REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
MINISTERE DE L'ENSEIGNEMENT SUPERIEUR
ET DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITE SAAD DAHLEB DE BLIDA
FACULTE DES SCIENCES
DEPARTEMENT D'INFORMATIQUE

MEMOIRE DE FIN D'ETUDES

Pour l'obtention

D'un diplôme de master en informatique.

Spécialité : Système Informatique et Réseaux

THÈME :

Etude exploratoire et mise en œuvre des solutions basées sur SDN pour Groupe Sonelgaz

Réalisé par :

Meridji Rania Meridji Ibtihel

Soutenue le : 16/07/2019 , devant :

Ferfera Soufiane	Président
Ould aissa	Examinateur
Mr. Ould-Khaoua Mohamed	Promoteur
Mme.DebbaghSouhila	Encadreur

2018/2019

Résumé

SDN est une technologie émergente qui permet de séparer le plan de contrôle du plan de données d'un équipement réseau. Ce qui en permet une évolution du matériel et du logiciel d'une façon indépendante.

SDN a déclenché un changement radical dans la conception des réseaux, et le marché s'est rapidement approprié cette technologie comme un ensemble de solutions/architectures permettant de supprimer les frontières existantes entre les mondes des applications et du réseau. D'autant plus que le déploiement d'applications est toujours plus aisé et dynamique, notamment grâce à la virtualisation et au Cloud.

Dans ce mémoire, nous avons évalué les performances de deux contrôleurs SDN, à savoir, Floodlight et Opendaylight. Puis nous en avons sélectionné le plus performant des deux et nous en avons testé les fonctionnalités.

Mots-clefs: SDN, Contrôleur, Floodlight, Openflow, Open vSwitch.

Abstract

SDN is an emerging technology that separates the control plane from the data plane of a network device. This allows the hardware and software to evolve independently.

SDN triggered a radical change in network design, and the market quickly embraced this technology as a set of solutions / architectures that removed the existing boundaries between the application and network worlds. Especially since the deployment of applications is always easier and dynamic, thanks to virtualization and the Cloud.

In this memoir, we evaluated the performance of two SDN controllers, namely, Floodlight and Opendaylight. Then we selected the best of both and we tested the features.

Keywords: SDN, controller, Floodlight, Openflow, Open vSwitch.

ملخص

اس دي ان هي تقنية ناشئة تفصل مستوى التحكم عن مستوى بيانات جهاز الشبكة. هذا يسمح للأجهزة والبرامج بالتطور بشكل مستقل.

تسببت اس دي ان في تغيير جذري في تصميم الشبكة، وسرعان ما تبنت السوق هذه التكنولوجيا كمجموعة من الحلول / الهندسة التي أزالت الحدود الحالية بين التطبيق وعالم الشبكات. خاصة وأن نشر التطبيقات يكون دائمًا أسهل وديناميكيًا، وذلك بفضل المحاكاة الافتراضية والسحابة.

في هذه المذكرات، قمنا بتقييم أداء وحدتي تحكم اس دي ان، وهما أوبن فلو وفلودلايت. ثم اخترنا الأفضل من الاثنين وقمنا باختبار الميزات.

كلمات مفتاحية: مراقب, الشبكات المبرمجة , أوبن فلو, فلودلايت, أوبن في سويش.

Remerciement

Merci à Dieu de sa grâce, source de notre force et courage tout au long de nos études universitaires.

Nous tenons à exprimer nos vifs remerciements et notre profonde gratitude à Mr. Ould Khaoua Mohamed pour son aide, son encouragement et de nous avoir dirigés dans notre travail.

Nous remercions également Mme. Debbagh Souhila et Mr. Kermiche Oussama de nous avoir encadré dans notre mémoire de fin d'études.

Nous remercions les membres du jury pour avoir accepté d'évaluer notre travail.

Un grand merci également à notre famille pour leur soutien aussi bien moral que financier et pour leurs sacrifices.

Nous ne pourrons terminer sans remercier tous ceux qui ont participé d'une manière ou d'une autre dans l'élaboration de ce projet de fin d'études.

Merci !

Dédicace :

Je dédie ce modeste travail à :

A l'homme de ma vie, mon exemple éternel, mon soutien moral et source de joie et de bonheur, celui qui s'est toujours sacrifié pour me voir réussir,

à toi mon père.

A la lumière de mes jours, la source de mes efforts, la flamme de mon cœur, ma vie et mon bonheur ; maman que j'adore.

À ma moitié ma jumelle ma très chère sœur Mon binôme « Rania »*qui m'a soutenu toute ma vie <3*

À mes très chers frères et mes belles sœurs

À toute ma famille : mes grands-parents, mes tantes, mes oncles et mes cousins et cousines

À tous mes amis et mes collègues

Enfin, Toute personne qui m'aime et que j'aime …

Ibtihel

Dédicace :

À MES CHERS PARENTS

Aucune dédicace ne saurait exprimer mon respect, mon amour éternel et ma considération pour les sacrifices que vous avez consenti pour mon

Instruction et mon bien être. Je vous remercie pour tout le soutien et l'amour que vous me portez depuis mon enfance et j'espère que votre bénédiction m'accompagne toujours.

À ma moitié ma jumelle ma très chère sœur Mon binôme « Ibtihel » qui m'a soutenu toute ma vie <3

À mes très chers frères et mes belles sœurs

À toute ma famille : mes grands-parents, mes tantes, mes oncles et mes cousins et cousines

À tous mes amis et mes collègues

Enfin, Toute personne qui m'aime et que j'aime …

Rania

La liste des figures

La liste des tableaux

Sommaire

Introduction générale

Dans le cadre de notre projet de fin d'étude sanctionnant un cursus de deux années de master en système informatique et réseaux, il nous a été donné de travailler sur la conception d'un réseau défini par logiciel (Software Defined Networking). Le présent document récapitule le travail effectué dans le cadre de ce projet.

Actuellement, tout le monde a pu remarquer qu'il y a une amélioration constante de la vitesse de l'internet avec un taux croissant de transfert de données qui est passé de 10 à 100 Gigabits par seconde. De plus, avec la mise en place de serveurs puissants et l'apparition du «Cloud computing», le problème de stockage et de traitement des données partiellement massives a été résolu. De même que celui de la saturation des adresses internet publiques, notamment, avec le lancement de la sixième génération du protocole internet IPV6. Toutefois, des problèmes subsistent et que tout le monde veut résoudre en mettant en place des solutions fortes, sûres et à faible coût sur lesquelles se baser et qui soient surtout faciles à implémenter et assez puissantes pour permettre leur résolution.

L'utilisation des appareils mobiles, les technologies émergentes telles que le cloud computing et la virtualisation, a impacté les modèles de trafic. La hausse des Big Data dans les centres de données pose la nécessité d'une grande capacité de réseau et mise à l'échelle de celui-ci. Pour répondre à ces besoins, les périphériques réseau deviennent plus complexes à gérer. En outre, il serait difficile et chronophage pour les administrateurs réseau de configurer ces périphériques dont le rôle est celui de prise de décision et d'acheminement de données. A cet effet, les équipements doivent se servir de règles implémentées en leur sein.

C'est dans ce contexte que la technologie Software Defined Networking (SDN) a vu le jour visant à consolider la programmation et le réseau, en permettant d'agir sur une infrastructure physique comme sur une entité logique, facilitant ainsi la gestion du réseau et sa mise à l'échelle. Tout comme l'idée des réseaux programmables a été introduite pour répondre à ces défis et faciliter l'évolution du réseau.

Par conséquent, SDN est un nouveau paradigme, qui a révolutionné l'architecture du réseau traditionnel. Il permet de séparer efficacement le plan de contrôle du plan de données et le déplacer vers un serveur centralisé nommé contrôleur. De cette façon, la complexité de la gestion du réseau est gérée par un contrôleur basé sur un logiciel fournissant ainsi une abstraction de l'infrastructure. Ce qui permet d'obtenir un plan de données simple et rend le plan de contrôle facile à gérer et de manière centralisée.

Le groupe Sonelgaz souhaitant investir dans un réseau SDN, il nous a proposé de le découvrir et de le mettre en œuvre afin de mesurer et de comparer les performances des contrôleurs Floodlight et Opendaylight, puis de choisir le plus satisfaisant à leurs exigences. Avec ce contrôleur, des problèmes liés à la sécurité (firewall et ACL) et la transmission de trafic (Forwarding et load balancing) seront résolus.

Notre mémoire retrace les différentes étapes qui ont donné lieu à cette solution. Des recherches bibliographiques ont été menées sur l'approche SDN. Ainsi, ce mémoire est organisé en quatre chapitres suivis d'une conclusion générale, à savoir :

Dans chapitre 1 : Réseau défini par logiciel (SDN), nous essayons de donner une vue générale sur les technologies des réseaux qui sont les motivations du Software-Defined Networking (SDN), la technologie du SDN et ses avantages.

Dans chapitre 2 : Les composants du SDN, nous essayons d'aborder les composants importants qui constituent les réseaux définis par logiciel.

Dans chapitre 3 : Evaluation des performances de Floodlight et Opendaylight, nous expliquons principalement les différentes démarches suivies pour déployer un réseau SDN, ensuite nous expliquons la raison pour laquelle on a choisi le contrôleur Floodlight en le comparant avec un autre contrôleur Opendaylight.

Dans chapitre 4 : Firewall et Access Control List dans le SDN, nous présentons la couche application du SDN, et nous définissons notre choix de domaine de firewall, ACL, Forwarding et load balancing afin de démontrer et de prouver l'utilité et l'objectif des réseaux SDN.

Nous terminons ce mémoire par une Conclusion générale dans laquelle nous présentons notre travail, ainsi que les perspectives envisagées.

Chapitre 1 :
Réseau défini par logiciel (SDN)

Introduction :

De nos jours, l'internet connait un énorme succès. Il est devenu un outil universel et indispensable pour les entreprises et la vie quotidienne de milliards d'individus. Avec l'explosion du nombre de terminaux mobiles, l'utilisation massive des réseaux sociaux et l'apparition de nouvelles tendances des technologies de l'information comme le Cloud Computing, la virtualisation, le BIG-DATA, et l'internet des objets (50 milliards d'objets connectés) qui sont génératrices de gros volumes de données, l'internet doit relever le défi de traiter ces volumes de trafic de plus en plus gigantesques [1].

Cependant, l'internet est devenu une infrastructure critique à cause de l'absence de changements dans le réseau cœur et de la rigidité des équipements déployés, qui rendent la mise en place et le déploiement de nouveaux services réseaux difficiles et coûteux [2].

Toutefois, l'un des problèmes qui subsiste et que tout le monde veut résoudre est la mise en place de solutions fortes, sûres et à faible coût sur lesquelles il est possible de se baser et qui soient faciles à traiter tout en étant assez puissantes pour la résolution des problèmes qui peuvent survenir.

Cela a donné aux chercheurs un motif pour passer à une architecture à contrôle centralisé, dynamique, facile à gérer et hautement évolutive d'où le SDN -Software-Defined-Networking c'est-à-dire le réseau défini par l'application. Ce qui a bouleversé l'approche traditionnelle des réseaux.

Dans ce chapitre, nous présentons les technologies des réseaux qui sont utilisées actuellement telles que la virtualisation, la centralisation, le Cloud, et le besoin d'avoir un réseau programmable et aussi nous présentons sur la technologie SDN.

1. Les motivations du Software-Defined Networking (SDN)

1.1. Réseaux traditionnels

Les équipements réseaux traditionnels utilisés actuellement comportent deux parties fondamentales : plan de contrôle et plan de données (le cerveau et le corps). La fonction de la première partie se résume en la prise de décision et le lancement des processus, comme le routage ou redirection du trafic, par contre la deuxième partie traite la mise en œuvre de toutes les décisions prises par la première partie, c'est-à-dire lorsqu'un paquet arrive sur un port d'un commutateur ou d'un routeur, celui-ci applique les règles de routage ou de commutation qui sont inscrites dans son système d'exploitation [3].

Il est difficile de gérer les réseaux et d'appliquer efficacement de nouvelles politiques, car chaque périphérique fonctionne avec un protocole d'un niveau donné et doit être configuré de manière appropriée pour communiquer les uns avec les autres. Donc, il n'est pas facile d'ajouter une règle parce

qu'il n'y a pas de moyen standardisé pour le faire, mais aussi parce qu'un périphérique risque d'entrer en conflit avec l'un des nombreux autres périphériques du réseau. En outre, les règles étaient initialement destinées à être statiques et donc le réseau est statique et mal adapté aux besoins et technologies actuels.

La vie d'un administrateur réseau est un cycle sans fin de réflexion, de travail fastidieux, de configurations, de mises à niveau et de remplacements. Pour les entreprises et les opérateurs, c'est une activité coûteuse. La mise en réseau n'a pas connu l'équivalent de la révolution des PC (*Perosnal Computer*) tel que banaliser le matériel, offrir un choix entre plusieurs systèmes d'exploitation et créer un marché d'applications concurrentes. Tous les fabricants des machines réseau ont développé des systèmes très compliqués et très incompatibles.

La figure ci-dessous montre l'architecture des équipements réseau traditionnels et leur fonctionnement. Ces réseaux traditionnels sont désavantagés par les points suivants :

- Complexité : l'ajout ou la modification d'équipements et l'implémentation des politiques réseaux sont complexes, longues et peuvent être source d'interruptions de service. Ce qui décourage les modifications et l'évolution du réseau.
- Passage à l'échelle : l'impossibilité d'avoir un réseau qui s'adapte au trafic a obligé les opérateurs à sur-provisionner leurs réseaux ce qui ajoute une complexité de gestion sur le plan de contrôle.
- Dépendance aux constructeurs : les constructeurs réalisent des produits avec des durées de vie et un manque de standard, d'interface ouverte. Ce qui restreint les opérateurs réseaux d'adapter le réseau à leurs propres besoins [4].

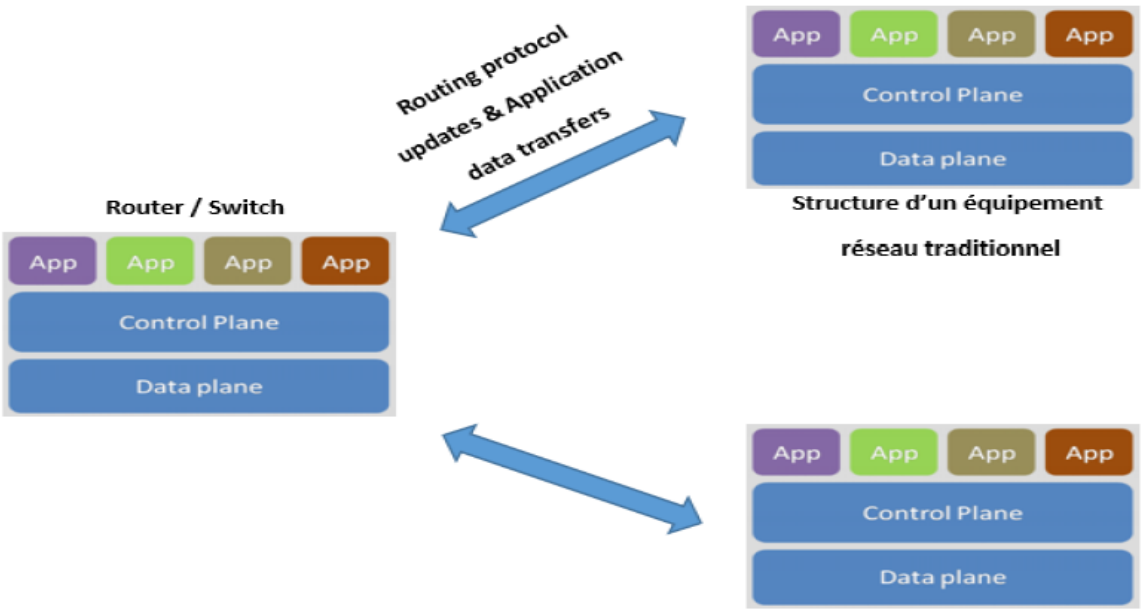


Figure 1.1 : Fonctionnements des structures des équipements réseaux traditionnels [4].

Une telle architecture statique est inadaptée aux besoins de stockages dynamiques des centres de données, des campus et environnements des prestataires de services réseaux d'aujourd'hui, ce qui cause un très large impact et conséquences qui sont mis en évidence dans le schéma suivant :

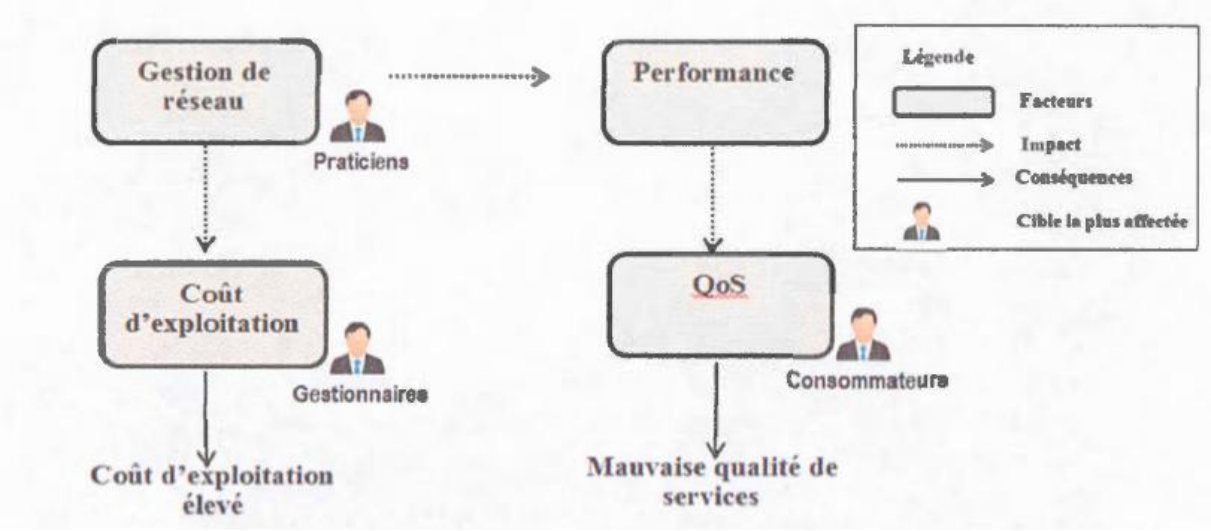


Figure 1.2 : Impacts et conséquences de l'architecture réseau traditionnel [5].

La figure 1.2 montre que la gestion des réseaux traditionnels d'aujourd'hui a une influence négative sur une grande partie de la population (praticiens, gestionnaires et consommateurs des services réseaux). Les conséquences qui en découlent sont, la mauvaise qualité de service pour les consommateurs, et les coûts d'exploitation élevés pour les gestionnaires. Il est alors extrêmement important d'étudier ce problème afin de palier à cette situation, ce qui représente d'énormes défis [5].

1.2. La virtualisation

La virtualisation est la capacité de simuler une plate-forme matérielle, telle qu'un serveur, un périphérique de stockage ou une ressource réseau, dans un logiciel. Toutes les fonctionnalités sont séparées du matériel et simulées comme une «instance virtuelle», avec la capacité de fonctionner comme le ferait une solution matérielle traditionnelle. Une solution virtualisée est généralement beaucoup plus portable, évolutive et économique qu'une solution matérielle traditionnelle [6].

La virtualisation correspond à l'ensemble des techniques matérielles et/ou logiciels qui permettent de faire fonctionner sur une seule machine plusieurs systèmes d'exploitation et/ou plusieurs applications, séparément les uns des autres, comme s'ils fonctionnaient sur des machines distinctes. La Virtualisation impacte quatre domaines majeurs :

- Le système d'exploitation
- Le serveur
- Le stockage des données
- La couche réseau [3]

La virtualisation des réseaux est le processus de combinaison des ressources matérielles du réseau et les fonctionnalités en une seule entité logicielle. La virtualisation du réseau dissocie complètement les ressources réseau du matériel sous-jacent. Tous les composants et fonctions réseau sont fidèlement répliqués dans le logiciel.

1.3. La centralisation

Dans le cadre d'un réseau, un système centralisé consiste au départ à mettre dans un seul endroit des périphériques jouant différents rôles, qui physiquement chacun d'entre eux pouvait être placé dans un local. Mais on les installe tous sur une seule machine en les virtualisant. Pour ainsi concentrer la gestion d'un réseau informatique, sous une administration d'une seule personne qui aura sous ses houlettes la charge de faire toutes les configurations possibles du réseau et répliquer enfin les mises à jour aux différents périphériques distants qui sont sous le contrôle de ce dernier.

C'est pourquoi en informatique, un contrôleur central centralise les plans de contrôle. La décision d'installer ou créer ce produit a été motivée par le souci :

- D'obtenir des informations en temps réel et à la demande.
- D'avoir un meilleur suivi de trafic, en gérant plusieurs périphériques depuis une console de gestion unique plus rapidement et plus facilement [7].

1.4. Cloud Computing

Un terme général pour la fourniture de service hébergé sur internet. Cloud Computing peut être brièvement décrit comme une infrastructure matérielle ou logicielle qui fournit des services de calcul à la demande, tels que le traitement, le stockage et la mise en réseau. Ceci est réalisé grâce au concept de virtualisation, qui résume la couche matérielle physique et la partage entre plusieurs hôtes, connues sous le nom de machines virtuelles (VMs). Le mécanisme responsable de l'extraction des ressources physiques et du contrôle de leur accès est connu sous le nom d'hyperviseur.

En général, le Cloud peut être classé comme privé, publique, hybride ou communautaire. Dans le Cloud Computing privé, l'administrateur VM a accès à l'hyperviseur, ce qui lui permet de gérer la disposition des VMs et de contrôler l'accès aux ressources physiques.

Dans le Cloud Computing public, hybride ou communautaire, l'administrateur VM n'a normalement pas accès à l'hyperviseur. Ainsi, l'administrateur ne peut pas déterminer le statut de la ressource physique VM, car le matériel est partagé entre plusieurs autres locataires [8].

Caractéristiques du cloud :

- Self-service à la demande : un utilisateur peut allouer des ressources sans interaction humaine avec le fournisseur [9].
- Accès réseau élargi : les ressources sont accessibles via le réseau par des systèmes hétérogènes [9].
- Partage de ressources : les ressources sont dynamiquement affectées, libérées puis réaffectées à différents utilisateurs. L'utilisateur n'a pas besoin de connaître la localisation (pays, région, centre de donnée) des ressources [9].
- Élasticité : les ressources peuvent être augmentées ou diminuées facilement, voire automatiquement, pour qu'elles s'adaptent aux besoins en temps réel. Les capacités offertes paraissent infinies, et l'utilisateur peut en consommer davantage, à tout moment, sans se soucier d'une éventuelle limite [9].
- Mesures : le système contrôle et optimise l'usage des ressources en mesurant régulièrement leur consommation. Ces mesures sont indiquées de manière transparente aux utilisateurs et au fournisseur pour le calcul de la facturation [9].

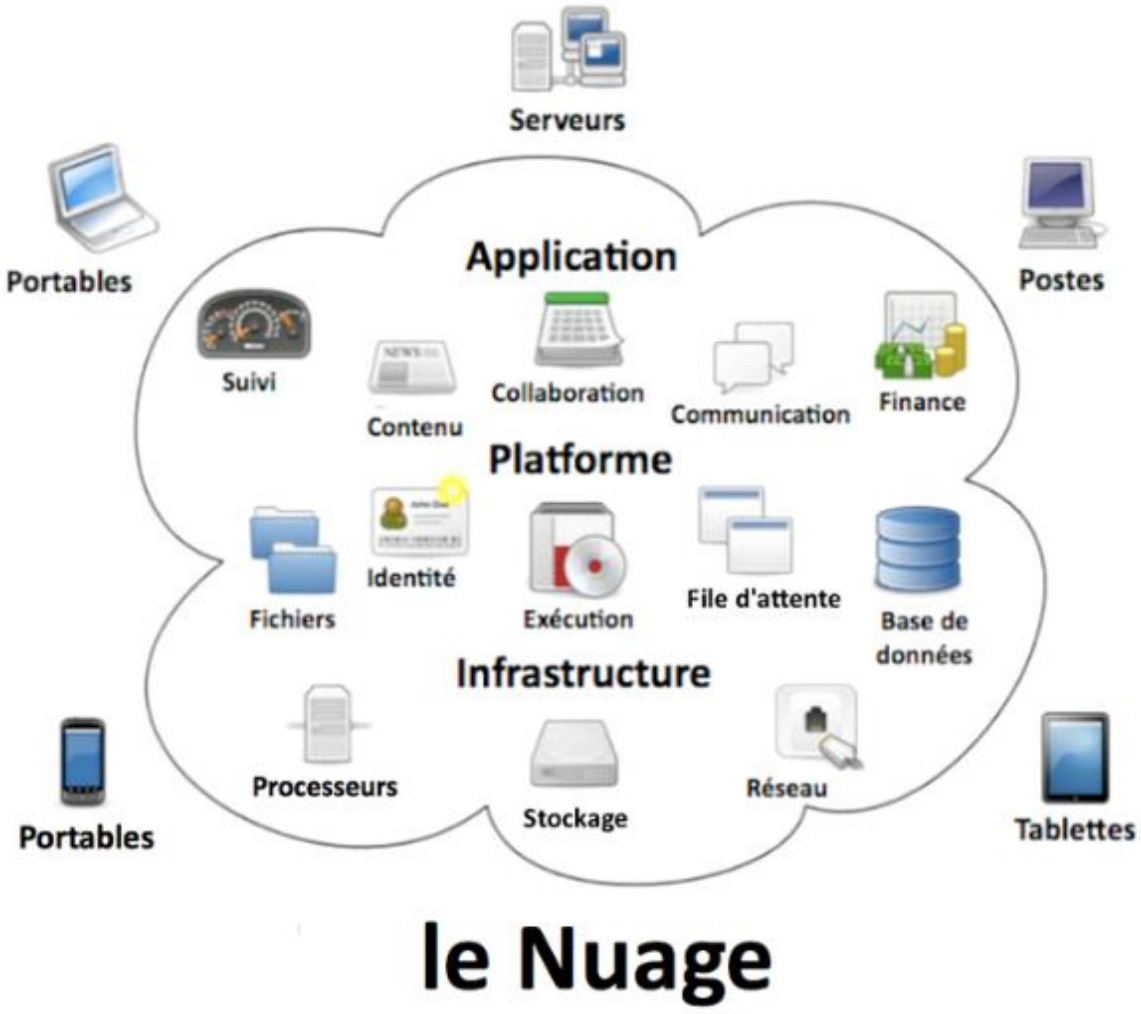


Figure 1.3 : Présentation du Cloud Computing [7].

1.5. Le besoin de réseau programmable

Les administrateurs réseau configurent généralement les équipements réseau à l'aide d'une interface de ligne de commande CLI (*Command Line Interface*) ou via l'interface utilisateur graphique GUI (*Graphical User Interface*). Ce modèle n'est pas sans problèmes.

La mise en œuvre de configurations de réseaux complexes peut exiger d'un ingénieur qu'il configure séparément plusieurs périphériques réseaux différents. C'est un processus gourmand en temps, fastidieux et source d'erreurs.

Tandis que le rythme du changement s'accélère, les entreprises passent au cloud, qui leur promet une agilité, une flexibilité et une évolutivité illimitées. Cependant, l'adoption croissante du cloud (ainsi que des applications mobiles, IoT et Big Data) a des répercussions profondes sur les infrastructures informatiques. En rendant les réseaux programmables, la programmabilité appliquée équivaut à une automatisation puissante, et donc il faudrait programmer le réseau pour qu'il agisse de la manière la plus adaptée aux besoins des entreprises [10].

2. Définition du SDN

SDN est une technologie qui rend le réseau programmable c'est-à-dire réseau défini par logiciel. Le SDN présente une architecture réseau où le plan de contrôle est totalement découplé du plan de données.

Le plan de contrôle gère la gestion des périphériques réseau, tandis que le plan de données est la couche matérielle responsable du transfert des paquets réseau selon les politiques définies dans le plan de contrôle. Ce découplage transforme les commutateurs/routeurs réseau en simples dispositifs de transfert, tandis que la commande logique est implémentée dans le contrôleur qui fonctionne comme un système d'exploitation réseau centralisé. Ainsi, le SDN est composé de deux entités : le contrôleur et les Dispositifs de transfert (switch ou routeur) [8].

Le SDN comme ensemble de solutions/architectures permettant de supprimer les frontières existantes entre les mondes des applications et du réseau. Ce qui le rend 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...).

2.1. Historique

Avant l'apparition des réseaux SDN tels que nous les connaissons aujourd'hui, plusieurs idées et travaux ont été proposés auparavant, notamment la programmation du réseau et la séparation des plans de contrôle et de données. Par exemple (AN, Active Network) et (PN, Programming Network) le projet de recherche dénommé DCAN (Devolved Control of ATM Networks).

Le début des réseaux SDN a commencé avec le projet Ethane, lancé en 2006 à l'université de Stanford. En effet, le projet Ethane définit une nouvelle architecture pour les réseaux d'entreprises. L'objectif d'Ethane était d'avoir un contrôleur centralisé pour gérer les règles (policies) et la sécurité dans le réseau. Ethane utilise deux composantes : un contrôleur pour décider si un paquet doit être transmis, et un switch Ethane composé de table et d'une chaîne de communication entre les deux [1]. La figure ci-dessous montre le développement de réseau SDN dans le temps :

- 2007-2011: l'apparition du réseau SDN.
- 2012-2013: la maturité du réseau SDN.
- 2014-2015: l'automatisation du réseau SDN.
- 2015-2016 : Les déploiements de SDN ont commencé à progresser.
- 2016-2019: la généralisation du réseau SDN.
- 2020: l'économie du SDN.

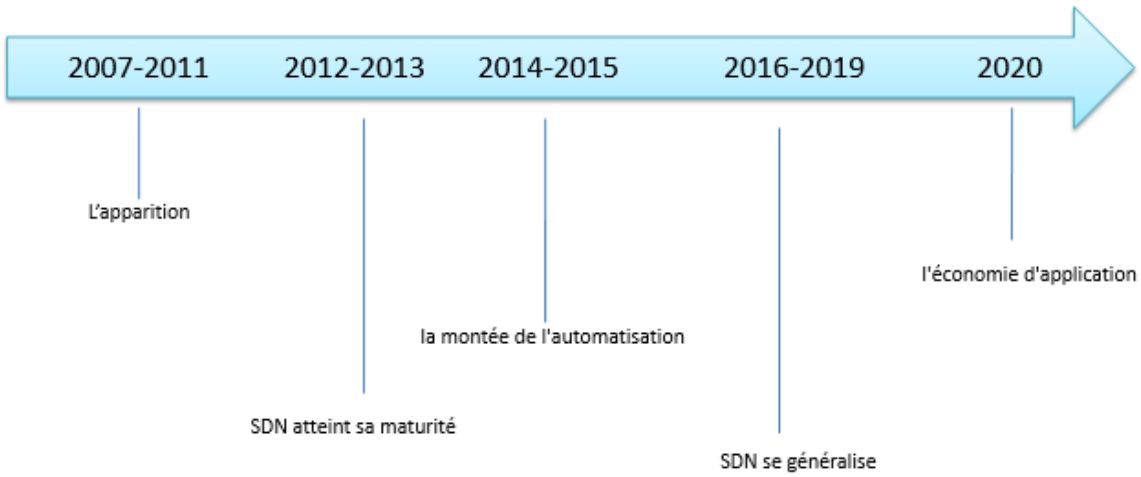


Figure 1.4 : le développement d'un réseau SDN.

Il y a une réalité SDN et cela conforte les analystes dans leurs prédictions : le métier d'administrateur réseau devrait prochainement changer [13]. Les chiffres suivants démontrent bien que l'engouement pour le SDN n'est pas juste une mode ou un simple intérêt technologique :

- Croissance des investissements depuis 2007
- 419 entreprises se proclament comme fournissant des solutions « SDN »

- Marché estimé à 35 milliards de dollars pour 2008

2.2. Architecture SDN

Le SDN présente une architecture réseau où le plan de contrôle est totalement découplé du plan de données, cela est illustré par la figure suivante :

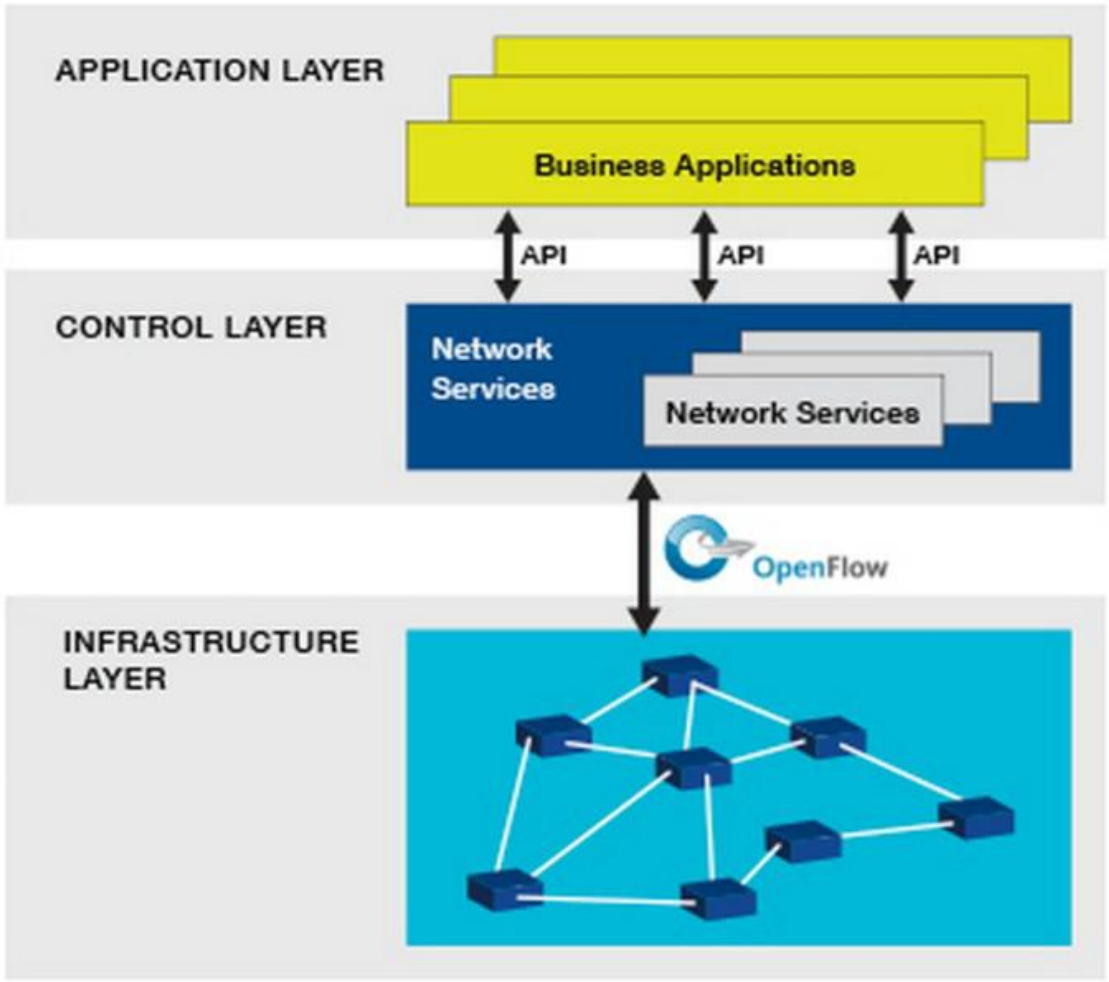


Figure 1.5 : Fonctionnement d'une structure d'équipement réseau utilisant la technologie SDN [15].

SDN se compose de 3 couches :

- **La couche infrastructure** (la couche de transmission) : c'est la couche la plus basse, elle contient les équipements de transmission (FEs : Forwarding Elements) tels que les switch physiques et virtuels. Son rôle principal est l'acheminement du trafic et qui supportent le protocole OpenFlow qu'ils partagent avec le contrôleur [12].
- **La couche de contrôle :** ces contrôleurs utilisent des interfaces southbound pour contrôler le comportement des FEs et communiquent via des APIs Northbound avec la couche supérieure pour superviser et gérer le réseau. C.à.d. offre une visibilité globale du réseau et des équipements d'infrastructure [12].

- **La couche applicative :** héberge les applications qui peuvent introduire de nouvelles fonctionnalités réseau, comme la sécurité, la configuration dynamique et la gestion et apporte l'automatisation à travers du réseau, et à l'aide des interfaces programmables alternative [12].

3. Les avantages du SDN et des réseaux programmables

- **Réseaux programmables**

Avec SDN, il est plus simple de modifier les stratégies réseau car il suffit de changer une politique de haut niveau et non de multiples règles dans divers équipements de réseau. De plus, la facilité de la conception et du contrôle du réseau, et surtout l'existence d'une structure de contrôle centralisée du trafic dans le réseau avec des connaissances globales et une puissance de calcul élevée, simplifient le développement de fonctions plus sophistiquées (centralisation et programmabilité) [16].

- **Flexibilité**

SDN apporte également une grande flexibilité dans la gestion du réseau. Il devient facile de rediriger le trafic, d'inspecter des flux particuliers, de tester de nouvelles stratégies ou de découvrir des flux inattendus. Donc il permet d'augmenter le taux d'innovation au niveau de l'infrastructure réseau, et cela conduit à l'apparition de nouvelles idées. Par exemple, les développeurs peuvent tester des applications au sein du réseau sans affecter les performances du réseau ou des services. Ce qui nous permet d'obtenir des performances plus élevées par rapport à celles qui existent actuellement [16].

- **Politique unifiée**

Avec son contrôleur, SDN garantit également une politique réseau unifiée et à jour, puisque le contrôleur est responsable de l'ajout de règles dans les commutateurs, il n'y a aucun risque qu'un administrateur de réseau ait oublié un commutateur ou installé des règles incohérentes entre les dispositifs [16].

- **Optimisation/Evolutivité**

Technologie à faible coût à comparer avec les équipements réseaux actuels que nous achetons fermés qu'il est impossible de développer ou de fusionner avec d'autres produits ou d'ajouter de nouveaux périphériques, (on devient indépendant du fournisseur), ainsi que de réduire le nombre d'ingénieurs, et gagner le temps de travail de plus de 50% c'est-à-dire réduire le temps de travail d'un mois à moins de 15 jours.

4. La comparaison entre réseau traditionnel et SDN

	Réseau SDN	Réseau traditionnel
Fonctionnalité	-Découple le plan de contrôle de celui du plan de données. -Offre un meilleur contrôle du réseau et la possibilité de le programmer.	-Le contrôle du réseau est complexe.
Configuration	-Configuration automatique à travers une centralisation du contrôle du réseau. -Optimisation de la configuration.	-Une configuration manuelle et la possibilité de faire des erreurs qui vont entrainer un comportement erroné du réseau.
Performances	-Contrôle global de l'information.	-Le problème de configuration statique
Innovation	-Implémentation facile de logiciels et des mises à jour dans le réseau. -Environnements de tests suffisants.	-Difficultés d'implémentation de logiciels et des mises à jour dans le réseau. -Environnements de tests limités.

Tableau 1.1 : Tableau comparatif entre le SDN et les réseaux traditionnels.

Conclusion

Est-ce que la virtualisation et le cloud computing résolvent toutes les problématiques ? Non, car il reste la problématique d'appliquer les règles réseau, d'où la sécurité n'est qu'un aspect parmi d'autres, notamment de Forwarding et de load balancing. C'est pourquoi le marché planche sur SDN.

Du coup dans ce chapitre, nous avons fourni une base théorique sur les réseaux définis par logiciels (SDN), en présentant l'architecture et les avantages de ce dernier.

Dans le chapitre suivant, nous présentons les différents composants d'un réseau SDN.

Chapitre 2 :

Les composants du SDN

Introduction

Le réseau défini par logiciel (SDN) modifie la façon dont les plans de contrôle (cerveau) et de transfert (corps) du réseau interagissent pour améliorer l'automatisation.

Le SDN est un concept devenu à la mode depuis quelques temps et la mise en réseau défini par logiciel consiste à mieux comprendre les composants importants qui constituent les réseaux définis par logiciel:

- Premièrement, les API orientées au sud transmettent les informations aux commutateurs et aux routeurs ci-dessous. Vous ne le savez peut-être pas, mais vous connaissez probablement déjà les API orientées vers le sud sous la forme d'OpenFlow. Considéré comme le principal standard SDN, OpenFlow est la première API en direction du sud et est un protocole fortement adopté.
- Deuxièmement, le contrôleur SDN agit en tant que «cerveau» du réseau. Il permet aux utilisateurs SDN d'avoir une vision centrale de l'ensemble du réseau et permet aux administrateurs réseau d'indiquer aux commutateurs et aux routeurs comment le plan de transfert doit diriger le trafic réseau.
- Enfin, les API orientées vers le nord transmettent les informations ci-dessus aux applications et à la logique métier, offrant ainsi aux administrateurs réseau la possibilité de structurer de manière pragmatique le trafic et de lancer les services [11]. (détailler dans le troisième chapitre)

Dans ce chapitre, nous traitons les différents composants d'un réseau SDN tel que protocole OpenFlow et les contrôleurs SDN.

1. Le protocole OpenFlow dans SDN

OpenFlow est un protocole de communication entre le contrôleur et le commutateur dans un réseau SDN. Publié par l'Open Networking Foundation (ONF) à Stanford. Il s'agit d'un standard ouvert utilisé par le contrôleur pour transmettre aux commutateurs des instructions qui permettent de programmer leur plan de données et d'obtenir des informations de ces commutateurs afin que le contrôleur puisse disposer d'une vue globale logique (abstraction) du réseau physique [12].

1.1 La genèse d'OpenFlow

L'histoire d'OpenFlow est intéressante et permet de mieux comprendre son rôle fondamental dans la conception de l'architecture SDN. OpenFlow a été initié comme un projet à l'université de Stanford lorsqu'un groupe de chercheurs exploraient la manière de tester de nouveaux protocoles dans

le monde IP (créer un réseau expérimental confondu avec le réseau de production) mais sans arrêter le trafic du réseau de production lors des tests.

C'est dans cet environnement que les chercheurs à Stanford ont trouvé un moyen de séparer le trafic de recherche du trafic du réseau de production qui utilise le même réseau IP. Ils ont découvert que bien que les constructeurs de matériel réseau concevaient leurs produits différemment, tous utilisaient des tables de flux (flow table) afin d'implanter les services réseau tels que les NATs, la QoS, les firewalls, etc. Par ailleurs, bien que l'implantation des tables de flux diffère entre ces constructeurs, les chercheurs ont découvert qu'ils pouvaient exploiter un ensemble de fonctions communes.

Le résultat de l'équipe de recherche à Stanford a été OpenFlow, qui fournit un protocole ouvert (open protocol) qui permet aux administrateurs de réseau de programmer les tables de flux (flow tables) dans leurs différents routeurs IP et commutateurs Ethernet dans un but (dans le cas de Stanford) de séparer le trafic de recherche du trafic de production, chacun avec son ensemble de fonctionnalités et caractéristiques de flux [12].

1.2 Canal OpenFlow (OpenFlow Channel)

Le canal OpenFlow est l'interface qui connecte chaque commutateur OpenFlow à un contrôleur. Cette interface permet au contrôleur de recevoir les messages du commutateur et de pouvoir le gérer à travers le réseau. Le canal doit être sécurisé afin d'assurer le bon déroulement des communications entre le commutateur et le contrôleur. Pour cela l'échange de message se fait au cours d'une session TCP (Transmission Control Protocol) établie via le port 6653 du serveur contrôleur ou à travers une connexion SSL/TLS (Secure Sockets Layer/ Transport Layer Security) [13].

1.3 Etablissement d'une connexion commutateur-contrôleur

Nous allons citer ci-dessous trois cas d'un établissement d'une connexion entre le commutateur OpenFlow et le contrôleur :

- **Cas n°1 : connexion établie**

Tout d'abord, il faut renseigner l'adresse IP du contrôleur au niveau du commutateur. Lors du démarrage du commutateur OpenFlow, ce dernier envoie un paquet « OFPT_HELLO » avec le numéro de version d'OpenFlow supportée. Le contrôleur vérifie la version d'OpenFlow supportée par le commutateur et lui répond par un message « OFPT_HELLO » en indiquant la version d'OpenFlow avec laquelle ils communiqueront. La connexion est ainsi établie.

La figure suivante présente les principales étapes de la connexion entre le contrôleur et le commutateur OpenFlow [14].

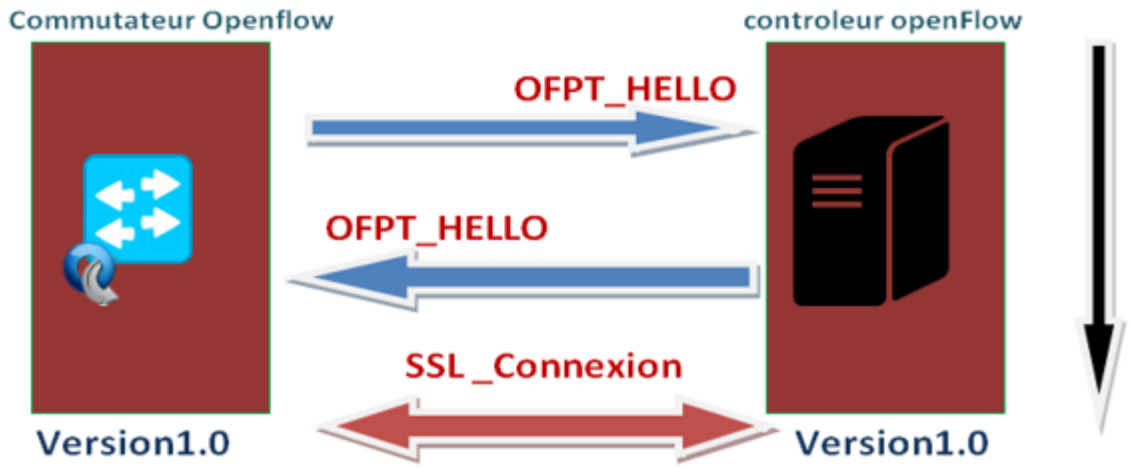


Figure 2.1: Connexion au contrôleur OpenFlow [14]

- **Cas n°2 : Échec de la connexion**

Comme dans le cas précédent le commutateur OpenFlow envoie un paquet «OFPT_HELLO» avec la version du protocole utilisé, le contrôleur s'aperçoit qu'il ne supporte pas la version OpenFlow du commutateur. Il lui retourne donc un paquet « OFPT_ERROR » en indiquant que c'est un problème de compatibilité, comme illustré dans la figure suivante [14] :

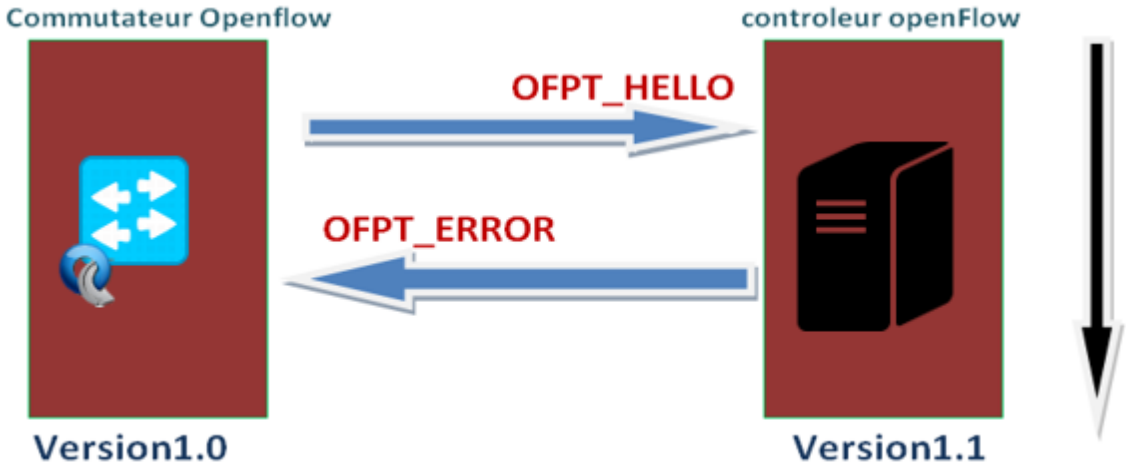


Figure 2.2 : Échec de la connexion au contrôleur OpenFlow [14].

- **Cas n°3 : Mode d'urgence**

Comme dans les cas précédents le commutateur envoie un paquet « OFPT_HELLO » au contrôleur, si celui-ci ne répond pas, il se met alors en mode urgence «EMERGENCY MODE». Le commutateur utilise sa table de flux par défaut, si toutefois un paquet ne correspond à aucun enregistrement dans la table il le supprime, comme le montre la figure ci-dessous [14] :

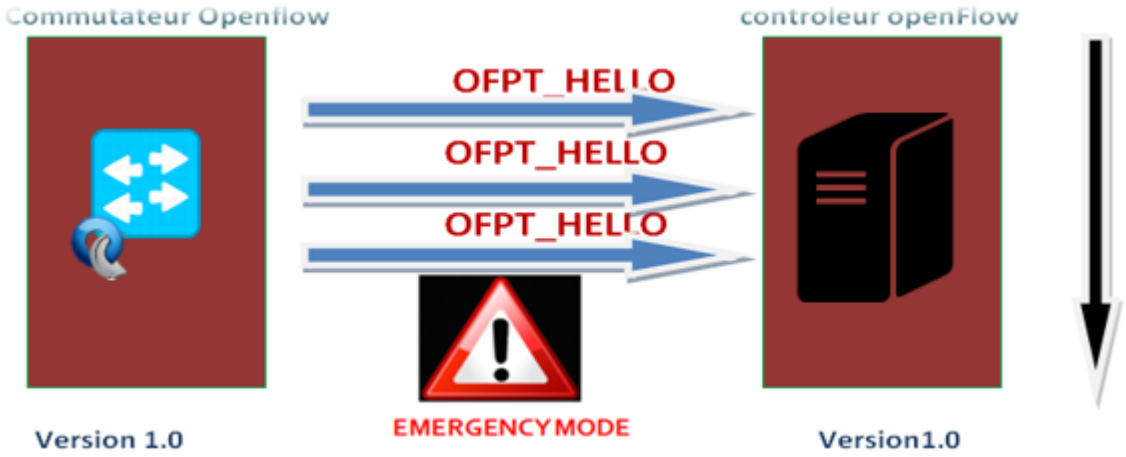


Figure 2.3: Mode d'urgence [14].

1.4 Structure d'un commutateur OpenFlow :

Les commutateurs contiennent des tables de flux qui sont utilisées pour effectuer des fonctions de transfert, indiquées dans les entêtes de paquets.

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.
- Fonction d'acheminement des données: déterminer le chemin de flux de données entrants à partir des règles définies par un contrôleur, ces règles indiquent pour des catégories de paquet données quel doit être le prochain saut sur la route [12].

Le comportement d'un switch OpenFlow est alors illustré par la figure 2.4 :

Quand un switch reçoit un paquet, il compare son en-tête avec les règles de la table des flux. Si le masque configuré dans le champ d'en-tête d'une règle correspond à l'en-tête du paquet, la règle avec la plus haute priorité est sélectionnée, puis le switch actualise son compteur pour cette règle. Finalement, le switch effectue les actions spécifiées par la règle sur le paquet (Ex : le switch envoie le paquet vers un port de sortie). Sinon, le switch notifie son contrôleur concernant ce paquet abrité dans

la mémoire tampon. Le switch encapsule le paquet utilisant un message PacketIn et l'envoie au contrôleur. Le contrôleur reçoit ce message et identifie l'action appropriée à ce paquet, il installe une ou plusieurs entrées dans le switch. Par la suite, les paquets tamponnés sont transmis selon les règles.

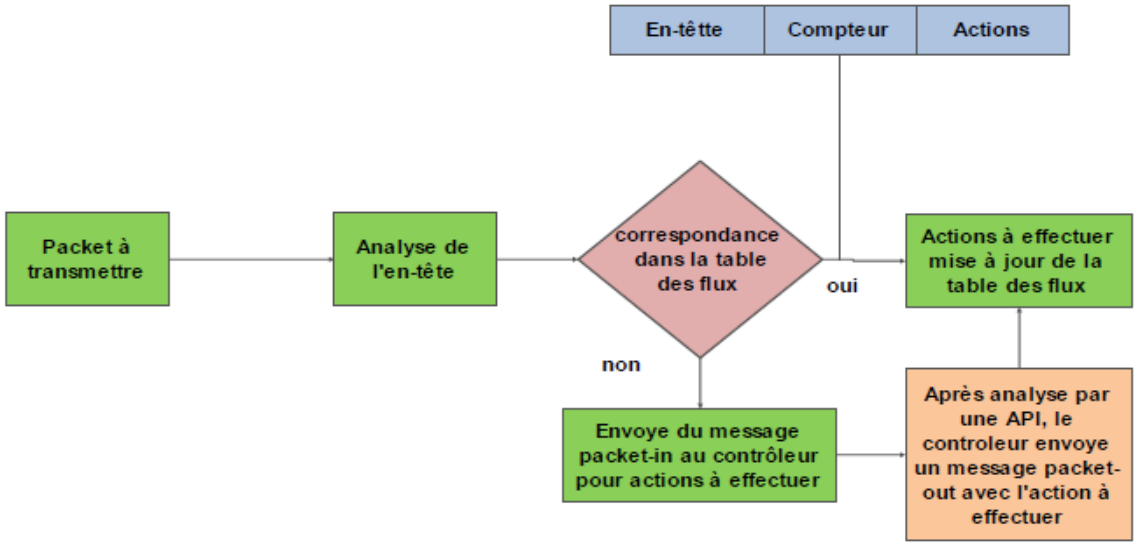


Figure 2.4 : Processus de transmission des paquets au sein des commutateurs Openflow [13].

S'il n'y a pas de correspondance dans la table de Flux :

- Le paquet est commuté vers le contrôleur via le canal sécurisé OpenFlow ou bien ;
- Le paquet sera abandonné ou bien ;
- Le paquet va continuer vers la table de Flux qui suit (Traitement Pipeline).

1.5 Table de flux :

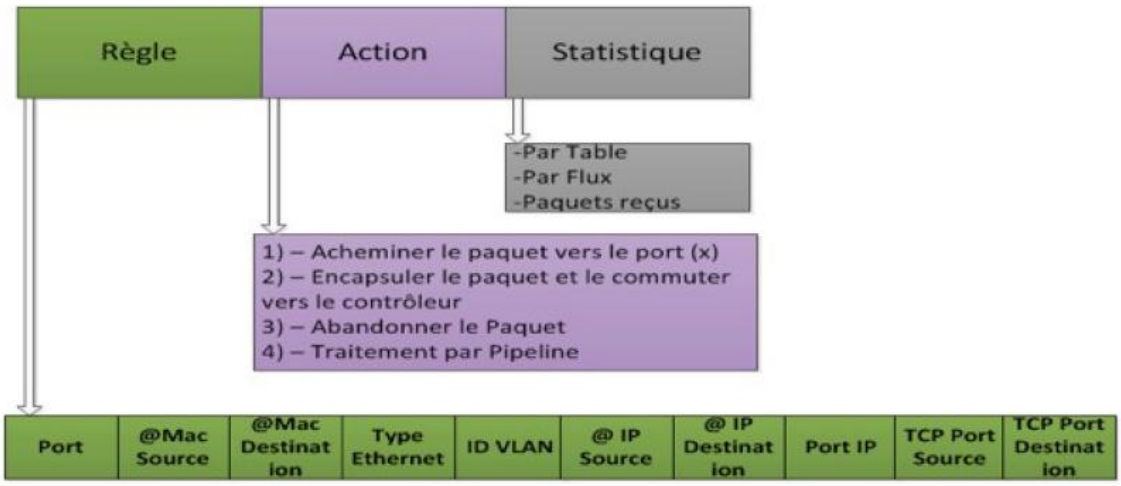


Figure 2.5 : Contenu d'entrées de flux [17].

Tel que le démontre la Figure 2.5 chaque table de flux du commutateur contient un ensemble d'entrées de flux qui sont composées de :

- **Le champ de correspondance (Match Fields ou règle) :** c'est la partie sur laquelle le contrôleur se base pour faire correspondre les paquets, cela consiste à vérifier le port d'entrée ou l'entête du paquet afin d'y appliquer une action X [17].
- **Compteurs** (Counters ou Statistiques) : Il est possible de disposer d'un certain nombre de statistiques. Dont on se sert pour la gestion des entrées dans les tables de flux, pour ensuite décider si une entrée de flux est active ou non [17].
- **Instructions** (Actions) : c'est une opération pouvant contenir en elle-même un ensemble d'actions à appliquer sur le paquet.

- Apply-Action : appliquer l'action spécifique immédiatement, sans aucun changement du Action set.
- Clear-Actions : Effacer toutes les actions dans l'action set immédiatement.
- Write-Actions : Fusionner les actions dans l'action set actuel, si une des actions du même type existe déjà dans l'action set actuel, l'écraser, autrement l'ajouter.
- Goto-Table : Indique la prochaine table de Flux dans le traitement pipeline (une ou plusieurs tables de flux) [17].

Action set :

Un ensemble d'actions associées qui s'accumulent au fur et à mesure que le paquet avance dans la chaine de traitement Pipeline est traité par chaque table de flux. Quand une instruction ne contient pas un **Goto-Table**, le traitement pipeline s'arrête et les actions dans le set actions sont exécutées [17].

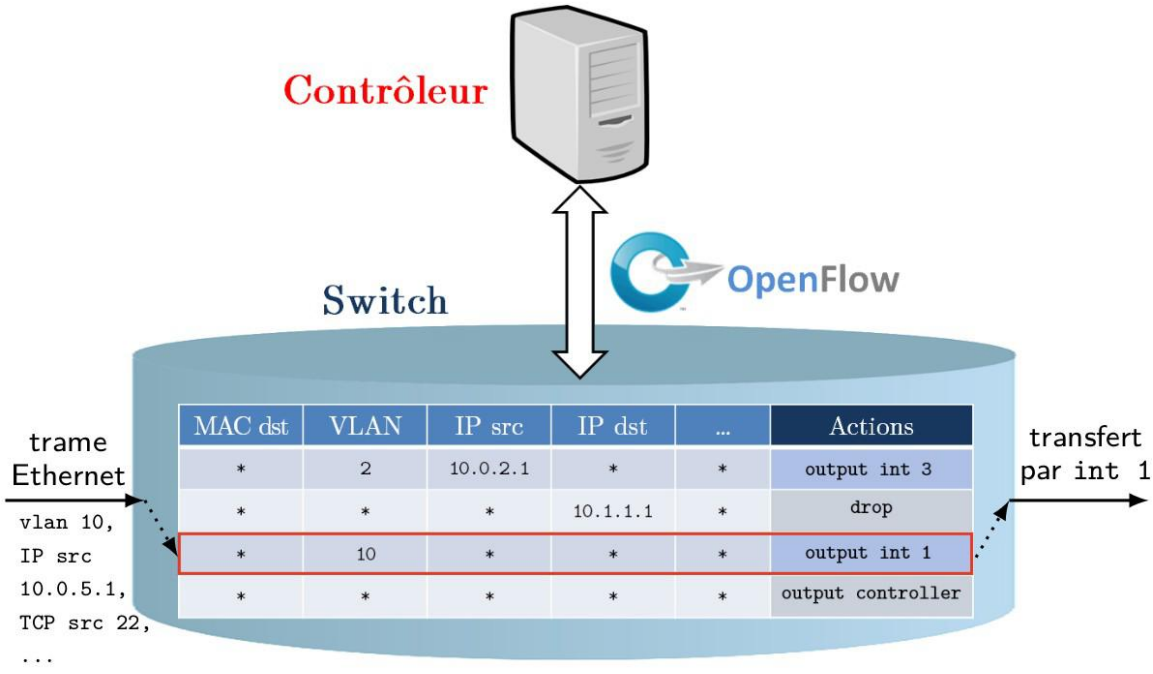


Figure 2.6 : Schématisation du modèle de flux OpenFlow [18].

La figure 2.6 illustre le traitement d'une trame parvenant à un switch OpenFlow contenant une seule table de flux : parmi les différents champs d'en-têtes de la trame, l'appartenance au VLAN 10 fait correspondre à un des flux du switch, dont l'action associée consiste en un transfert par l'interface 1[18].

1.6 Messages OpenFlow :

Le protocole de commutateur OpenFlow prend en charge trois types de message : contrôleur à commutateur, asynchrone et symétrique.

- Les messages symétriques :

Sont initiés par le commutateur ou le contrôleur et envoyés sans sollicitation. Par exemple, on trouve les messages HELLO qui sont échangés une fois que le canal sécurisé a été établi.

- Hello : les messages Hello sont échangés entre le commutateur et le contrôleur au démarrage de la connexion.
- Echo : les messages de demande / réponse d'écho peuvent être envoyés à partir du commutateur ou du contrôleur, et renvoient une réponse d'écho. Ils servent principalement à vérifier la qualité de vie d'une connexion contrôleur-commutateur.
- Erreur : le commutateur utilise les messages d'erreur pour signaler les problèmes à l'autre côté de la connexion. Ils sont principalement utilisés par le commutateur pour indiquer l'échec d'une requête initiée par le contrôleur [19].

- Les messages de contrôleur à 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 [6].

a) 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 [12].
- Le contrôleur utilise la paire de message FEATURES_REQUEST et FEATURES_REPLY afin d'interroger le commutateur au sujet des fonctionnalités qu'il supporte.
- La paire de message GET_CONFIG_REQUEST et GET_CONFIG_REPLY est utilisée afin d'obtenir la configuration du commutateur [12].

b) Les messages command from controller sont au nombre 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 [12].

- 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.

c) Des statistiques sont obtenues du commutateur par le contrôleur via la paire de message STATS_REQUEST et STATS_REPLY.

d) 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. 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 files d'attente spécifiques et ainsi fournir à ces paquets un niveau de QoS désiré [12].

e) 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 [12].

- La communication asynchrone :

Est initiée par le commutateur OpenFlow et est utilisée pour informer le contrôleur du passage des paquets ou du changement de l'état du commutateur.

- PacketIn : Pour tous les paquets qui ne disposent pas d'une entrée de flux correspondante ou dans le cas où l'action correspondante est l'envoi au contrôleur.
- Flow-Removed : informe le contrôleur de la suppression d'une entrée de flux d'une table de flux.
- Port-status : informe le contrôleur d'une modification sur un port.
- Role-status : informe le contrôleur d'un changement de son rôle. Lorsqu'un nouveau contrôleur choisit lui-même le maître, le commutateur est censé envoyer des messages d'état de rôle à l'ancien contrôleur maître.
- Flow-monitor : informe le contrôleur d'une modification dans un tableau de flux [19].

2. Contrôleurs SDN

Toute l'intelligence du réseau est externalisée dans un point logiquement centralisé appelé contrôleur SDN. Ce dernier offre une connaissance globale de l'infrastructure physique et des abstractions pour la configurer. Le contrôleur SDN est un composant programmable. Il peut être vu

comme un système d'exploitation qui expose une *application programming interface* (API) « Northbound API » pour spécifier des applications de contrôle pour les réseaux programmables [20].

Le contrôleur peut ajouter, modifier ou supprimer des règles dans les tableaux de flux, à travers l'API OpenFlow, d'une façon réactive (comme réponse à l'arrivée d'un paquet) ou proactive (installer les règles à l'avance dans le commutateur) [21]. Ce qui devrait être mis en évidence est que l'analyse comparative et la sélection d'un contrôleur est considérée de nos jours vraiment difficiles car il est un problème à multiples facettes. Le contrôleur le plus approprié peut être sélectionné que si un ensemble spécifique d'exigences est pris en considération [22].

Trois types d'interfaces permettent aux contrôleurs de communiquer avec leur environnement : interface Sud, Nord et Est/Ouest. Nous les détaillons par la suite.

2.1 Quelques Contrôleurs SDN

- ✓ **NOX**

NOX est le premier contrôleur SDN. Il a été initialement développé par Nicira Networks et a été le premier à soutenir le protocole OpenFlow. C'est Open-source et écrit en C ++.

Aujourd'hui est considéré comme inactif NOX (il n'y a pas eu de changements majeurs), il est devenu la base de nombreuses solutions de contrôleurs SDN ultérieures [16].

- ✓ **POX**

POX est la version la plus récente, en Python de NOX, fournit un cadre pour le développement et le test d'un contrôleur OpenFlow, mais les performances POX sont nettement inférieures à celles des autres contrôleurs et ne convient donc pas au déploiement d'entreprise [16].

- ✓ **Beacon**

Beacon est un contrôleur OpenFlow développé en Java et publié en 2010. Il a été utilisé dans plusieurs projets de recherche, considéré comme plus approprié pour les réseaux d'entreprise et de centres de données.

Aujourd'hui, Beacon est considéré comme inactif, Il utilise une approche statique dans laquelle un nombre fixe de commutateurs sont affectés à un thread de travail [22], ce contrôleur a également été utilisé dans d'autres projets tels que Floodlight ou Opendaylight.

- ✓ **OpenDaylight**

Le 8 avril 2013, la fondation open source OpenDaylight (ODL), a été annoncée comme faisant partie de la Linux Foundation et dirigée par IBM et Cisco. Ce contrôleur est basé sur Java et est dérivé de la conception originale de Beacon. Il prend en charge OpenFlow et d'autres API sud et inclut des fonctionnalités essentielles.

ODL est une plate-forme ouverte modulaire pour la personnalisation et l'automatisation de réseaux de toute taille et d'échelle. Le projet ODL est né du mouvement SDN, avec une orientation claire sur programmabilité du réseau. ODL peut également être utilisé pour obtenir une administration centralisée du réseau. Jusqu'à ce jour, la communauté ODL a introduit plusieurs versions, visant à améliorer le cadre grâce à des fonctionnalités supplémentaires, de restructuration composant existant.

ODL comprend un support pour le plus large ensemble de protocoles à toute plate-forme SDN. Par exemple, OpenFlow, ainsi que des protocoles traditionnels, y compris NETCONF, le BGE, ODL est le contrôleur qui est compatible avec la plus grande variété de protocoles [22].

- ✓ **Floodlight**

Il est sous licence Apache et est l'une des contributions importantes de Réseaux Big Switch à la communauté open source écrit en java [22]. Floodlight est l'un des contrôleurs SDN les plus populaires prenant en charge les commutateurs physiques et virtuels compatibles OpenFlow. Est basé sur un contrôleur de bacon de l'Université de Standford. L'architecture Floodlight est modulaire [24].

Le tableau 2 montre la comparaison basée sur les fonctionnalités des contrôleurs SDN open source les plus populaires [23].

	Prog language	**Gui**	**Doc**	**Modularity**	**Distributed/ Centralized**	**Platform support**	**prod**	**Southbound APIs**	**Northbound APIs**	**Partener**	**Multithreading support**	**Open-stack support**
ONOS	Java	Web Based	Good	High	D	Linux, MAC OS, And Windows	fair	OF1.0,1.3, NETCONF	REST API	ON.LAB, AT&T, Ciena, Cisco,Ericsson, Fujitsu,Huawei, Intel, NecNsf. Ntt Comunnication ,Sk Telecom	Y	N
Open-Day- Light	Java	Web Based	Very good	High	D	Linux, MAC OS, And Windows	fair	OF1.0,1.3, 1.4, NETCONF/ YANG, OVSDB, PCEP, BGP/LS, LISP, SNMP	REST API	Linux Foundation With Memberships Covering Over 40 Companies, Such As Cisco, IBM, NEC	Y	Y
NOX	C++	Python + QT4	Poor	Low	C	Most Supported On Linux	fair	OF 1.0	REST API	Nicira	NOX_MT	N
POX	Python	Python + QT4	Poor	Low	C	Linux, MAC OS, And Windows	high	OF 1.0	REST API	Nicira	N	N
RYU	Python	Yes	Fair	Fair	C	Most Supported On Linux	high	OF 1.0,1.2, 1.3, 1.4, NETCONF, OFCONFIG	REST For South bound	Nippo Telegraph And Telephone Corporation	Y	Y

Beacon	Java	Web Based	Fair	Fair	C	Linux, MAC OS, And Windows	fair	OF 1.0	REST API	Standford University	Y	N
Maestro	Java	-	Poor	Fair	C	Linux, MAC OS, And Windows	fair	OF 1.0	REST API	RICE, NSF	Y	N
Flood-Light	Java	Web/ Java Based	Good	Fair	C	Linux, MAC OS, And Windows	fair	OF 1.0, 1.3	REST API	Big Switch Networks	Y	N
Iris	Java	Web Based	Fair	Fair	C	Linux, MAC OS, And Windows	fair	OF 1.0, 1.3, OVSDB	REST API	ETRI	Y	N
MUL	C	Web Based	Fair	Fair	C	Most Supported On Linux	fair	OF 1.4,1.3, 1.0, OVSDB, OFCONFIG	REST API	Kulcloud	Y	Y
Runos	C++	Web Based	Fair	Fair	D	Most Supported On Linux	fair	OF 1.3	REST API	ARCCN	Y	N
Lib-Fluid	C++	-	Fair	Fair	-	Most Supported On Linux	fair	OF 1.0,1.3	-	ONF	Y	N

Tableau 2.2 : comparaison entre les contrôleurs [23].

2.2 Interfaces de communications

Le contrôleur interagit avec les autres couches à travers les interfaces Sud et Nord et avec les autres contrôleurs à travers une interface Est-Ouest.

- ✓ **Interfaces Sud :**

Ce sont des interfaces de communication permettant au contrôleur d'interagir avec les switches /routeurs de la couche d'infrastructure. Le protocole le plus utilisé et déployé comme interface de Sud est OpenFlow, c'est un élément fondamental pour construire des solutions SDN.

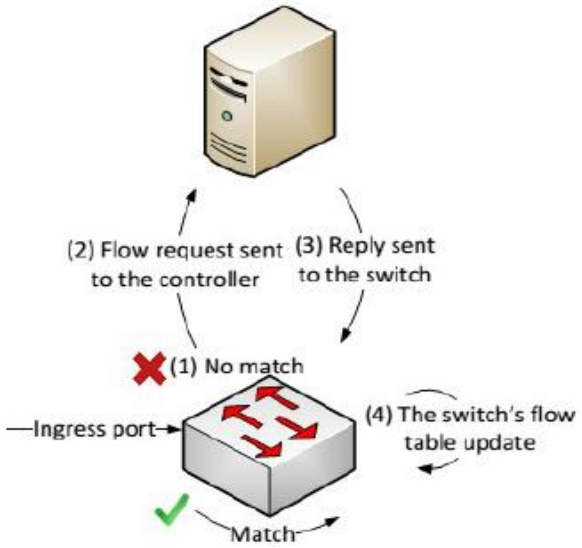


Figure 2.7 : Processus de communication entre les commutateurs et le contrôleur [1].

Quand un paquet est reçu par le commutateur, son champ d'entête est examiné et comparé avec le champ 'Match' dans les entrées de la table de flux. Si le match est identifié, le commutateur exécute l'action correspondante dans la table de flux. Par contre, s'il n'y a pas de match (1), une demande est envoyée au contrôleur (2) sous la forme d'un 'Packet_in', puis le contrôleur décide selon sa configuration une action pour ce paquet, et envoie une nouvelle règle de transmission sous la forme d'un 'Packet_out' et 'Flow-mod' au commutateur (3). Enfin, la table de flux du commutateur est actualisée (4) [1].

- ✓ **Interfaces Nord :**

L'interface nord entre le plan applicatif et le contrôleur a pour objectif de faire transiter les informations provenant des besoins des applications pour que le contrôleur puisse ouvrir le meilleur réseau logiciel avec les bonnes qualités de services, la sécurité adéquate et la gestion nécessaire pour que les opérations puissent s'exécuter sans problèmes. Le protocole de base pour réaliser ces transmissions est fondé sur l'API REST [1].

- ✓ **Est/Ouest :**

Les interfaces côté Est/Ouest sont des interfaces de communication qui permettent généralement la communication entre les contrôleurs dans une architecture multi-contrôleurs pour synchroniser les états du réseau [1].

Conclusion

Au cours de ce chapitre, nous avons traité les composants essentiels de cette solution afin d'appliquer ses concepts à notre contexte.

Le prochain chapitre de ce mémoire sera réservée à notre propre travail qui sera scindé en deux chapitres contenant chacun une contribution.

Chapitre 3 :

Evaluation des performances de Floodlight et OpenDaylight

Introduction

Software Defined Networking (SDN) est une architecture émergente qui découple plan de contrôle des réseaux et des données physiques. Le contrôleur constitue le «cerveau» du réseau, et construit des réseaux intelligents et flexibles.

Ce chapitre représente la première contribution de notre travail, nous commençons d'abord par une présentation et une comparaison entre les contrôleurs Floodlight et Opendaylight puis nous présentons l'émulateur Mininet et OpenVswitch et à la fin une évaluation de performance des contrôleurs.

1. Contrôleur SDN

1.1. Floodlight

L'une des contributions majeures de Big Switch Networks à la communauté open source, est un contrôleur OpenFlow basé sur Java et sous licence Apache nommé Floodlight. L'architecture de ce contrôleur repose sur Big Network Controller (BNC).

Comme de nombreux autres contrôleurs, lorsque vous exécutez Floodlight, les opérations du contrôleur orientées vers le nord et le sud deviennent actives. Autrement dit, lorsque vous exécutez Floodlight, le contrôleur et l'ensemble des applications de module configurées s'exécutent également. API (Application Programming Interface) REST (representational state transfer) nordiques exposées par tous les modules en cours deviennent disponibles via le port REST spécifié. Toute application peut interagir (récupérer des informations et invoquer des services) avec le contrôleur en envoyant des commandes HTTP REST. En revanche, en direction du sud, le module fournisseur de Floodlight commencera à écouter sur le port Transmission Control Protocol (TCP) spécifié par OpenFlow les connexions provenant des commutateurs OpenFlow.

Le terme «architecture modulaire» est utilisé pour décrire l'architecture du contrôleur Floodlight. L'architecture de base comprend divers modules, ce module traduit également les messages OpenFlow en événements Floodlight. Floodlight propose également des exemples d'applications, notamment un firewall [25]. (Voir annexe 1).

1.2. OpenDaylight

OpenDaylight (ODL) est un Framework open source pour la migration vers une architecture de réseau SDN. Il a été déployé dans des réseaux de centres de données, d'entreprises et d'opérateurs. ODL

est une plate-forme modulaire permettant de personnaliser et d'automatiser des réseaux de toutes tailles et de toutes échelles. Il fournit l'abstraction, la programmabilité et l'ouverture qui ouvrent la voie à une infrastructure intelligente, définie par logiciel. ODL fournit une plate-forme commune flexible qui sous-tend une grande variété d'applications [26]. (Voir annexe 2).

1.3. Comparaison entre Floodlight et Opendaylight :

Caractéristiques	Floodlight	OpenDaylight
Développé par	Big Switch Networks	Linux Foundation
Soutenu par	Big Switch Networks	Cisco, HP, IBM, Juniper, VMWare, etc.
Ecrit en Langage	Java	Java
Langages supportés	Java, Python	Java
Open source	Oui	Oui
Interface utilisateur	Web	Web
Virtualisation	Mininet, Open vSwitch	Mininet, Open vSwitch
Interface	southbound (OpenFlow), northbound (Java, REST)	southbound (OpenFlow et autres protocoles), northbound (Java RPC)
Plateforme	Linux, Mac, Windows	Linux, Windows
Documentation	Site officiel	Moyenne

Tableau 3.1 : Comparaison entre Floodlight et Opendaylight.

2. Émulateur du réseau Mininet

Mininet est un *émulateur de réseau* qui crée un réseau d'hôtes virtuels, de commutateurs, de contrôleurs et de liens [27]. Autrement dit, Mininet est un émulateur d'environnement SDN, développé par l'université de Stanford, et peut être utilisé pour construire des réseaux virtuels [1].

Mininet prend en charge la recherche, le développement, l'apprentissage, le prototypage, les tests, le débogage et toute autre tâche pouvant bénéficier d'un réseau expérimental complet sur un ordinateur [27].

Mininet :

- Fournit un banc d'essai réseau simple et peu coûteux pour le développement d'applications OpenFlow
- Permet à plusieurs développeurs simultanés de travailler indépendamment sur la même topologie.
- Permet d'effectuer des tests de topologie complexes sans avoir besoin de câbler un réseau physique
- Inclut une CLI sensible à la topologie et à OpenFlow pour le débogage ou l'exécution de tests à l'échelle du réseau.
- Prend en charge les topologies personnalisées et inclut un ensemble de base de topologies paramétrées.
- Est utilisable hors de la boîte sans programmation, mais fournit également une API Python simple et extensible pour la création et l'expérimentation de réseaux [27].

Le tableau ci-dessus liste quelques fonctionnalités de l'émulateur mininet.

Général	**MININET**
Logiciel gratuit	Oui
Open source	Oui
publiquement téléchargeable	Oui
Prise en charge Windows	Non
Prise en charge Linux	Oui
Entièrement fonctionnelle IOS	Non
Mode simulation	Non
Mode d'émulation	Oui
Compatible avec le monde réel Contrôleurs	Oui
Performance exactitude des résultats	Dépend des ressources
soutien IUG	Oui

Tableau 3.2 : fonctionnalité de mininet.

Les hôtes crées par Mininet exécutent un logiciel standard du réseau Linux (standard Linux network software), les commutateurs sont de type Open vSwitch qui supportent le protocole OpenFlow et SDN, via :

- Ligne de commande,
- Interface interactive,
- Script Python.

(Plus de détails dans la partie annexe 3).

3. Open vSwitch

C'est un composant essentiel dans notre émulation, Open vSwitch (OVS) est un outil très utilisé dans le monde SDN. C'est un commutateur virtuel sous licence open source Apache 2.0 que l'on utilise pour interconnecter des machines virtuelles [28].

Open vSwitch est donc une implémentation logicielle d'un switch Ethernet. Concrètement, il est constitué d'un service (**ovs-vswitchd**) et d'un module kernel (**openvswitch_mod**). Le service permet de commuter effectivement les paquets vers les bons ports virtuels, alors que le module kernel permet de capturer le trafic provenant des interfaces réseau [29]. Pour fonctionner comme n'importe quel switch, Open vSwitch utilise la notion de ports. Chaque port est constitué d'une ou plusieurs interfaces, qui correspondent à des interfaces du système hôte (logiques ou physiques). Il présente plusieurs fonctionnalités d'un commutateur couche 2 et même d'un commutateur couche 3, par exemple : [30]

- Tag de VLANs (norme 802.1Q).
- Protocole spanning tree.
- Qualité de Service (QoS).

Il supporte aussi le protocole OpenFlow.

Interagir avec Open vSwitch

✓ Pour connaitre l'état global d'un switch :

```
ovs-vsctl show
```

✓ La création d'un switch se fait par la commande :

```
ovs-vsctl add-br <virtual name of switch>
```

✓ Pour ajouter un flux directement dans la table de flux sans avoir à utiliser un contrôleur, nous utilisons la commande :

```
dpctl
mininet> dpctl add-flow in_port=1,actions=output:2
mininet> dpctl add-flow in_port=2,actions=output:1
```

Cette commande permet de transférer les paquets arrivant à port 1 vers port 2 et vice versa.

✓ Pour vérifier le contenu de la table de flux :

```
dpctl dump-flows
```

4. Évaluation des performances de Floodlight et OpenDaylight

Le contrôleur dans le plan de contrôle est l'élément fondamental utilisé pour toutes les opérations de gestion du plan de données. Par conséquent, les performances et les capacités du contrôleur lui-même sont extrêmement importantes [31]. L'efficacité du contrôleur est un terme utilisé pour désigner les différents paramètres tels que la performance, l'évolutivité, la fiabilité et la sécurité qui caractérisent un contrôleur. Presque toutes les études existantes se concentrent sur un ou plusieurs de ces paramètres.

La performance est l'attribut le plus souvent testé d'un contrôleur SDN, les indicateurs de performance les plus courantes pour un contrôleur SDN dans la littérature sont le débit et la latence. Le débit est le nombre de transactions par seconde qu'une application peut gérer (Quantité fournie dans un temps donné). La latence est le temps nécessaire pour un paquet pour arriver à sa destination à travers le réseau. L'objectif est d'obtenir un débit maximal et une latence minimale.

De plus, les outils utilisés pour évaluer leurs performances doivent être précis et efficaces pour mesurer différents paramètres d'évaluation. Dans la plupart des cas l'émulation et l'expérimentation sont utilisés comme techniques d'évaluation des performances [22]. Bien que les bancs d'essai de matériel fournissent des mesures plus proches des valeurs réelles dans un environnement de production, leur coût est toutefois considérable pour le milieu de la recherche. Par conséquent, les évaluations basées sur l'émulation sont une pratique courante [31].

L'étalonnage du contrôleur est une procédure couramment utilisée pour l'analyse des performances des contrôleurs SDN. Et aussi un outil ou une méthode utilisée pour effectuer une comparaison.

Analyse comparative

Comme déjà mentionné, l'analyse comparative est une procédure couramment utilisée pour l'analyse des performances des contrôleurs SDN dans différents scénarios de réseau. D'une manière générale, l'analyse comparative est définie comme l'acte de l'exécution d'un programme d'ordinateur ou d'autres opérations, afin d'évaluer la performance d'un objet en se basant sur du matériel ou du logiciel. Dans le contexte des contrôleurs SDN, l'étalonnage a été utilisé par les chercheurs pour évaluer les performances du contrôleur.

Afin d'évaluer les performances de plusieurs contrôleurs SDN, notamment leur évolutivité, leur efficacité et leur robustesse, des travaux ont été menés utilisant des techniques, des outils et des bancs d'essais. La majorité de ces travaux utilisent l'outil Cbench pour évaluer les performances en fonction de la latence et du débit [31].

Cbench (controller benchmarker) est un programme permettant de tester les contrôleurs OpenFlow en générant des événements d'entrée de paquet pour les nouveaux flux. Cbench émule un groupe de commutateurs qui se connectent à un contrôleur, envoient des messages entrant dans les paquets et surveillent les modulations de flux qui sont abaissées. Si on pense qu'un changement apporté peut avoir un impact sur les performances, il peut être utile d'utiliser Cbench pour le mesurer [32].

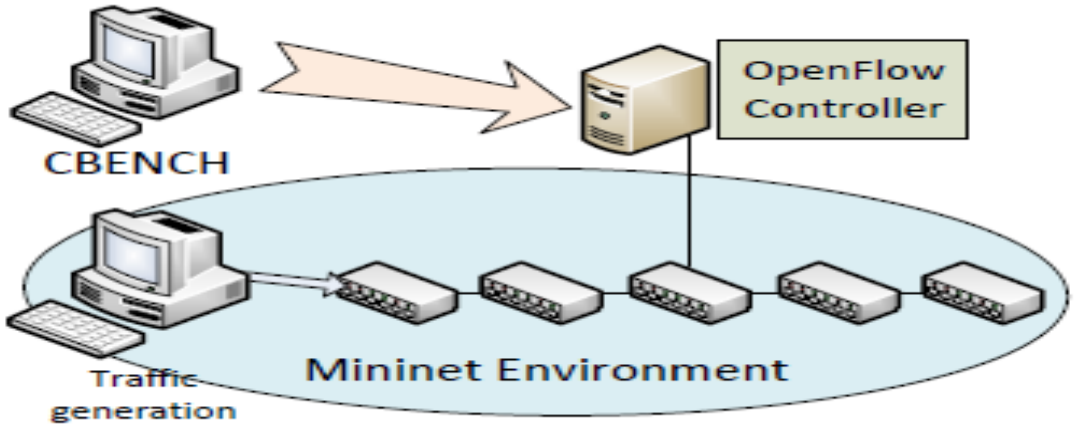


Figure 3.1 : Scénario virtuel de test [33].

La figure 3.1 montre un exemple du scénario virtuel de test sur lequel le serveur a un contrôleur OpenFlow en cours d'évaluation et installé. Dans l'hôte, il exécute le logiciel de référence Cbench pour insister sur la capacité du contrôleur. Il existe également un autre hôte capable de générer du trafic virtuel sur le réseau à l'aide du Mininet. Par contre, dans notre modèle l'évaluation s'est faite dans un environnement d'émulation utilisant un seul hôte exécutant l'outil Cbench. Cet environnement était approprié pour obtenir les résultats de test [33].

CBench teste les performances en envoyant des messages asynchrones. Pour la latence, les messages sont en série, c'est-à-dire qu'il envoie un paquet sous forme de message au commutateur émulé et attend une réponse avant d'envoyer le suivant. Nous exécutons plusieurs itérations avec un nombre variable de commutateurs émulés pour observer l'impact des commutateurs sur le contrôleur. D'autre part, avec les mêmes paramètres, nous testons le débit du contrôleur en cours d'exécution. Cependant, les paquets ne sont pas envoyés en série et les demandes sont envoyées sans attendre de réponse. Une exécution, CBench génère les messages de flux qu'un contrôleur peut gérer par seconde [31].

Afin de réaliser ces tests sur les contrôleurs SDN, nous avons utilisé un laptop Dell, équipé d'un processeur Intel Core i3 et qui utilise Ubuntu comme système d'exploitation. Nous avons également installé dans cette machine Floodlight et Opendaylight. Nous avons par la suite exécuté l'outil Cbench.

Nous avons testé le débit et la latence des contrôleurs avec différents nombres de commutateurs : 8, 16 et 32. On a supposé que chaque commutateur supporte 100 @MAC. Chaque test a été exécuté pour un minimum de 10 Minutes et répété dix (10) fois mais nous n'avons conservé que le résultat des dernières boucles, car les deux premières boucles peuvent être considérées comme un échauffement pour le contrôleur.

4.1. Tests de débit

Nous avons testé la performance du débit du contrôleur avec la ligne de commande suivante :

```
$ cbench -c 127.0.0.1 -p 6653 -m 10000 -l 3 -s 16 -M 100 -t
```

Floodlight

A partir des résultats présentés dans la figure 3.2, nous avons observé qu'avec 8 commutateurs, Floodlight produit en moyenne de 47258 réponses /sec par commutateur. L'augmentation du nombre de commutateurs a entrainé une légère augmentation du nombre de réponses, de 47258 à 8 commutateurs à 54219 avec 16 commutateurs et à 61957 avec 32 commutateurs.

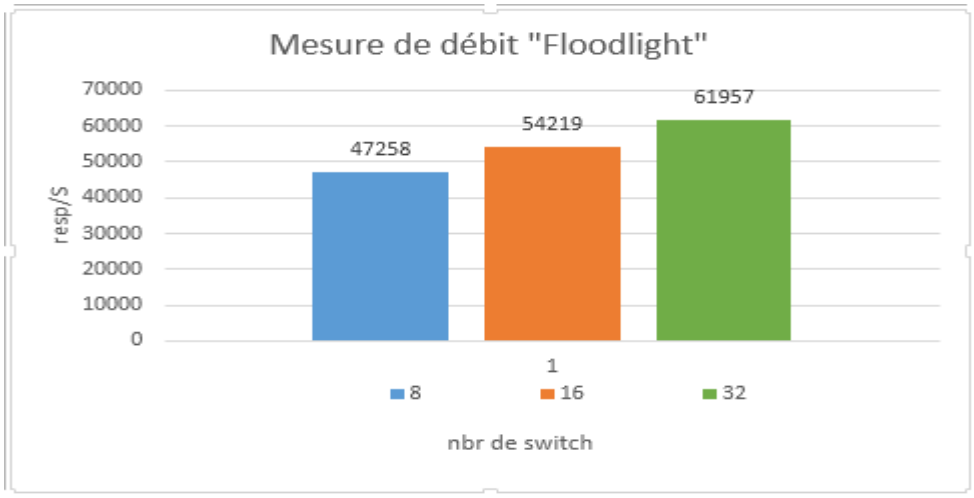


Figure 3.2 : Mesure de débit « Floodlight »

(Voir dans annexe 4 les résultats de test).

Opendaylight

Le Cbench a échoué à produire un résultat dans plusieurs tests et a montré des réponses nulles. Les réponses moyennes par commutation enregistrées sur le contrôleur Opendaylight étaient respectivement de 0, 0 et 0 pour 8, 16 et 32 commutateurs. Ces résultats sont présents dans la figure 3.3.

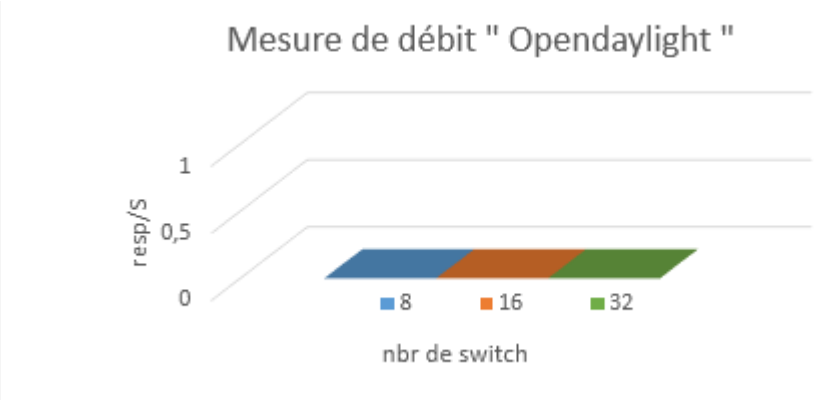


Figure 3.3 : Mesure de débit « ODL»

4.2. Tests de latence

A cet effet, nous avons exécuté Cbench en mode latence qui envoie un PacketIn au contrôleur et attend une réponse avant d'envoyer un autre paquet.

La ligne de commande utilisée pour mesurer la latence est la suivante :

```
$ cbench -c 127.0.0.1 -p 6653 -m 10000 -l 3 -s 16 -M 100
```

Floodlight

La figure ci-dessous montre que le Cbench a réussi à produire des résultats dans plusieurs tests, nous avons observé qu'avec 8 commutateurs, Floodlight produit en moyenne 5653 réponses /sec par commutateur. L'augmentation du nombre de commutateurs a entrainé une légère augmentation du nombre de réponses, de 5653 à 8 commutateurs à 5905 avec 16 commutateurs et à 6131 avec 32 commutateurs.

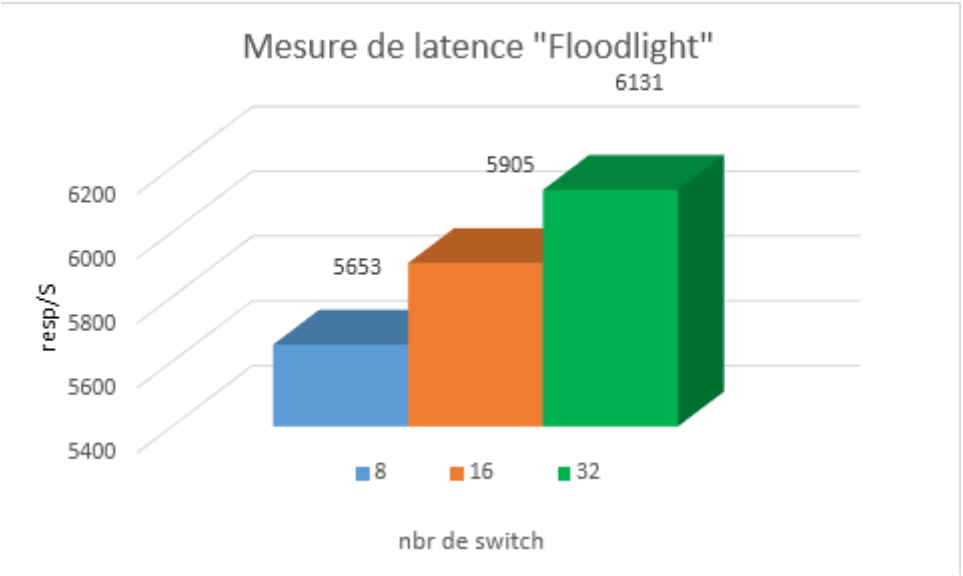


Figure 3.4 : Mesure de latence « Floodlight »

Opendaylight

Le Cbench a échoué à produire un résultat dans plusieurs tests et a montré des réponses nulles. Les réponses moyennes par commutation enregistrées sur le contrôleur Opendaylight étaient respectivement de 0, 0 et 0 pour 8, 16 et 32 commutateurs. Ces résultats sont présents dans la figure 3.5.

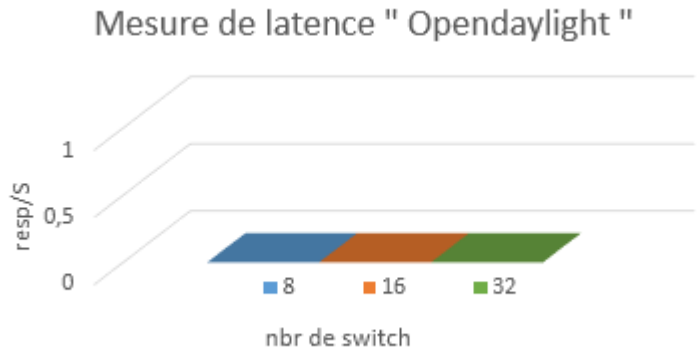


Figure 3.5: Mesure de latence « ODL »

4.3. Discussions

Dans les tests en mode latence : D'après les résultats, nous avons observé que le contrôleur Floodlight fournit des résultats assez cohérents avec les études menées précédemment. Par rapport à Floodlight, le nombre de réponses sur ODL était très faible de manière inattendue.

Dans les tests en mode débit : L'ODL a de nouveau fourni des réponses très faibles par rapport à Floodlight.

Selon les résultats, nous avons remarqué qu'Opendaylight a plusieurs problèmes de performance. Nous avons également constaté que de nombreux tests Opendaylight ne produisaient aucun résultat. C'est évident qu'un grand nombre de tests échouent si nous augmentons le nombre de commutateurs, la raison est que le protocole OpenFlow utilisé avec Opendaylight fonctionne mal et nous avons lu des articles pour confirmer ces résultats. Suite à ces résultats d'émulation, nous avons opté pour le choix du contrôleur Floodlight.

Conclusion

Nous avons présenté dans ce chapitre les différents outils utilisés pour déployer un réseau SDN, que ce soit le contrôleur, l'émulateur mininet ou l'Open vSwitch. Ainsi qu'une analyse comparative des contrôleurs OpenDaylight et Floodlight en utilisant l'outil cbench. Nous avons constaté que le benchmarking d'OpenDaylight avec Cbench n'a pas beaucoup de succès. Il semble qu'ODL pose plusieurs problèmes de performances.

En effet, Floodlight est un contrôleur doté de nombreuses fonctionnalités, offrant une grande modularité et la capacité de prendre en charge un large éventail d'applications. De plus, il est soutenu par les communautés les plus connues, dans le but de diriger la transformation vers les réseaux du futur.

Dans la suite, nous allons démontrer comment mettre en place un réseau SDN, en utilisant Floodlight, et nous allons tester les fonctionnalités réseau qui existent dans Floodlight.

Chapitre 3 :

Firewall et Access Control List dans le SDN

Introduction

Floodlight est un contrôleur SDN-OpenFlow qui est livré avec des applications intégrées. Un ensemble de fonctionnalités communes est utilisé par Floodlight pour sonder et contrôler un réseau OpenFlow, tandis que les applications qui s'y ajoutent offrent différentes fonctionnalités pour répondre aux différents besoins des utilisateurs sur le réseau. L'architecture Floodlight comprend trois composants interdépendants : le contrôleur Floodlight, les applications construites sous forme de modules Java compilés avec Floodlight et les applications construites sur l'API REST Floodlight. Java a été utilisé pour développer des modules de gestion de réseau et les a compilés avec le système de chargement de module de contrôleur de Floodlight [34].

1. Northbound

Un réseau défini par logiciel a été proposé et son idée principale est la centralisation du plan de contrôle dans un contrôleur SDN. Et la séparation du plan de contrôle du plan de données qui communiquent entre eux en utilisant un protocole appelé OpenFlow. Avec sa séparation et sa centralisation, le contrôleur SDN fournit des services réseau utiles via des interfaces orientées vers le nord et expose des API ouvertes permettant aux développeurs de mettre en œuvre des applications SDN innovantes. Pour cette raison, il est beaucoup plus facile de gérer de manière flexible et dynamique le réseau SDN que le réseau traditionnel. De plus, afin de permettre un accès facile aux fonctionnalités du réseau, les contrôleurs SDN actuels fournissent des services RESTful en tant que service Web externe. Ainsi, les administrateurs réseau peuvent utiliser les services de base et diverses fonctionnalités réseau du contrôleur SDN en appelant des API REST (via protocole HTTP) au lieu de programmer une logique complexe dans le contrôleur. Techniquement, la tâche de programmation compliquée et fastidieuse est simplifiée aux opérations CRUD des méthodes HTTP (c'est-à-dire, GET, POST, etc.), afin que les administrateurs puissent facilement interroger et manipuler les états du réseau [35].

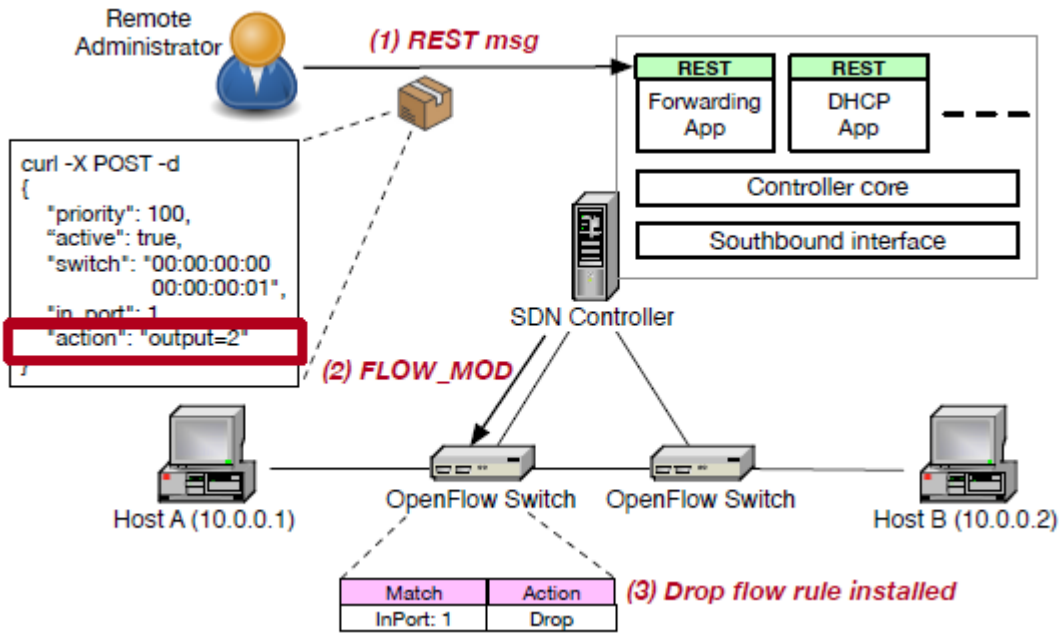


Figure 4.1 : Un scénario d'utilisation abusive de l'API REST

Dans la figure 4.1 est présentée une topologie réseau simple composée d'un contrôleur SDN Floodlight et de deux commutateurs OpenFlow. Et, en supposant que l'administrateur installe une règle de flux sur le commutateur en utilisant un service RESTful fourni par l'application de transfert exécutée sur le contrôleur afin que l'hôte A puisse communiquer avec l'hôte B, il doit créer un message REST adapté à la syntaxe, puis l'envoyer au contrôleur via le protocole HTTP. Le processus d'installation de la règle de flux sur le commutateur via le service RESTful est le suivant. Tout d'abord, l'administrateur crée un message REST composé de certaines ressources (URI, méthode, etc.) dans l'en-tête HTTP et des informations sur les règles de flux dans le champ de données conformément au format requis par Floodlight. Ensuite, après avoir envoyé le message au service RESTful, le service vérifie si le message correspond à sa spécification. Si le message REST est légal, le contrôleur crée un message FLOW MOD, puis l'envoie au commutateur.

Curl

Curl est un utilitaire de ligne de commande qui permet d'exécuter des requêtes HTTP avec différents paramètres et méthodes. Au lieu d'accéder aux ressources Web dans la barre d'adresse du navigateur, il est possible d'utiliser la ligne de commande pour obtenir ces mêmes ressources, extraites au format texte dans ce cas format **JavaScript Object Notation** (json) [36].

Au lieu de décrire comment passer des appels REST à l'aide d'un client à interface graphique tel que Postman, la méthode la plus conventionnelle de documentation de la syntaxe de requête consiste à utiliser client url (curl).

2. Application module de Floodlight

Les applications suivantes ont été implémentées en tant que modules Floodlight en Java.

2.1. Static Flow Entry Pusher

Application permettant d'installer une entrée de flux spécifique (correspondance + action) sur un commutateur spécifique c'est-à-dire permet à un utilisateur d'insérer manuellement des flux dans un réseau OpenFlow, l'utilisateur doit définir l'entrée au format JSON. Activé par défaut, expose un ensemble d'API REST pour ajouter, supprimer et interroger des entrées de flux.

L'ajout d'un flux

Par exemple, pour insérer un flux sur le commutateur 1 qui prend les paquets du port 1 et les envoie sur le port 2, vous pouvez composer la chaîne JSON et utiliser simplement une commande curl pour envoyer le HTTP POST au contrôleur.

```
    curl -X POST -d '{"switch":"00:00:00:00:00:00:00:01", "name":"flow-mod-1",
"cookie":"0", "priority":"32768", "in_port":"1","active":"true",
"actions":"output=2"}'  http://<controller_ip>:8080/wm/staticentrypusher/json
```

2.2. Forwarding

Est une application de transfert de paquets réactif par défaut de Floodlight. Il insère des flux dans des commutateurs qui acheminent les paquets de la source à la destination après que les périphériques ont été appris. La fonctionnalité de routage n'est pas fournie actuellement.

On devrait voir le trafic de contrôle OpenFlow comme suit : Le premier hôte communique avec le deuxième, ce qui entraîne l'envoi d'un message Packet_in au contrôleur. Le contrôleur envoie ensuite un message Packet_out. Le deuxième hôte voit la demande et envoie une réponse. Cette réponse est transmise au contrôleur, qui l'envoie au premier hôte et enfonce une entrée de flux.

Le premier hôte connaît maintenant l'adresse MAC du second et peut envoyer son ping via une requête d'écho ICMP. Cette demande, ainsi que la réponse correspondante du deuxième hôte, vont toutes les deux au contrôleur et aboutissent à une entrée de flux enfoncée (avec les paquets réellement envoyés).

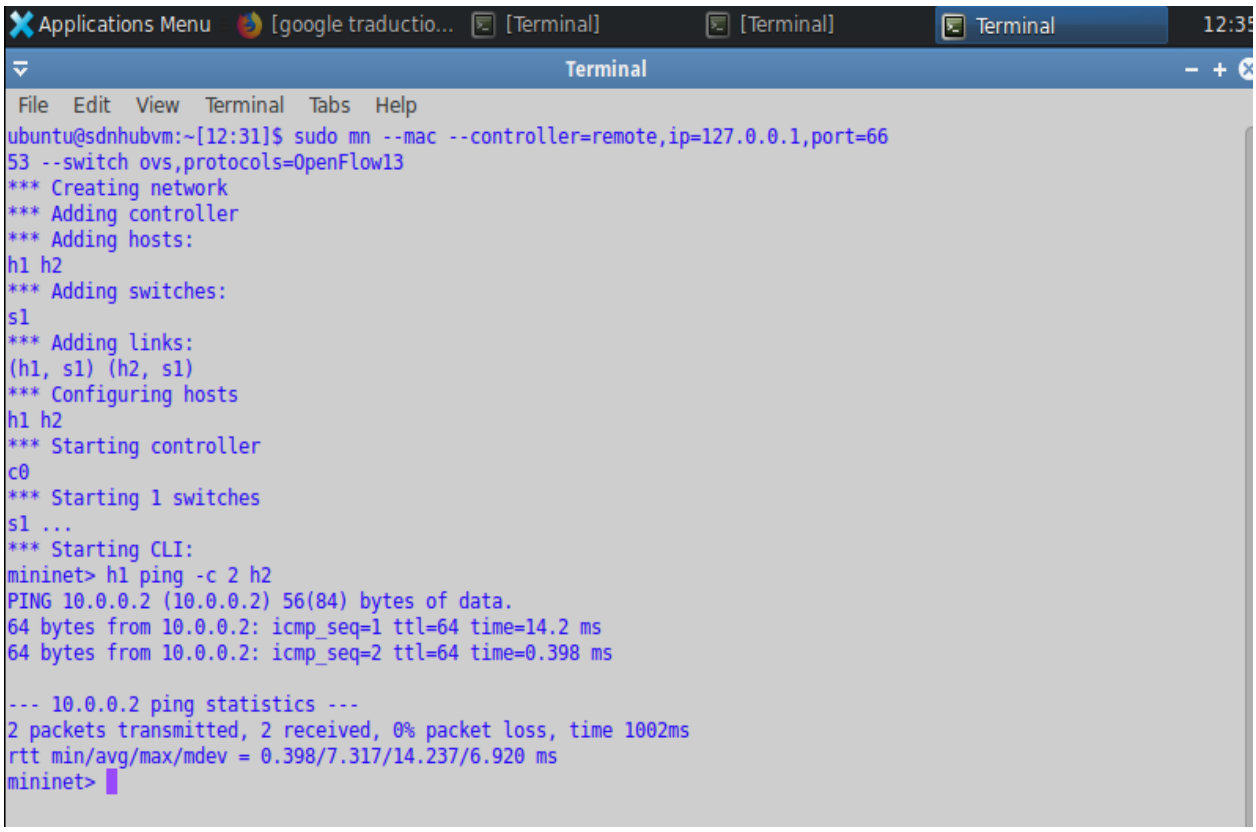


Figure 4.2 : ping entre h1 et h2.

Le ping entre h1 et h2 prend 14.2ms (première tentative) par contre dans la deuxième tentative il prend 0.3ms. On a remarqué que la 2eme tentative a pris un temps beaucoup plus bas que la 1ere, veut dire que lorsque le paquet est reçu par le commutateur, il n'a pas trouvé leur correspondance dans la table de flux, donc une demande est envoyée au contrôleur. Par contre dans le 2eme cas une entrée de flux couvrant le trafic ping ICMP était précédemment installée dans le commutateur, par conséquent, aucun trafic de contrôle n'a été généré et les paquets transitent immédiatement par le commutateur.

2.3.Firewall

Chaque ordinateur connecté à internet (et d'une manière plus générale à n'importe quel réseau informatique) est susceptible d'être victime d'une attaque d'un pirate informatique.

Pour parer à ces attaques, une architecture sécurisée est nécessaire. Pour cela, le cœur d'une telle architecture est basé sur un firewall. Un système firewall fonctionne sur le principe du filtrage de paquets [37].

L'application du firewall a été implémentée en tant que module Floodlight qui applique les règles de la liste de contrôle d'accès (ACL) sur les commutateurs compatibles OpenFlow du réseau à l'aide de flux et en surveillant le comportement des paquets entrants. Les règles ACL ne sont ici que des ensembles de conditions permettant ou interdisant un flux de trafic au niveau de son commutateur

d'entrée. Expose un ensemble d'API REST pour activer / désactiver le firewall et ajouter / supprimer / lister des règles. L'application est chargée et désactivée par défaut mais après l'activation le firewall refuse tout le trafic sauf si une règle explicite ALLOW est créée.

Chaque entrée de paquet déclenchée par le ou les premiers paquets d'un flux de trafic est comparée à l'ensemble des règles de firewall existantes. Les règles de firewall sont triées en fonction des priorités assignées et sont comparées aux champs d'en-tête de PacketIn. La règle de firewall correspondant à la priorité la plus élevée détermine l'action (autoriser / refuser) du flux [38].

Implémentation et la présentation du firewall

La commande avec laquelle on a fait le test du firewall :

```
    sudo mn --controller=remote,ip=127.0.0.1,port=6653 --switch
ovsk,protocols=OpenFlow13 --mac --topo=tree,3
```

La topologie :

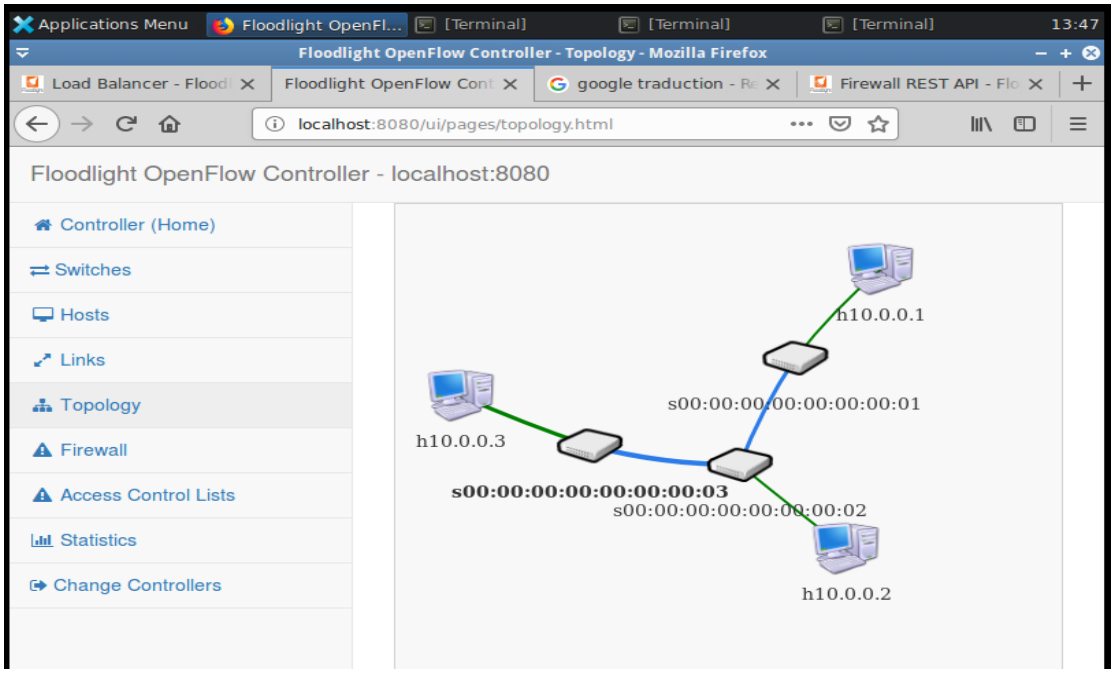


Figure 4.3 : l'affichage graphique de la conception du réseau.

Cette commande indique si le firewall est activé ou désactivé :

```
$ curl http://localhost:8080/wm/firewall/module/status/json
```

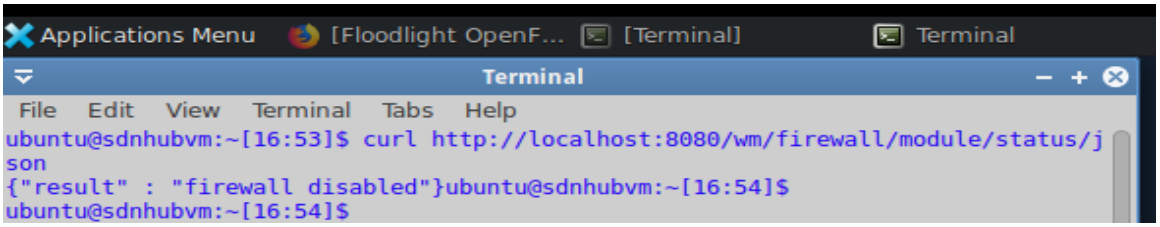


Figure 4.4 : Etat du firewall.

La commande active le module de firewall compilé avec le contrôleur floodlight et bloque toutes les communications entre les hôtes du réseau. Pour permettre la communication, des règles doivent être insérées dans le firewall afin que tout paquet entrant soit vérifié avec la règle avant qu'une action ALLOW ou DENY ne soit mise en œuvre sur le paquet.

```
curl http://localhost:8080/wm/firewall/module/enable/json -X PUT -d
''
```

Figure 4.5 : activation du firewall.

Pour implémenter le firewall en utilisant un scénario de réseau c'est : « autoriser tous les flux passent par le commutateur `00:00:00:00:00:00:00:01` ». Et « autoriser tous les flux passent par le commutateur `00:00:00:00:00:00:00:02` ».

N'oubliez pas que le firewall a déjà été mis en œuvre et activé. Par conséquent, toutes les communications sont bloquées. Donc, le firewall doit être configuré pour permettre les communications ping entre les hôtes conformément à la politique de l'organisation. Pour ce faire, le firewall est conçu pour accepter les règles via le service API REST. Les commandes suivantes permettent d'insérer les règles:

```
curl-X POST -d '{"switchid":
"00:00:00:00:00:00:00:01"}' http://localhost:8080/wm/firewall/rules/json
```

```
curl-X POST -d '{"switchid":
"00:00:00:00:00:00:00:02"}' http://localhost:8080/wm/firewall/rules/json
```

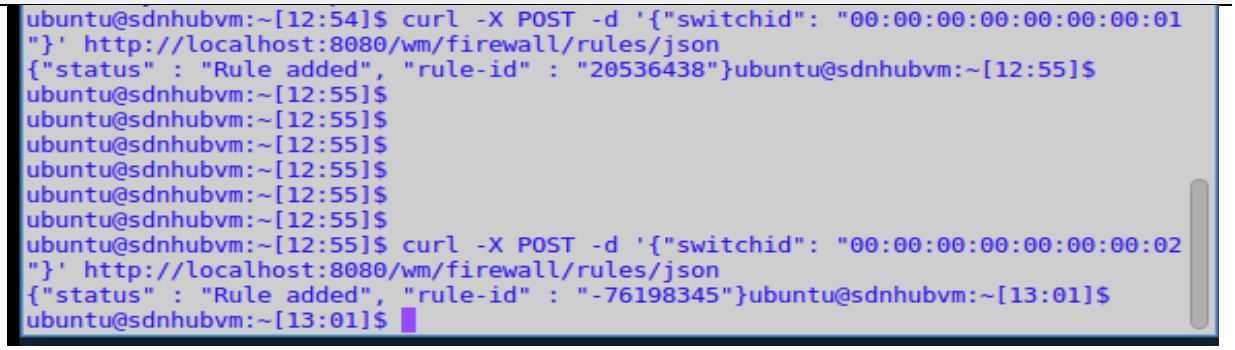


Figure 4.6 : l'ajout d'une règle firewall.

La méthode «POST» insérera les règles dans le fichier de règles du firewall. Chaque paquet entrant sera comparé à ces règles et si une correspondance est trouvée, l'action de la règle (autoriser / refuser) est appliqué.

La figure ci-dessus montre les résultats d'un ping aprés l'activation du firewall et avant l'ajout des règles montre que le trafic est refusé.

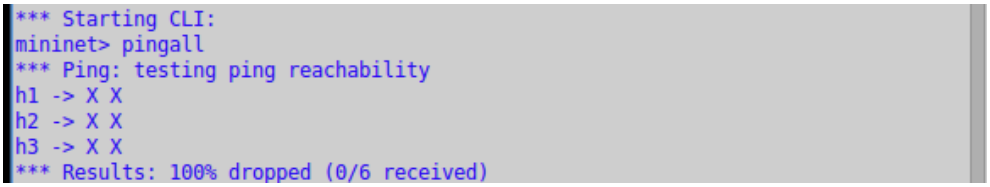


Figure 4.7 : ping avant l'ajout des règles.

Les résultats d'un autre ping après l'ajout des règles d'autorisation. Alors, nous observons, que h1 peut pinger que h2 (et vice versa).

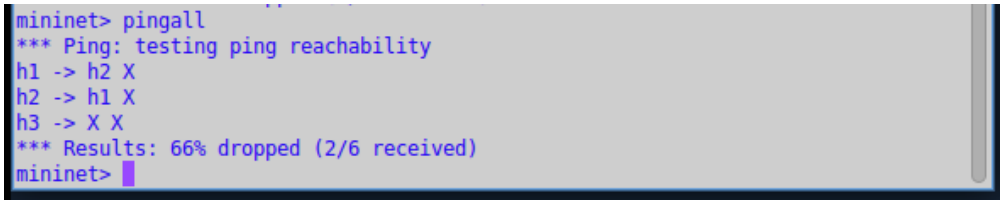


Figure 4.8 : ping après l'ajout des règles.

- Ajout d'une règle ALLOW pour tous les flux entre l'hôte mac 00:00:00:00:00:0a et l'hôte 00:00:00:00:00:0b

```
curl -X POST -d '{"src-mac": "00:00:00:00:00:0a", "dst-mac":
"00:00:00:00:00:0a"}' http://localhost:8080/wm/firewall/rules/json<font></font>
```

```
    curl -X POST -d '{"src-mac": "00:00:00:00:00:0b", "dst-mac":
"00:00:00:00:00:0b"}' http://localhost:8080/wm/firewall/rules/json<font></font>
```

- Ajout d'une règle ALLOW pour que le ping fonctionne entre les hôtes IP 10.0.0.3 et 10.0.0.7.

```
    curl -X POST -d '{"src-ip": "10.0.0.3/32", "dst-ip": "10.0.0.7/32", "dl-
type":"ARP" }'   http://localhost:8080/wm/firewall/rules/json<font></font>

    curl -X POST -d '{"src-ip": "10.0.0.7/32", "dst-ip": "10.0.0.3/32", "dl-
type":"ARP" }' http://localhost:8080/wm/firewall/rules/json<font></font>
    <font></font>

    curl -X POST -d '{"src-ip": "10.0.0.3/32", "dst-ip": "10.0.0.7/32", "nw-
proto":"ICMP" }' http://localhost:8080/wm/firewall/rules/json<font></font>

    curl -X POST -d '{"dst-ip": "10.0.0.7/32", "dst-ip": "10.0.0.3/32", "nw-
proto":"ICMP" }' http://localhost:8080/wm/firewall/rules/json<font></font>
```

2.4. ACL

Une liste de contrôle d'accès est un script de configuration d'autorisation ou de refus le passage des paquets qui s'appliquent aux adresses ou aux protocoles. Les ACL représentent un outil puissant pour contrôler le trafic entrant ou sortant d'un réseau.

```
    curl -X POST -d '{"src-ip":"10.0.0.1/32","dst-
ip":"10.0.0.2/32","action":"deny"}' http://<controller_ip>:8080/wm/acl/rules/json
```

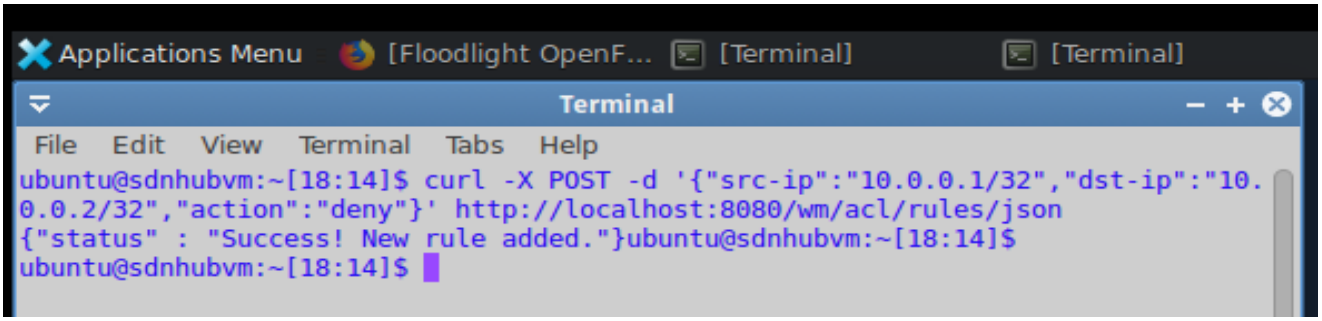


Figure 4.9 : l'ajout d'une règle ACL.

Les figures 4.10 et 4.11 montrent le test de règle d'ACL.

La figure ci-dessus montre le résultat d'un ping avant l'ajout de règle ACL.

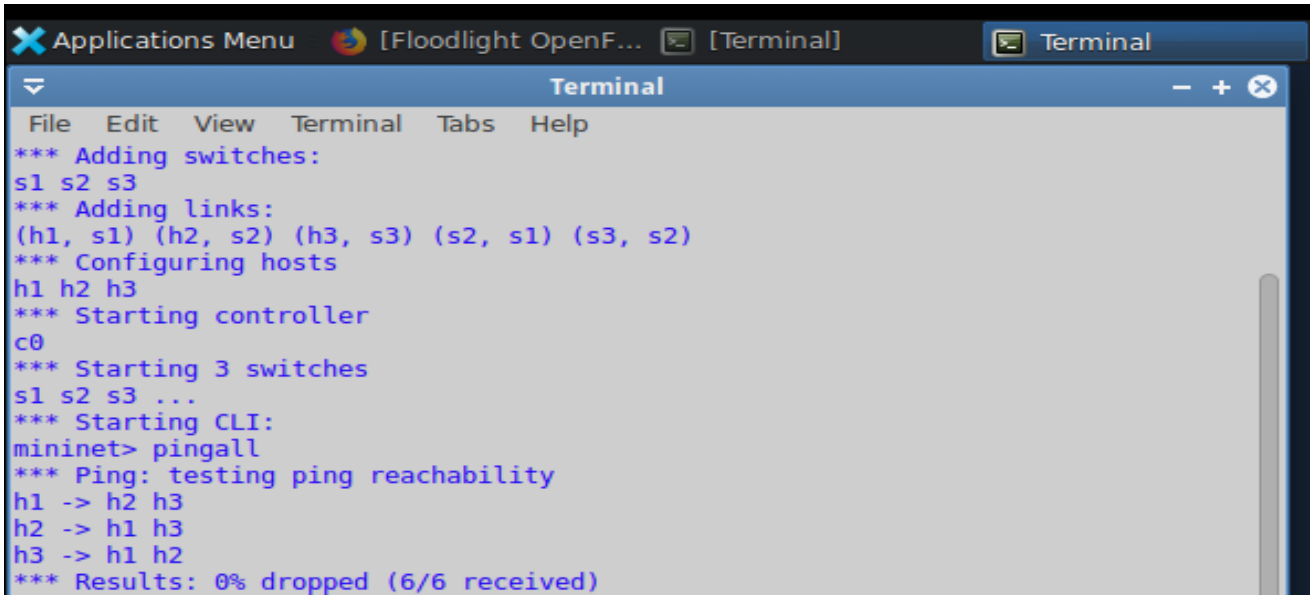


Figure 4.10 : ping avant l'ajout des règles.

Nous avons remarqué que avant l'ajout de la règle le ping entre h1 et h2 marche bien.

La figure ci-dessous montre le résultat d'un ping aprés l'ajout de règle ACL.

Figure 4.11 : ping aprés l'ajout des règles.

Nous avons remarqué qu'après l'ajout de la règle le ping entre h1 et h2 ne marche pas car la règle d'ACL est installée.

2.5. Load Balancer

Le controleur floodlight a un module d'application de l'équilibreur de charge simple à exécuter dans les services ping, tcp, udp. L'accès à ce module s'effectue via une API REST.Cette application fournit une adresse IP virtuelle, qui représente une implémentation de la politique d'équilibrage de charge sur le contrôleur . Une fois que le contrôleur a reçu les paquets de requêtes réseau des clients dont l'adresse de destination est l'adresse IP virtuelle, le contrôleur sélectionne les paquets en fonction de l'algorithme Round-Robin Scheduling afin de répondre à la demande.

1/ Démarrer Floodlight

2/ Démarrer mininet (Cette commande créera un réseau complexe avec 8 hôtes et 7 commutateurs).

```
sudo mn --controller=remote,ip=127.0.0.1,port=6653 --switch ovsk,protocols=OpenFlow13 --mac --topo=tree,3
```

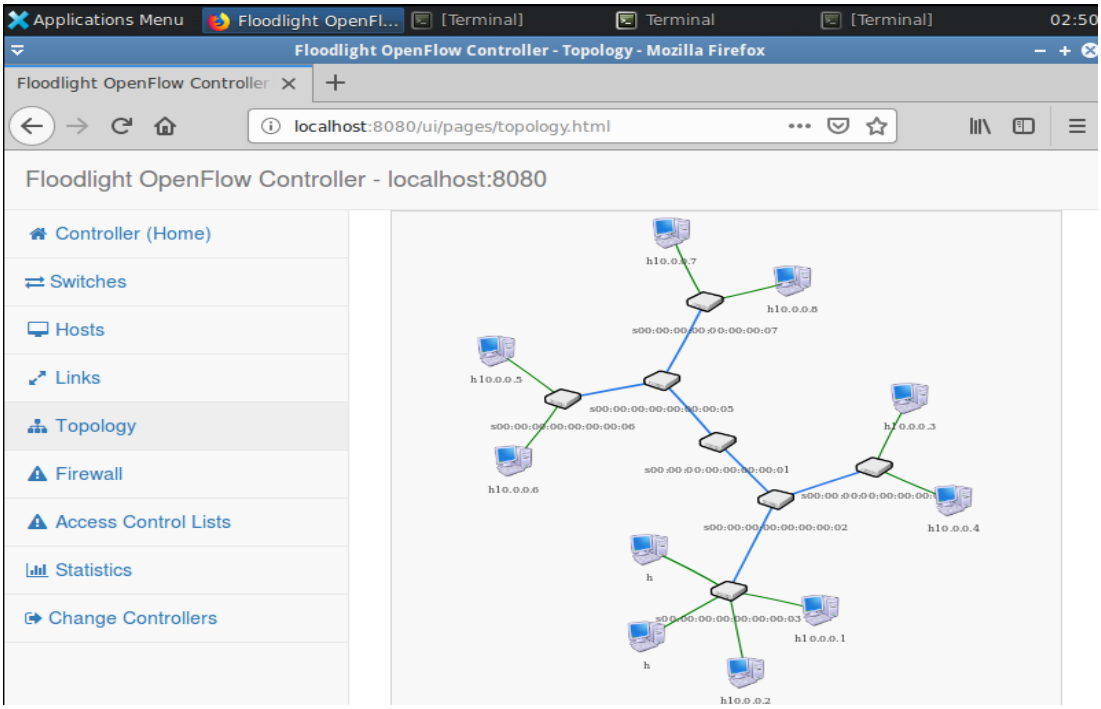


Figure 4.12 : l'affichage graphique de la conception du réseau pour le test du Load Balancer.

3/ En Mininet faire 'Pingall'

4/ L'API REST enverra des commandes au module d'équilibrage de charge (configurez les vips, les pools et les membres de l'équilibreur de charge). Les commandes sont les suivantes :

```
     curl -X POST -
d '{"id":"1","name":"vip1","protocol":"icmp","address":"10.0.0.100","port":"8"}' htt
p://localhost:8080/quantum/v1.0/vips/

     curl -X POST -
d '{"id":"1","name":"pool1","protocol":"icmp","vip_id":"1"}' http://localhost:8080/q
uantum/v1.0/pools/

     curl -X POST -
d '{"id":"1","address":"10.0.0.3","port":"8","pool_id":"1"}' http://localhost:8080/q
uantum/v1.0/members/

     curl -X POST -
d '{"id":"2","address":"10.0.0.4","port":"8","pool_id":"1"}' http://localhost:8080/q
uantum/v1.0/members/
```

5/ Dans mininet, faites "h3 ping-c1 10.0.0.100", puis "h4 ping-c1 10.0.0.100". Les deux requêtes ping doivent réussir alors qu'elles auraient dû être gérées par deux hôtes réels différents (10.0.0.3 et 10.0.0.4). Stratégie de test d'équilibrage de charge: une application serveur qui surveille le port 8 s'exécute sous des hôtes h3 et h4.

Après cela, il autorisera les paquets de requête accessibles à l'équilibrage de charge 10.0.0.100 vers l'hôte h3 et h4.

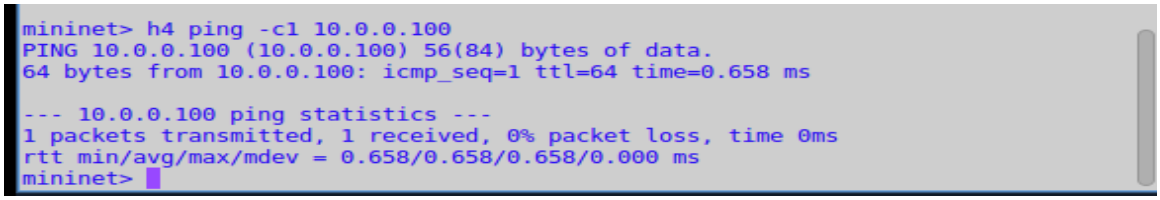


Figure 4.13 : ping h4 vers le serveur virtuel.

Comparaison des fonctionnalités

- La comparaison des fonctionnalités dans le SDN et dans le réseau traditionnel montré dans le tableau suivant :

	Réseau traditionnel	**SDN**
Forwarding	Lorsque la table mac ne contient pas l'adresse mac le switch fait une diffusion.	Lorsque la table de flux ne contient pas une entrée de correspondance de paquet, une demande est envoyée au contrôleur sous la forme d'un 'Packet_in'.
	On a besoin d'un protocole stp pour éviter les boucles.	On n'a pas besoin d'un protocole stp car on a une vue global sur la topologie.
	Le chemin est bouché par le trafic et le commutateur n'a pas de programmabilité donc les règles ne peuvent pas être changées *dynamiquement* comme nous le souhaitons.	Le chemin est bouché par le trafic, une partie du réseau est fortement encombrée, le contrôleur peut reprogrammer les tables de transfert pour rediriger le trafic sensible autour des liaisons et des commutateurs encombrés.
Firewall et ACL	Configuration distribuée.	Configuration centralisée, avec une architecture de contrôleur, la stratégie serait créée une fois.
	Avoir autant de politiques de sécurité mises en œuvre et maintenues manuellement serait un cauchemar administratif.	Avoir autant de politiques de sécurité mises en œuvre et maintenues automatiquement lorsqu'un nouvel ordinateur virtuel est créé.

Tableau 4.1 : comparaison entre l'architecture réseau traditionnel et le SDN

Conclusion

Dans ce chapitre, nous avons atteint le but du projet qui est l'étude exploratoire et la mise en œuvre des solutions basées sur SDN. On a testé les fonctionnalités du contrôleur Floodlight, des fonctionnalités en tant qu'application activée par défaut comme Static Flow Entry Pusher qui permet d'insérer manuellement des flux et Forwarding qui permet de transférer des paquets de manière réactive. Et des applications de sécurité comme firewall et ACL. Ainsi qu'une comparaison entre ces fonctionnalités dans un réseau SDN par rapport au réseau traditionnel, Nous avons constaté que ces fonctionnalités sont automatisées et centralisées, on a donc confirmé les avantages du réseau SDN.

Conclusion générale

Avec la découverte du SDN de nombreuses nouvelles perspectives ont eu lieu. C'est le pouvoir de renforcer l'agilité, la sécurité ou encore les capacités du réseau grâce à de nouvelles solutions logicielles embarquant toujours plus d'intelligence. Car c'est bien là que réside le point fort du SDN qui désormais n'évolue plus avec la lourdeur du matériel mais avec la célérité du logiciel. Le SDN va donc changer le quotidien des administrateurs réseau, qui seront moins pris par des opérations de configuration répétitives et fastidieuses et pourront ainsi se consacrer à des tâches présentant plus de valeur ajoutée pour leur entreprise.

Grâce au SDN, les réseaux sont programmables, et la mise à disposition d'interfaces de programmation d'applications permet de programmer les équipements du réseau en utilisant différents langages.

Les travaux menés durant ce projet se positionnent dans le cadre de la découverte des réseaux SDN et leurs concepts, en mettant l'accent sur le protocole OpenFlow, ainsi qu'un des composants clés de cette technologie qui est le contrôleur. Dans notre cas, notre démarche s'est d'abord basé sur une étude comparative des contrôleurs Floodlight et OpenDaylight, ensuite nous avons démontré l'efficacité du contrôleur Floodlight en testant les fonctionnalités qu'il offre dans le domaine de la sécurité et de la transmission du trafic.

Les résultats de notre projet étaient très satisfaisants, et nous pouvons dire que nous avons atteint nos objectifs qui étaient de :

- Surpasser les défauts des réseaux traditionnels
- Détailler l'aspect de sécurité des réseaux SDN (Firewall, ACL)
- Equilibrer la charge du réseau (Load Balancer).

Ce stage a été une expérience enrichissante d'un point de vue technique, de par les technologies abordées, autant sur le plan d'administration réseau et système que sur le plan de la découverte du monde de l'entreprise.

Cependant, le développement d'un tel projet n'est jamais totalement achevé et certaines idées n'ont pas pu être réalisées à cause de contraintes de temps. C'est dans ce sens et afin d'améliorer ce présent travail que nous proposons de :

- Etudier le SDN dans un environnement du Cloud Computing et DATACENTER.
- Réaliser une application de gestion de VLAN basée sur un réseau SDN.
- Déployer le SD-WAN.

Références

[1] Benamrane, F. « Étude des Performances des Architectures du Plan de Contrôle des Réseaux Software-Defined Networks ». Thèse de doctorat, Université Mohammed V, Rabat, Soutenue le Mercredi 18 Janvier 2017.

[2] Said Seddiki, M. « Allocation dynamique des ressources et gestion de la qualité de service dans la virtualisation des réseaux ». Thèse de doctorat, Ecole doctorale IAEM Lorraine, Soumis le 14 décembre 2015.

[3] Altufaili, M. (2015). « Intelligent Network Bandwidth Allocation using SDN », International Journal of Engineering Research & Technology, Volume. 4 – Issue.

[4] Réseau SDN : mode d'emploi et pièges à éviter, dossier ZDNet : https://www.zdnet.fr/actualites/reseau-sdn-mode-d-emploi-et-pieges-a-eviter-39797891.htm , 2014.

[5] Patrice kuekem, T. «implémentation d'un plan de contrôle pour les réseaux multicouches if/optiques », mémoire, université du Québec à Montréal, Soutenu JUIN 2016.

[6] Wiley, J. & Sons, I. (2016). « Network Virtualization For Dummies », VMware Special Edition, 111 River St.

[7] Systeme centralise: https://www.memoireonline.com/04/17/9838/m_Etude-d-une-mise-en-place-d-un-systeme-centralise-a-integration-des-sites-distants-avec-les-technolo21.html

[8] Vicentini, C. Santin, A. Viegas, E. & Abreu, V. (2018). «SDN-based and multitenant-aware resource provisioning mechanism for cloud-based big data streaming». Journal of Network and Computer Applications, pages 2_4.

[9] Buche, X « Cloud universitaire avec Openstack », mémoire d'ingénieur CNAM, Université Lille 1, Soutenu le 1 er décembre 2015.

[10] Paradis, T. « Software-Defined Networking », mémoire de Master, School of Information and Communication Technology KTH Royal Institute of Technology Stockholm, Sweden, le 20 Janvier 2014.

[11] Didacticiel de mise en réseau définie par logiciel - Notions de base

https://www.sdxcentral.com/networking/sdn/definitions/software-defined-networking-tutorial/

[12] Aceres, E. « Le Protocole OpenFlow dans l'Architecture SDN », EFORT 2016.

[13] Open Network Foundation, «OpenFlow Switch Specification», version 1.4.0, October 2013.

[14] Ferreira, T. & Cloarec, V. « Le projet OpenFlow », Telecom Lille 1, 2010.

[15] Le SDN pour les nuls, JRES 2015 – Montpellier.

[17] Protocole_openflow:
[https://www.supinfo.com/articles/single/1010-protocole-openflow]

[18] Tury, M. (2015). «Les risques d'OpenFlow et du SDN».

[19] Openflow spécification, 26 mars 2015.

[20] Aouadj, M. «AirNet : le modèle de virtualisation "Edge-Fabric" comme plan de contrôle pour les réseaux programmables», thèse de doctorat de l'université de Toulouse, Présentée et soutenue le 08 novembre 2016.

[21] BenChahed, S. «Mise en œuvre des aspects de gestion des réseaux définis par logiciel (réseaux SDN)», Maitrise ès sciences appliquées (génie informatique), école polytechnique de Montréal, soutenue octobre 2015.

[22] Sakellaropoulou, D. «A Qualitative Study of SDN Controllers», thèse de doctorat de l'université de Athens, Septembre 2017.

[23] Salman, O. Elhajj, I. Kayssi, A. & Chehab, A. «SDN Controllers: A Comparative Study» Conference Paper, April 2016.

[24] Akcay, H. & Yiltas-Kaplan, D. «Web-Based User Interface for the Floodlight SDN Controller». Int. J. Advanced Networking and Applications, Volume: 08, 22 mars 2017.

[25] Série cinq de la série SDN : Floodlight, un contrôleur OpenFlow: https://thenewstack.io/sdn-series-part-v-floodlight

[26] Cas d'utilisation, Visibilité et contrôle : https://www.opendaylight.org/use-cases-and-users/by-function/visibility-and-control

[27] Mininet Vue d'ensemble : http://mininet.org/overview/ ,2018.

[28] Qualité de production, commutateur virtuel ouvert multicouche https://www.openvswitch.org/

[29] Open vSwitch: http://www.linusnova.com/2013/04/open-vswitch

[30] What is Open vSwitch: https://northboundnetworks.com/blogs/sdn/what-is-open-vswitch

[31] SDN Controllers: Benchmarking & Performance Evaluation (12 /02/2019), IEEE

[32] Cbench: https://floodlight.atlassian.net/wiki/spaces/floodlightcontroller/pages/1343657/Cbench+ew

[33] Fernandez, P. (2013) «Evaluating OpenFlow Controller Paradigms», La douzième conférence internationale sur les réseaux, Université d'Etat du Ceará.

[34] Sangeetha, E. «Implementation of address learning/packet forwarding, firewall and load balancing in floodlight controller for SDN network management» international journal of information and engineering. Volume. 3 (no.1), Avril, 2015.

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

[36] curl intro and installation (2017): https://idratherbewriting.com/learnapidoc/docapis_install_curl.html

[37] https://www.frameip.com/firewall/

[38] https://floodlight.atlassian.net/wiki/spaces/floodlightcontroller/pages/1343616/Firewall/

Annexe 1 : Installation du Floodlight

1. Prérequis

- OS Recommandé : n'importe quelle distribution Linux.
- Installation de JDK, et ant.

```
# yum install java-1.8.0-openjdk java-1.8.0-openjdk-devel
# yum install ant
```

2. Téléchargement du Floodlight

- Télécharger la source à partir du site github.
- Lancer la commande ant (Ant est un projet du groupe Apache-Jakarta. Son but est de fournir un outil écrit en Java pour permettre la construction d'applications (compilation, exécution de tâches post et pré compilation, ...).

```
$ git clone git://github.com/floodlight/floodlight.git
$ cd floodlight
$ ant
```

3. Exécution du Floodlight

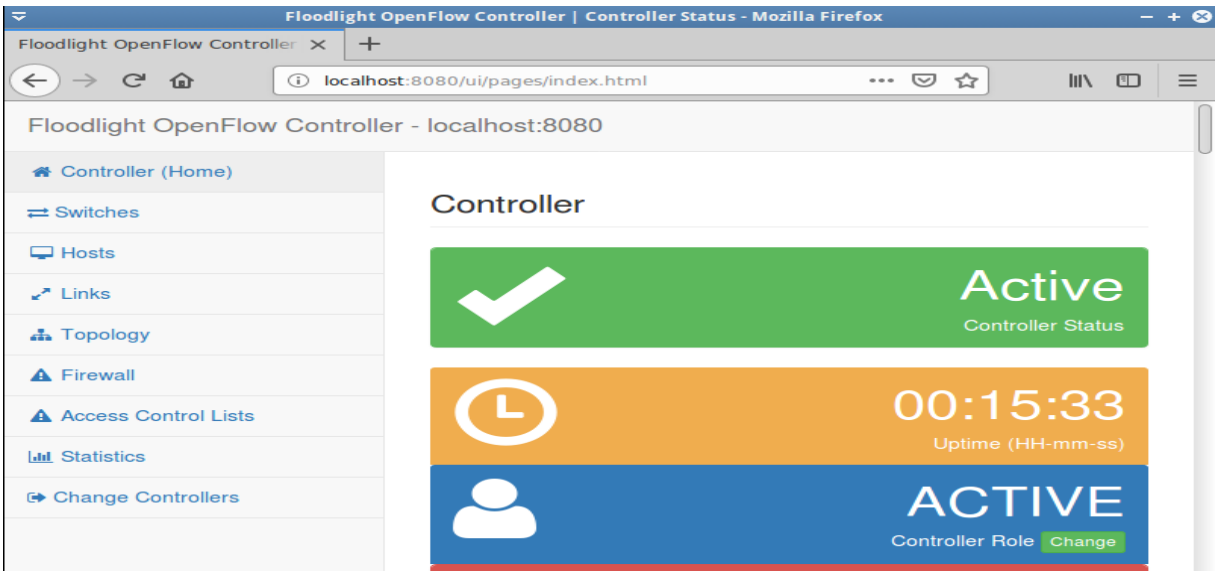
- Avant de lancer la commande ci-dessous, la commande java doit être ajoutée à la variable d'environnement PATH.
- Exécuter le fichier **floodlight.jar** généré par ant.

```
$ java -jar target/floodlight.jar
```

4. Accéder au contrôleur Floodlight

- Ouvrir un navigateur web,
- Saisir l'URL ci-dessous :

```
http://floodlight_ip_address:8080/ui/index.html
```


Floodlight OpenFlow Controller | Controller Status - Mozilla Firefox
Floodlight OpenFlow Controller
localhost:8080/ui/pages/index.html
Floodlight OpenFlow Controller - localhost:8080
Controller (Home)
Switches
Hosts
Links
Topology
Firewall
Access Control Lists
Statistics
Change Controllers
Controller
Active
Controller Status
00:15:33
Uptime (HH-mm-ss)
ACTIVE
Controller Role
Change

Annexe 2 : Installation d'OpenDaylight

1. Téléchargement d'OpenDaylight

Téléchargez le logiciel OpenDaylight à partir du site Web OpenDaylight. Sur un hôte Linux ou Mac OS, nous pouvons utiliser la commande wget pour télécharger le fichier tar.

```
$wget
https://nexus.opendaylight.org/content/groups/public/org/opendaylight/integration/distribu
tion-karaf/0.4.0-Beryllium/distribution-karaf-0.4.0-Beryllium.tar
```

Installez OpenDaylight en extrayant le fichier tar :

```
$ tar -xvf distribution-karaf-0.4.0-Beryllium.tar.gz
```

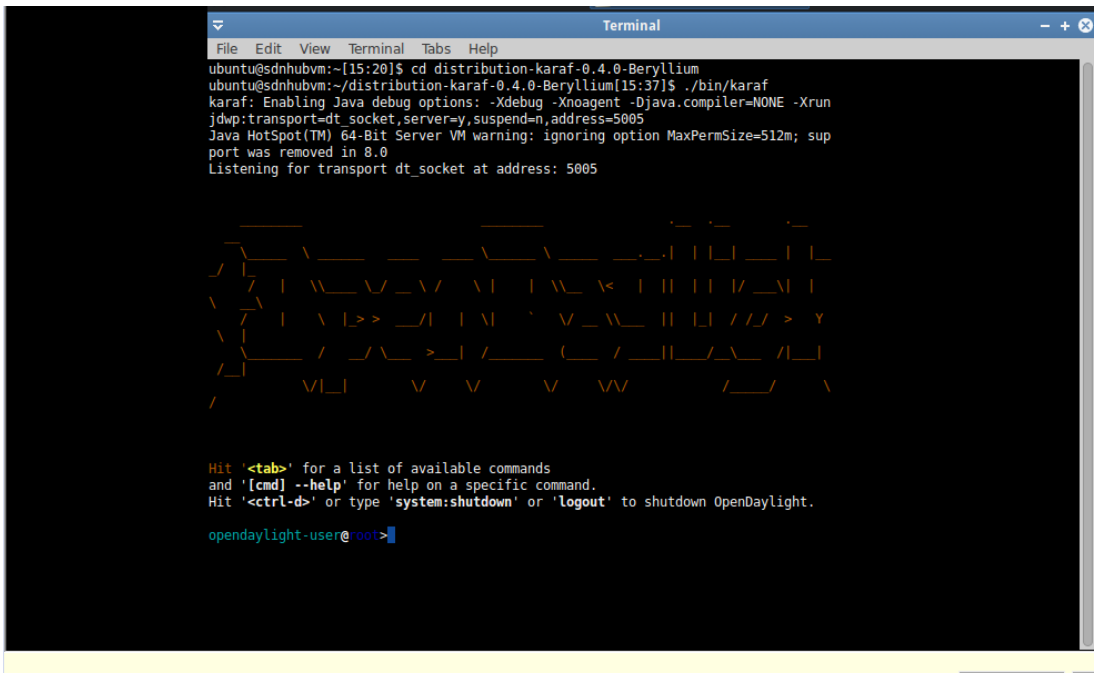
Cela crée un dossier nommé distribution-karaf-0.4.0-Beryllium qui contient le logiciel OpenDaylight et ses plugins. OpenDaylight est emballé dans un conteneur karaf. Karaf est une technologie de conteneur qui permet aux développeurs de placer tous les logiciels requis dans un seul dossier de distribution. Cela facilite l'installation ou la réinstallation d'OpenDaylight lorsque cela est nécessaire, car tout se trouve dans un dossier.

2. Exécution d'OpenDaylight

Pour exécuter OpenDaylight, exécutez la commande karaf dans le dossier de distribution du pa quet.

```
$ cd distribution-karaf-0.4.0-Beryllium

$ ./bin/karaf
```

Maintenant, le contrôleur OpenDaylight est en cours d'exécution :



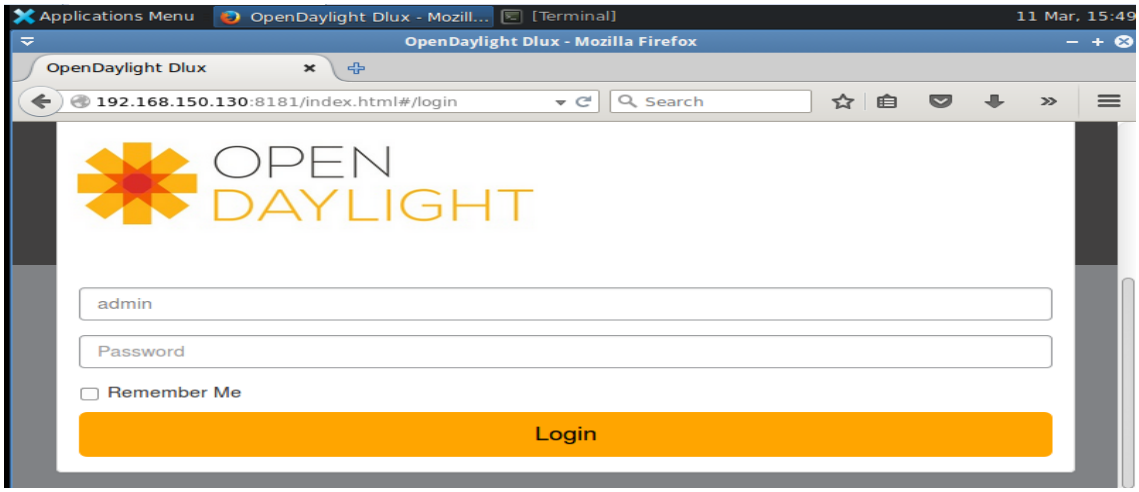
Installez le minimum de fonctionnalités requises pour tester OpenDaylight et l'interface graphique OpenDaylight :

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

    opendaylight-user@root> feature:list

    opendaylight-user@root> feature:list --installed
```

3. Accéder au contrôleur OpenDaylight

- Ouvrir un navigateur web,
- Saisir l'URL ci-dessous :

```
    http://OpenDaylight_ip_address:8181/ui/index.html
```



Annexe 3 : Installation de mininet

L'installation et la configuration de Mininet se fait au niveau d'un nœud indépendant de notre contrôleur, mais il est possible de les installer sur la même machine, les étapes de configuration peuvent être résumées dans les points suivants :

1. Téléchargement de mininet

```
git clone git: //github.com/mininet/mininet
```

La commande git télécharge la plus récente version de Mininet :

```
cd mininet
git tag  #  list available versions
git checkout –b 2.2.0b3 # or whatever version you wish to install
```

2. Installation de mininet

Une fois nous avons accédé au dossier mininet, la commande suivante permet d'installer tous les packages Mininet :

```
mininet/util/install.sh -a
```

3. Création d'une topologie avec 3 méthodes

3.1. Création d'une topologie avec la ligne de commande

Mininet fournit, en plus de la topologie ≪minimal ≫, la topologie ≪single≫, ≪linear≫ et ≪tree≫. Pour charger l'une de ces topologies, on utilise l'option **«--topo»**, Par exemple:

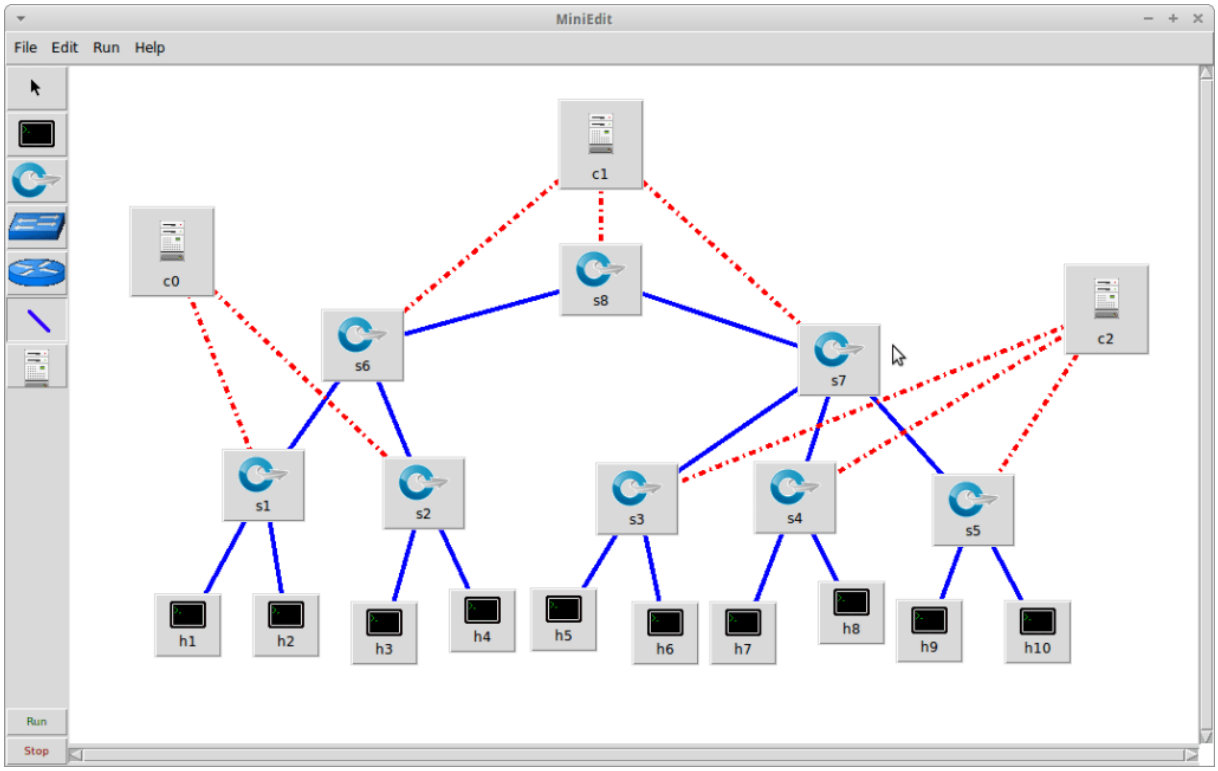
```
$ Sudo mn --topo tree
```

Les commandes CLI de Mininet : net, dump, pingall

3.2. Création d'une topologie avec une Interface interactive

L'émulateur de réseau Mininet inclut MiniEdit , un éditeur graphique simple pour Mininet. MiniEdit est un outil expérimental créé pour montrer comment Mininet peut être étendu. Le script MiniEdit se trouve dans le dossier des exemples de Mininet. Pour exécuter MiniEdit, exécutez la commande suivante :

```
$ sudo ~/mininet/examples/miniedit.py
```



3.3. Création d'une topologie avec un Script Python

Si nous souhaitons travailler avec une topologie autre que tree, linear ou simple, et même sans la ligne de commande, il faut créer une topologie personnalisée à l'aide des scripts Python. Par exemple, si on veut créer 2 hôtes h1 et h2, les deux sont connectes a un switch s1.

Voici le script Python à écrire :

```
class Test_Topo(Topo):
def __init__(self):
"Create P2P topology"
# Initialize topology
Topo.__init__(self)
# Add hosts and switches
h1 = self.addHost('h1')
h2 = self.addHost('h2')
s1 = self.addSwitch('s1')
# Add links
self.addLink(h1,s1)
self.addLink(h2,s1)
topos = { 'toptest': (lambda: Test_Topo()) }
```

Après, nous sauvegardons le code dans un fichier toptest.py, et nous exécutons la commande suivante :

```
$ sudo mn  --custom  /chemin/vers/toptest.py  --topo toptest
```

Annexe 4 : Installation du Cbench

Pour installer Cbench, nous avons suivi la procédure suivante :

```
$ sudo apt-get install autoconf automake libtool libsnmp-dev libpcap-dev
$ git clone git://gitosis.stanford.edu/oflops.git
$ cd oflops; git submodule init && git submodule update
$ git clone git://gitosis.stanford.edu/openflow.git
$ cd openflow; git checkout -b release/1.0.0 remotes/origin/release/1.0.0
$ wget http://hyperrealm.com/libconfig/libconfig-1.4.9.tar.gz
$ tar -xvzf libconfig-1.4.9.tar.gz
$ cd libconfig-1.4.9
$ ./configure
$ sudo make && sudo make install
$ cd ../../netfpga-packet-generator-c-library/
$ sudo ./autogen.sh && sudo ./configure && sudo make
$ cd ..
$ sh ./boot.sh ; ./configure --with-openflow-src-dir=<absolute path to
openflow branch>; make
$ sudo make install
$ cd cbench
```

Exécution du Cbench

- Les options les plus utilisées avec Cbench sont :

M	Nombre de MACs / hôtes à émuler par switch
s	Nombre de switches à émuler
t	throughput (mode débit)

Un exemple utile :

```
./cbench -c localhost -p 6653 -m 10000 -l 10 -s 16 -M 100 -t
```

Les résultats de test :

Tests de débit de Floodlight

```
ubuntu@sdnhubvm:~/oflops/cbench[11:47] (master)$ cbench -c localhost -p 6653 -m
10000 -l 3 -s 8 -M 1000 -t
cbench: controller benchmarking tool
   running in mode 'throughput'
   connecting to controller at localhost:6653
   faking 8 switches offset 1 :: 3 tests each; 10000 ms per test
   with 1000 unique source MACs per switch
   learning destination mac addresses before the test
   starting test with 0 ms delay after features_reply
   ignoring first 1 "warmup" and last 0 "cooldown" loops
   connection delay of 0ms per 1 switch(es)
   debugging info is off
11:48:28.138 8   switches: flows/sec:  44480  51952  56272  0  51323  51952  583
47  63985   total = 37.795840 per ms
11:48:38.240 8   switches: flows/sec:  62969  60243  63298  0  64441  63941  767
29  66891   total = 45.850375 per ms
11:48:48.342 8   switches: flows/sec:  76070  61273  76729  0  63942  57429  686
65  82576   total = 48.666502 per ms
RESULT: 8 switches 2 tests min/max/avg/stdev = 45850.37/48666.50/47258.44/1408.0
6 responses/s
ubuntu@sdnhubvm:~/oflops/cbench[11:48] (master)$
```

```
untu@sdnhubvm:~/oflops/cbench[14:23] (master)$ cbench -c localhost -p 6653 -m 10000 -l 3 -s 16 -M 1000 -t
ench: controller benchmarking tool
 running in mode 'throughput'
 connecting to controller at localhost:6653
 faking 16 switches offset 1 :: 3 tests each; 10000 ms per test
 with 1000 unique source MACs per switch
 learning destination mac addresses before the test
 starting test with 0 ms delay after features_reply
 ignoring first 1 "warmup" and last 0 "cooldown" loops
 connection delay of 0ms per 1 switch(es)
 debugging info is off
:24:45.877 16  switches: flows/sec:  18111  18945  18450  28037  29790  29764  26949  0  28884  20687  30544
1284  30344  0  29403  33735   total = 37.461483 per ms
:24:55.985 16  switches: flows/sec:  38366  34524  36574  27338  38366  38366  38367  0  38366  38367  38366
8366  41885  0  33091  38367   total = 51.839123 per ms
:25:06.091 16  switches: flows/sec:  38367  38367  38367  38368  38366  38366  38366  0  38366  38367  42400
8366  38366  0  51153  50501   total = 56.600382 per ms
SULT: 16 switches 2 tests min/max/avg/stdev = 51839.12/56600.38/54219.75/2380.63 responses/s
untu@sdnhubvm:~/oflops/cbench[14:25] (master)$
```

```
ubuntu@sdnhubvm:~/oflops/cbench[13:49] (master)$ cbench -c localhost -p 6653 -m
10000 -l 3 -s 32 -M 1000 -t
cbench: controller benchmarking tool
   running in mode 'throughput'
   connecting to controller at localhost:6653
   faking 32 switches offset 1 :: 3 tests each; 10000 ms per test
   with 1000 unique source MACs per switch
   learning destination mac addresses before the test
   starting test with 0 ms delay after features_reply
   ignoring first 1 "warmup" and last 0 "cooldown" loops
   connection delay of 0ms per 1 switch(es)
   debugging info is off
13:50:01.732 32  switches: flows/sec:  12347  22860  13521  17183  12433  10539
 10957  12525  11969  11998  12149  13279  12818  10866  21827  8273  5109  1039
4  11306  11557  11035  12422  11119  10947  9930  10760  10888  11180  10863  4
405  0  11400   total = 36.885863 per ms
13:50:11.856 32  switches: flows/sec:  25579  12792  25579  15434  25579  25579
 25579  25579  25579  20765  25579  25579  25579  25579  12792  12793  17338  12
789  19515  12787  25579  25579  25324  25579  25579  12787  23539  25579  25579
  19848  23738  24991   total = 70.086370 per ms
13:50:21.974 32  switches: flows/sec:  12787  23813  12788  25575  12787  12787
 12787  23396  12787  15428  12787  12787  12787  12787  25575  25575  12788  25
577  16174  25579  12787  12787  12787  12787  12787  25579  12787  12787  12787
  12787  25579  25579   total = 53.828514 per ms
RESULT: 32 switches 2 tests min/max/avg/stdev = 53828.51/70086.37/61957.44/8128.
93 responses/s
```