

qui peut être utilisé pour construire l'émulation de réseau OFNet

4.4.2 Compilation d'une topologie

Une fois que le fichier de topologie est créé, il doit être compilé sur une image réseau. OFNet utilise l'analyseur pour lire le fichier topologique et créer une image réseau qui poura etre contruite pour l'émulation de réseau. Le nom de la commande pour ce faire est "topoc." La syntaxe de topoc est simple :

```
topoc fichier.topo fichier.net
```

une fois la compilation éffectuée et le fichier.net créé, une image de notre topologie s'affichera pour nous permettre de faire une verification comme vous pouvez le voir sur la figure 4.2 ci-dessous

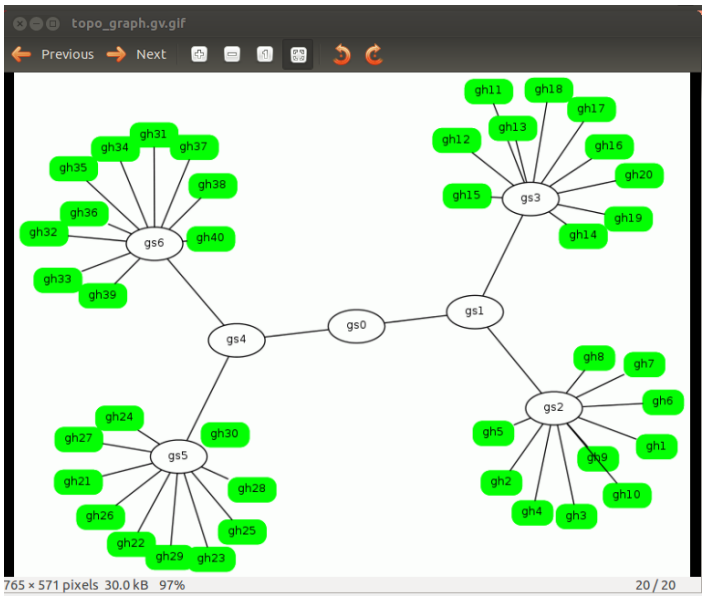

FIGURE 4.2 Image de la topologie

Afin d'exécuter notre fichier.net on utilise la commande "ctopo" ce qui émulera notre topologie et la connectera directement au controleur qu'on aura deja lancé au préalable. par exemble :
`ctopo fichier.net` pour lancer notre fichier.net.
`start_floodlight_controller.sh` pour lancer notre controleur.(nous verrons plus en detail l'installation et l'exécution de chaque controleur dans la suite de ce mémoire). il est interessant de voir aussi que OFnet nous permet de visualiséé le trafic de manière claire ou nous pouvons apprécier de façon distincte le deplacement des flux a travers notre topologie via des commande comme :
`hsh gh3 ping -c 1 10.0.0.30; fstate` qui nous donnera la figure 4.3 ou on peut voir le chemin empreinté par les paquets pour effectuer un ping entre l'hote h3 et l'hote ayant l'adresse IP 10.0.0.30 et l'adresse MAC 00 :00 :00 :00 :00 :30.

ou encore la commande `ofevent hsh gh31 ping -c 1 10.0.0.23` qui nous permettera d'avoir plusieur figures.
l'une représentra sur notre topologie chemins empreinté par les paquets du plan de controle (figure 4.4) . une autre représentera un diagramme de séquence pour tout les paquets du plan de controle echangés entre les commutateurs liés aux controleur(figure 4.5).

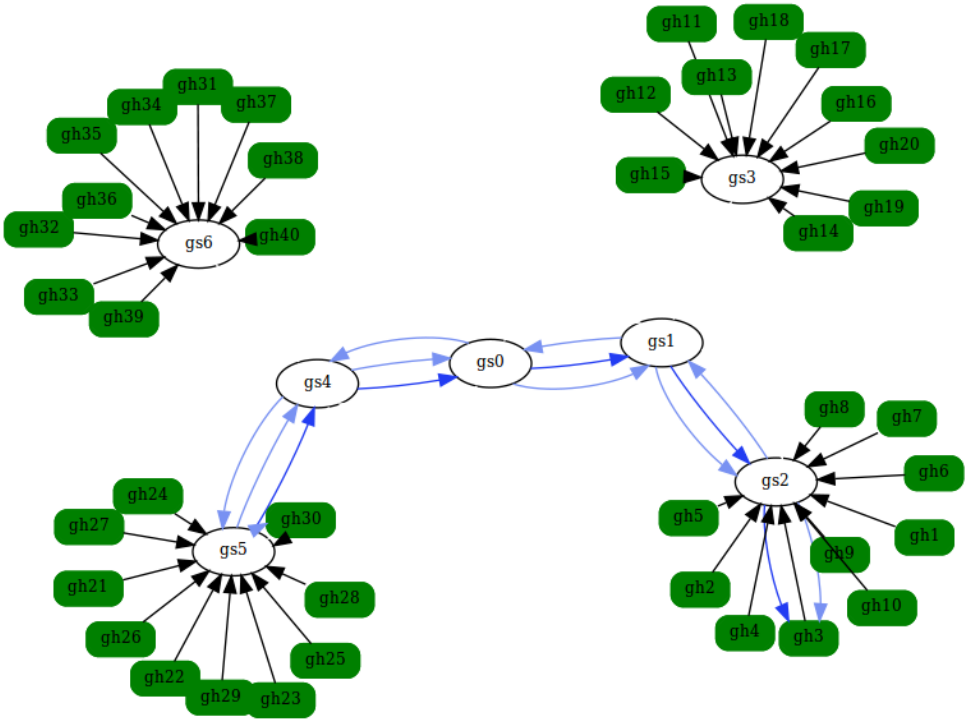

FIGURE 4.3 Aperçu des paquets echangés avec un ping grâce à la commande fstate

4.4.3 Le générateur de trafic

OFNet contient un générateur de trafic intégré qui peut générer différents types de trafic réseau et fournit des informations très utiles sur les caractéristiques de performance du contrôleur [36]. La figure 4.6 montre les graphiques réalisés en temps réel de diverses mesures utiles que OFNet fournit avec le générateur de trafic constituant ce qu'on appel tableau de bord (dashboard) qui nous permettera de faire une comparaison de ces caractéristiques de performance par rapport aux différents controleurs utilisés dans ce mémoire.
Pour cela on utilise les deux commandes "trafficup" et "trafficdown" qui éxécutent une série de commande "tctrl" pour démarrer le trafic sur notre réseau et afficher le tableau de bord relatif au controleur connecté comme vous pouvais le voir sur l'exemple de la figure 4.6.
L'axe des abscisses represente le temps et pour cette exemple equivaut à une periode de 310 seconde

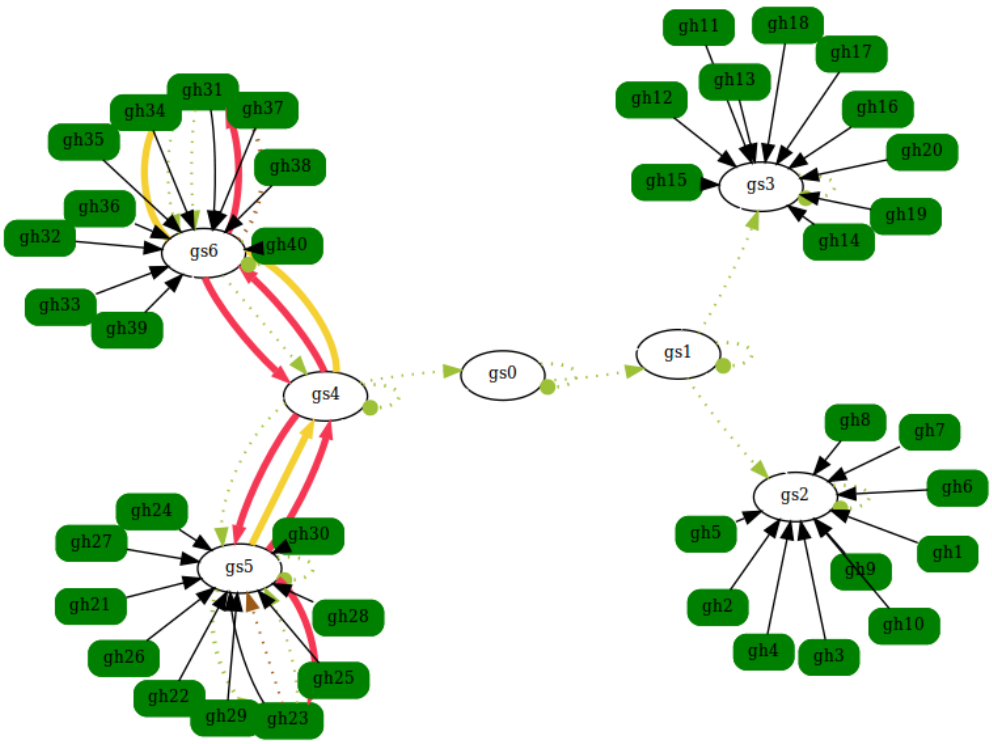

FIGURE 4.4 Aperçu des paquets du plan de controle echangés avec un ping grâce à la commande ofevent

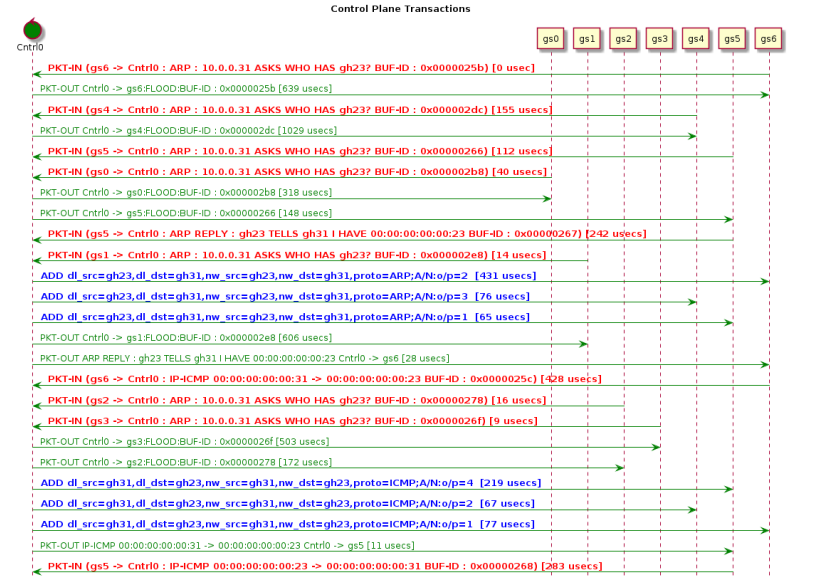

FIGURE 4.5 diagramme des séquences du plan de controle

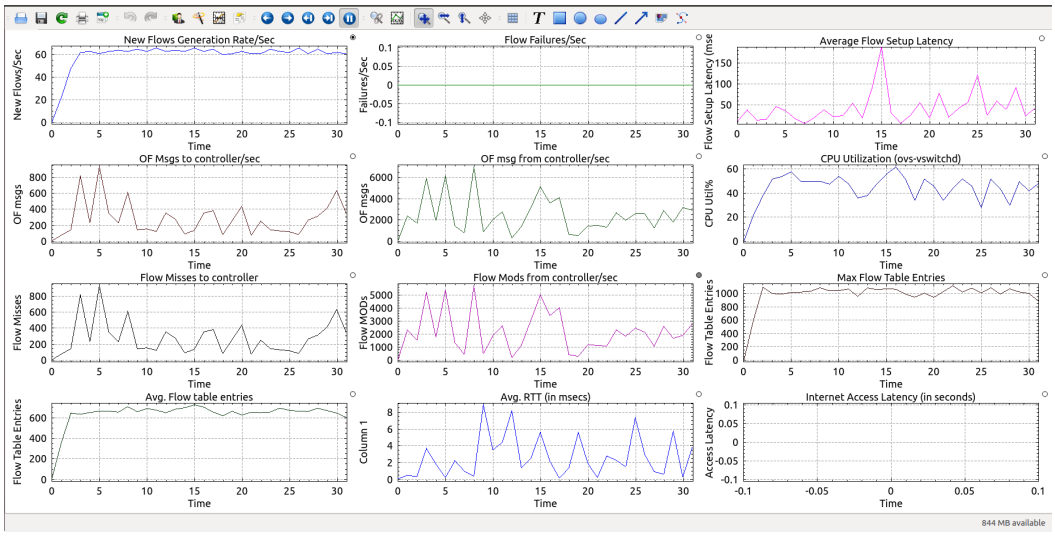

FIGURE 4.6 Tableau de bord du générateur de trafic

4.5 Contrôleur Openflow

La communauté OpenFlow a assisté au développement de nombreux contrôleurs écrits dans de multiples langages de programmation tels que C, C ++, Java, JavaScript, Ruby et Python.Il existe deja de nombreux articles publiés par les développeurs de ces contrôleurs OpenFlow individuellement, ce qui a poussé certains à faire des recherches afin d'effectuer des comparaisons d'évaluation entre ces contrôleurs en utilisant différentes méthodologie et environnement. Dans cette partie du mémoire, nous travaillerons sur l'implémentations de divers contrôleurs connues par leurs caracéristiques distinctes.

4.5.1 Contrôleur Openflow Beacon

Beacon est un contrôleur OpenFlow qui est développé à l'aide du langage de programmation Java qui est connu sous le nom de langage multiplate-forme ou Write-once-run-anywhere (WORA) afin qu'il puisse s'exécuter sur de nombreuses plates-formes depuis des serveurs multi-sites vers Android. Certaines des principales caractéristiques que le contrôleur Beacon possède sont [37] :

— Stabilité : il est en développement depuis plus de six ans ; Depuis début de 2010, il a également été utilisé dans plusieurs projets de recherche et déploiements d'essais.
— Est considérée comme un logiciel Open Source.
— Rapide : Beacon utilise le Multithread de ce fait il est considéré comme l'un des contrôleurs OpenFlow les plus rapides

— développement rapide : Beacon est facile à mettre en service. Java et Eclipse simplifient le développement et le débogage des applications.
— Dynamique : le code de Beacon peut être démarré / arrêté / rafraîchi / installé au moment de l'exécution, par exemple, il est possible de remplacer l'application Running Learning Switch sans déconnecter les commutateurs connectés.
— Beacon intègre en option le serveur Web d'enterprise Jetty et un cadre d'interface utilisateur extensible personnalisé.

Exécution et évaluation du contrôleur Beacon Le code source du controleur Beacon (version 1.0.2) se trouve deja dans la machine virtuel d'OFNet, mais nous avons préféré telecharger la derniere version 1.0.4 pour de meilleur performmance et eviter certains bugs lié a la version 1.0.2. pour le telechargement la commande suivante a été utilisé `git://gitosis.stanford.edu/beacon-1.0.4.git`
On exécutera les codes lui correspondant en tant que projet sur l'espace de travail d'Eclipse ce qui nécessitera des ressources CPU supplémentaires.

Selon les directives du guide d'analyse comparative OpenFlow Le Java 6 JRE / JDK basé sur Oracle est recommandé qui est deja fournit avec la VM d'OFNet. Pour l'installation de Oracle JDK, on utilise les commandes suivantes :

```
sudo apt-get install python-software-properties
sudo add-apt-repository ppa:webupd8team/java
sudo apt-get update
```

Ensuite, on exécute la commande suivante pour Oracle JDK 6 :

```
sudo apt-get install oracle-java6-installer
```

Après avoir téléchargé et installé tous les fichiers requis et Java il est temps d'exécuter le contrôleur,et cela en utilisant la commande suivant :

```
cd ~/beacon-1.0.4/ ./beacon
```

une fois le controleur connécté on peut accéder à son interface graphique GUI en entrant http ://127.0.0.1 :8080/ui/index.html dans une page de notre navigateur. comme vous pouvez le voir sur la figure 4.7 ci-dessous :

Nous utilisons l'interface d'OFNet pour montrer sur la figure 4.8 la topologie émuler pour nos tests.

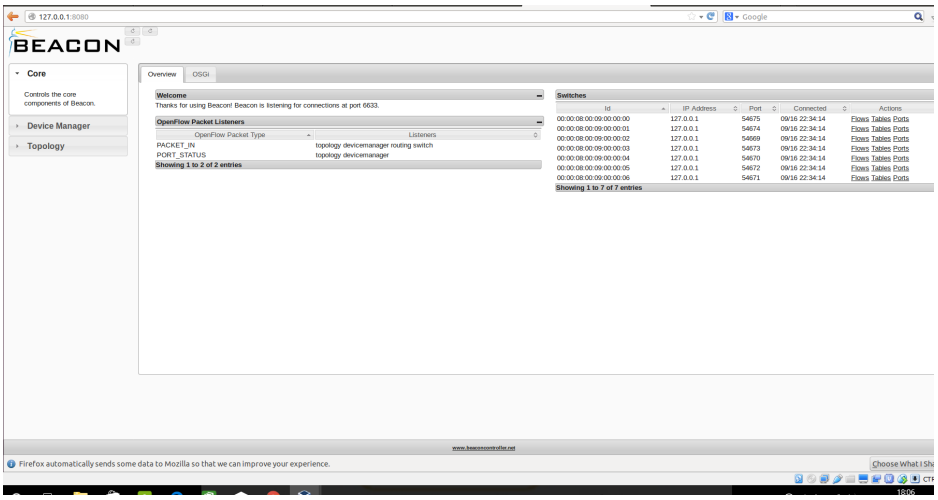

FIGURE 4.7 Interface graphique de Beacon

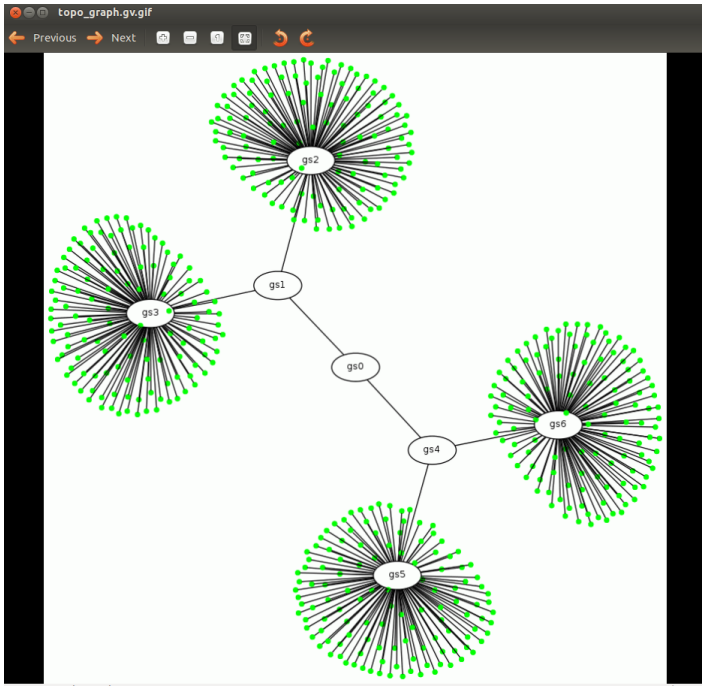

FIGURE 4.8 Topologie utilisée avec Beacon

Après l'installation et l'exécution du contrôleur, nous pouvons exécuter le generateur de trafic afin d'obtenir les caracterisque de Beacon.

Les résultats obtenus sont sur la figure 4.9 généré sur une periode de 1400 seconde : On constate que les valeurs concernant chaque graphe corréspondent à(nous avons pris en compte uniquement les paramétres qui varient et nous apportent des informations utiles a notre comparaison) :

— La quantité de flux génerées par rapport au temps : 60 flux/sec
— Fiabilité de notre controleur : nul ou trés minime.
— Les messages OF vers le controleur : 250 OF msgs/sec
— Les messages Of émis par le controleur : 1700 OF msgs/sec
— Utilisation du CPU des ovs-vswitch : 45%
— Paquets qui ne sont pas arrivés au controleur : 180/sec
— Flow_mods envoyés par le controleur : 1500 paquet/sec
— Le maximum d'entrées dans la table de flux : 1000 entrées/sec
— La moyenne des entrées dans la table de flux : 650 entrées/sec

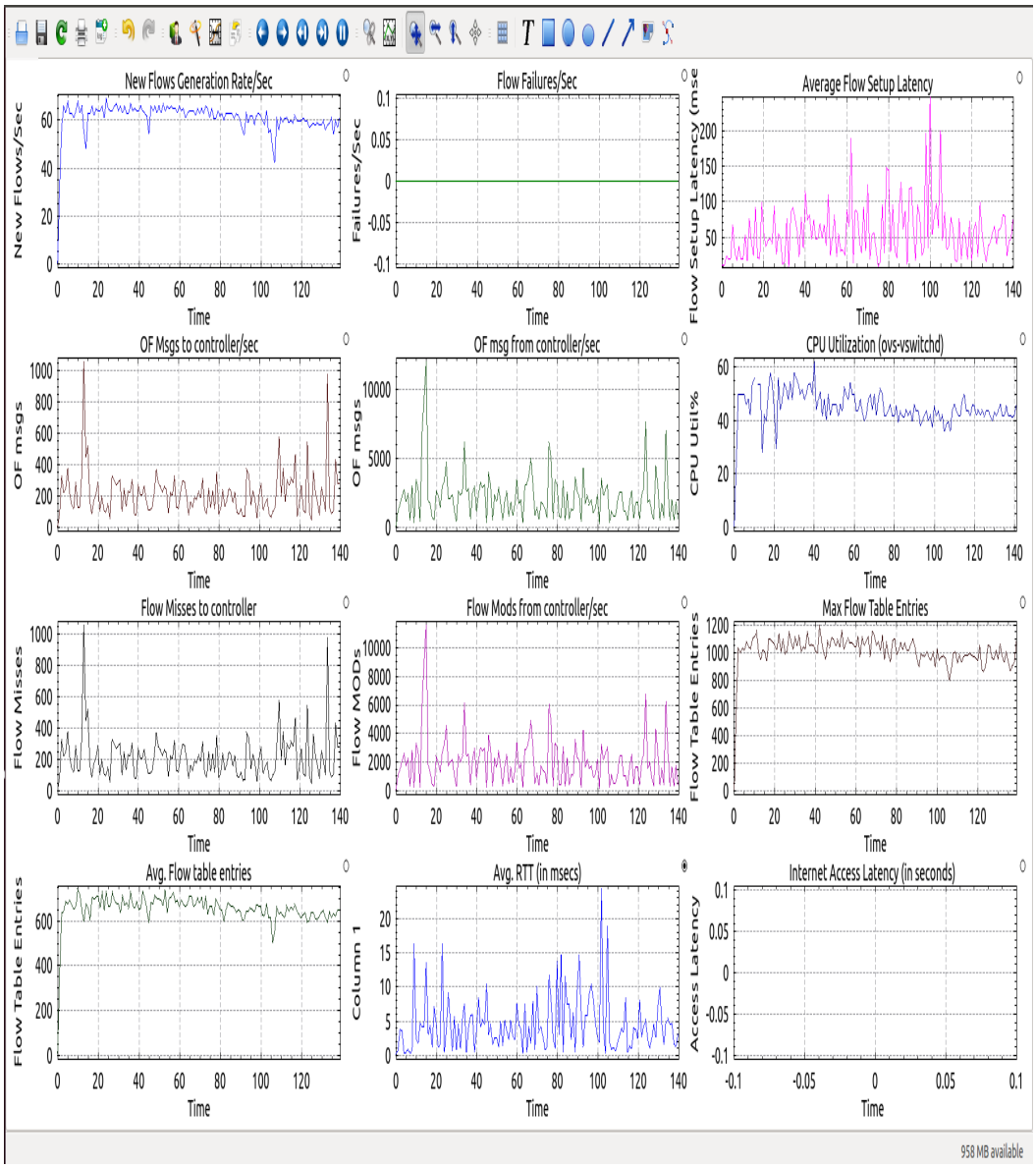

FIGURE 4.9 Tableau de bord des performances pour le controleur

4.5.2 Le contrôleur Floodlight

Le contrôleur SDN open source Floodlight est écrit en langage Java, multiplate-forme et est considéré comme un contrôleur OpenFlow d'entreprise. Il fournit les caractéristiques principales suivantes :

— Logiciel Open Source sous licence Apache, qui permet d'être utilisé à peu près pour n'importe quel objectif [9].

— Assis sur une solide base, car il est bien soutenu par les communautés de développeurs, y compris Big Switch [38].

— Floodlight est considéré comme assez simple à utiliser, à construire et à exécuter. C'est donc une option populaire pour les ingénieurs qui commencent à apprendre et à tester OpenFlow.
— Floodlight est considéré comme assez simple à utiliser, à construire et à exécuter. C'est donc une option populaire pour les ingénieurs qui commencent à apprendre et à tester OpenFlow[9].

Exécution et évaluation du contrôleur Floodlight Puisque Floodlight est écrit en Java et fonctionne ainsi dans une machine virtuele java (JVM), il existe aussi sur la VM d'OFNet mais on peut le telecharger en utilisant la commande suivante :

```
Sudo git clone git://github.com/floodlight/floodlight.git
```

Pour éxécuter floodlight on utilise la commande suivante :

```
start_floodlight_controller.sh
```

De la méme façon qu'avec Beacon on veut avoir accés a l'interface graphique de Floodlight qu'on peut voir sur la figure 4.10 avec la topologie de test qu'on a créé.

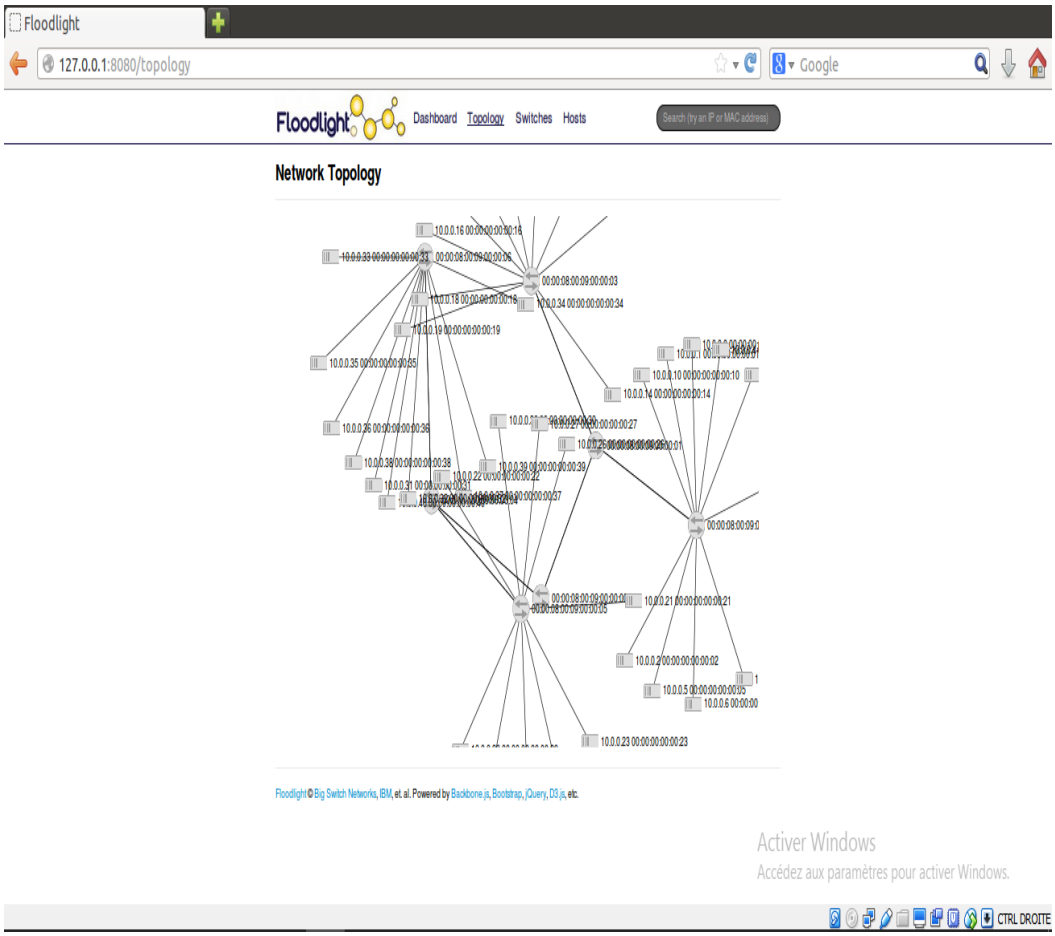

FIGURE 4.10 Interface graphique et topologie pour Floodlight

Après l'installation et l'exécution du contrôleur, nous pouvons exécuter le generateur de trafic afin d'obtenir les caracterisque de Floodlight.
Les résultats obtenus sont sur la figure 4.11 :

On constate que les valeurs concernant chaque graphe corréspondent à :

— La quantité de flux génerées par rapport au temps : 60 flux/sec
— Fiabilité de notre controleur : nul ou trés minime.
— Les messages OF vers le controleur : 200 OF msgs/sec
— Les messages OF émis par le controleur : 450 OF msgs/sec
— Utilisation du CPU des ovs-vswitch : 25%
— Paquets qui ne sont pas arrivés au controleur : 180/sec
— Flow_mods envoyés par le controleur : 180 paquet/sec
— Le maximum d'entrées dans la table de flux : 200 entrées/sec
— La moyenne des entrées dans la table de flux : 160 entrées/sec

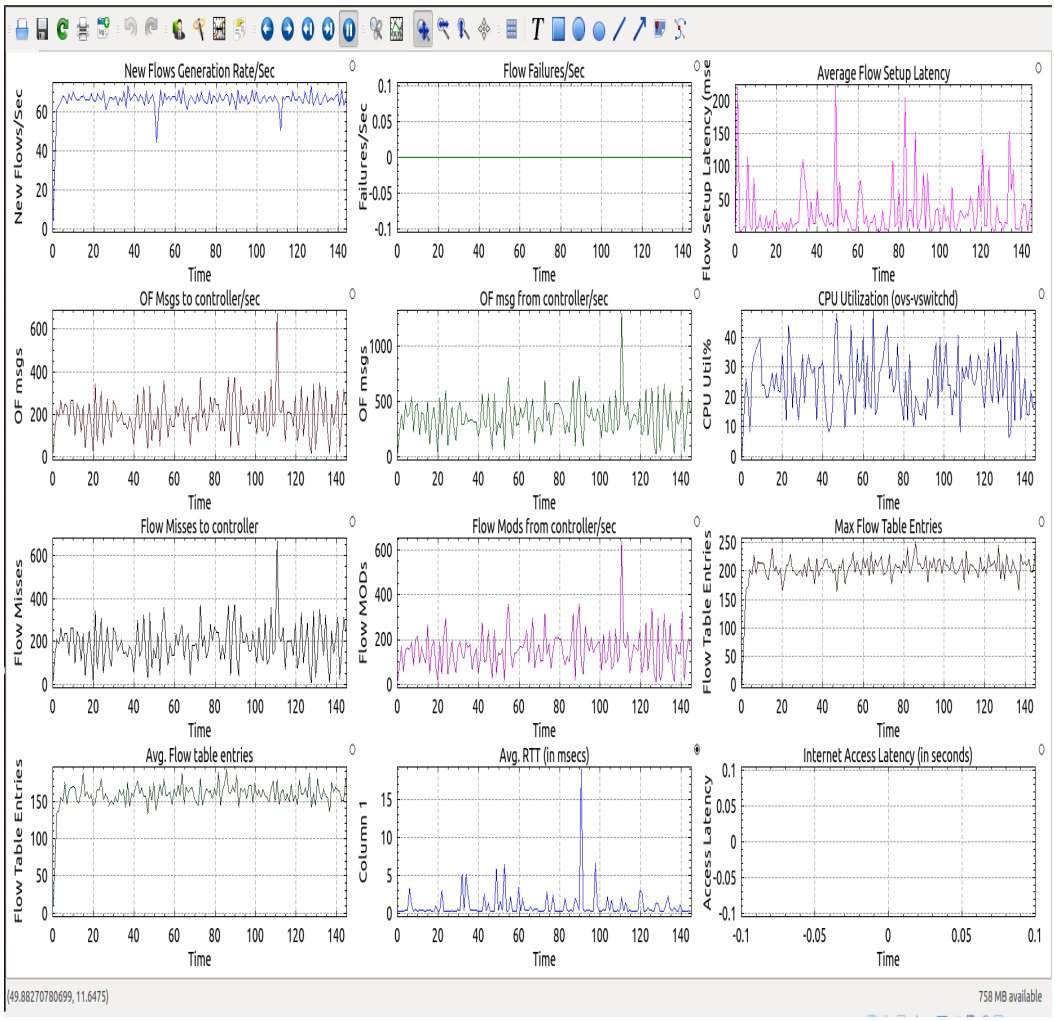

FIGURE 4.11 Tableau de bord des performances pour le controleur Floodlight

Comme le montre les figures 4.9 et 4.11, nous constatons qu'à partir des résultats de l'évaluation des contrôleurs OpenFlow , ou une juxtaposition des différents graphiques ce trouvant sur les tableaux de bord de nos contrôleurs testé que pour les graphes relatifs aux entrées de flux que le contrôleur Beacon a des valeurs plus élevé comparé a celle de floodlight. pour les graphes relatifs a l'utilisation du CPU des commutateurs virtuelles Floodlight a des valeurs plus basses que beacon. pour les graphes relatifs aux messages OF émis par le contrôleur Beacon obtient de plus grand résultats. pour les résultats relatifs Flow_mods émis par chaque contrôleur, c'est le contrôleur Beacon qui en emmet le plus comparé a Floodlight. et enfin pour les graphes relatifs la moyennes des messages d'entrée à la table de flux c'est encore Beacon qui a les plus grande valeurs.

4.6 Controleur Opendaylight

Hébergé par la Fondation Linux, OpenDaylight Project (ODL) est un projet SDN open source destiné à améliorer la création de réseau (SDN) par logiciel.
Le controleur OpenDaylight, renommé OpenDaylight Platform est soutenu par la communauté et supporté par l'industrie. Il est ouvert à tous, y compris les utilisateurs finaux et les clients, et offre une plate-forme partagée pour ceux qui ont des objectifs SDN afin d'oeuvrer ensemble pour trouver de nouvelles solutions.
Sous la fontadion Linux, OpenDaylight inclut un support pour le protocole OpenFlow, mais peut également supporter d'autres interface southbound SDN ouvertes.
Le contrôleur OpenDaylight peut se déployer dans une variété d'environnements de réseau de production. Il peut prendre en charge un contrôleur modulaire, mais peut fournir un support pour d'autres normes SDN et protocoles à venir.
Le contrôleur OpenDaylight est implémenté uniquement dans un logiciel et est conservé dans sa propre machine virtuelle Java (JVM). Cela signifie qu'il peut être déployé sur des plates-formes matérielles et système d'exploitation prenant en charge Java.

Exécution et évaluation du contrôleur Opendaylight la machine virtuelle d'OFNet pocéde la version 6 de java par defaut mais pour exécuter le controleur Opendaylight il nous faut la version 8 donc nous avons du l'intaller et cela en suivant les commandes suivante :

```
$ sudo apt-get install python-software-properties
$ sudo apt-add-repository ppa:webupd8team/java
$ sudo apt-get update
$ sudo apt-get install openjdk-8-jdk
$ echo "export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64" >> ~/.bashrc
$ source ~/.bashrc
```

Pour installer et lancer le controleur Opendaylight nous avons utilisé les commandes suivantes [39] :

```
$ cd Downloads

$ wget https://nexus.opendaylight.org/content/groups/public/org/
opendaylight/integration/distribution-karaf/
0.6.1-Carbon/distribution-karaf-0.6.1-Carbon.tar.gz

$ tar -xvf distribution-karaf-0.6.1-Carbon.tar.gz
```

```
$ cd ~/distribution-karaf-0.6.1-Carbon
```

```
$ ./bin/karaf
```

la figure 4.12 nous montre que Opendaylight a éfféctivement démarrer mais nous devons telecharger un certain nombre de fonctionalié pour le faire fonctionner en mode commutateur d'apprentissage et cela en utilisant les commandes suivantes :

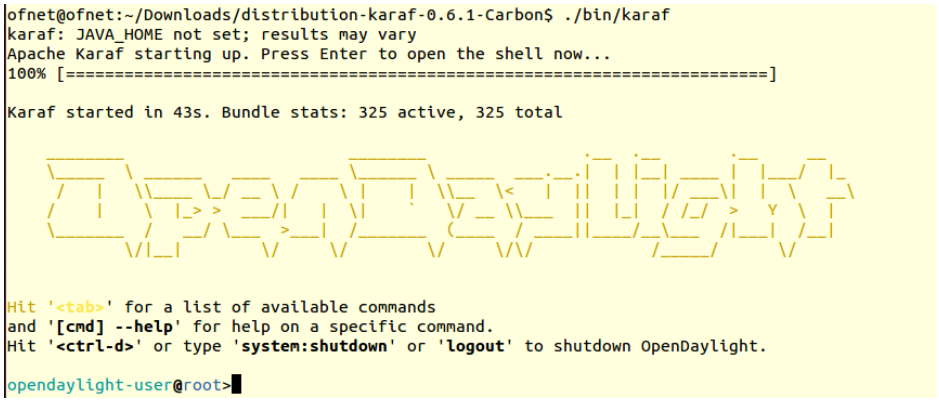

FIGURE 4.12 Connection du controleur ODL

```
opendaylight-user@root> feature:install odl-restconf odl-l2switch-
switch-ui odl-mdsal-apidocs odl-dlux-all
```

Nous avons par la suite lancé notre topologie de test que l'on peut voir sur l'intérface graphique d'Opendaylight figure 4.13 qui cette fois-ci s'exécute sur lo port 8181.

Après l'installation et l'exécution du contrôleur, nous pouvons exécuter le generateur de trafic afin d'obtenir les caracterisque de Opendaylight.
Les résultats obtenus sont sur la figure 4.14

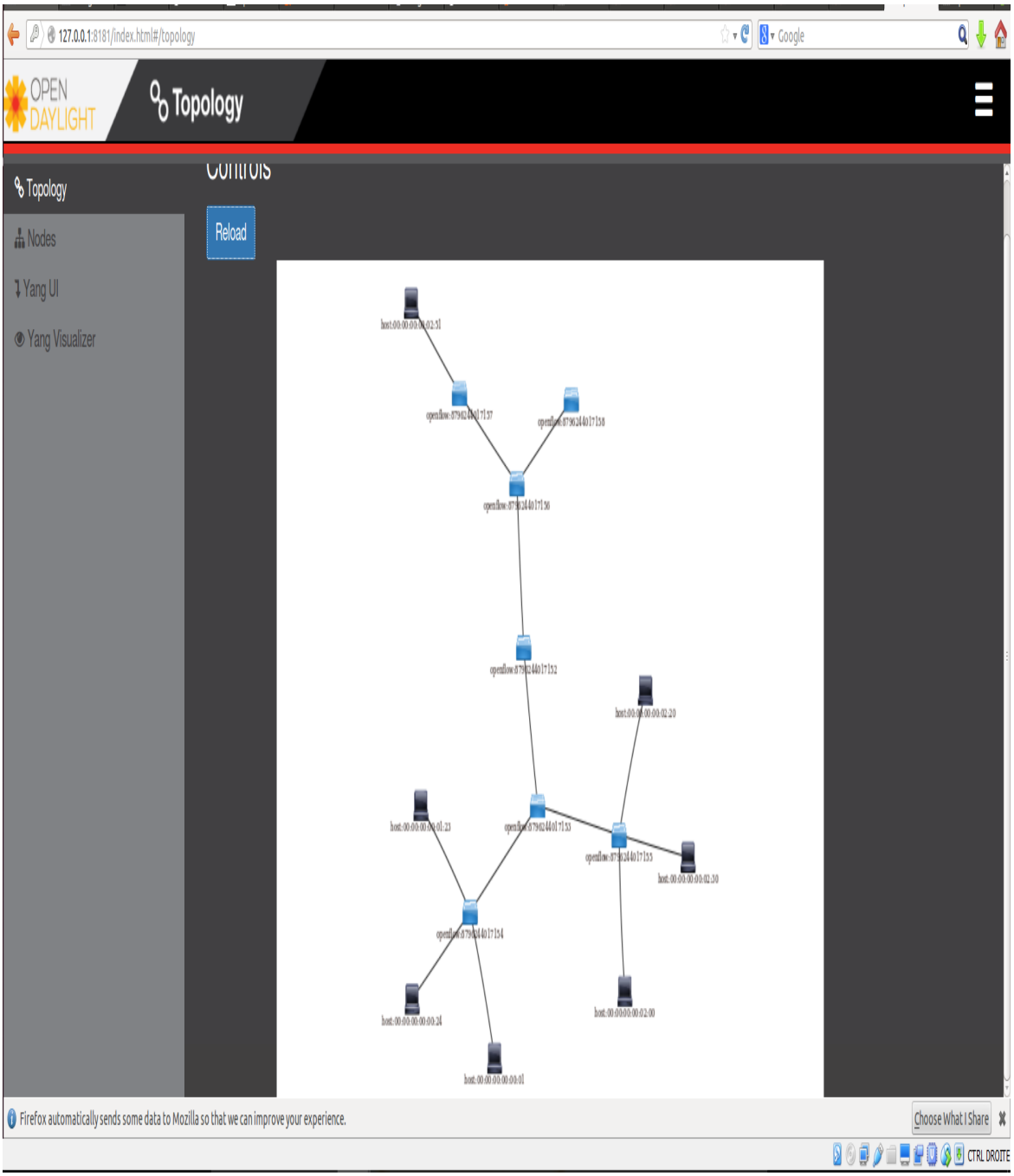

FIGURE 4.13 Interface graphique du controleur ODL

Nous avons aussi remarqué que les commandes fstae et ofevent ne nous affiché pas des information sur la visualisation des paquets du plan de controle et du plan de donnée et cela méme si le controleur fonctionne parfaitement et que les ping passent.
Nous remarquons sur le tableau de bord relatif au controleur ODL que les valeurs correspondantes aux paquets du plan de controle. C'est à dire ceux des paquets émis ou reçu par

le controleur, les Flow_mods ainsi que les tables de flux sont beaucoup plus petite que les valeurs des deux controleur précédents.

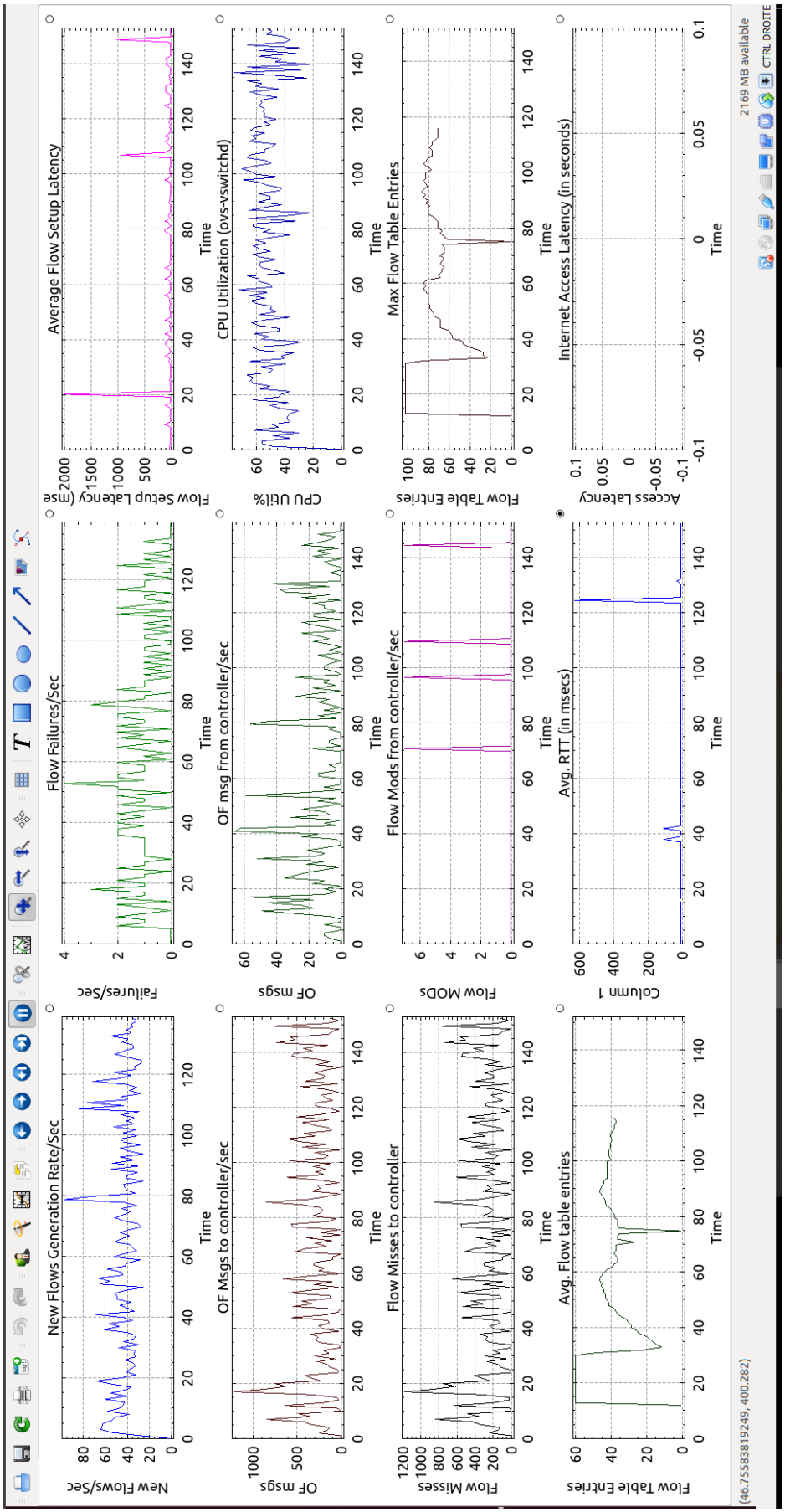

FIGURE 4.14 Tableau de bord des performances pour le controleur ODL

Chapitre 5

Conclusion générale

Dans ce mémoire, l'évaluation de la performance de certains contrôleurs OpenFlow exécutant une application de commutation d'apprentissage a été réalisée ainsi que des analyses pour la technologie SDN (réseaux définis par logiciel) et le protocole OpenFlow. Au début du mémoire, une étude de base sur SDN a été réalisée pour explorer cette nouvelle technologie, en particulier son utilisation de différente maniére associé à différentes technologies et implanté dans différents milieux. Ensuite, une étude du protocole OpenFlow a été présentée pour explorer l'architecture du protocole OpenFlow et des contrôleurs OpenFlow.

Suite a ce constant et a la comparaison fait précédement, on remarque que les deux contrôleurs ce resssemble quelque peu etant donné que Beacon est a la base de Floodlight mais que Beacon communique beacoup plus avec ses commutateurs et ainsi les solicite beaucoup plus, cela explique la différence entre l'utilisation des ressources CPU des ovsvswitches et cela concordre parfaitremant avec toutes les études faites précédement ou Beacon a le plus grand débit. On peut on déduire que Beacon a un mode de foctionnement beacoup plus réactif comparé a celui de Floodlight.

Le controleur OpenDaylight n'a pas été pris en compte pour la comparaiosn des performance et cela par rapport a l'instabilité des résultats obtenu en utiliasnt OFNetce qui pourrait fausser notre conclusion, méme si on peut tout de méme dire qu'a partir des résultats obtenus et en comparaison avec des études faites précédement sur lui qu'il presente un mode de fonctionnement trés proactive et donc concorde avec nos résultats.

Bien que tous les contrôleurs testés aient un support multi-thread, ils ont néanmoins démontré une certaine différence dans les comportements d'évolutivité. La différence de comportement de débit pour les contrôleurs peut être causée par deux facteurs : le premier est

l'algorithme qu'un contrôleur utilise pour distribuer les messages entrants entre les threads, et le second est les mécanismes et les bibliothèques utilisés pour l'interaction entre le réseau et le Contrôleur [7].

Traveaux futur

il est également utile d'effectuer un autre test de benchmarking pour les contrôleurs OpenFlow à la pointe de la technologie. Outre l'évaluation implémentée dans ce mémoire, l'évolutivité et la fiabilité sont deux facteurs importants à prendre en considération lors du choix d'un contrôleur OpenFlow. L'analyse comparative de la scalabilité peut tester le nombre maximal de connexions OpenFlow simultanées prises en charge par un contrôleur et un test de fiabilité peut caractériser le comportement du contrôleur lorsqu'un message OpenFlow malveillant essaie de compromettre le contrôleur OpenFlow[2]. La sécurité aussi qui est un paramétre crucial dans un choix d'équipement surtout que centraliser le controle du réseau fera des controleurs une cible privilégiére pour les hacker malveillant.

Difficultés rencontrés

1. Au début.le sujet nous a parru un peu vaste, et la difficulté était de trouver une structure à notre projet.
2. En ce qui concerne Mininet, Le fait d'avoir très peu de référence qui explique le fonctionnement et surtout l'architecture de cette plateforme, Rend la compréhension un peu difficile.
3. L'apprentissage du langage de programmation python.
4. Des bugs dans le fonctionnement de Mininet. qui ne sont pas expliquées dans MIninet.org et qui nous ont induit en erreur ce qui nous a fait perdre beaucoup de temps.
5. L'apprentissage d'OFNet et les problémes d'instabilité rencontré avec le controleur Opendaylight.
6. La redaction compléte du manuscrit en LaTeX.
7. l'apprentissage du systeme d'éxploitation Linux.
8. la tentative de faire la simulation avec un autre logiciel que mininet (Pktblaster untilisant Docker)

Bibliographie

[1] Turner J Taylor, D. Diversifying the internet. In Proceedings of the IEEE Global.Telecommunications Conference., 2005.

[2] Hassan Abdalkreim Ahmed Sonba. Performance comparison of the state of the art openflow controllers. Master's thesis, Halmstad University, 2014.

[3] Tejada Muntanerč. Evaluation of openflow controllers. Master's thesis, KTH, 2012.

[4] M. Casado N. Gude, T. Koponen and S. Shenker. Nox : towards an operating system for networks. *ACM SIGCOMM Computer Communication Review*, 38(3) :105–110, 2008.

[5] D. Erickson. Beacon open flow controller, fevrier 2013. [Online]. Available at : https ://openflow.stanford.edu/display/Beacon.

[6] A.L. Cox Z. Cai and T. S. Eugene Ng. *Maestro : A system for scalable openflow control*. Technical Report, Rice University, 2010.

[7] D. Zimarina V. Pashkov A. Shalimov, D. Zuikov and R. Smeliansky. Advanced study of sdn/openflow controllers. In Proceedings of the 9th ACM Central and Eastern European Software Engineering Conference in Russia, 2013.

[8] D. Erickson. The beacon openflow controller. In Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking, 2013.

[9] P. Lappas. Floodlight documentation, decembre 2013. [Online]. Available at : http ://floodlight.openflowhub.org.

[10] K. Wan. Ryu openflow controller. Master's thesis, University Oregon, 2013. [Online]. Available at : http ://osrg.github.io/ryu.

[11] D. Saikia. Mul documentation, juin 2014. [Online]. Available at : http ://sourceforge.net/p/mul/wiki/MuL

[12] Open Networking Foundation. *Software-defined networking : The new norm for networks*. ONF White Paper, 2012.

[13] Open networking foundation. Software-defined networking : The new norm for networks, decembre 2012. [Online]. Available at : http ://www.opennetworking.org.

[14] Parulkar G. Das, S. and N. McKeown. Why openflow/sdn can succeed where gmpls failed. In European Conference and Exhibition on Optical Communication., 2012.

[15] M. Nispel. A different take on software defined networks, juin 2013. Extreme Networks [Online]. Available at : http ://www.extremenetworks.com/a-different-take-on-software definednetworks.

[16] QuinStreet Enterprise. Software defined networking : 39l'année, fevrier 2016. [Online]. Available at : http ://www.zdnet.fr/actualites/software-defined-networking-39-des-entreprises-vont-deployer-dans-l-annee-39832408.htm.

[17] X. Gong X. Que Y. Hu, W. Wendong and C. Shiduan. Reliability-aware controller placement for software-defined networks. *IFIP/IEEE International Symposium on Integrated Network Management*, pages 672–675, 2013.

[18] D. Butnariu R. Wang and J. Rexford. *OpenFlow-based server load balancing gone wild.* In Proceeding of the USENIX Conference on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services (Hot-ICE), Boston, MA, USA, 2011.

[19] Q. Hao F. Hu and K. Bao. *IEEE Communications Surveys and Tutorials*, (99) :1–1.

[20] A. Shalimov and R. Smeliansky.

[21] A. Shalimov P. Ivashchenko and R. Smeliansky. High performance inkernel sdn/openflow controller, juin 2014. [Online]. Available at : https ://www.usenix.org/sites/default/files/ons2014-poster-ivashchenko.pdf.

[22] A. Tootoonchian S. H. Yeganeh and Y. Ganjali. On scalability of software defined networking. *IEEE Communications Magazine*, 51(2) :136–141, 2013.

[23] G. Gibb R. Sherwood and G. Parulkar. Flowvisor : A network virtualization layer. *OpenFlow Switch Consortium*, 2009. Tech Rep.

[24] A. Tootoonchian and Y. Ganjali. Hyperflow : A distributed control plane for openflow. *In Proceedings of the 2010 internet network management conference on Research on enterprise networking*, 2010.

[25] N. Gude J. Stribling L. Poutievski M. Zhu T. Koponen, M. Casado and S. Shenker. A distributed control platform for large-scale production networks. *In OSDI*, 10 :1–6, 2010.

[26] M. P. Fernandez. Comparing openflow controller paradigms scalability : Reactive and proactive. *In Advanced Information Networking and Applications (AINA), 2013 IEEE 27th International Conference*, pages 1009–1016, march 2013.

[27] G. Kontesidou and Z. Kyriakos. Open flow virtual networking : A flow-based network virtualization architecture. Master's thesis, Royal Institute of Technology(KTH)n, 2009.

[28] B. Heller B. Lantz and N. McKeown. A network in a laptop : rapid prototyping for software-defined networks. *In Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, page 19, 2010.

[29] M. Casado. List of openflow software projects, juin 2014. https ://yuba.stanford.edu/ casado/of-sw.html.

[30] J. Shin B. Lee, S. H. Park and S. Yang. Iris : The openflow-based recursive sdn controller. *In 2014 16th International Conference on Advanced Communication Technology (ICACT)*, pages 1227–1231, 2014.

[31] OpenDaylight Project Proposal R. R. Krishnan, V. Bhupatiraju. Dynamic flow management, juin 2016. [Online]. Available : https ://wiki.opendaylight.org/images/4/42/OpenDaylight-lb-prj-propv3. pdf.

[32] J. Mccauley. Pox : A python-based openflow controller., mai 2014. [Online]. Available at : http ://www.noxrepo.org.

[33] N. Foster J. Rexford C. Monsanto, J. Reich and D. Walker. Composing software defined networks. *In NSDI*, pages 1–13, Avril 2013.

[34] J. Shin B. Lee, S. H. Park and S. Yang. Iris : The openflow-based recursive sdn controller. *In 2014 16th International Conference on Advanced Communication Technology (ICACT)*, pages 1227–1231, Fevrier 2014.

[35] Intel Processors. "intel core i7-5500u processor (4m cache, up to 3.00 ghz)". [Online]. Available at :https ://ark.intel.com/fr/products/85214/Intel-Core-i7-5500U-Processor-4M-Cache-up-to-3_00-GHz.

[36] Ganesh H. Shankar. Ofnet an opensource openflow network emulation, visual debugging, performance monitoring and traffic generation environment, Avril 2016. [Online]. Available at : http ://sdninsights.org/.

[37] D. Erickson. Open flow benchmarking guidelines, janvier 2013. [Online]. Available at : http ://openflow.stanford.edu/display/Beacon/Benchmarking..

[38] Big Switch Networks R. Sherwood. Big switch, juillet 2014. [Online]. Available at : http ://http ://www.bigswitch.com.

[39] OpenDaylight. sit officiel opendaylight, juin 2017. [Online]. Available at : https ://www.opendaylight.org/.

Annexe A

Definitions

A.1 MPLS

Etant une technologie largement utiliser chez Djezzy nous tenions a citer un peu plus de details quand a son principe de fonctionnement en annexe.

A.1.1 Definition des Réseaux MPLS

Dans les réseaux informatiques et les télécommunications, MultiProtocol Label Switching (MPLS) est un mécanisme de transport de données basé sur la commutation d'étiquettes ou "labels", qui sont insérés à l'entrée du réseau MPLS et retirés à sa sortie.
MPLS doit permettre d'améliorer le rapport performance/prix des équipements de routage, d'améliorer l'efficacité du routage (en particulier pour les grands réseaux) et d'enrichir les services de routage (les nouveaux services étant transparents pour les mécanismes de commutation de label, ils peuvent être déployés sans modification sur le coeur du réseau).

A.1.2 Comparaison IP/MPLS

Dans un réseau IP typique, le trafic n'emprunte pas de chemin particulier. Chaque routeur calcule le routage localement. Chaque paquet est routé d'un nœud à un autre, suivant l'adresse de destination portée par les paquets, chaque nœud recalculant à chaque fois la route. Cette architecture de routage IP est considérée comme conventionnelle.

Avec le MPLS, chaque paquet IP se verra assigner un " shim header " contenant un " label " à son entrée dans le réseau et le trafic empruntera un chemin défini préalablement, le

LSP (Label Switched Path). Le premier routeur calcule la route et assigne le label, les autres ne font que de la commutation.

A.1.3 Principe de fonctionnement

Basée sur la permutation d'étiquettes, un mécanisme de transfert simple offre des possibilités de nouveaux paradigmes de contrôle et de nouvelles applications. Au niveau d'un LSR (Label Switch Router) du nuage MPLS, la permutation d'étiquette est réalisée en analysant une étiquette entrante, qui est ensuite permutée avec l'étiquette sortante et finalement envoyée au saut suivant. Les étiquettes ne sont imposées sur les paquets qu'une seule fois en périphérie du réseau MPLS au niveau du Ingress E-LSR (Edge Label Switch Router) où un calcul est effectué sur le datagramme afin de lui affecter un label spécifique. Ce qui est important ici, est que ce calcul n'est effectué qu'une fois. La première fois que le datagramme d'un flux arrive à un Ingress E-LSR. Ce label est supprimé à l'autre extrémité par le Egress E-LSR. Donc le mécanisme est le suivant : Le Ingress LSR (E-LSR) recoit les paquets IP, réalise une classification des paquets, y assigne un label et transmet les paquets labellisés au nuage MPLS. En se basant uniquement sur les labels, les LSR du nuage MPLS commutent les paquets labellisés jusqu'à l'Egress LSR qui supprime les labels et remet les paquets à leur destination finale.

L'affectation des étiquettes aux paquets dépend des groupes ou des classes de flux FEC (forwarding équivalence classes). Les paquets appartenant à une même classe FEC sont traités de la même manière. Le chemin établi par MPLS appelé LSP (Label Switched Path) est emprunté par tous les datagrammes de ce flux. L'étiquette est ajoutée entre la couche 2 et l'en-tête de la couche 3 (dans un environnement de paquets) ou dans le champ VPI/VCI (identificateur de chemin virtuel/identificateur de canal virtuel dans les réseaux ATM). Le switch LSR du nuage MPLS lit simplement les étiquettes, applique les services appropriés et redirige les paquets en fonction des étiquettes. Ce schéma de consultation et de transfert MPLS offre la possibilité de contrôler explicitement le routage en fonction des adresses source et destination, facilitant ainsi l'introduction de nouveaux services IP. Un flux MPLS est vu comme un flux de niveau 2.5 appartenant niveau 2 et niveau 3 du modèle de l'OSI.

A.2 Définitions diverses

Mininet : Est un émulateur logiciel qui permet de faire des prototypes d'un grand réseau sur une seule machine.

Mininet peut être utilisé pour créer rapidement un réseau virtuel réaliste exécutant les codes de noyau, de commutateur et de logiciel sur un ordinateur personnel. Il permet aussi à l'utilisateur de créer rapidement, d'interagir, de personnaliser et de partager un prototype de réseau (SDN) défini par logiciel pour simuler une topologie de réseau qui utilise des commutateurs Openflow.

Il existe de nombreuses raisons qui déclenchent l'utilisation d'un nouvel émulateur appelé Mininet ; D'abord, il existe seulement quelques périphériques réseau disponibles pour la mise en œuvre de la norme SDN car ce n'est pas encore une technologie étendue du point de vue industriel. De plus, la mise en œuvre d'un réseau avec un grand nombre de périphériques réseau est très difficile et coûteuse. Par conséquent, pour surmonter ces problèmes, la stratégie en mode virtuel a été menée dans le but de prototypage et d'émulation de ce type de technologies de réseau et la plus importante est MININET. Il a donc la capacité d'imiter différents types d'éléments de réseau tels que ; Hôte, les commutateurs de couche 2, les routeurs de couche 3 et les liens. Il fonctionne sur un noyau Linux unique et il utilise la virtualisation dans le but d'émuler un réseau complet en utilisant un seul système. Cependant, l'hôte, les commutateurs, les routeurs et les liens créés sont des éléments du monde réel, bien qu'ils soient créés au moyen de logiciels.

LaTeX : LaTeX est un langage créé pour séparer le fond de la forme lors de la création d'un document ou d'une publication, il décrit comment il veut hiérarchiser l'information. Ensuite, son code est traité par un logiciel : LaTeX choisit alors les meilleurs agencements et la disposition optimale pour chacun des éléments du document.
En résumé, LaTeX est un langage de description donnant à l'auteur les moyens d'obtenir des documents mis en page de façon professionnelle sans avoir à se soucier de leur forme. La priorité est donnée à l'essentiel : le contenu.
LaTeX se prononce « latec » ou « latèque », mais certainement pas « latex ».Le nom est l'abréviation de Lamport TeX. On écrit souvent LaTeX.

Kernel : Un noyau de système d'exploitation, ou simplement noyau, ou kernel (de l'anglais), est une des parties fondamentales de certains systèmes d'exploitation. Il gère les ressources de l'ordinateur et permet aux différents composants — matériels et logiciels — de communiquer entre eux.

Cloud cumputing :Le cloud computing, ou l'informatique en nuage ou nuagique, est l'exploitation de la puissance de calcul ou de stockage de serveurs informatiques distants par l'intermédiaire d'un réseau, généralement internet. Ces serveurs sont loués

à la demande, le plus souvent par tranche d'utilisation selon des critères techniques (puissance, bande passante, etc.) mais également au forfait.

SSH : Secure Shell (SSH) est à la fois un programme informatique et un protocole de communication sécurisé. Le protocole de connexion impose un échange de clés de chiffrement en début de connexion. Par la suite, tous les segments TCP sont authentifiés et chiffrés. Il devient donc impossible d'utiliser un sniffer pour voir ce que fait l'utilisateur.

NAT :En réseau informatique, on dit qu'un routeur fait du **network address translation** (NAT) (« traduction d'adresse réseau » - voir la section « traduction » ou « translation ») lorsqu'il fait correspondre des adresses IP à d'autres adresses IP. En particulier, un cas courant est de permettre à des machines disposant d'adresses qui font partie d'un intranet et ne sont ni uniques ni routables à l'échelle d'Internet, de communiquer avec le reste d'Internet en semblant utiliser des adresses externes uniques et routables.

Putty : est un émulateur de terminal doublé d'un client pour les protocoles SSH, Telnet, rlogin, et TCP brut. Il permet également d'établir des connexions directes par liaison série RS-232. À l'origine disponible uniquement pour Windows, il est à présent porté sur diverses plates-formes Unix (et non-officiellement sur d'autres plates-formes).

Xming :est un portage sous Windows du système de fenêtrage X ouvert des systèmes Unix, Linux et BSD. Il est fondé sur le serveur X.org et compilé avec MinGW. Il permet ainsi de rediriger l'affichage vers Windows d'une application graphique tournant sur une machine distante, sous un autre système supporté par X.org.

JVM : est un appareil informatique fictif qui exécute des programmes compilés sous forme de bytecode Java.(Java Virtual Machine)

Git : est un logiciel de gestion de versions décentralisé. C'est un logiciel libre créé par Linus Torvalds, auteur du noyau Linux, et distribué selon les termes de la licence publique générale GNU version 2. En 2016, il s'agit du logiciel de gestion de versions le plus populaire qui est utilisé par plus de douze millions de personnes.

La virtualisation : consiste à faire fonctionner un ou plusieurs systèmes d'exploitation1/ applications comme un simple logiciel, sur un ou plusieurs ordinateurs-serveurs/ système d'exploitation, au lieu de ne pouvoir en installer qu'un seul par machine. Ces ordinateurs virtuels sont appelés serveur privé virtuel (Virtual Private Server ou VPS) ou encore environnement virtuel (Virtual Environment ou VE).

QoS : La qualité de service (QDS) ou quality of service (QoS) est la capacité à véhiculer dans de bonnes conditions un type de trafic donné, en termes de disponibilité, débit, délais de transmission, gigue, taux de perte de paquets... La qualité de service est

un concept de gestion qui a pour but d'optimiser les ressources d'un réseau (en management du système d'information) ou d'un processus (en logistique) et de garantir de bonnes performances aux applications critiques pour l'organisation. La qualité de service permet d'offrir aux utilisateurs des débits et des temps de réponse différenciés par applications (ou activités) suivant les protocoles mis en œuvre au niveau de la structure.

Apache ANT : Ant est un logiciel créé par la fondation Apache qui vise à automatiser les opérations répétitives du développement de logiciel telles que la compilation, la génération de documents (Javadoc) ou l'archivage au format JAR, à l'instar des logiciels Make.

Ant est écrit en Java et son nom est un acronyme pour « Another Neat Tool » (un autre chouette outil). Il est principalement utilisé pour automatiser la construction de projets en langage Java, mais il peut être utilisé pour tout autre type d'automatisation dans n'importe quel langage.

Multithread : Un processeur est dit multithread s'il est capable d'exécuter efficacement plusieurs threads simultanément. Contrairement aux systèmes multiprocesseurs (tels les systèmes multi-cœur), les threads doivent partager les ressources d'un unique cœur : les unités de traitement, le cache processeur et le translation lookaside buffer ; certaines parties sont néanmoins dupliquées : chaque thread dispose de ses propres registres et de son propre pointeur d'instruction. Là où les systèmes multiprocesseurs incluent plusieurs unités de traitement complètes, le multithreading a pour but d'augmenter l'utilisation d'un seul cœur en tirant profit des propriétés des threads et du parallélisme au niveau des instructions. Comme les deux techniques sont complémentaires, elles sont parfois combinées dans des systèmes comprenant de multiples processeurs multithreads ou des processeurs avec de multiples cœurs multithreads.

IT : Stands for "Information Technology," and is pronounced "I.T." It refers to anything related to computing technology, such as networking, hardware, software, the Internet, or the people that work with these technologies. Many companies now have IT departments for managing the computers, networks, and other technical areas of their businesses. IT jobs include computer programming, network administration, computer engineering, Web development, technical support, and many other related occupations. Since we live in the "information age," information technology has become a part of our everyday lives. That means the term "IT," already highly overused, is here to stay

Karaf Apache Karaf est un projet open source de la fondation Apache écrit en Java qui propose une distribution d'un environnement OSGi pour la création de serveurs. Karaf

peut être utilisé sur tout système d'exploitation fournissant une machine virtuelle Java.

OSGI L'OSGi Alliance (précédemment connue en tant qu'Open Services Gateway initiative) est une organisation qui spécifie une plate-forme de services fondée sur le langage Java qui peut être gérée de manière distante. Le cœur de cette spécification est un framework (canevas) qui définit un modèle de gestion de cycle de vie d'une application, un répertoire (registry) de services, un environnement d'exécution et des modules. Fondés sur ce framework, un grand nombre de couches (layers) OSGI, d'API et de services ont été définis.