République Algérienne Démocratique et Populaire
Université Abou Bakr Belkaid – Tlemcen
Faculté des Sciences
Département d'Informatique

Mémoire de fin d'études

Pour l'obtention du diplôme de Master en Informatique

Option : Réseaux et Systèmes Distribués (R.S.D)

Thème

Etude et implémentation d'une solution loadbalacing dans un réseau SDN

Réalisé par :

- **LOURIACHI Zineb**

Présenté le 02 Octobre 2020 devant le jury composé de MM :

- Mr BENAISSA Mohamed *(Président)*
- Mme LABRAOUI Nabila *(Encadreur)*
- Mr. MEHIAOUI Sid Ahmed *(Co-Encadreur)*
- Mr BAMBRIK Ilyes *(Examinateur)*

Année universitaire : 2019-2020

Résumé

Aujourd'hui, les approches SDN bouleversent les systèmes traditionnels des réseaux. Elles initient un «véritable changement » dans les infrastructures réseaux des entreprises et opérateurs avec des promesses en termes de réduction de coûts, d'agilité, de flexibilité, de rapidité de mise en œuvre et d'adaptation. A ce niveau plusieurs modèles de SDN se développent en parallèle pour mieux répondre à des usages spécifiques. Les travaux portent globalement sur la virtualisation et la programmabilité intrinsèque des équipements (nouveaux composants programmables…).

Le travail que nous avons réalisé durant notre projet de fin d'étude, était à mettre en œuvre une plateforme SDN sous le contrôleur Floodlight avec la fonctionnalité de partage de charge, ce qui nous a conduit à comprendre les aspects clés de ce nouveau paradigme.

Dans le contexte de la virtualisation , l'intégration des solutions SDN dans les environnements Cloud est très importante, ce qui mène à développer de nouvelles solutions plus avancées tels que le loadbalancing afin d'interagir avec l'environnement virtualisé comme la solution Cloud (opensource) OpenStack LBaaS (Load-balancer-as-a-Service).

Mots clés: SDN, contrôleur, partage de charge , Cloud, Floodlight, OpenFlow, OpenVSwitch.

Abstract

Today, SDN approaches are disrupting traditional network systems. They initiate a "real change" in the network infrastructures of companies and operators with promises in terms of cost reduction, agility, flexibility, speed of implementation and adaptation. At this level, several SDN models are developing in parallel to better respond to specific uses. The work generally focuses on virtualization and the intrinsic programmability of equipment (new programmable components, etc.).

The work we did was to implement an SDN platform under the Floodlight controller with loadbalancing functionality which led us to understand the key aspects of this new paradigm.

In the context of virtualization, the integration of SDN solutions in Cloud environments is very important, which leads to the development of new more advanced solutions like a loadbalancing in order to interact with the virtualized environment such as the Cloud solution (opensource) OpenStack LBaaS (Load-balancer-as-a-Service).

Keywords: SDN, controller, loadbalancing, Cloud, Floodlight, OpenFlow, OpenVSwitch.

ملخص

تعمل اليوم أساليب الشبكات المبرمجة (أس دي أن) على تعطيل أنظمة الشبكات التقليدية. لقد اظهروا "تغييرًا حقيقيًا" في البنية التحتية لشبكة الشركات والمشغلين بوعود من حيث خفض التكلفة وخفة الحركة والمرونة وسرعة التنفيذ والتكيف. في هذا المستوى، يتم تطوير العديد من نماذج أس دي أن بالتوازي للاستجابة بشكل أفضل لاستخدامات محددة. يركز العمل بشكل عام على المحاكاة الافتراضية وقابلية البرمجة الجوهرية للمعدات (مكونات جديدة قابلة للبرمجة، ...إلخ).

العمل الذي قمنا به هو تنفيذ منصة أس دي أن تحت وحدة تحكم فلودلايت مع وظيفة توزيع العبئ التي أدت بنا إلى فهم الجوانب الرئيسية لهذا النموذج الجديد.

في سياق المحاكاة الافتراضية، يعد دمج حلول أس دي أن في بيئات السحابة أمرًا مهمًا للغاية، مما يؤدي إلى تطوير حلول موازنة العبئ جديدة أكثر تقدمًا من أجل التفاعل مع البيئة الافتراضية مثل حل السحابة (مفتوح المصدر) اوبن LBaaS (موازن التحميل كخدمة).

الكلمات المفتاحية: أس دي أن، وحدة تحكم، توزيع العبئ ، حوسبة سحابية، فلودلايت، اوبن فلو، اوبن في سويتش.

Remerciement

Ma gratitude ainsi que ma dévotion se dirigent en premier lieu vers mon Créateur, le Tout Puissant et Miséricordieux « ALLAH » qui m'a offert la vie et m'a toujours guidé lors de ma route. Dieu qui m'a donné une force, un courage et plein de volontés et de patiences afin de faire face aux obstacles et de mener à bien mes travaux, ce modeste mémoire inclus.

Je tiens à exprimer mes remercîments les plus sincères à mon encadreur, Madame LABRAOUI Nabila, pour avoir accepté de m'encadrer et aussi pour l'aide qu'elle m'a accordé tout au long de mon travail, moyennant ses critiques constructives et ses suggestions pertinentes.

Je tiens à exprimer aussi ma gratitude et mon respect à mon Co-encadreur Monsieur MEHIAOUI Sid Ahmed pour ses conseils multiformes et la richesse de ces connaissances, cela m'a permis de mener à terme ce travail. J'ai eu le plaisir de travailler avec lui. Je n'oublierai jamais ses conseils, ses remarques, ses critiques et surtout ses sacrifices. Je lui exprime toute ma reconnaissance.

Je remercie les membres du jury de m'avoir fait l'honneur d'accepter d'examiner ce travail. Je tiens à exprimer ma reconnaissance à Monsieur BENAISSA Mohamed qui a accepté de présider le jury, et je n'oublierai surement pas Monsieur BEMBRIK Ilyes pour le temps consacré à examiner ce mémoire, ainsi que d'avoir bien voulu faire partie du jury.

Je sais gré à mes enseignants du Département d'Informatique, qui m'ont accompagné et aidé à m'améliorer durant toute ma formation.

Je souhaite adresser mes remerciements aux personnes qui nous ont apporté leur aide et contribuer à l'élaboration de ce mémoire ainsi qu'à la réussite de cette année universitaire sans oublier le corps professoral et administratif de notre faculté : la faculté des Sciences.

Dédicaces

Ce modeste travail est dédié :

A tous ceux qui me sont chères ;

A tous ceux qui m'aiment ;

A tous ceux que j'aime.

ZINEB

TABLES DES MATIERES

INTRODUCTION GENERALE

Durant ces dernières années, les réseaux informatiques traditionnelles ont connu de grands défis et avec l'arrivée des réseaux définis par logiciels (SDN) a permis de faciliter la gestion des réseaux en proposant la possibilité de définir leurs politiques sous forme de programmes de contrôle. Ceci soulage les administrateurs du devoir de configurer chaque équipement à part afin de fixer une politique donnée.

En effet, SDN permet principalement de centraliser la logique déterminant les politiques de gestion d'un réseau dans une ou plusieurs unités appelées contrôleurs. Ces contrôleurs communiquent avec le reste des équipements du réseau via des interfaces ouvertes.

L'architecture SDN a été adoptée par plusieurs grandes entreprises et universités à savoir, Google et Stanford. D'autre part, les fabricants des équipements réseaux tels que Cisco, HP, Huawei et Juniper offrent désormais des solutions SDN. Ainsi, ils commencent actuellement à intégrer dans leurs nouveaux équipements les protocoles ouverts propres aux SDN tel que OpenFlow.

La solution SDN n'a pas séparé juste le plan contrôle par rapport aux données dans les réseaux mais a apporté de nouvelles approches et nouvelles fonctionnalités qui n'existent pas auparavant dans les équipements des réseaux traditionnels où ces taches spécifiques ont été assurer par des serveurs dédiés ou bien des équipements spécifiques tel que le serveur de partage de charge, Firewall, ect..

INTERET DU SUJET

Dans le cadre de notre stage au niveau de l'opérateur ALGERIE-TELECOM et en prenant en considération la croissance de l'infrastructure de l'entreprise en termes de réseaux et plateformes et vue la nécessité de migration vers de nouvelles solutions SDN, nous avons choisi notre projet en mettant en œuvre une solution de partage de charge entre plusieurs serveurs dans un environnement SDN.

Le but de ce projet est d'implémenter la solution loadbalancing sous le contrôleur floodlight et d'évaluer les performances de la méthode appliquée.

Plan du mémoire

Ce mémoire s'organiser comme suit :

- **Chapitre 1** : présente l'évolution des architectures réseaux vers les réseaux définis par logiciels (SDN) en abordant cette technologie et ces protocoles.
- **Chapitre 2 :** présente une solution SDN sous le contrôleur open source Floodlight et d'autres outils tel que OVS et mininet.
- **Chapitre 3 :** présente l'implémentation d'une plateforme SDN sous Floodlight, mettre en œuvre la solution du loadbalancing et évaluer ses performances.

Nous terminerons par une conclusion et quelques perspectives pour des travaux futurs.

CHAPITRE I :

Evolutions des architectures des réseaux vers SDN

Sommaire :

CHAPITERE.I Evolutions des architectures des réseaux vers SDN

I.1 Introduction

Si on va aborder le sujet des réseaux traditionnels, on va constater que ce sont à la base des systèmes de calculs distribués, généralement constituer des nœuds hétérogènes (switches, routeurs, firewalls, …) et ces réseaux-là doivent être configuré d'une façon cohérente ç-à-d les services qui vont être implémentés doivent être cohérents pour ne pas interférer avec les autres services donc le problème de ce réseau qu'il est très difficile de le faire évoluer et s'il y a un changement quelconque dans le réseau ce qu'il affecte le réseau tout entier, c'est pour ça que les opérateurs ont un très peu du changement dans le réseau et l'évolution de leurs datacenters ou de leurs réseaux est très lente.

C'est pour cela, les réseaux définis par logiciel « SDN » ont encaissé une réputation dans les milieux industriels et de recherches, un modèle qui vient brouiller la balle réseau avec le souhait de changer notre perception de l'infrastructure.

Dans ce chapitre nous allons parler brièvement des réseaux traditionnels et les problèmes reliés à ces réseaux. On va revenir aussi à la source de la mutation des SDN, comprendre les facteurs qui ont poussé à l'arrivée de ce paradigme. Ensuite nous présenterons les principes de fonctionnement du protocole SDN, ses interfaces et le protocole utilisé qui est l'OpenFlow.

I.2 Réseaux d'entreprises : WAN, Campus

Les dernières années ont été marquées par un changement radical de la perception des équipements de travail en entreprise et dans les campus universitaires. En effet la mode BYOD , ou chaque employée et étudiant utilise son propre matériel sur le campus, ce qui nous pousse à repenser le réseau pour y infuser un souffle de flexibilité sans lequel l'administrateur du réseau est très vite pris de court par la quantité de trafic à gérer manuellement et par les opérations manuelles qui pourrait permettre de connecter tous les utilisateurs à internet par exemple. Le SDN dont l'un des avantages est l'automatisation et la programmation des réseaux nous semble apparemment comme une évidence pour combler les lacunes des réseaux de campus qui deviennent de plus en plus complexe. Une des applications les plus intéressantes pour les campus au-delà de la

virtualisation, est la programmation d'un réseau conscient d'applications. Le nombre d'applications tournant sur les équipements des utilisateurs est considérable ce qui conduit dans la plupart des cas à une surcharge sur le réseau et peut induire une latence sur les applications déployés dans un but business ou académique. SDN permet de gérer ces applications en introduisant des priorités pour les différentes applications et en installant un équilibreur de charge (Load Balancer), éliminant ainsi toute forme de congestion, ce qui offre aux professionnels et étudiants un environnement de travail fluide.

Pour le réseau WAN qui est un réseau longue distance à l'échelle d'un pays où d'un continent, il permet l'interconnexion de plusieurs LAN ou MAN sur des grandes distances géographiques. Le plus grand WAN est le réseau internet qui est un WAN public. Il est à noter que les organisations internationales peuvent créer des WAN privés comme par exemple des réseaux banquiers. Les WAN utilisent des infrastructures de fibres optiques, de câbles sous-marins internationaux ou des transmissions par satellite.

I.3 Fournisseurs de services Réseaux

Les fournisseurs de services ce sont des personnes physiques ou morales qui fournissent un ou plusieurs services aux utilisateurs d'un système de télécommunication[1]. Plus précisément les fournisseurs de services réseaux se sont des entreprises ou des organisations qui vendent de la bande passante ou un accès réseau en fournissant un accès direct à la dorsale Internet aux fournisseurs de services Internet et généralement un accès à ses points d'accès réseau.

Pour la fibre optique et dans le cadre des besoins spécifiques aux entreprises et aux administrations, les services des opérateurs permettent de répondre à des besoins parfois élevés de sécurité et de disponibilité des services par exemple une double adduction d'un site ç-à-d deux accès fibres optiques distincts pour le même site mais surtout basés sur des engagements contractuels (disponibilité, délai de réparation…). Pour les particuliers, les services correspondent à des besoins domestiques ne nécessitant pas aujourd'hui de tels engagements.[2]

Sur les réseaux FttH (Fibe to the Home), les fournisseurs d'accès Internet proposent des débits symétriques pouvant aller jusqu'à **200 Mbits/s** ce qui permet une pleine **simultanéité des usages** et ouvre des possibilités nouvelles comme le multi-écrans, la télévision Haute Définition, les services en ligne, la télémédecine, etc… Le

raccordement final du particulier ou de l'entreprise sur un réseau FttH est réalisé à la suite de l'ouverture commerciale du réseau et sur contractualisation de l'abonné pour un service avec un FAI [2].

Il y a aussi le lien de peering qui est un accord entre deux opérateurs, pour s'échanger directement le trafic de leur réseau respectif. Un lien de peering était gratuit au début de l'Internet, mais il est de plus en plus souvent payant. L'opérateur qui envoie plus de trafic vers l'autre étant celui qui paye [3]. Les FAI peuvent s'engager dans le peering, où plusieurs FAI s'interconnectent à des points d'appairage ou à des points d'échange Internet (IXP), permettant le routage des données entre chaque réseau. Le matériel, les logiciels et les spécifications du réseau, ainsi que l'expertise du personnel de gestion du réseau sont importants pour garantir que les données suivent l'itinéraire le plus efficace et que les connexions en amont fonctionnent de manière fiable. Un compromis entre coût et efficacité est possible.

I.4 La virtualisation

Si on définit la virtualisation c'est tout simplement une technologie qui permet de créer et d'exécuter une ou plusieurs représentations virtuelles d'un ordinateur ou de ses différentes ressources sur une même machine physique. Autrement dit elle permet d'exécuter sur une machine hôte, dans un environnement isolé, **des systèmes d'exploitations** ou **des applications**, on parle alors de virtualisation système ou virtualisation applicative, ces ordinateurs virtuels sont appelés VPS ou encore VE.

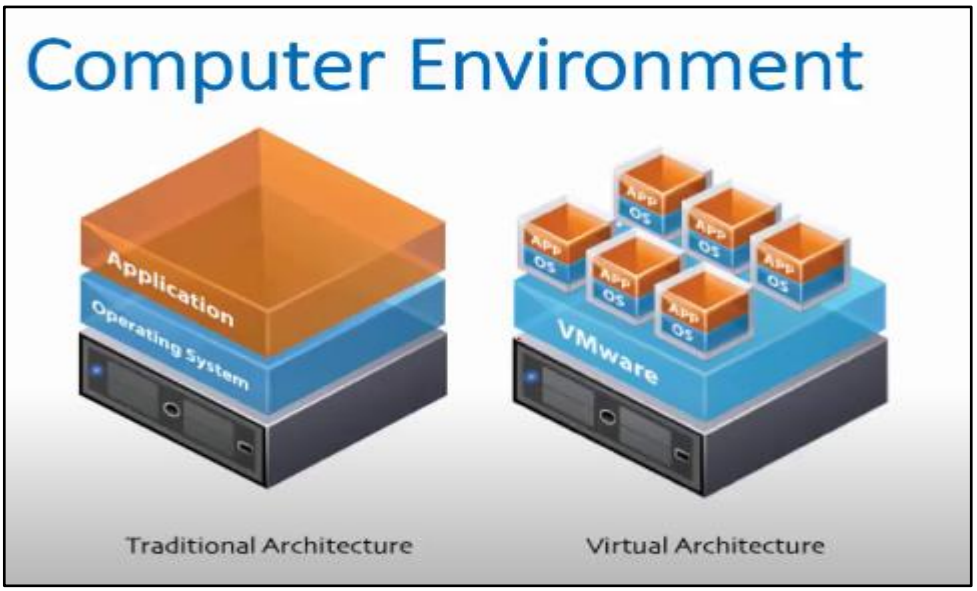


Figure I-1 : Architecture traditionnelle VS Architecture Virtuelle

I.5 Problèmes et challenges liés aux architectures actuelles des réseaux

Généralement un nœud réseau dans une architecture traditionnelle est constitué de trois couches ou plans, le Plan de gestion c'est le responsable de la configuration(protocole, Access-List…) ou de l'interaction entre l'utilisateur et le nœud réseau, c'est à travers ce plan qu'on peut configurer ses liens à travers des interfaces web CLI ou GUI, ces actions sont faites soit par un administrateur réseau ou le directeur. On a aussi le Plan de contrôle qui est la partie intelligente dans l'équipement, c'est lui le responsable de ce qui est tables et protocoles de routages, de l'interaction avec les nœuds ou les routeurs adjacents, il est responsable par exemple de la BDD des états des liens et c'est lui aussi qui donne les instructions aux plan de données pour gérer ou traiter ces paquets. En plus on a le Plan de donnée c'est la partie d'exécution, il suit les ordres du plan de contrôle, il ne fait que traiter ou transporter les paquets/tables selon les instructions de ce plan.

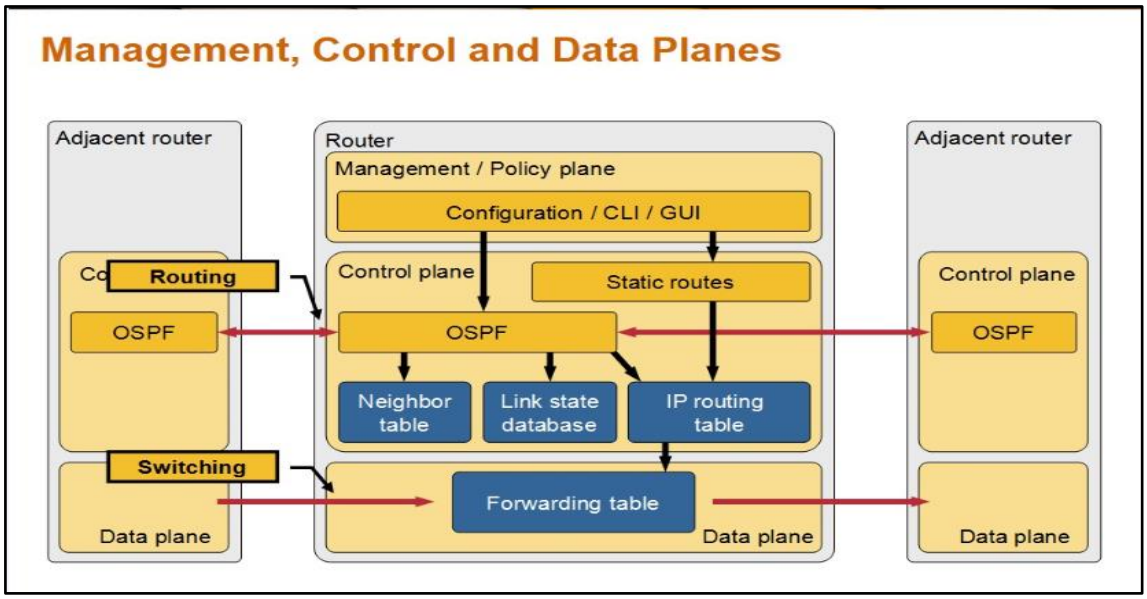


Figure I-2 : Architecture d'un nœud réseau, exemple d'un routeur

Parmi les inconvénients et les problèmes des réseaux actuels, on peut citer :

- *Lent* Supposant que j'ai 100 switches et je veux les configurer, automatiquement je vais la faire sur les 100 équipements et si j'ai par exemple une nouvelle technologie à utiliser, aussi je dois la charger sur les 100 équipements ou bien changer les 100 équipements et apporter cette nouvelle technologie donc surement l'interaction avec mon réseaux sera lente.

➢ *Erreurs/fautes humains* Je peux saisir par exemple une adresse IP fausse donc j'aurai surement des surprises.

➢ *Temps de convergence énorme* Dans le cas où un lien ou un réseau X est en panne. La figure I.3 montre qu'on aura une perte de temps.

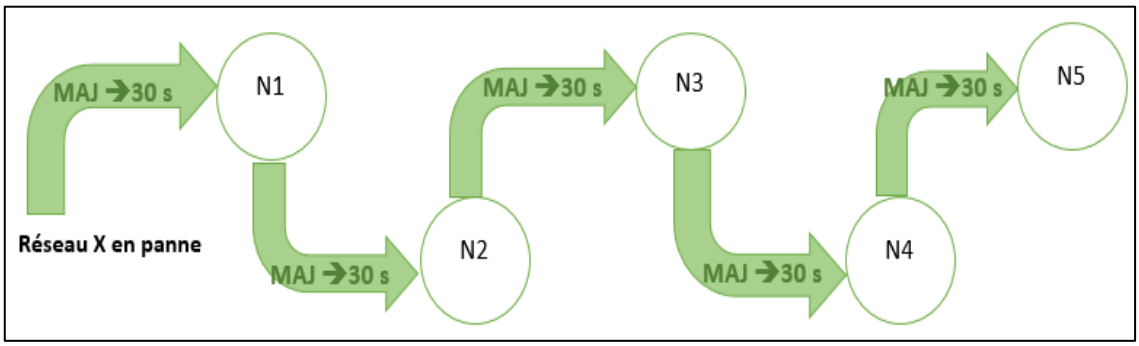


Figure I-3 : Schéma explicatif d'une conversion énorme

➢ *Limite d'automatisation* je n'ai pas un équipement qui gère les autres.

Exemple : si l'appareil A tombe en panne ➔ faire ça et si équipement B n'a pas de connexion ➔ faire cela ç-à-d on n'a pas un coordinateur.

➢ *Limite d'innovation* l'espace d'innovation est limiter.

Pour cela le besoin nous a mené à laisser que le plan de données dans les équipements et on enlève le cerveau(plan de contrôle)de chaque appareil en les rassemblant dans un contrôleur(cerveau, appareil) donc on va se baser maintenant sur deux plans: le plan de contrôle et le plan de données comme montre la figure ci-dessous.

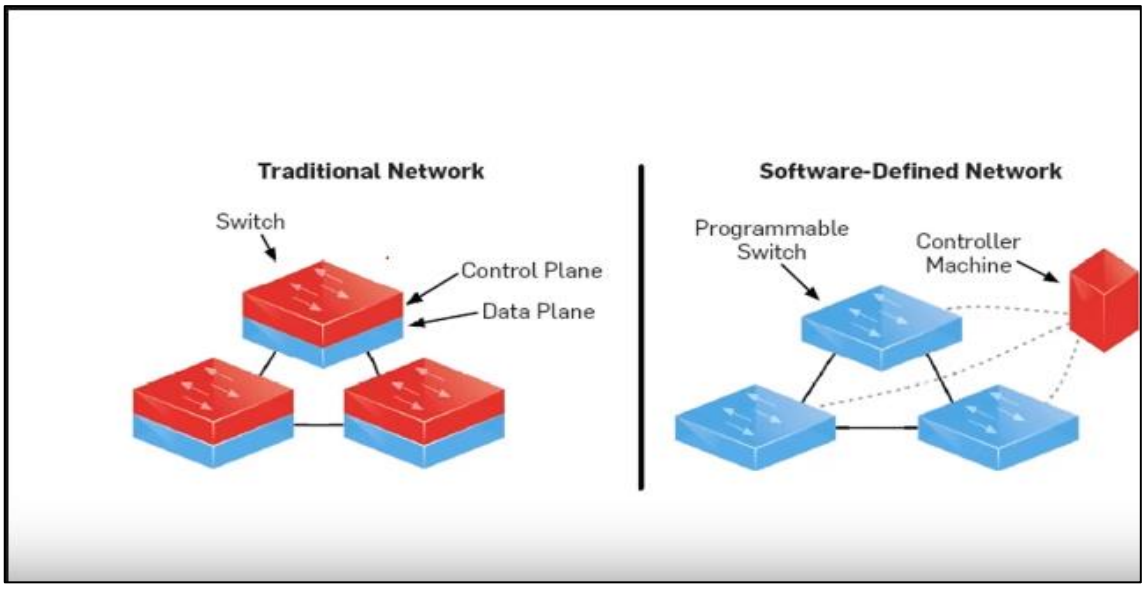


Figure I-4 : Réseaux traditionnels VS SDN

I.6 La naissance et historique du SDN

L'histoire c'est passée en 2007 quand Martin Casado et deux autres personnes ont créé l'entreprise NICIRA, en parallèle Martin travaillait sur sa thèse de doctorat basé sur le développement d'OpenFlow. Ce dernier lui permet d'exécuter le premier OVS (supporte OpenFlow) à travers son entreprise. Cinq ans plus tard, VMWare a acheté NICIRA car elle a senti que cette notion d'OpenFlow va faire une mutation dans le domaine des réseaux dans les prochaines années et a laissé Martin de devenir le directeur.

Maintenant le rôle ou le principe d'OpenFlow est de séparer le plan de contrôle, le mettre dans un équipement nommé contrôleur, du plan de données qui va devenir qu'un exécutable des ordres du contrôleur comme ça la gestion deviendra centralisée et plus facile. Donc OpenFlow devient l'intermédiaire entre les deux et c'est là où qu'elle est née la notion du SDN.

parmi les avantages du SDN, on site:

- Diminution de la charge du travail ;
- Diminution du cout ;
- Flexibilité.

I.7 L'architecture de référence SDN proposé par l'ONF

SDN signifie Software Defined Networking, ç-à-d un réseau défini par logiciel. Sa définition académique, consistait à voir le SDN comme une architecture qui sépare les fonctions de contrôle et de transfert des données du réseau afin d'avoir une infrastructure physique complètement libre de tout service réseau. Le SDN est donc reconnu aujourd'hui comme une architecture permettant d'ouvrir le réseau aux applications. La base d'une architecture SDN est de découpler la couche de contrôle de la couche de données, cela est illustré dans la Figure I-5.

L'architecture SDN est basée sur trois couches essentielles :

- *Couche application* (business application) Contient les programmes à utiliser pour fonctionner des logiciels ou des protocoles de routage ou des outils qu'on va les ajouter au contrôleur pour les utiliser et les exécuter sur mon équipement.

- *Couche contrôle* (contrôleur SDN) Il contrôle les logiciels ou les applications qu'il reçoit de la couche application en utilisant des API et il se concentre sur les équipements qui se situe dans la couche infrastructure comme OpenFlow Switches.

- *Couche infrastructure* Il contient des équipements réseaux comme des OpenFlow Switches, on n'aura pas des switches ou des routeurs ordinaires ç-à-d pas de tables d'adresses MAC et pas de tables de routages, on aura que des OVS (Open Virtuel Switch) qui fonctionnent avec OpenFlow en utilisant tables de flux.

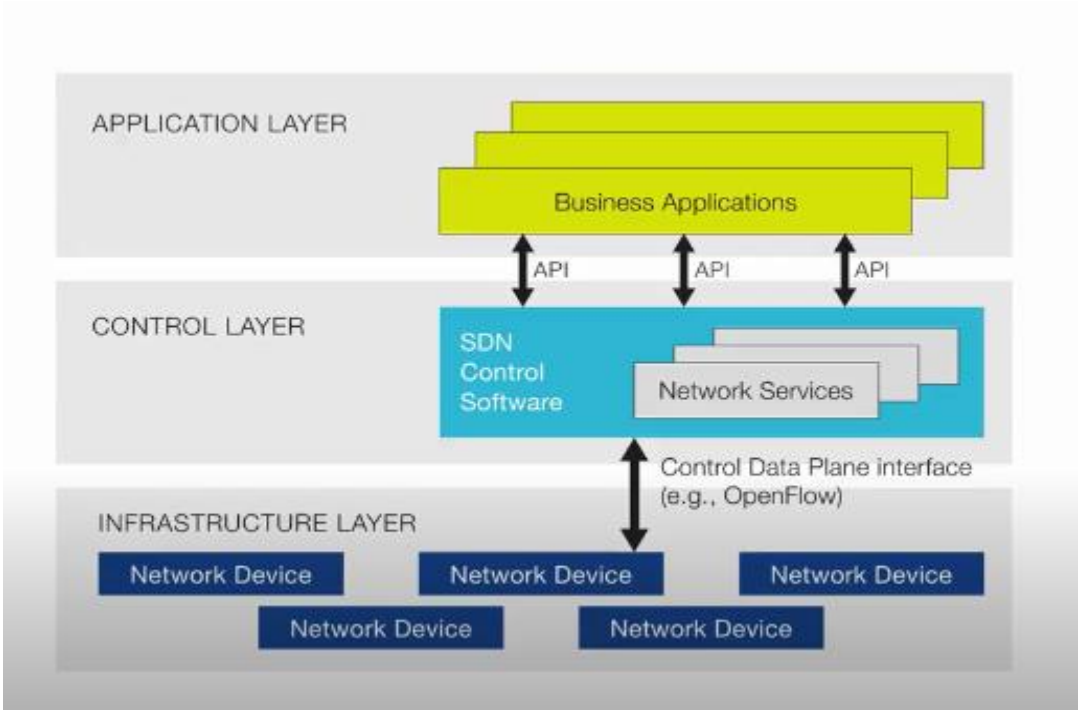


Figure I-5 : Architecture du SDN

I.8 Le contrôleur SDN et les interfaces North Bound/South Bound

Comme déjà vu précédemment, les équipements réseaux vont adapter des règles introduites par les applications. On site par exemple une abstraction habile appelée **contrôleur** qui réunit les plans de contrôles de tous les équipements du réseau, aperçoit le réseau dans sa totalité et introduit directement les règles de traitement des données dans chaque équipement, et entre chaque couche du SDN en trouve :

Interface direction nord (NBI) C'est l'intermédiaire entre le contrôleur SDN et la couche application, l'intégration est fournie généralement via les interfaces: XML, JSON [4].

Interface direction sud (SBI) C'est l'intermédiaire entre le contrôleur SDN et le matériel (exécutables) de la couche infrastructure, comme OpenFlow. Cette interface permet au SDN de se connecter aux éléments de réseau (NE) et de les gérer, l'intégration de ces NE aux systèmes de gestion de réseau (NMS) comme SDN est assurée par les interfaces suivantes: SNMP , CLI , FTP / SFTP , Telnet / SSH …[4]

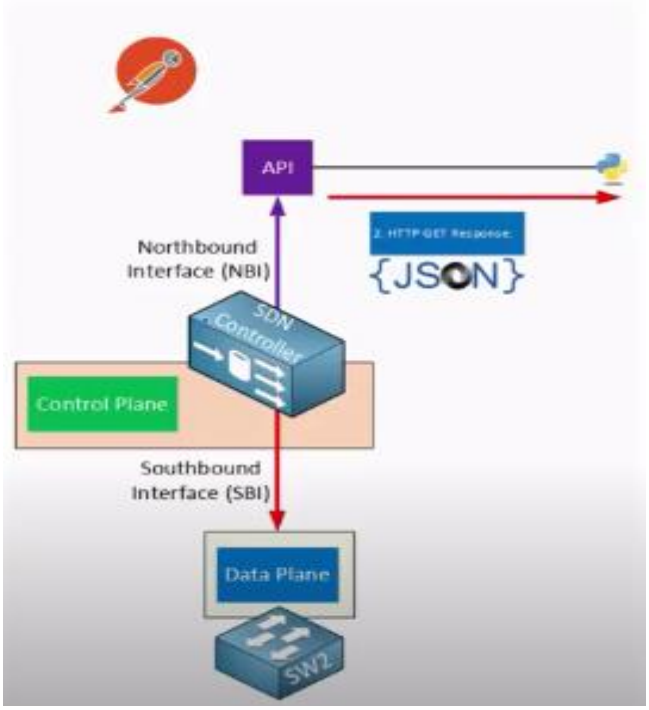


Figure I-6 : NBI & SBI

I.9 Le protocole OpenFlow

Le protocole Open Flow permet l'échange de messages entre le contrôleur et le Switch, OpenFlow est l'intermédiaire entre le contrôleur et les équipements (OpenFlow switches) donc il aide à concrétiser la séparation des 2 couches. Généralement on trouve deux types d'OpenFlow soit pure ou hybride.

Pure: Le switch ou le commutateur est basé que sur OpenFlow.

Hybride: c'est un commutateur ordinaire qui est basé sur un OS précis et en même temps il supporte OpenFlow..

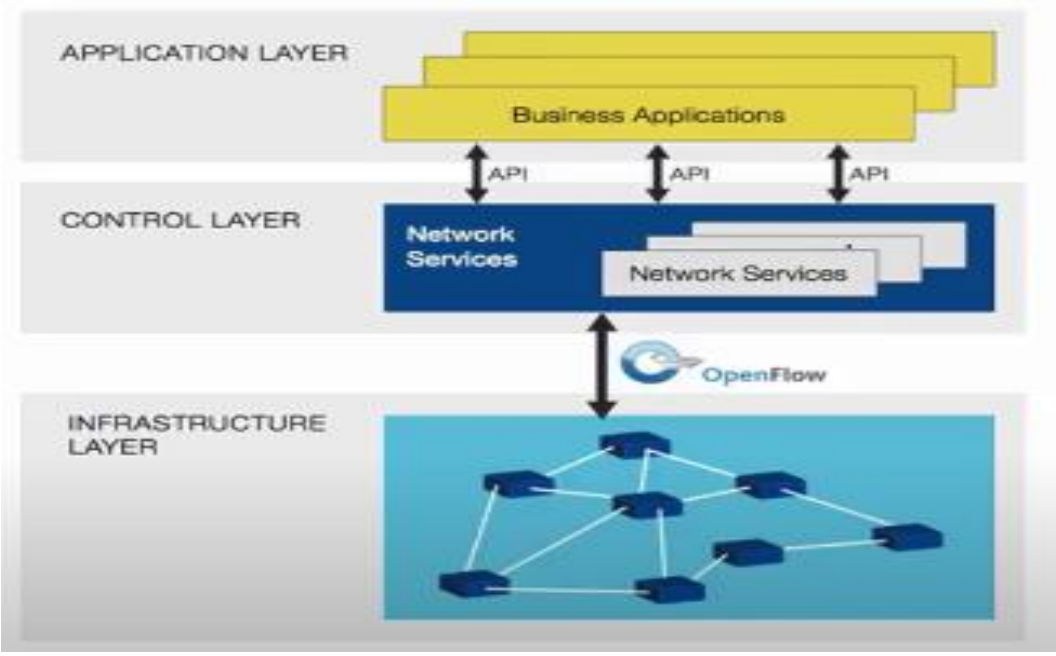


Figure I-7 : OpenFlow dans l'architecture SDN

I.8.1 Les différentes versions du protocole OpenFlow

Un livre blanc pose en 2008 les bases de ce protocole. La dernière version de spécification d'un switch OpenFlow est 1.5.1. Un rapport annuel de 2016 positionne OpenFlow dans une phase décisive:

- son évolution étant prise en charge par une communauté Open Source ;
- en étant portée par des acteurs commerciaux qui intégreront cette technologie dans leur produit.[5]

Le tableau suivant montre les différentes versions d'OpenFlow avec description des additions apportées par chaque version:

Version Open Flow	Additions apportées par la version
V 1.0	- Les paquets Ethernet et IP sont identifiés selon l'adresse source et de destination, port source ou destination UDP et TCP. - Champs Ethernet-type et Vlan pour la couche 2. - Champs protocole DC et ECN pour la couche 3.
V 1.1	- Ajout d'un pipeline, de la table de groupe et des métadonnées. - Ajout des champs d'identification MPLS.

V 1.2	- Model OXM qui apporte un support IPv6 et une structure de correspondance plus flexible. - Un Switch peut à présent être connecté à plusieurs contrôleurs en mode Master/Slave.
V 1.3	- Ajout de la table de mesure(meter table) qui mesure le taux de paquets assigné à une entrée. - Amélioration du support des contrôleurs multiples.
V 1.4	- Amélioration d'OXM et de la structure TLV ce qui apporte un gain de synchronisation.
V1.5	- Notion des tables de sortie qui permet de traiter les paquets sur le port de sortie en même que sur les portes d'entrée.

Tableau I-1 : Tableau comparatif des différentes versions d'Open Flow

I.8.2 Le fonctionnement du protocole Open Flow

Le commutateur OpenFlow communique avec le contrôleur via une interface de communication sur un canal sécurisé, via une connexion TLS par une session de base TCP avec un port 6653/6633. Par la suite le contrôleur peut ajouter des entrées dans les tables, modifier ou supprimer des entrées dans la table de flux, chaque entrée se compose :

- De champs de correspondances.
- De compteurs.
- D'un ensemble d'instructions à appliquer aux paquets.

Il peut aussi gérer les cas extrêmes tel que par exemple si on a un paquet qui n'a pas d'entrées dans la table de flux donc le commutateur va forwarder vers le contrôleur qui lui peut décider de son sort soit le forwarder soit le jeter.

Donc ce protocole n'a pas besoin d'être lié à un OS bien précis pour fonctionner, il est écrit dans des tables de transfert alors avec ça on éliminera l'intermédiaire de l'OS. Si le lien entre le commutateur est le contrôleur est direct, on constate que le port Open Flow est physique sinon il est logique.

I.8.3 Les messages et les tables des Flux d'Open Flow

❖ La communication entre le commutateur et le contrôleur est réaliser en envoyant et recevant des messages alors il existe trois types de messages d'OpenFlow, messages contrôleur vers OVS, messages asynchrones et messages symétriques

a- Messages contrôleur vers commutateur : (messages de sorties) Les messages contrôleur-commutateur représentent la catégorie la plus importante des messages OpenFlow. Ils se divisent en sous-catégories :

a-1) Switch configuration contient : le message SET_CONFIG qui positionne les paramètres de configuration du commutateur, la paire FEATURES_REQUEST et FEATURES_REPLY permettent d'interroger le commutateur au sujet des fonctionnalités qu'il supporte et la paire de message GET_CONFIG_REQUEST et GET_CONFIG_REPLY est utilisée pour obtenir la configuration du commutateur.

a-2) Command from controller qui inclue : PACKET-OUT permet d'émettre des paquets de données au commutateur pour leur acheminement du plan de données, le message FLOW_MOD modifie les entrées de flux existantes dans le commutateur, PORT_MOD est utilisé pour modifier l'état d 'un port OpenFlow.

a-3) statics la paire de message STATS_REQUEST et STATS_REPLY permet d'avoir des statistiques du commutateur par le contrôleur.

a-4) queue configuration le message 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'attentes spécifiques et ainsi fournir à ces paquets un niveau de QoS désiré.

a-5) barrier le message BARRIER_REQUEST 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. [6]

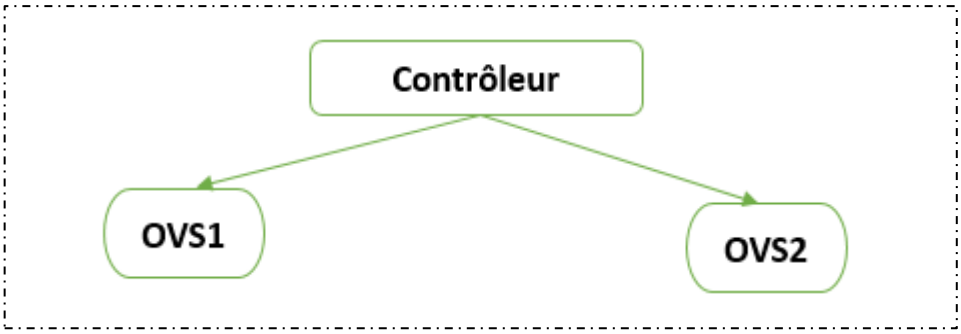


Figure I-8 : Chemin des messages du contrôleur vers les OVS

b- Messages asynchrones *: (messages d'entrés)* du commutateur vers contrôleur. Les messages PACKET_IN sont utilisés pour passer les paquets de données au contrôleur pour leur prise en charge, pour relayer le trafic du plan de contrôle et contiennent une partie de l'en-tête et un buffer_ID à utiliser par le contrôleur si le commutateur dispose d'une mémoire suffisante pour mémoriser les paquets qui sont envoyés au contrôleur sinon émettent le paquet entier au contrôleur. Le message FLOW_REMOVED pour informer le contrôleur qu'une entrée de flux a été supprimée de la table de flux. Le message PORT_STATUS est utilisé afin de communiquer un changement de configuration du port ou changement d'état du port. Finalement le commutateur utilise le message ERROR pour notifier des erreurs au contrôleur.[6]

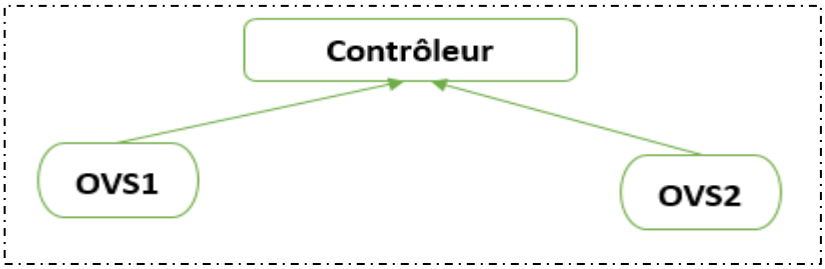


Figure I-9 : Chemin des messages asynchrones

c- Messages symétriques : Les messages symétriques qui sont envoyées dans un canal sécurisé peuvent être émis indistinctement par le contrôleur ou le commutateur(OVS) sans avoir été sollicité par l'autre entité. On a les messages HELLO qui sont échangés afin de déterminer le numéro de version OpenFlow le plus élevé supporté par le commutateur et le contrôleur; Le protocole spécifie que la plus petite des deux versions doit être utilisée pour la communication contrôleur-commutateur, on a aussi les messages ECHO qui sont utilisés par n'importe quel entité pendant le fonctionnement du canal pour s'assurer que la connexion est toujours en vie et afin de mesurer la latence

et le débit courants de la connexion. Chaque message ECHO_REQUEST doit être renvoyé par un message ECHO_REPLY. [6]

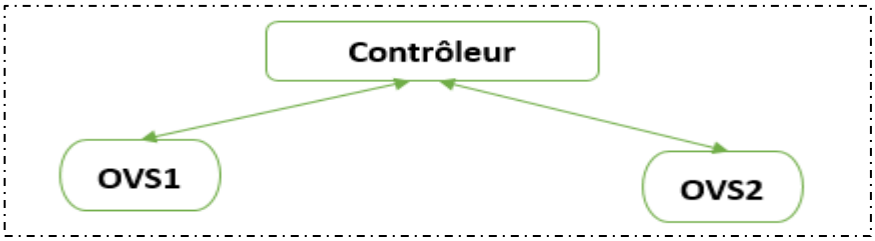


Figure I-10 : Chemin des messages symétriques

Les messages associés peuvent être regroupés à l'aide de messages groupés(bundle). Ce bundle est généralement configuré comme une commande. Ce groupe est exécuté dans un ordre spécifique par le commutateur.

❖ Un switch qui supporte OpenFlow est constitué de plusieurs tables : des tables de flux et des tables de groupe.

Tables de flux : sont constituer des champs de correspondances et des champs d'instructions.

La table de groupe : génère généralement les cas exceptionnels comme le load-balancing, le multicast, forwarding des données vers plusieurs ports d'une façon aléatoire.

❖ Les entrées de flux se divisent en deux parties : réactive et proactive

Réactive : Si on prend l'architecture ci-dessous comme exemple, X veut communiquer avec Y donc X va communiquer avec OVS1, à son tour il va envoyer la demande au contrôleur donc le contrôleur lui désigne la sortie et l'OVS1 va remplir sa table de flux à fur et à mesure.

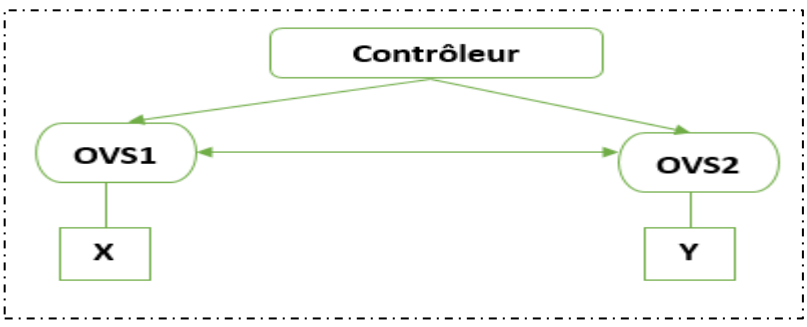


Figure I-11 : Exemple descriptif des entrées de flux proactives

Proactive : La table de flux de l'OVS est rempli dès le début.

I.10 Les modèle de déploiement SDN : SDN-WAN, SDN-LAN

Tout comme le SDN, le WAN défini par logiciel sépare les plans de contrôle et de données du WAN et permet le contrôle des éléments WAN, physiques et virtuels et bien que le SDN soit une architecture, le SD-WAN est une technologie achetable.

Une grande partie de la technologie qui compose le SD-WAN n'est pas nouvelle; il s'agit d'un partage de la bande passante du réseau entre les points de connexion d'une façon dynamique. Cette technologie est attirante pour de nombreuses entreprises pour sa facilité de déploiement, sa gestion centralisée et ses coûts réduits. Lerner estime qu'un SD-WAN peut être jusqu'à deux fois et demie moins cher qu'une architecture WAN traditionnelle.[7]

Pour le SD-LAN aussi qui s'appuie sur les principes du SDN dans le centre de données et du SD-WAN pour apporter des avantages spécifiques(Adaptabilité, Flexibilité, Rentabilité, Evolutivité aux réseaux d'accès filaires et sans fil). Tout cela en assurant une continuité d'activité essentielle à la couche d'accès au réseau.

En termes simples SD-LAN est une architecture axée sur les applications et les politiques qui marquent les couches matérielles et logicielles en créant des réseaux auto-organisés et gérés de manière centralisée, plus simples à exploiter, à intégrer et à mettre à l'échelle.[7]

I.11 Les contrôleurs SDN

En termes de contrôleurs, on note l'existence d'une multitude parmi lesquels on peut énumérer :

Nox et Pox *:* qui sont les deux premiers contrôleurs OpenFlow. NOX sur C++ et POX sur Python**.**

Becon : qui est un contrôleur OpenFlow rapide, multiplateforme, modulaire basé sur Java.

Ryu : un contrôleur SDN capable de configurer les équipements réseaux en utilisant OpenFlow, NetConf et OF-config.

FloodLight : un contrôleur supporté par BigSwitch ; il est sous licence Apache et écrit en Java.

OpenDayLight : Le projet OpenDaylight est un projet open source pour le Software Defined Network qui a été annoncé publiquement le 8 Avril 2013. C'est un projet de collaboration hébergé par la Fondation Linux. Il contient un contrôleur modulaire et souple.

La liste n'étant pas exhaustive, on peut y ajouter : MUL, Maestro, Trema, Jaxon, Helios, SNAC, NodeFlow, Flowvisor, Ovs-controller, RouteFlow.

I.12 Conclusion

On essaie toujours d'améliorer et d'évoluer les architectures réseaux pour avoir des meilleures réponses, meilleurs réactions et surtout meilleurs solutions. C'est pourquoi le monde se dirige vers le SDN.

Comme on a vue dans ce chapitre d'une façon théorique que SDN est une séparation de la couche de contrôle de la couche de donnée, il est basé sur un contrôleur centralisé. Il nous permet la simplicité de l'architecture du réseau, la flexibilité du réseau et aussi la programmabilité des réseaux.

Ce fameux contrôleur SDN qui est situé dans la couche de contrôle va communiquer avec les autres équipements (OVS) en utilisant le protocole OpenFlow qui est écrit dans des tables de transfert directement sans OS et il nous permet une suppression complète du plan de contrôle.

Nous présenterons dans le chapitre suivant les outils que nous allons utiliser dans notre implémentation.

CHAPITRE II:

Outils utilisés pour l'implémentation des réseaux SDN

Sommaire :

CHAPITERE.II Outils utilisés pour l'implémentation des réseaux SDN

II.1 Introduction

Après que nous sommes familiarisés avec les fondements du SDN, nous allons à présent nous intéresser à son implémentation en incluant une solution de partage de charge dans le cas d'un réseau SDN.

II.2 Infrastructure

L'approche du SDN a apporté sa part de changements dans la couche physique, outre la simplification du matériel de transmission. Contradictoirement au protocole STP qui bloque les routes redondantes pour éviter les boucles, le SDN opte sur le partage de charge entre les liens, augmentant la capacité de transport des données et mettant à profit les architectures à très larges bandes passantes.

Dans le cadre de ce mémoire nous allons déployer une solution loadbalacing dans un réseau SDN. le déploiement ne se fera pas avec des switches matériels, nous exploiterons la nature portable et orientée logiciel du SDN pour implémenter l'infrastructure dans un environnement virtuel. Cette option est plus intéressante pour notre projet vu la richesse des switches virtuels en possibilités confrontés à leurs équivalents matériels.

II.2.1 Open Virtuel Switch

Open Virtuel Switch OVS est un commutateur logiciel multicouches crée par NICIRA, soutenu après par la fondation Linux. Son implémentation principale présentant dans les réseaux virtualisés et les data center, par sa mobilité et sa réaction dynamique aux changements d'états, elle s'étale aux architectures physiques ou l'OVS, peut être installé sur un matériel open source. Nous avons choisi d'utiliser l'OpenVSwitch pour notre projet étant l'un des meilleurs commutateurs SDN du moment.

II.2.1.1 Anatomie d'OVS

L'OVS est principalement composé des éléments suivants : [8]

- **ovsdb-server** : est un serveur BDD ou sont stockés les différentes configurations du switch, celui-ci permet de garder une continuité dans le comportement du switch au-delà du redémarrage de celui-ci.

- **ovs-vswitchd** : est le processus principal dans le fonctionnement du switch, il permet entre autres la communication avec le contrôleur via OpenFlow ou encore la récupération des données se trouvant dans la BDD.

- **openvswitch.ko** (Module Kernel) : Le Kernel est le noyau du système d'exploitation, c'est la partie qui s'occupe réellement de la commutation et du tunneling.

La communication de l'utilisateur avec ces composantes constitue une étape importante dans l'implantation de l'OVS. Trois outils d'espace utilisateur sont à disposition pour configurer et surveiller l'état du switch **ovs-vsctl** , **ovs-ofctl** et **ovs-dpctl**, chacun de ces outils offre un ensemble de commandes permettant d'interagir avec le switch.

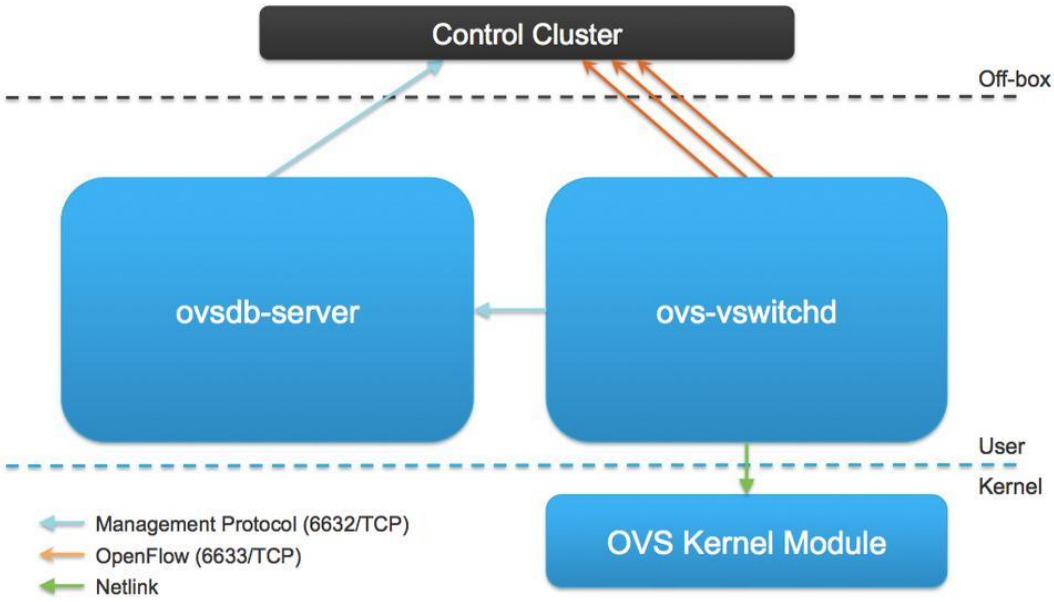


Figure II-1 : Composants d'OVS

II.2.1.2 Fonctionnement d'Open vSwitch

L'OVS est constituée de bridges qui relient les différentes machines virtuels, cependant on peut connecter cette partie virtuelle aux réseaux physiques en attribuant des ports et des interfaces à la multitude de ponts dont le commutateur dispose.

Pour configurer l'OVS, on communique avec ses différentes composantes en utilisant les outils de l'espace utilisateur cités dans la section précédente. Ci-dessous nous allons

aborder certaines des commandes les plus en vue dans une architecture axée sur Open Flow.

- La commande **ovs-vsctl add-br br0** permet de créer un pont, par défaut une interface du même nom (br0).
- La commande **ovs-vsctl add-port br0 eth1** permet la création des ports/interfaces.
- La commande **ovs-vsctl show** pour connaitre l'état global du switch.

Pour l'architecture SDN, nous devons ensuite connecter le switch à un contrôleur, pour le faire on introduit **ovs-vsctl set-contoller br0 tcp= ip-controler :6633** ou 6633 le port d'écoute du protocole Open Flow. A partir de là pour communiquer avec Open Flow l'outil privilégié est **ovs-ofctl**.

- La commande **ovs-ofctl -O Open Flow 13 dump-flows br0** affichera les différents flux installés sur le switch.
- **Add-flow** permet d'ajouter des flux directement sur le switch sans passer par un contrôleur. Dans l'exemple présent on ajoute un simple flux **ovs-ofctl -O Open Flow 13 add-flow br0 in_port=2 actions=output :3** ou tous les paquets reçus dans le port 2 sont renvoyés par le port 3. On peut vérifier que le flux a été bien ajouté en repassant par la commande dump-flows.

II.2.2 Mininet

Mininet est un émulateur de réseau open source, il fonctionne sur le noyau de Linux et utilise Python pour son API. Il permet de créer des topologies réseaux qui se composent d'un ensemble d'hôtes, de switches, de contrôleurs et des liens virtuels. Il fournit la capacité de créer ces composants via : ligne de commande, interface interactive ou script Python.

Mininet est très performant car il utilise les techniques de virtualisation réseaux de Linux comme l'espace de noms réseaux pour émuler plusieurs hôtes et il supporte Open Flow pour interagir avec le controlleur SDN.

II.2.2.1 La ligne de commande Mininet

Mininet permet de réaliser maintes formes de topologies et des réseaux virtuels en utilisant la commande **mn .**

sudo mn : elle crée une topologie par default minimal qui inclut un contrôleur, deux switches et deux hôtes comme montre la figure ci-dessous.

exit pour arrêter mininet ou bien **sudo mn -c** pour supprimer les fichiers caches.

```
emrc2020@emrc2020:~$ cd mininet
emrc2020@emrc2020:~/mininet$ sudo mn
[sudo] Mot de passe de emrc2020 :
*** No default OpenFlow controller found for default switch!
*** Falling back to OVS Bridge
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller

*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet>
```

Figure II-2 : Exécution d'une topologie existante par défaut dans mininet

Mininet nous donne la main à réaliser des topologies personnalisées, selon nos besoins et tout ça grâce au nombre d'options offertes par cet émulateur, on site quelques-unes :

- **--topo** en utilisant cette option, mininet fournit des topologies de base : single (topologie centralisée d'un seul switch), linear(linéaire) et tree(arbre), par exemple: **sudo mn --topo single** single tout court donne la même topologie que minimal, ç-à-d créer un hôte relier avec un switch, mais on peut ajouter également comme argument un chiffre, qui indique le nombre d'hôtes à créer **sudo mn --topo single,4 .**
- **--controller** relier la topologie minient au contrôleur , Exemple : **sudo mn --controller=remote ip= « ip-controlleur»** .
- **--switch** offre la possibilité de choisir le modèle de switch parmi ceux supportés par l'émulateur.
- **--mac** simplifie l'adresse mac des nœuds réseau en les faisant correspondre à leur identifiant. Exemple : **sudo mn --mac --contoller=remote,ip= « ip-controlleur»**

- **ipbase** permet de changer la base de l'adresse ip des switches. Exemple : **sudo mn --contoller=remote,ip= "ip-controlleur" ipbase=172.16.0.0/8**
- **--link tc** personnaliser la bande passante et la latence des liaisons. Exemple : **sudo mn --controller=remote,ip="ip-controlleur" --link=tc,bw=10,delay=10ms**

Après avoir créé la topologie en insérant la commande **mn** suivie des options voulues, on peut accéder à la CLI du mininet identifiable par l'expression **mininet>** au début de chaque ligne. Les commandes principales de la CLI sont :

1- **mininet> h1 ping h2** ou bien **mininet> pingall** vérifier la connectivité ç-à-d tester si les paquets sont routés correctement.

2- **mininet> help** pour accéder à toutes les commandes utilisées sous Mininet.

3- **mininet> net** afficher toute les liaisons du réseau.

4- **mininet> nodes** afficher tous les nœuds réseau.

5- **mininet> dump** afficher les informations liées aux nœuds du réseau.

6- **mininet> iperf** permet de mesurer la bande passante entre deux hôtes.

7- **mininet> xterm h1** ouvrir un terminal pour chaque hôte où h1 est le nom de l'hôte.

8- **mininet> dpctl dump-flows** permet d'afficher les différents flux installés sur les nœuds autrement dit analyser les règles insérées dans chaque commutateur.

9- **mininet> h1 route** permet de montrer la destination de h1 où h1 est un hôte.

10- **mininet> link s1 h1 down** permet de supprimer le lien entre s1 et h1 (h1:hôte et s1:switch).

11- **mininet> link s1 h1 up** mettre un lien entre s1 et h1 après on tape **pingall** pour avoir le trafic.

II.2.2.2 Topologies personnalisées via l'API Mininet

Mininet fournit une interface de programmation Python, l'une de ces utilisations consiste en la création de topologies personnalisées en y définissant toutes les caractéristiques voulues des éléments du réseau donc on ne doit pas être limiter à des topologies prédéfinies comme tree, linear ou simple, et même se débarrasser de la CLI.

Par défaut, on a un script python qui contient une topologie de deux switches et deux hôtes relié entre eux, ce fichier s'appelle topo-2sw-2host.py. Ce fichier .py se situe dans le dossier custom et ce dernier se trouve dans le dossier mininet, dans le terminal on tape : **cd mininet/custom** et pour ouvrir le fichier .py : **sudo nano topo-2sw-2host.py**

```
"""Custom topology example

Two directly connected switches plus a host for each switch:

   host --- switch --- switch --- host

Adding the 'topos' dict with a key/value pair to generate our newly defined
topology enables one to pass in '--topo=mytopo' from the command line.
"""

from mininet.topo import Topo

class MyTopo( Topo ):
    "Simple topology example."

    def build( self ):
        "Create custom topo."

        # Add hosts and switches
        leftHost = self.addHost( 'h1' )
        rightHost = self.addHost( 'h2' )
        leftSwitch = self.addSwitch( 's3' )
        rightSwitch = self.addSwitch( 's4' )

        # Add links
        self.addLink( leftHost, leftSwitch )
        self.addLink( leftSwitch, rightSwitch )
        self.addLink( rightSwitch, rightHost )


topos = { 'mytopo': ( lambda: MyTopo() ) }
```

Figure II-3 : Exemple d'un script Python d'une topologie existante dans mininet

Donc si on veut créer une topologie personnalisée, il faut sauvegarder le fichier .py correspondant dans le même endroit ç-à-d dans le répertoire mininet/custom, de plus on appelle la topologie par l'option : **--custom=« nom du script » --topo= « nom de la topologie »** .

```
emrc2020@emrc2020:~$ cd mininet/custom/
emrc2020@emrc2020:~/mininet/custom$ sudo mn --custom topo-2sw-2host.py --topo my
topo
[sudo] Mot de passe de emrc2020 :
*** No default OpenFlow controller found for default switch!
*** Falling back to OVS Bridge
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s3 s4
*** Adding links:
(h1, s3) (s3, s4) (s4, h2)
*** Configuring hosts
h1 h2
*** Starting controller

*** Starting 2 switches
s3 s4 ...
*** Starting CLI:
mininet>
```

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2
h2 -> h1
*** Results: 0% dropped (2/2 received)
```

Figure II-4 : Exécution de la topologie du script existant par défaut dans mininet

Exemple d'exécution de la topologie précédente sur le contrôleur Floodlight : **sudo mn --mac --controller=remote,ip= « ip-controlleur » --custom=topo-2sw-2host.py --topo=mytopo –mac**

Le résultat est affiché par exemple sur la GUI du contrôleur Floodlight qui se présente dans la figure ci-dessous :

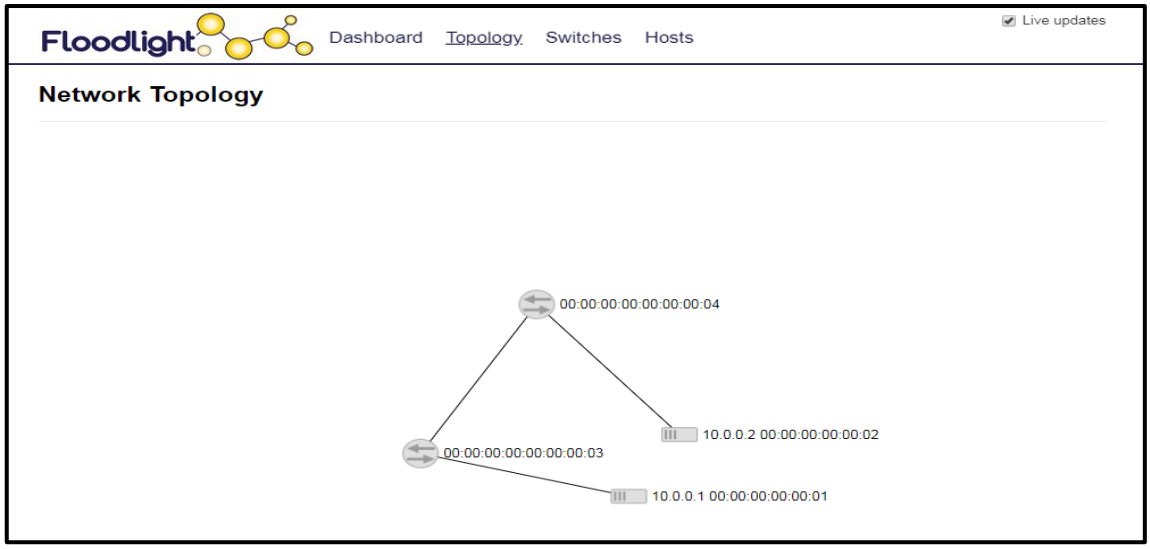


Figure II-5 : Une topologie par défaut affiché dans sur le contrôleur Floodlight

Si nous souhaitons personnaliser une topologie déjà créée, nous appliquons une des commandes citées ci-dessous :

- Pour ajouter un switch : **self.addSwitch('s1')**
- Pour ajouter un hôte : **self.addHost('h1')**
- Pour ajouter un lien : **self.addLink(h1, s1)**

II.3 Le contrôleur SDN Floodlight

Floodlight est un contrôleur OpenFlow open source de hautes performances écrit en Java. Il a été développé sur la base de Beacon, un contrôleur Open Flow expérimental de l'Université de Stanford et il est désormais pris en charge par une large communauté de développeurs. BigSwitch Networks soutient Floodlight en tant qu'entreprise qui propose principalement des solutions pour les datacenters.

Actuellement, Floodlight implémente Open Flow version 1.0 à 1.5 et fonctionne avec tous les commutateurs, routeurs, commutateurs virtuels et points d'accès qui prennent également en charge cette version. Floodlight est publié sous la licence Apache et fournit un certain nombre d'applications réseau, en plus de l'infrastructure de contrôle pour contrôler les composants réseau compatibles avec OpenFlow.[9]

II.3.1 Architecture de Floodlight

Floodlight offre plusieurs fonctionnalités et abstractions pour contrôler un réseau OpenFlow. La prise en charge de ces fonctionnalités est une architecture modulaire, ainsi qu'une série d'applications de base. Dans la figure ci-dessous nous présentons l'architecture Floodlight et les relations entre chacun des modules Java du Contrôleur et l'API RESTful NorthBound. [9]

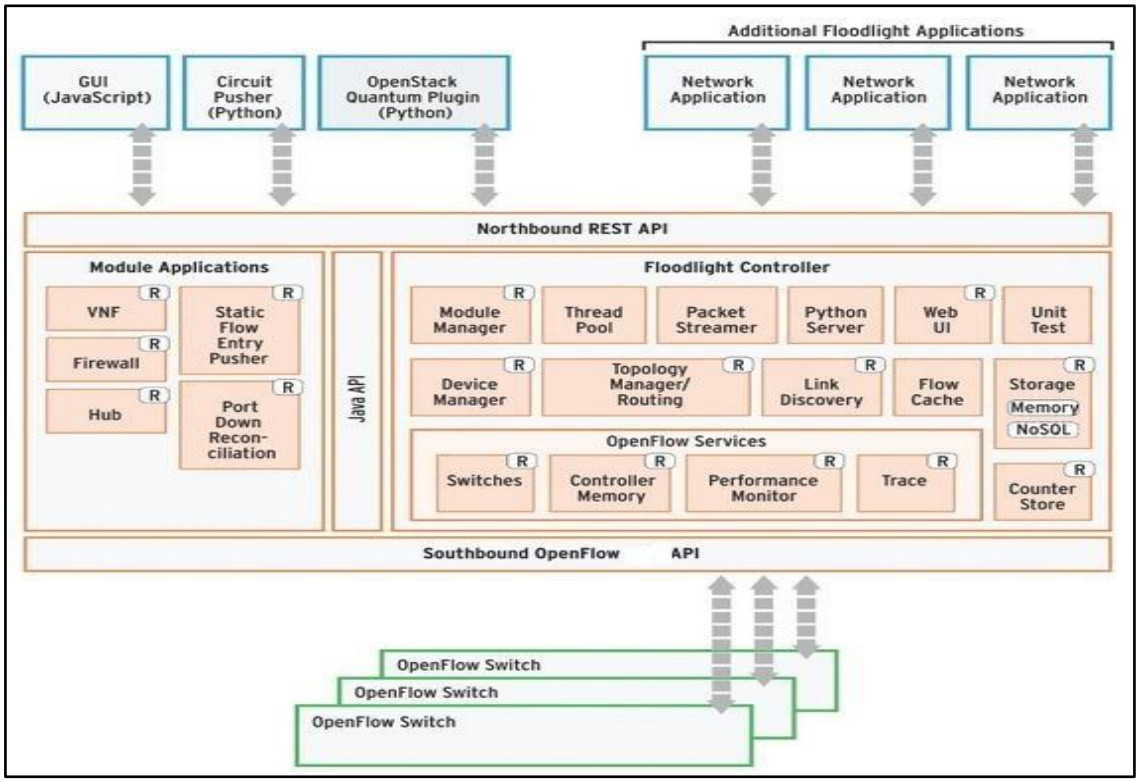


Figure II-6 : Architecture de Floodlight

Pour une utilisation optimale des ressources, Floodlight s'appuie sur le multi-threading et peut gérer plusieurs millions de nouveaux flux par seconde. L'API Java permet le développement de modules personnalisés en Java et une interface rapide avec le contrôleur principal. Les modules sont chargés via un système de modules séparé lors de démarrage du contrôleur Floodlight. Vous pouvez ainsi tirer une partie de toutes les fonctionnalités du contrôleur et de l'API OpenFlow et répondre rapidement aux événements sur le réseau, tels que l'émergence de nouveaux paquets ou de nouveaux flux.

L'interface graphique de Floodlight est l'une des applications standard reposant sur l'API REST (figure II.7). De plus, un Circuit Pusher basé sur Python installe automatiquement des règles OpenFlow persistantes pour la connexion entre deux hôtes.[10]

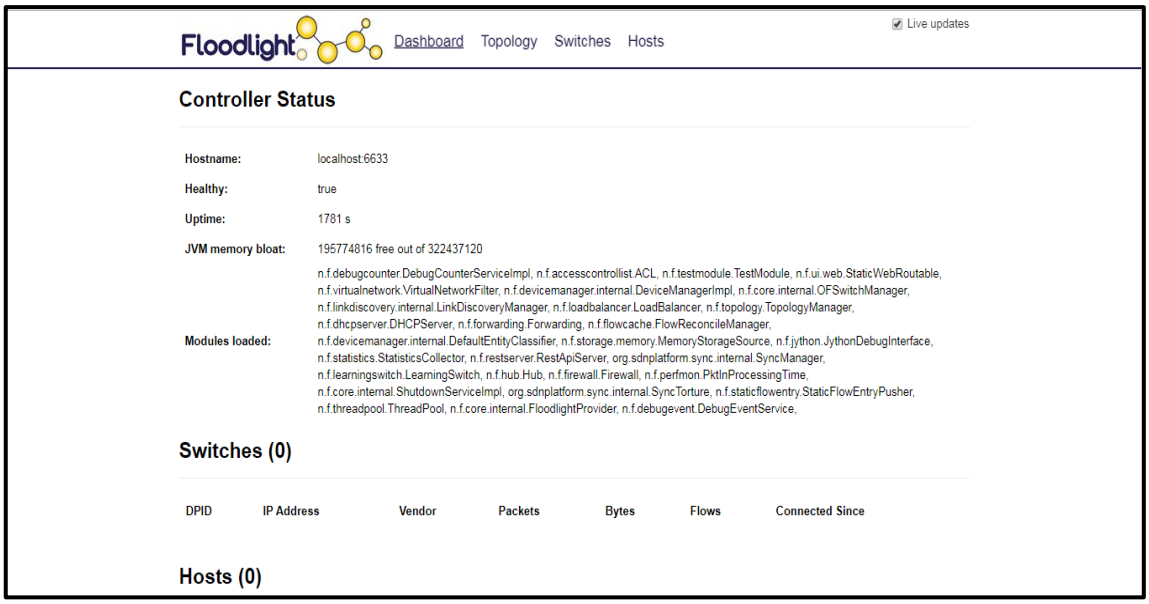


Figure II-7 : Interface GUI de Floodlight

Ici cette capture montre que la topologie est vide, vu qu'aucun équipement n'est connecté au contrôleur. Toute topologie générée sur Mininet apparaitra graphiquement sur le contrôleur. Dans un premier temps seul les switches apparaissent, pour ce qui est des hôtes il suffit de faire circuler un flux de données (**pingall**) entre eux pour qu'ils soient détectés par le contrôleur. En entrant la commande **sudo mn --controller=remote,ip=ip-controlleur --topo=linear,2 --mac** , on aura une topologie linéaire et le résultat est illustré dans la figure ci-dessous.

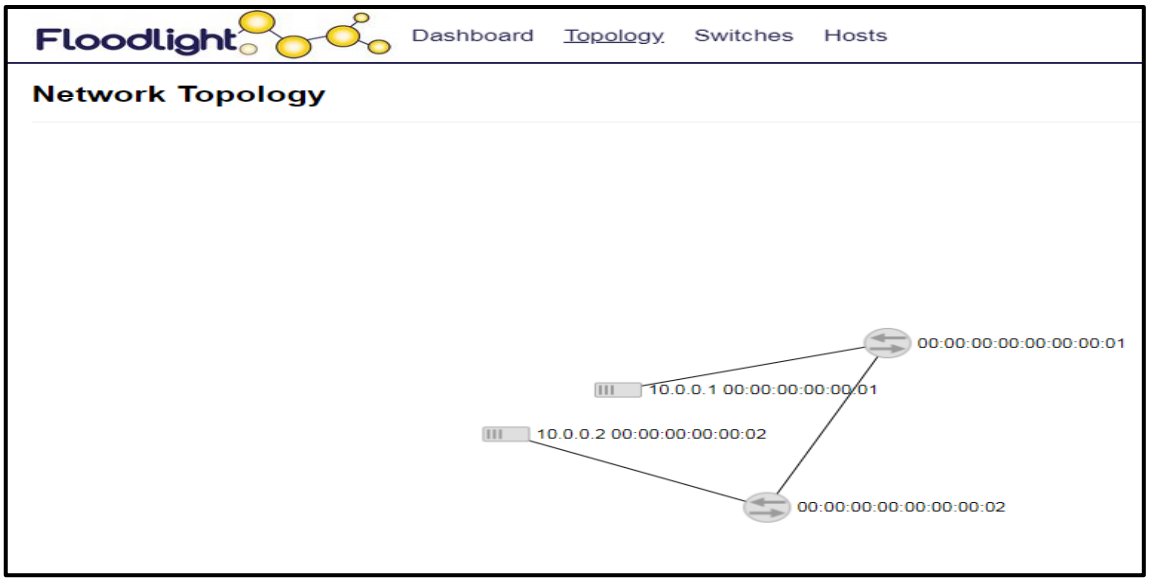


Figure II-8 : Vue globale d'une topologie sur l'GUI Floodlight

II.3.2 Les principales fonctionnalités avancées de Floodlight

Dans les réseaux traditionnels, le Spanning Tree ou les protocoles de routage assument souvent la tâche de gestion de la topologie, par exemple en garantissant l'absence de boucles. En raison des algorithmes distribués de ces protocoles, cependant, un certain nombre de difficultés surviennent, telles qu'une configuration complexe, un nombre limité de sauts ou de longs temps de convergence pour les changements dans l'infrastructure de réseau sous-jacente. L'exploitation de plusieurs chemins entre la source et la destination d'un flux de données implique un effort considérable et l'utilisation d'autres protocoles. En revanche, les contrôleurs SDN ont une vue centrale de tous les composants du réseau et peuvent donc grandement simplifier la gestion de la topologie. Pour que cela se produise, Floodlight implémente un mécanisme sophistiqué de détection automatique de la topologie d'un réseau OpenFlow. À l'aide d'un module de découverte de liens, le contrôleur génère à la fois des paquets LLDP et broadcast (appelés BDDP) et les envoie régulièrement à tous les commutateurs voisins.

En supposant que tous les commutateurs reçoivent des messages LLDP et transfèrent les paquets de broadcast, Floodlight peut identifier les connexions actives en recevant ses propres messages et en calculant la topologie du réseau.[11]

II.3.2.1 Interconnection entre plusieurs nuages OpenFlow

Un autre concept intéressant et important est la combinaison de commutateurs OpenFlow directement interconnectés avec des nuages appelés Island Open Flow, les nuages peuvent se connecter à d'autres nuages via des liens de broadcast. Avec certaines restrictions, Floodlight prend aussi en charge la combinaison d'équipements OpenFlow avec des composants réseaux traditionnels.

En raison des paquets de broadcast, il est important d'éviter les boucles dans les connexions avec des commutateurs non OpenFlow. Pour cette raison, tout Island OpenFlow ne peut avoir qu'une seule connexion à un équipement non OpenFlow. De plus, les Island OpenFlow et non OpenFlow ne doivent pas former eux-mêmes une boucle. La figure ci-dessous montre une topologie typique qui autorise l'interconnexion des équipements OpenFlow et non OpenFlow.[11]

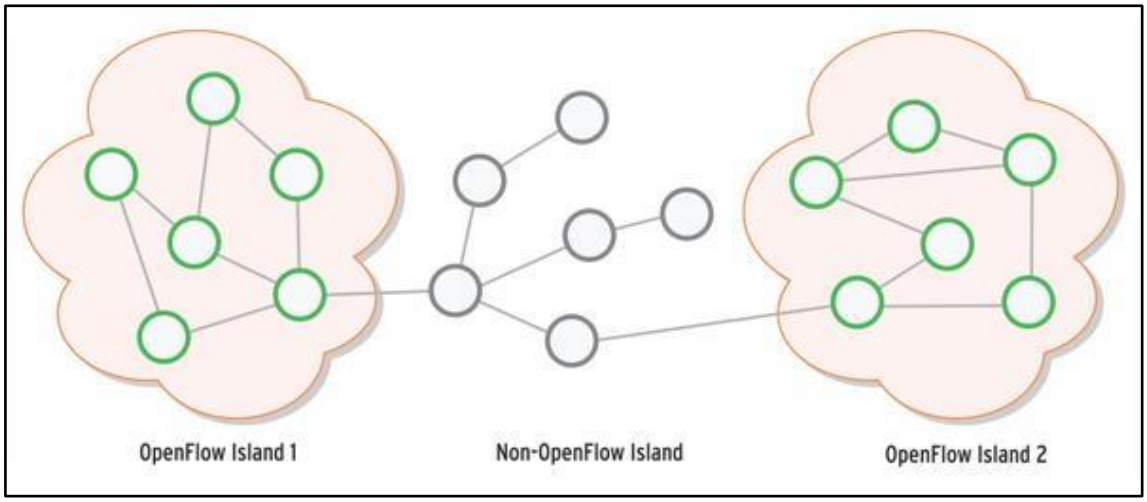


Figure II-9 : Exemple d'interconnexion entre réseaux OpenFlow et non OpenFlow

II.3.2.2 Transfert de paquets : Le forwarding

Floodlight propose actuellement le module avancé « le Learning Switch » implémente un comportement similaire aux commutateurs standard : le Learning Switch détecte et apprend les nouveaux périphériques en fonction de leurs adresses MAC.

Lorsque Floodlight détecte un nouveau flux, le module Learning Switch identifie les switches d'entrée et de sortie, ainsi que tous les autres switches sur le chemin le plus court entre la source et la destination. Une fois qu'un chemin a été trouvé, le module installe les règles OpenFlow appropriées pour gérer les nouveaux flux sur tous les commutateurs participants.

Par rapport au module de forwarding simple, les performances du module Learning Switch sont meilleures car après avoir installé les règles OpenFlow, le transfert de paquets est effectué uniquement dans le chemin de transfert des switches et l'interaction avec le contrôleur n'est pas nécessaire pour chaque message.

Les avantages du SDN, OpenFlow et Floodlight sont clairement visibles. Même les implémentations relativement simples offrent des avantages significatifs, et par rapport aux réseaux traditionnels, les configurations complexes du protocole Spanning Tree deviennent inutiles même pour les grands réseaux maillés.[11]

II.3.2.3 Le partage de charge « Loadbalancing »

L'une des fonctionnalités avancées de Floodlight est le partage de charge réseau sur plusieurs serveurs (Loadbalancing). Floodlight fournit déjà une version rudimentaire de

cette approche. Le module loadbalancing se charge pour la distribution des flux UDP, TCP ou ICMP vers différents serveurs. Aucun composant réseau ne gère la tâche de partage de charges; la tâche est plutôt distribuée sur l'ensemble du réseau. [11]

II.3.2.4 Plus de sécurité : Le Module Parfeu (Firewall)

Floodlight offre une solution de pare-feu apatride Floodlight via un module réactif qui stocke de façon centralisée les configurations ACL à l'échelle du réseau pour tous les switch OpenFlow. Les règles du pare-feu (ALLOW ou DENY) peuvent être configurées de manière relativement pratique pour les correspondances Open Flow arbitraires et triées par priorité grâce à une API REST.

Le pare-feu Floodlight permet ainsi de configurer de manière centralisée les règles ACL à l'échelle du réseau qui sont automatiquement appliquées à tous les switches OpenFlow (ou sélectionnés). L'exemple simple suivant illustre les options de configuration. Les règles autorisent exclusivement les connexions dans le sous-réseau 192.168.1.0/24 au port TCP 80.[10]

```
TCP | 192.168.1.0/24 | 80 | ALLOW | 1 TCP | 192.168.1.0/24 | * | DENY | 2
```

II.4 Conclusion

Le contrôleur SDN Floodlight OpenFlow offre déjà des performances impressionnantes et de nombreuses applications, les développeurs Floodlight travaillent sur des améliorations intéressantes, telles qu'un transfert plus efficace dans les réseaux sans fil, le partage de charge multi-path dans le réseau, et l'ingénierie du trafic à l'approche de l'application.

Dans le chapitre suivant et afin de mettre en pratique les notions théoriques acquises, nous allons implémenter la solution loadbalancing de Floodlight dans un réseau SDN et présenter les performances de cette dernière.

CHAPITRE III:

Implémentation d’une solution loadbalacing dans le SDN sous Floodlight

Sommaire :

CHAPITERE.III Implémentation d'une solution loadbalacing dans le SDN sous Floodlight

III.1 Introduction

Les réseaux subirent de profonds changements en passant d'une infrastructure de communication à un média de gestion d'une multitude de services. Avec la croissance rapide du stockage de l'information dans le monde entier, rechercher et obtenir une donnée est devenu un comportement commun. L'explosion du trafic sur le World Wide Web a entrainé une rapide augmentation du nombre de requêtes sur les sites Web populaires. La plupart des services fournis par ces sites peuvent ainsi recevoir un nombre très important de connexions des clients dans une période très courte.

Ce trafic grandissant peut entrainer des congestions importantes sur les serveurs informatiques et des temps de réponses élevés. Ces perturbations peuvent avoir pour conséquence une instabilité du site et du service pour les utilisateurs. Pour un site Web populaire par exemple, la congestion réseau et la surcharge des serveurs peuvent amener à des problèmes sérieux.

L'amélioration de la performance des serveurs et la garantie de la fiabilité et de la disponibilité sont des notions importantes. C'est pourquoi il arrive un moment ou la charge doit être répartie entre plusieurs serveurs via un mécanisme de partage de charge (loadbalancing) qui régule les requêtes clientes à travers différents serveurs.

III.2 Principe de fonctionnement du loadbalancing

III.2.1 Qu'est-ce que le loadbalancing

Le loadbalancing (Partage de charge) est une technique qui permet de distribuer une charge entres plusieurs serveurs appartenant au même groupe ou cluster (appelé aussi ferme de serveur). Il s'agit d'une technologie efficace pour optimiser la qualité de services.

Dans les réseaux traditionnels, les différentes architectures informatiques permettant le partage de charge des serveurs peuvent se distinguer par le niveau d'abstraction et la gestion de leur adresse IP :

1. Une seule adresse IP est visible par les hôtes clients et leurs requêtes sont prises en charge par un serveur loadbalancer, et à son tour il va les rediriger chaque requête vers l'un des serveurs.

2. Une seule adresse IP virtuelle est visible par les hôtes des clients qui représente tous les serveurs, mais il n'y a pas de serveur loadbalancer.

3. Toutes les adresses IP des serveurs sont visibles par les hôtes des clients, Il n'y a pas de serveur loadbalancer, la redirection des requêtes peut être effectué par un serveur DNS accompagné.[12]

III.2.2 Méthodes utilisées dans le loadbalancing.

Les algorithmes de partage de charge ont pour but de répartir la charge entre les serveurs pour optimiser leur utilisation et diminuer le temps d'exécution des requêtes. Ils jouent une part importante dans le temps de réponse au client ainsi que dans la disponibilité de la ressource, il existe deux approches dans ce contexte à savoir statique et dynamique. [12]

III.2.2.1 Les algorithmes statiques

- *Random*

Dans la répartition aléatoire, chaque requête entrante est envoyée à un serveur sélectionné aléatoirement. Cet algorithme est très rapide et il assure un bon partage de charge sur une période de temps donnée. Cependant, si le nombre de requêtes est faible, la probabilité d'un partage de charge est forte.[12]

- *Round Robin*

L'algorithme de round-robin transfère chaque requête entrante au prochain serveur de la liste. Tous les serveurs sont traités de manière égalitaire sans prendre en compte le volumes des connexions.[12]

- *Weighted Round Robin*

L'algorithme de round-robin à poids est mis en place pour mieux gérer les serveurs ayant des capacités de traitement différents. Chaque serveur se voit assigner un poids qui indique sa capacité de traitement. L'ordonnancement tiendra compte de ce poids pour transférer les requêtes aux serveurs.[12]

III.2.2.2 Algorithmes dynamiques

Ces algorithmes sont dits dynamiques car ils ont besoin de collecter dynamiquement les informations des clients ou des serveurs pour déterminer vers quel serveur sera envoyée la requête du client. Les informations sont accessibles et peuvent être récupérées par le protocole ICMP, par un agent installé sur les serveurs, par une inspection des paquets ou par une analyse du temps de réponse.[12]

III.2.3 Le loadbalancing dans les réseaux SDN.

Parmi les solutions innovantes qui a apporté le paradigme SDN c'est le loadbalancing, ce dernier a remis en cause le principe de fonctionnement de partage de charges dans les réseaux traditionnels ce qui permet une meilleure gestion et partage de charge dans les réseaux. Les applications loadbalancer intégrées dans le contrôleur prend des décisions de contrôle de la trajectoire des données sans avoir à s'appuyer sur des algorithmes définis par les équipements réseau traditionnels. Ce qui permet d'économiser du temps en ayant le contrôle sur tout un réseau d'applications et de serveurs. La tâche de partage de charges entre des serveurs sera à la charge du contrôleur suite à des algorithmes connus dans le domaine de loadbalancing.[12]

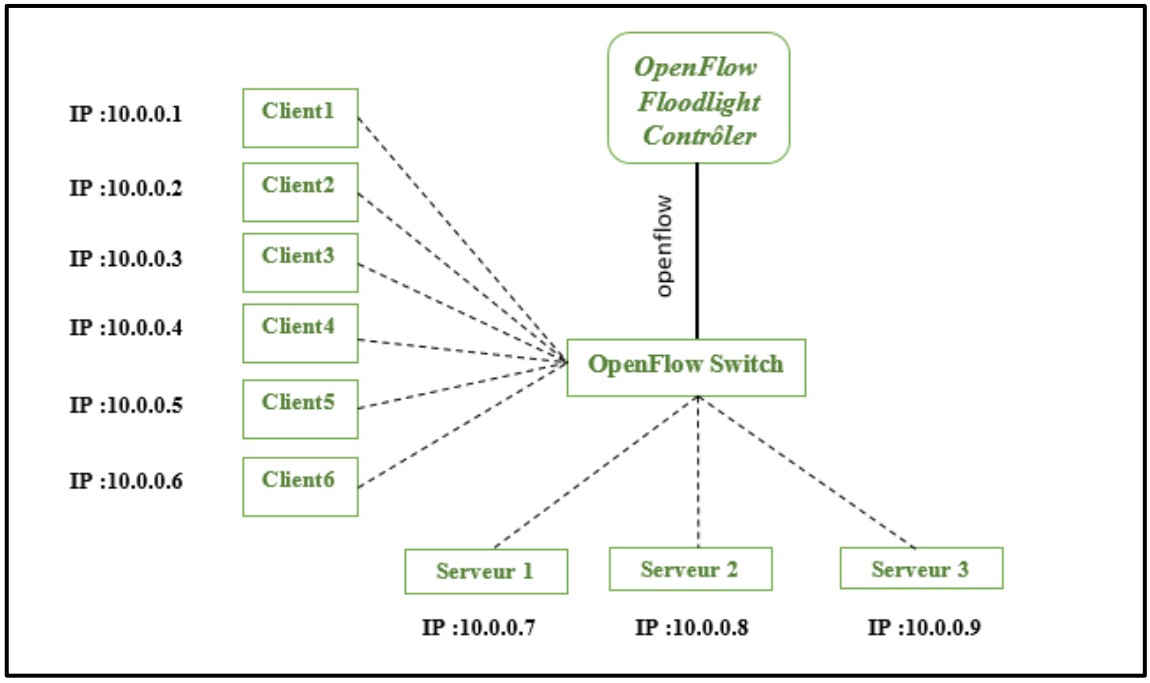


Figure III-1 : Exemple d'une architecture loadbalancing dans un réseau SDN

Ce concept de loadbalancing dans les réseaux SDN offre plusieurs avantages à savoir :

- ✓ Réduction des coûts.
- ✓ Scalabilité.
- ✓ Une plus grande fiabilité.
- ✓ Flexibilité de configuration.
- ✓ Réduction du temps de déploiement.
- ✓ Capacité construction d'un réseau sans nécessité de logiciel/matériel spécifique de l'équipementier.

III.3 La mise en service d'une solution loadbalacing

Afin de mettre en pratique les connaissances acquises dans les chapitres précédents nous allons implanter une solution SDN sous le contrôleur Floodlight avec la fonctionnalité avancé de loadbalancing. Ce qui nous permet d'aborder les aspects clés de cette solution et évaluer ses performances.

La figure ci-dessous montre un partage de charge basé sur SDN qui s'exécute sur le contrôleur.

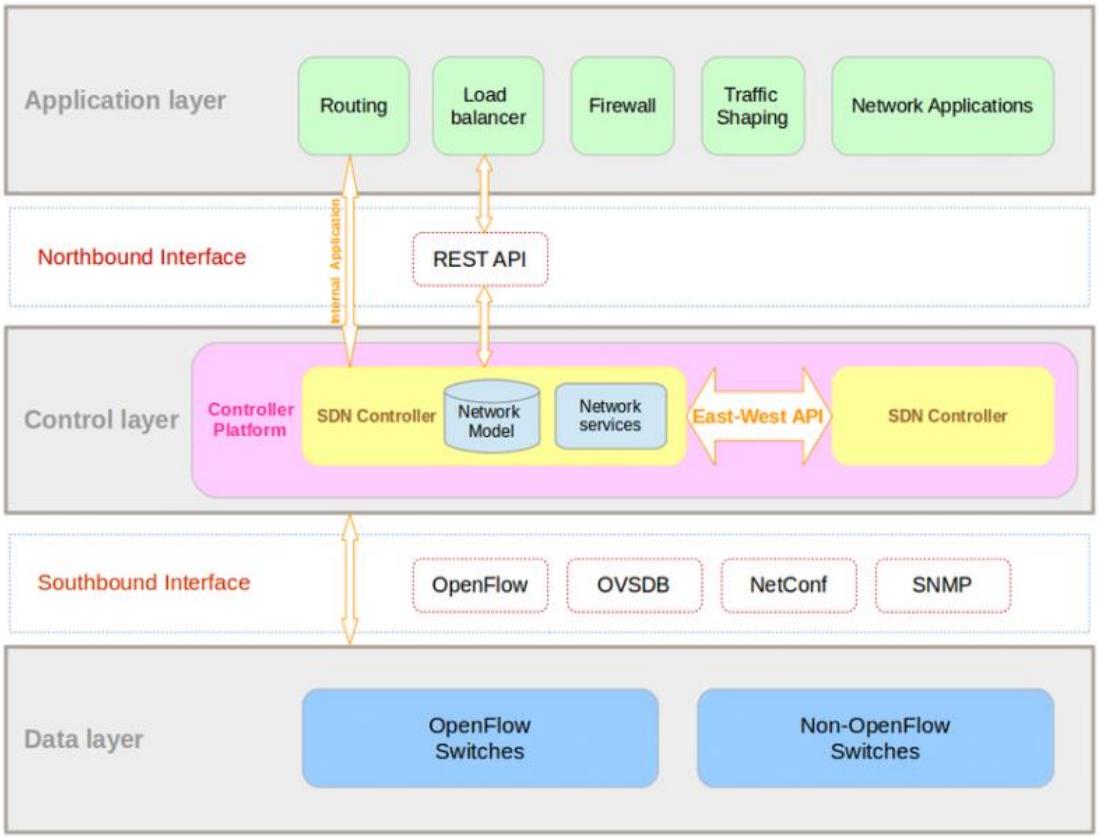


Figure III-2 : Equilibreur de charge basé sur SDN

L'application du loadbalancing communique avec le contrôleur SDN à l'aide de l'API en direction northbound comme l'API REST. Le contrôleur a une vue sur la topologie complète du réseau et les fonctions réseaux telles que le routage et le partage de charge sont effectuées en modifiant les tables de flux dans les switches OpenFlow qui comprend le plan de données.[12]

III.3.1 Scénario de la solution

Le scénario proposé dans notre implémentation d'un réseau SDN, un réseau similaire à une topologie dans un data center composé de trois couches : accès, distribution(agrégation) et Core.

- Accès : S1,S2,S3,S4
- Agrégation : S5,S6,S7,S8
- Core : S9,S10

Les hôtes (clients) sont raccordées à la couche accès et les serveurs à la couche Core comme il est mentionné dans la figure ci-dessous.

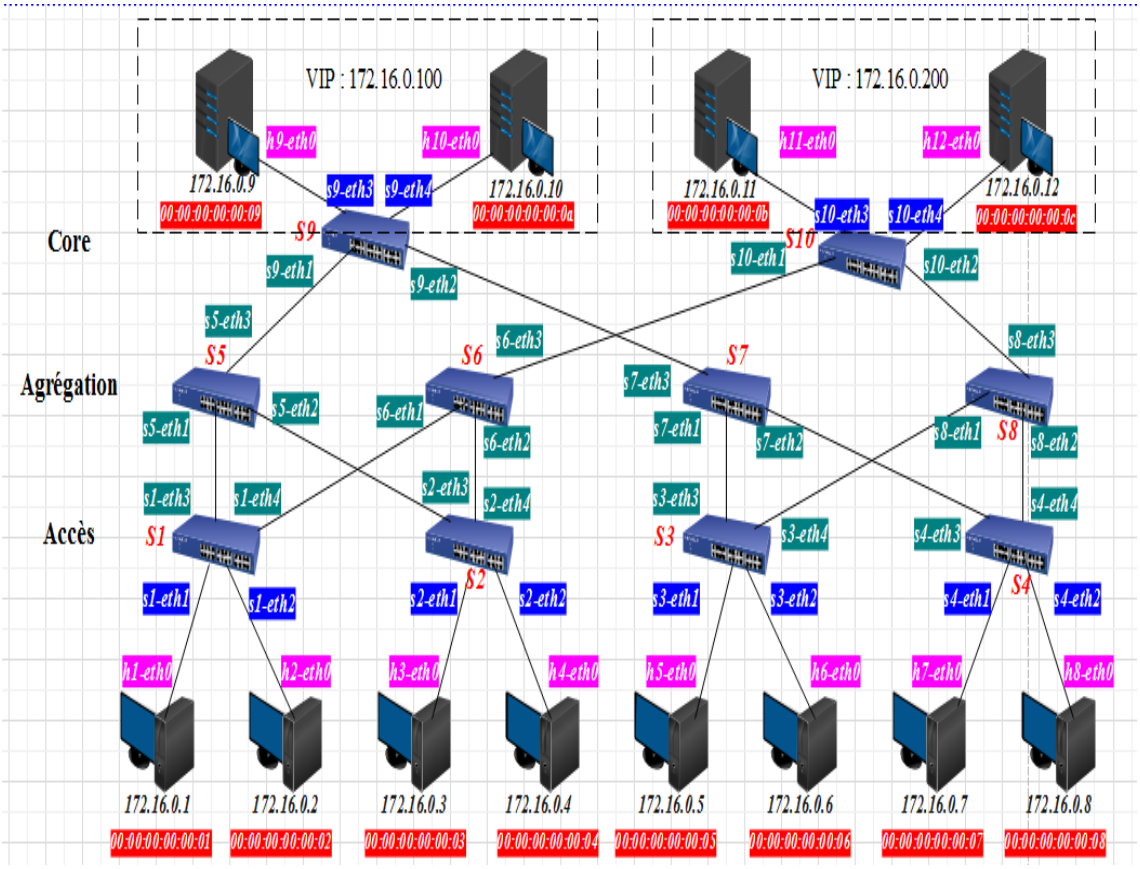


Figure III-3 : topologie de la solution

La configuration de base des différents hôtes est donnée par la table suivante :

	Hôtes	Adresse MAC	Adresse IP
Hôtes Clients	**h1**	00 : 00 : 00 : 00 : 00 : 01	172.16.0.1
	h2	00 : 00 : 00 : 00 : 00 : 02	172.16.0.2
	h3	00 : 00 : 00 : 00 : 00 : 03	172.16.0.3
	h4	00 : 00 : 00 : 00 : 00 : 04	172.16.0.4
	h5	00 : 00 : 00 : 00 : 00 : 05	172.16.0.5
	h6	00 : 00 : 00 : 00 : 00 : 06	172.16.0.6
	h7	00 : 00 : 00 : 00 : 00 : 07	172.16.0.7
	h8	00 : 00 : 00 : 00 : 00 : 08	172.16.0.8
Hôtes Serveurs	**h9**	00 : 00 : 00 : 00 : 00 : 09	172.16.0.9
	h10	00 : 00 : 00 : 00 : 00 : a	172.16.0.10
	h11	00 : 00 : 00 : 00 : 00 : b	172.16.0.11
	h12	00 : 00 : 00 : 00 : 00 : c	172.16.0.12

Tableau III-1 : Informations des hôtes de la topologie

Afin de servir les requêtes montantes à partir des hôtes (h1,h2,h3,h4,h5,h6,h7,h8), le partage de charge sera réparti entre les serveurs h9/h10 et les serveurs h11/h12.

III.3.2 Etapes d'installation des composants de la plateforme

pour accomplir notre projet, nous utiliserons les packages et outils suivants :

III.3.2.1 Le contrôleur Floodlight

Nous avons l'installé sur une machine virtuelle sous l'OS Ubuntu 14.04.6 sur une adresse IP : 192.168.1.8. Pour plus de détail sur la méthode de son installation voir annexe 1.

Lors du chargement de Floodlight, des modules chargés apparurent dans la page d'accueil ainsi que d'autres propriétés. Les résultat montré dans la capture ci-dessous de la page d'accueil de GUI de Floodlight.

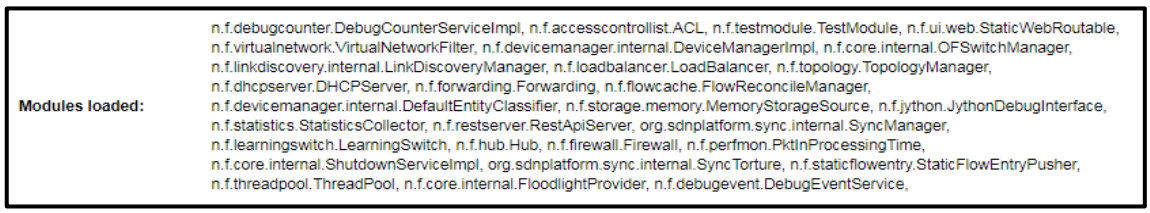


Figure III-4 : Quelques modules chargés affichées dans La GUI du contrôleur Floodlight

Sinon on peut les vérifier dans le fichier ‘floodlightdefault.properities’ qui se localise dans le répertoire floodlight dans le chemin /src/main/resources comme montre les captures ci-dessus

```
floodlight.modules=\
net.floodlightcontroller.jython.JythonDebugInterface,\
net.floodlightcontroller.storage.memory.MemoryStorageSource,\
net.floodlightcontroller.core.internal.FloodlightProvider,\
net.floodlightcontroller.threadpool.ThreadPool,\
net.floodlightcontroller.debugcounter.DebugCounterServiceImpl,\
net.floodlightcontroller.perfmon.PktInProcessingTime,\
net.floodlightcontroller.debugevent.DebugEventService,\
net.floodlightcontroller.staticflowentry.StaticFlowEntryPusher,\
net.floodlightcontroller.restserver.RestApiServer,\
net.floodlightcontroller.topology.TopologyManager,\
net.floodlightcontroller.forwarding.Forwarding,\
net.floodlightcontroller.linkdiscovery.internal.LinkDiscoveryManager,\
net.floodlightcontroller.ui.web.StaticWebRoutable,\
net.floodlightcontroller.loadbalancer.LoadBalancer,\
net.floodlightcontroller.firewall.Firewall,\
net.floodlightcontroller.devicemanager.internal.DeviceManagerImpl,\
net.floodlightcontroller.accesscontrollist.ACL,\
net.floodlightcontroller.statistics.StatisticsCollector
org.sdnplatform.sync.internal.SyncManager.authScheme=CHALLENGE_RESPONSE
org.sdnplatform.sync.internal.SyncManager.keyStorePath=/etc/floodlight/auth_cred$
org.sdnplatform.sync.internal.SyncManager.dbPath=/var/lib/floodlight/
org.sdnplatform.sync.internal.SyncManager.port=6642
net.floodlightcontroller.forwarding.Forwarding.match=vlan, mac, ip, transport
net.floodlightcontroller.forwarding.Forwarding.flood-arp=NO
net.floodlightcontroller.core.internal.FloodlightProvider.openFlowPort=6653
```

```
net.floodlightcontroller.loadbalancer.LoadBalancer,\
net.floodlightcontroller.firewall.Firewall,\
net.floodlightcontroller.devicemanager.internal.DeviceManagerImpl,\
net.floodlightcontroller.accesscontrollist.ACL,\
net.floodlightcontroller.statistics.StatisticsCollector
org.sdnplatform.sync.internal.SyncManager.authScheme=CHALLENGE_RESPONSE
org.sdnplatform.sync.internal.SyncManager.keyStorePath=/etc/floodlight/auth_cred$
org.sdnplatform.sync.internal.SyncManager.dbPath=/var/lib/floodlight/
org.sdnplatform.sync.internal.SyncManager.port=6642
net.floodlightcontroller.forwarding.Forwarding.match=vlan, mac, ip, transport
net.floodlightcontroller.forwarding.Forwarding.flood-arp=NO
net.floodlightcontroller.core.internal.FloodlightProvider.openFlowPort=6653
net.floodlightcontroller.core.internal.FloodlightProvider.role=ACTIVE
net.floodlightcontroller.linkdiscovery.internal.LinkDiscoveryManager.latency-his$
net.floodlightcontroller.linkdiscovery.internal.LinkDiscoveryManager.latency-upd$
net.floodlightcontroller.core.internal.OFSwitchManager.defaultMaxTablesToReceive$
net.floodlightcontroller.core.internal.OFSwitchManager.maxTablesToReceiveTableMi$
net.floodlightcontroller.core.internal.OFSwitchManager.clearTablesOnInitialHands$
net.floodlightcontroller.core.internal.OFSwitchManager.clearTablesOnEachTransiti$
net.floodlightcontroller.core.internal.OFSwitchManager.keyStorePath=/path/to/you$
net.floodlightcontroller.core.internal.OFSwitchManager.keyStorePassword=your-key$
net.floodlightcontroller.core.internal.OFSwitchManager.useSsl=NO
net.floodlightcontroller.core.internal.OFSwitchManager.supportedOpenFlowVersions$
net.floodlightcontroller.restserver.RestApiServer.keyStorePath=/path/to/your/key$
net.floodlightcontroller.restserver.RestApiServer.keyStorePassword=your-keystore$
net.floodlightcontroller.restserver.RestApiServer.httpsNeedClientAuthentication=$
```

```
net.floodlightcontroller.linkdiscovery.internal.LinkDiscoveryManager.latency-upd$
net.floodlightcontroller.core.internal.OFSwitchManager.defaultMaxTablesToReceive$
net.floodlightcontroller.core.internal.OFSwitchManager.maxTablesToReceiveTableMi$
net.floodlightcontroller.core.internal.OFSwitchManager.clearTablesOnInitialHands$
net.floodlightcontroller.core.internal.OFSwitchManager.clearTablesOnEachTransiti$
net.floodlightcontroller.core.internal.OFSwitchManager.keyStorePath=/path/to/you$
net.floodlightcontroller.core.internal.OFSwitchManager.keyStorePassword=your-key$
net.floodlightcontroller.core.internal.OFSwitchManager.useSsl=NO
net.floodlightcontroller.core.internal.OFSwitchManager.supportedOpenFlowVersions$
net.floodlightcontroller.restserver.RestApiServer.keyStorePath=/path/to/your/key$
net.floodlightcontroller.restserver.RestApiServer.keyStorePassword=your-keystore$
net.floodlightcontroller.restserver.RestApiServer.httpsNeedClientAuthentication=$
net.floodlightcontroller.restserver.RestApiServer.useHttps=NO
net.floodlightcontroller.restserver.RestApiServer.useHttp=YES
net.floodlightcontroller.restserver.RestApiServer.httpsPort=8081
net.floodlightcontroller.restserver.RestApiServer.httpPort=8080
net.floodlightcontroller.statistics.StatisticsCollector.enable=FALSE
net.floodlightcontroller.statistics.StatisticsCollector.collectionIntervalPortSt$
```

Figure III-5 :Capture du fichier ‘floodlightdefault.properities’

III.3.2.2 L'émulateur de réseau Mininet

Comme déjà cité précédemment, cet outil est utilisé pour émuler un réseau OpenFlow. Son installation est simple, il faut juste partir au site officiel du mininet (http://mininet.org/download/) et choisir la façon dont vous voulez l'installer. Nous avons choisi l'option 2: « Installation native à partir de la source » en suivant les étapes mentionnées dans le site. Nous avons l'installé sur une autre machine sous l'OS Ubuntu 20.04 avec une adresse IP : 192.168.1.5

III.3.2.3 L'utilitaire de capture de trafic Wireshark

Nous avons utilisé WireShark pour analyser les paquets OpenFlow et autres protocoles. Cela peut être configuré sous Linux et Windows à la fois. Wireshark est disponible dans le référentiel de packages officiel d'Ubuntu 20.04 LTS et versions ultérieures.

C'est vraiment facile à installer, nous avons l'installer sous la machine Mininet, en tapant tout simplement dans le terminal : **sudo apt-get install wireshark** et après que l'installation est terminée, on le lance, son interface est figurée ci-dessous.

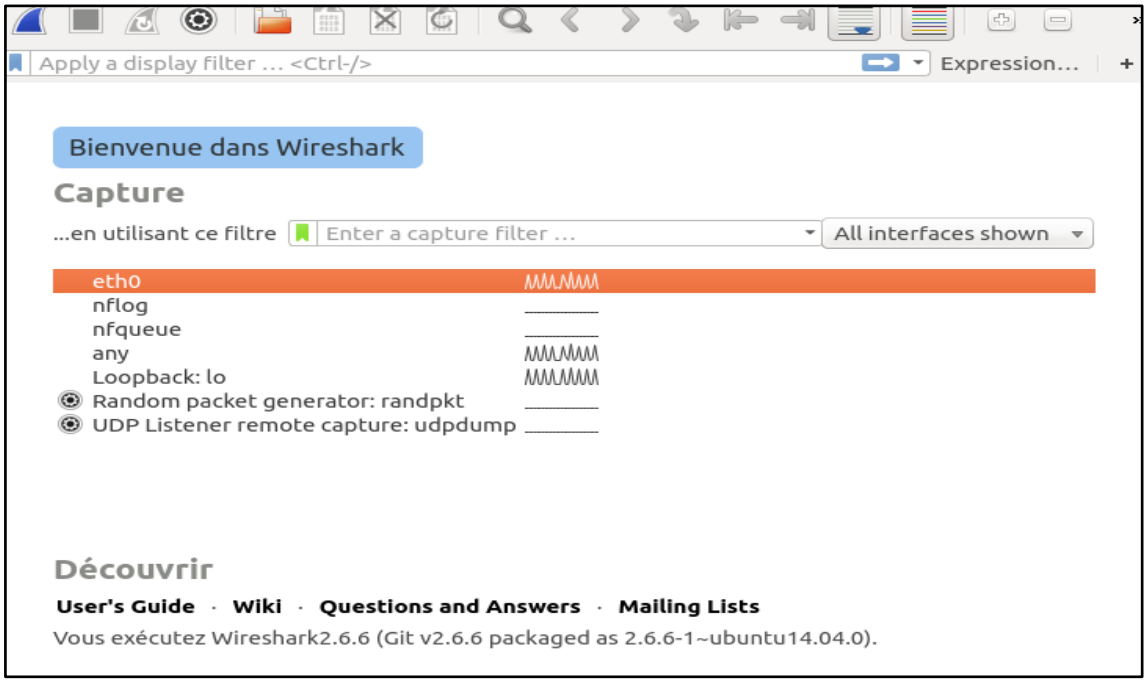


Figure III-6 : Interface de wireshark

III.3.3 Intégration et tests

III.3.3.1 Création et intégration de la topologie dans le contrôleur floodlight

Notre topologie a été implémentée sous l'émulateur mininet et sauvegardé dans le répertorie **mininet/custom** en utilisant un script en langage Python qui se suit :

```
class MyTopo( Topo ):

    def __init__( self ):
        "Creation de la topologie version 1 pour le memoire."

        # Initialization de la topologie
        Topo.__init__( self )

        #L'ajout des hotes clients
        h1 = self.addHost('h1', cls=Host, ip='172.16.0.1', defaultRoute=None)
        h2 = self.addHost('h2', cls=Host, ip='172.16.0.2', defaultRoute=None)
        h3 = self.addHost('h3', cls=Host, ip='172.16.0.3', defaultRoute=None)
        h4 = self.addHost('h4', cls=Host, ip='172.16.0.4', defaultRoute=None)
        h5 = self.addHost('h5', cls=Host, ip='172.16.0.5', defaultRoute=None)
        h6 = self.addHost('h6', cls=Host, ip='172.16.0.6', defaultRoute=None)
        h7 = self.addHost('h7', cls=Host, ip='172.16.0.7', defaultRoute=None)
        h8 = self.addHost('h8', cls=Host, ip='172.16.0.8', defaultRoute=None)

        #L'ajout des hotes Serveurs
        h9 = self.addHost('h9', cls=Host, ip='172.16.0.9', defaultRoute=None)
        h10 = self.addHost('h10', cls=Host, ip='172.16.0.10', defaultRoute=None)
        h11 = self.addHost('h11', cls=Host, ip='172.16.0.11', defaultRoute=None)
        h12 = self.addHost('h12', cls=Host, ip='172.16.0.12', defaultRoute=None)

        #L'ajout des switches
        s1 = self.addSwitch('s1', protocols='OpenFlow13' , cls=OVSKernelSwitch)
        s2 = self.addSwitch('s2', protocols='OpenFlow13' , cls=OVSKernelSwitch)
        s3 = self.addSwitch('s3', protocols='OpenFlow13' , cls=OVSKernelSwitch)
        s4 = self.addSwitch('s4', protocols='OpenFlow13' , cls=OVSKernelSwitch)
        s5 = self.addSwitch('s5', protocols='OpenFlow13' , cls=OVSKernelSwitch)
        s6 = self.addSwitch('s6', protocols='OpenFlow13' , cls=OVSKernelSwitch)
        s7 = self.addSwitch('s7', protocols='OpenFlow13' , cls=OVSKernelSwitch)
        s8 = self.addSwitch('s8', protocols='OpenFlow13' , cls=OVSKernelSwitch)
        s9 = self.addSwitch('s9', protocols='OpenFlow13' , cls=OVSKernelSwitch)
        s10 = self.addSwitch('s10', protocols='OpenFlow13' , cls=OVSKernelSwitch)
```

```
        #L'ajout des liens
        self.addLink(h1, s1)
        self.addLink(h2, s1)
        self.addLink(h3, s2)
        self.addLink(h4, s2)
        self.addLink(h5, s3)
        self.addLink(h6, s3)
        self.addLink(h7, s4)
        self.addLink(h8, s4)
        self.addLink(s1, s5)
        self.addLink(s1, s6)
        self.addLink(s2, s5)
        self.addLink(s2, s6)
        self.addLink(s3, s7)
        self.addLink(s3, s8)
        self.addLink(s4, s7)
        self.addLink(s4, s8)
        self.addLink(s5, s9)
        self.addLink(s7, s9)
        self.addLink(s6, s10)
        self.addLink(s8, s10)
        self.addLink(s9, h9)
        self.addLink(s9, h10)
        self.addLink(s10, h11)
        self.addLink(s10, h12)

topos = { 'mytopo1': (lambda: MyTopo() ) }
```

Figure III-7 : Code source de la topologie en python

Cela générera les différents éléments de la couche physique qui sera connectée au contrôleur Floodlight à distance par la commande suivante : **sudo mn --controller=remote,ip=ip-conrolleur –custom=nom-script –topo=nom-topologie --mac**

Le contrôleur détecte les switches OpenFlow et les hôtes suite au lancement du script comme montre la figure ci-dessous :

```
emrc2020@emrc2020:~$ cd mininet/custom/
emrc2020@emrc2020:~/mininet/custom$ sudo mn --custom topoMemoire1.py --topo mytopo --controller=remote,ip=192.168.1.3 --mac
[sudo] Mot de passe de emrc2020 :
*** Creating network
*** Adding controller
Connecting to remote controller at 192.168.1.3:6653
*** Adding hosts:
h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12
*** Adding switches:
s1 s2 s3 s4 s5 s6 s7 s8 s9 s10
*** Adding links:
(h1, s1) (h2, s1) (h3, s2) (h4, s2) (h5, s3) (h6, s3) (h7, s4) (h8, s4) (s1, s5) (s1, s6) (s2, s5) (s2, s6) (s3, s7) (s3, s8) (s4, s7) (s4, s8) (s5, s
9) (s6, s10) (s7, s9) (s8, s10) (s9, h9) (s9, h10) (s10, h11) (s10, h12)
*** Configuring hosts
h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12
*** Starting controller
c0
*** Starting 10 switches
s1 s2 s3 s4 s5 s6 s7 s8 s9 s10 ...
*** Starting CLI:
```

Figure III-8 : Détection des nœuds réseaux par le contrôleur

Après un ping global pour vérifier la connectivité des hôtes, le résultat est montré au-dessous :

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12
h2 -> h1 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12
h3 -> h1 h2 h4 h5 h6 h7 h8 h9 h10 h11 h12
h4 -> h1 h2 h3 h5 h6 h7 h8 h9 h10 h11 h12
h5 -> h1 h2 h3 h4 h6 h7 h8 h9 h10 h11 h12
h6 -> h1 h2 h3 h4 h5 h7 h8 h9 h10 h11 h12
h7 -> h1 h2 h3 h4 h5 h6 h8 h9 h10 h11 h12
h8 -> h1 h2 h3 h4 h5 h6 h7 h9 h10 h11 h12
h9 -> h1 h2 h3 h4 h5 h6 h7 h8 h10 h11 h12
h10 -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h11 h12
h11 -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h12
h12 -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11
*** Results: 0% dropped (132/132 received)
mininet>
```

Figure III-9 : Un ping global dans la topologie

Une fois la topologie créée, tout le détail sera affiché dans la page d'accueil de Floodlight comme montre la figure III.10

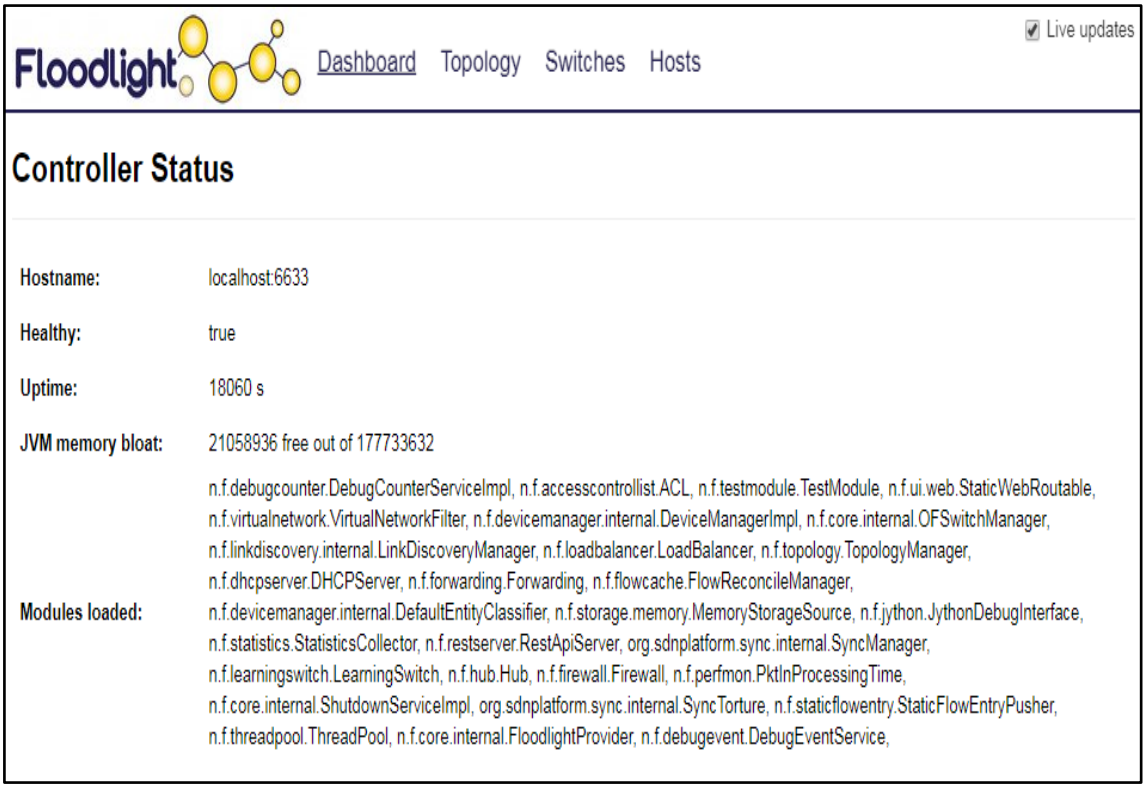


Figure III-10 : Interface web de floodlight

Si on veut afficher la topologie, on clique sur « Topology » comme montre la figure III.11

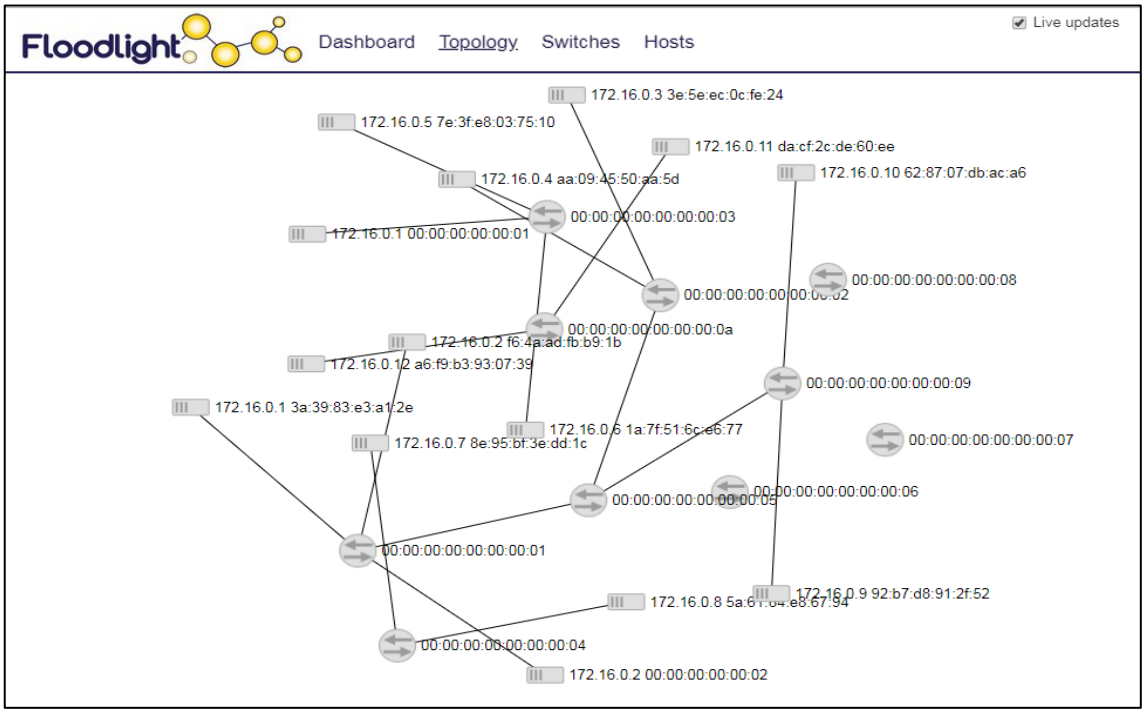


Figure III-11 : Topologie du scénario affichée sur Floodlight

Si on clique sur le lien « Switches », on aura les DataPath_ID et les adresses IP des switches qui compose notre topologie ainsi que le nombre de ports et des informations sur ces derniers tel que le flux circulant dedans (figure III.12)

Switches (10)

DPID	IP Address	Vendor	Packets	Bytes	Flows	Connected Since
00:00:00:00:00:00:00:05	/192.168.1.6:40688	Nicira, Inc.	0	0	0	17/09/2020 à 16:59:19
00:00:00:00:00:00:00:06	/192.168.1.6:40704	Nicira, Inc.	3	294	7	17/09/2020 à 16:59:19
00:00:00:00:00:00:00:02	/192.168.1.6:40690	Nicira, Inc.	1	98	3	17/09/2020 à 16:59:19
00:00:00:00:00:00:00:01	/192.168.1.6:40696	Nicira, Inc.	2	196	4	17/09/2020 à 16:59:19
00:00:00:00:00:00:00:09	/192.168.1.6:40694	Nicira, Inc.	0	0	0	17/09/2020 à 16:59:19
00:00:00:00:00:00:00:04	/192.168.1.6:40700	Nicira, Inc.	0	0	0	17/09/2020 à 16:59:19
00:00:00:00:00:00:00:07	/192.168.1.6:40702	Nicira, Inc.	0	0	0	17/09/2020 à 16:59:19
00:00:00:00:00:00:00:08	/192.168.1.6:40692	Nicira, Inc.	0	0	0	17/09/2020 à 16:59:19
00:00:00:00:00:00:00:03	/192.168.1.6:40698	Nicira, Inc.	0	0	0	17/09/2020 à 16:59:19
00:00:00:00:00:00:00:0a	/192.168.1.6:40706	Nicira, Inc.	3	294	7	17/09/2020 à 16:59:19

Figure III-12 : Liste des switches de la topologie affiché sur l'interface web du Floodlight

Et même chose en cliquant sur « hosts », le résultat est figuré ci-dessous

Hosts (12)

MAC Address	IP Address	Switch Port	Last Seen
00:00:00:00:00:08	172.16.0.8	00:00:00:00:00:00:00:04-2	17/09/2020 à 17:01:40
00:00:00:00:00:06	172.16.0.6	00:00:00:00:00:00:00:03-2	17/09/2020 à 17:01:40
00:00:00:00:00:0c	172.16.0.12	00:00:00:00:00:00:00:0a-4	17/09/2020 à 17:01:55
00:00:00:00:00:0b	172.16.0.11	00:00:00:00:00:00:00:0a-3	17/09/2020 à 17:02:00
00:00:00:00:00:07	172.16.0.7	00:00:00:00:00:00:00:04-1	17/09/2020 à 17:01:45
00:00:00:00:00:05	172.16.0.5	00:00:00:00:00:00:00:03-1	17/09/2020 à 17:01:45
00:00:00:00:00:03	172.16.0.3	00:00:00:00:00:00:00:02-1	17/09/2020 à 17:02:00
00:00:00:00:00:09	172.16.0.9	00:00:00:00:00:00:00:09-3	17/09/2020 à 17:01:40
00:00:00:00:00:01	172.16.0.1	00:00:00:00:00:00:00:01-1	17/09/2020 à 17:02:00
00:00:00:00:00:04	172.16.0.4	00:00:00:00:00:00:00:02-2	17/09/2020 à 17:01:40
00:00:00:00:00:02	172.16.0.2	00:00:00:00:00:00:00:01-2	17/09/2020 à 17:02:00
00:00:00:00:00:0a	172.16.0.10	00:00:00:00:00:00:00:09-4	17/09/2020 à 17:01:55

Figure III-13 : Liste des hôtes de la topologie affiché sur l'interface web du Floodlight

Pour vérifier la connectivité du protocole du SDN qui est OpenFlow, On fait des tests, on tape par exemple dans mininet :

h1 ping h4

```
mininet> h1 ping h4
PING 172.16.0.4 (172.16.0.4) 56(84) bytes of data.
64 bytes from 172.16.0.4: icmp_seq=1 ttl=64 time=7.13 ms
64 bytes from 172.16.0.4: icmp_seq=2 ttl=64 time=0.455 ms
64 bytes from 172.16.0.4: icmp_seq=3 ttl=64 time=0.110 ms
64 bytes from 172.16.0.4: icmp_seq=4 ttl=64 time=0.113 ms
64 bytes from 172.16.0.4: icmp_seq=5 ttl=64 time=0.103 ms
64 bytes from 172.16.0.4: icmp_seq=6 ttl=64 time=0.048 ms
```

Figure III-14 :Un ping entre h1 et h4

Pour qu'une hôte communique avec une autre, Floodlight choisi le meilleur chemin (best path) en constituant une table de flux dans tous les switches concernées par ce chemin et le transfert des paquets se fait par le forwarding suivant cette table.

Exemple : entre h1 et h4 passent via S1-S5-S2 comme montre les captures ci-dessus

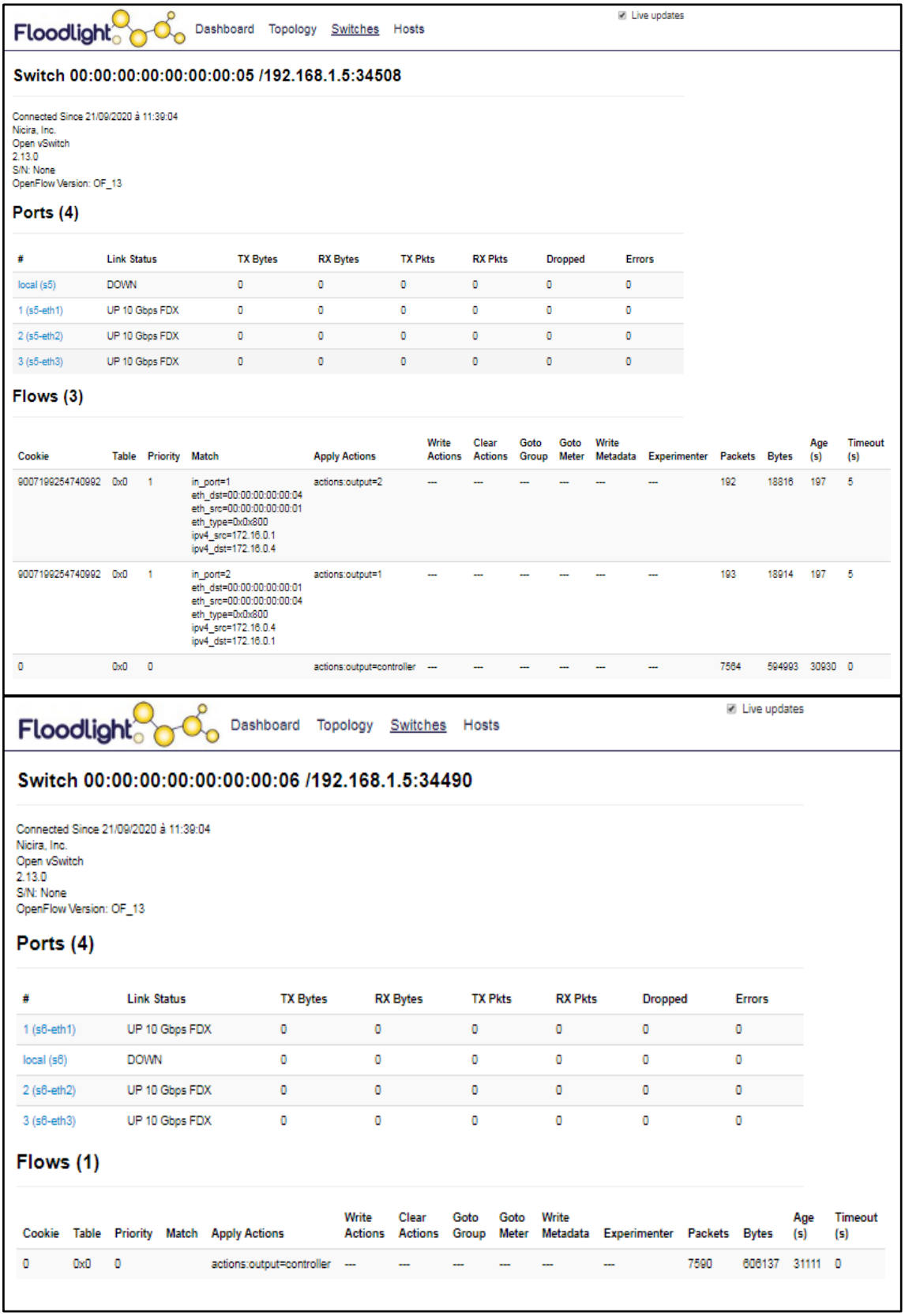


Figure III-15 : Capture des switches 5 et 6 après le ping entre h1 et h4

III.3.3.2 Implémentation du script de loadbalancing

Floodlight dispose d'un module de partage de charge intégré. Pour mettre en œuvre cette fonctionnalité, des requêtes au format JSON via une API REST seront envoyé au contrôleur. Les requêtes JSON contiennent l'adresse IP virtuelle (VIP), le type de service, le protocole et le pool de serveurs qui correspond au VIP. Ce qui suit décrit la procédure de gestion des paquets du module de partage de charge de Floodlight pour les requêtes ARP et IPv4.

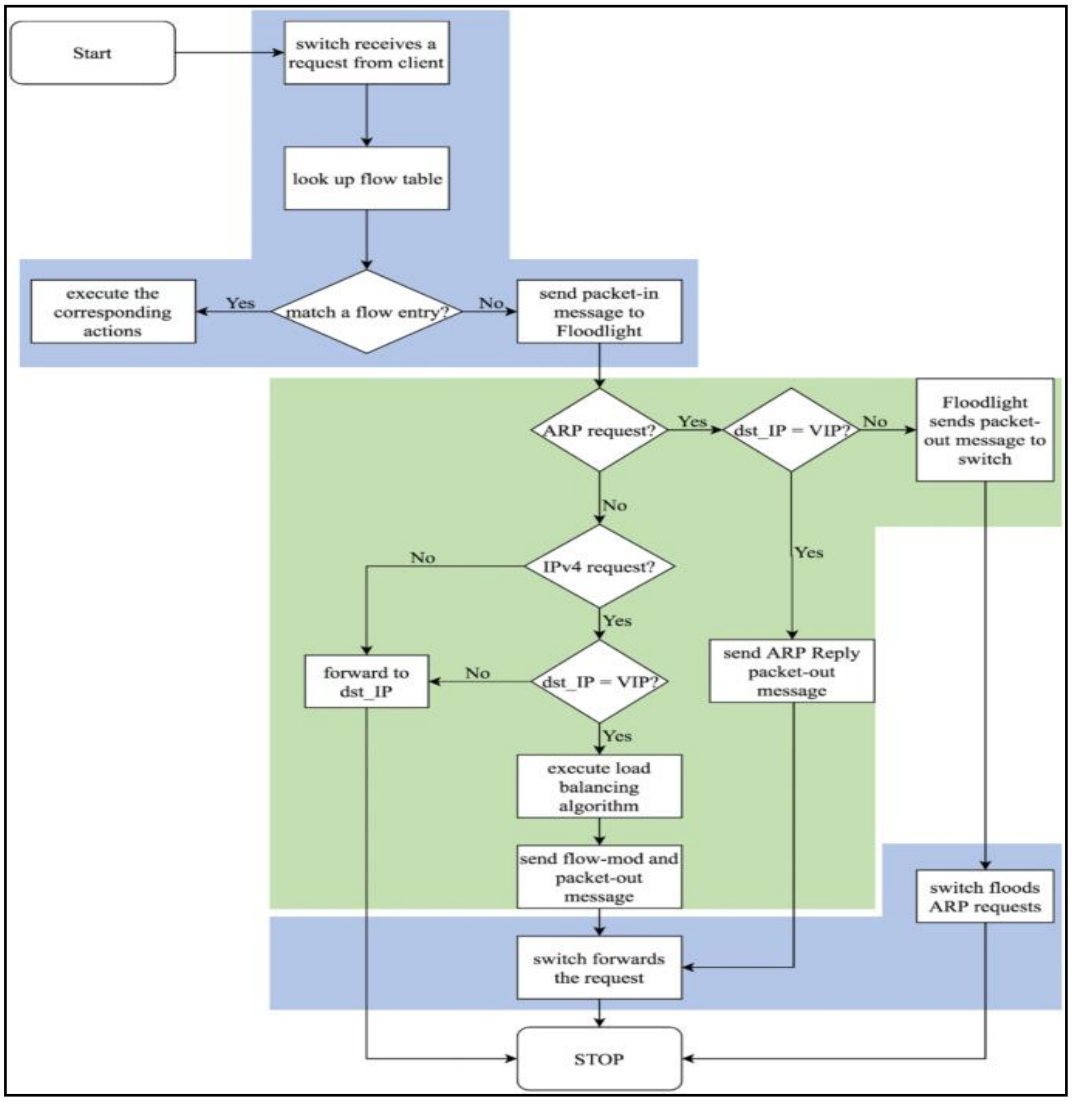


Figure III-16 : Gestion des paquets du module loadbalancing de Floodlight pour les requêtes ARP et IPv4

Comme le montre la figure ci-dessus (III.16), dans un premier temps, l'hôte (client) envoie une requête ARP au vSwitch OpenFlow (OVS). L'OVS va le rediriger vers le contrôleur Floodlight. Lorsque ce paquet ARP est reçu, Floodlight génère un message

REPLAY contenant un paquet de réponse ARP avec une fausse adresse MAC. Lorsque le message de réponse ARP est reçu, le client commence alors à envoyer la demande IPv4 (c'est-à-dire TCP, UDP ou ICMP).

Lorsqu'une nouvelle demande IPv4 arrive à OVS. Il envoie ensuite ce message à Floodlight car aucune entrée de flux ne correspond à l'en-tête du paquet. Lorsque Floodlight reçoit ce message d'entrée de paquet, son module de partage de charge examine le message. Si son adresse IP de destination est la même que le VIP, l'algorithme d'équilibrage de charge est exécuté pour sélectionner le serveur approprié. Floodlight envoie ensuite des messages flow-mod et un message de sortie de paquet aux switchs OVS. Il installe des entrées de flux bidirectionnelles dans la table de flux des switchs, indiquant comment transférer des paquets pendant que d'autres paquets avec la même connexion arrivent.

En ce qui concerne les stratégies de partage de charge, Floodlight utilise la politique de tourniquet (Round robbin) pour partager les charges vers les serveurs, qui est basée sur les connexions et non sur le volume de trafic. Lorsqu'un message entrant contenant une demande client arrive, Floodlight sélectionne à son tour les serveurs pour traiter les demandes client.

Dans cette partie nous allons mettre en œuvre cette solution de loadbalaning sur un réseau SDN.[12] Pour l'intégration du loadbalancing dans notre travail, on a opté à partager la charge entre les serveurs h9 et h10 et aussi entre les serveurs h11 et h12. Le détail est mentionné dans le script « loadbalancer.sh » suivant :

```
#!/bin/sh

curl -X POST -d '{"id":"1","name":"vip1","protocol":"icmp","address":"172.16.0.100","port":"100"}' http://192.168.1.8:8080/quantum/v1.0/vips/
curl -X POST -d '{"id":"1","name":"pool1","protocol":"icmp","vip_id":"1"}' http://192.168.1.8:8080/quantum/v1.0/pools/
curl -X POST -d '{"id":"1","address":"172.16.0.9","port":"100","pool_id":"1"}' http://192.168.1.8:8080/quantum/v1.0/members/
curl -X POST -d '{"id":"2","address":"172.16.0.10","port":"100","pool_id":"1"}' http://192.168.1.8:8080/quantum/v1.0/members/

curl -X POST -d '{"id":"2","name":"vip2","protocol":"tcp","address":"172.16.0.200","port":"200"}' http://192.168.1.8:8080/quantum/v1.0/vips/
curl -X POST -d '{"id":"2","name":"pool2","protocol":"tcp","vip_id":"2"}' http://192.168.1.8:8080/quantum/v1.0/pools/
curl -X POST -d '{"id":"3","address":"172.16.0.11","port":"200","pool_id":"2"}' http://192.168.1.8:8080/quantum/v1.0/members/
curl -X POST -d '{"id":"4","address":"172.16.0.12","port":"200","pool_id":"2"}' http://192.168.1.8:8080/quantum/v1.0/members/
```

Figure III-17 : Code source du script load-balancer

III.3.3.3 Test et analyse du trafic.

Après le lancement de Floodlight (en suivant les étapes mentionnée dans annexe1) et la topologie dans Mininet, on exécute le script loadbalancer.sh après ça on ouvre deux terminaux de h1 et h2 (des hôtes clients) par la commande **xterm h1 h2**

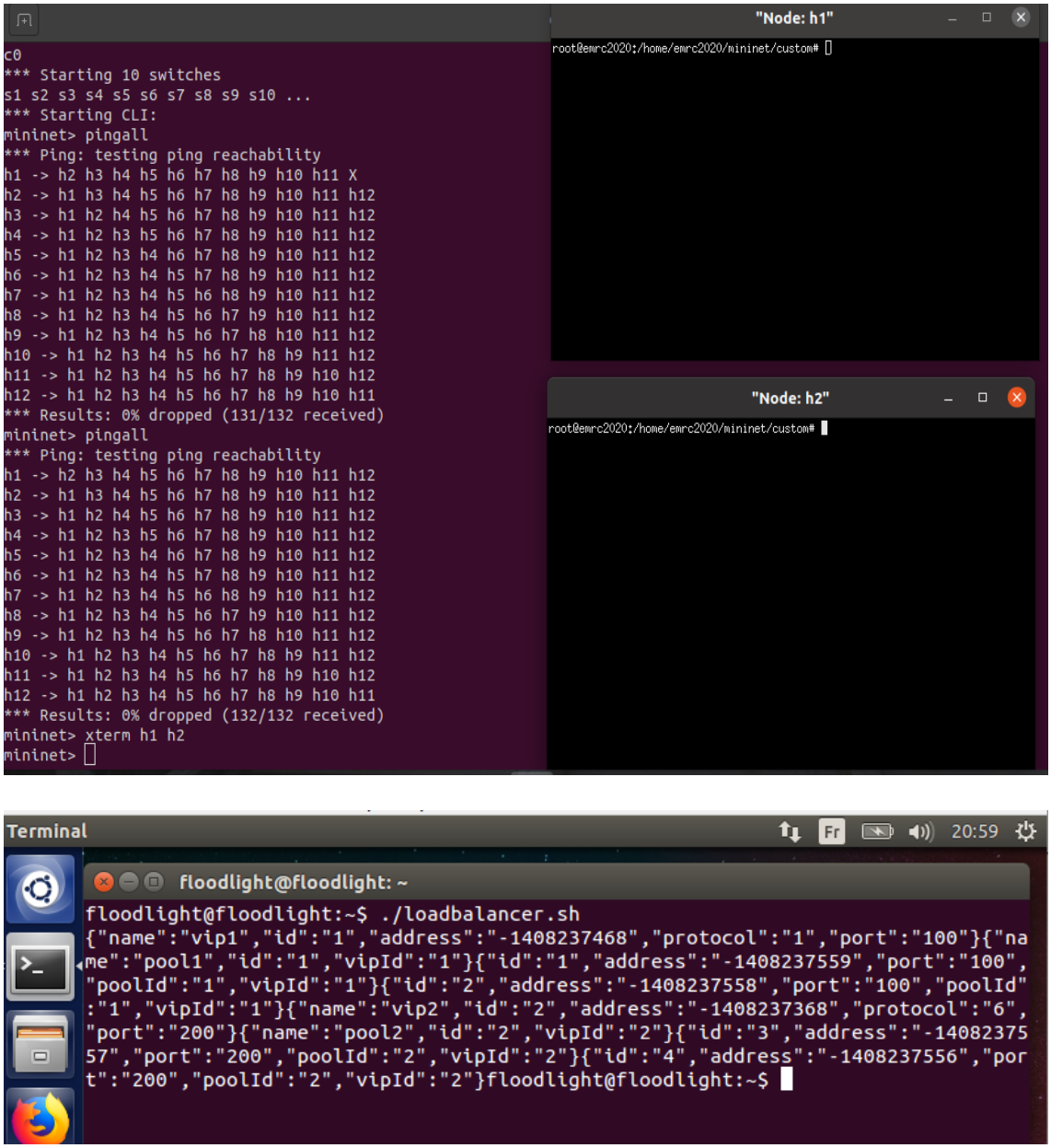


Figure III-18 : L'ouverture des terminaux h1 et h2, le lancement du Script loadbalancer

Pour tester le loadbalacing entre les deux serveurs 172.16.0.9 et 172.16.0.10 , nous avons lancé un ping vers 172.16.0.100 (VIP) à partir des terminaux de h1 avec une taille de paquets de 9500 bytes et de h2 avec une taille de paquets standard (64 bytes) parallèlement.

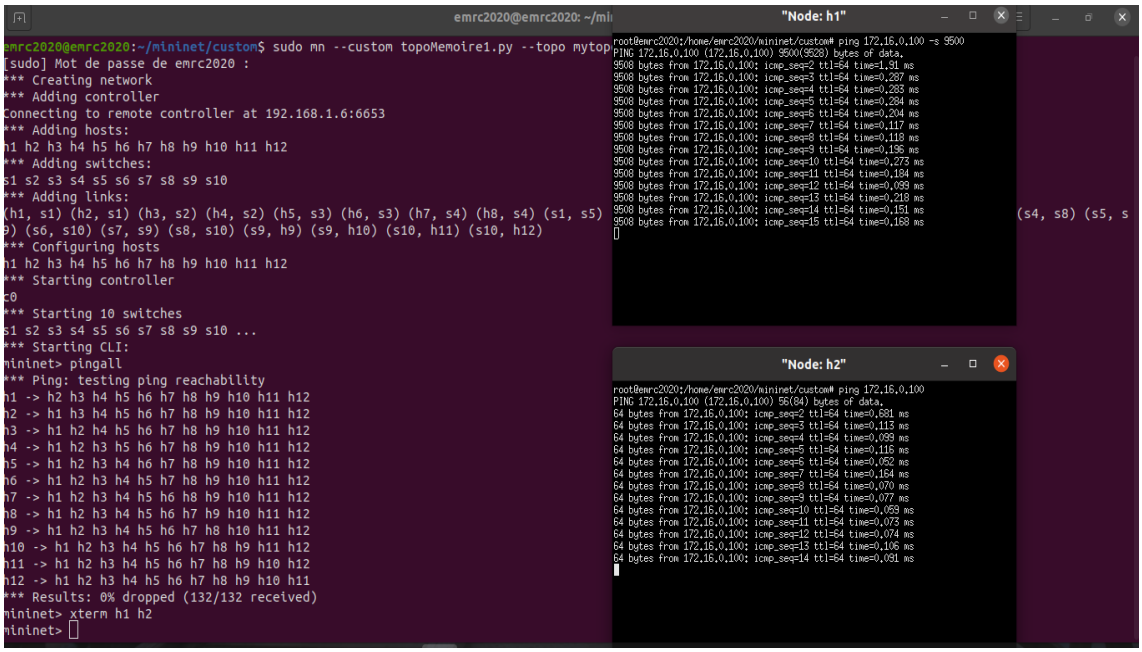


Figure III-19 : Lancement du Ping entre h1 et la VIP 172.16.0.100

Nous constatons que le contrôleur a partagé le trafic des deux hôtes entre les deux serveurs car le Traffic h1 est redirigé vers 172.16.0.9 et le Traffic de h2 est redirigé vers 172.16.0.10

Afin de vérifier ce partage nous avons fait des captures de flux dans l'interface de Floodlight ainsi que des captures de trafics dans wireshark dans le terminal du switch S1.

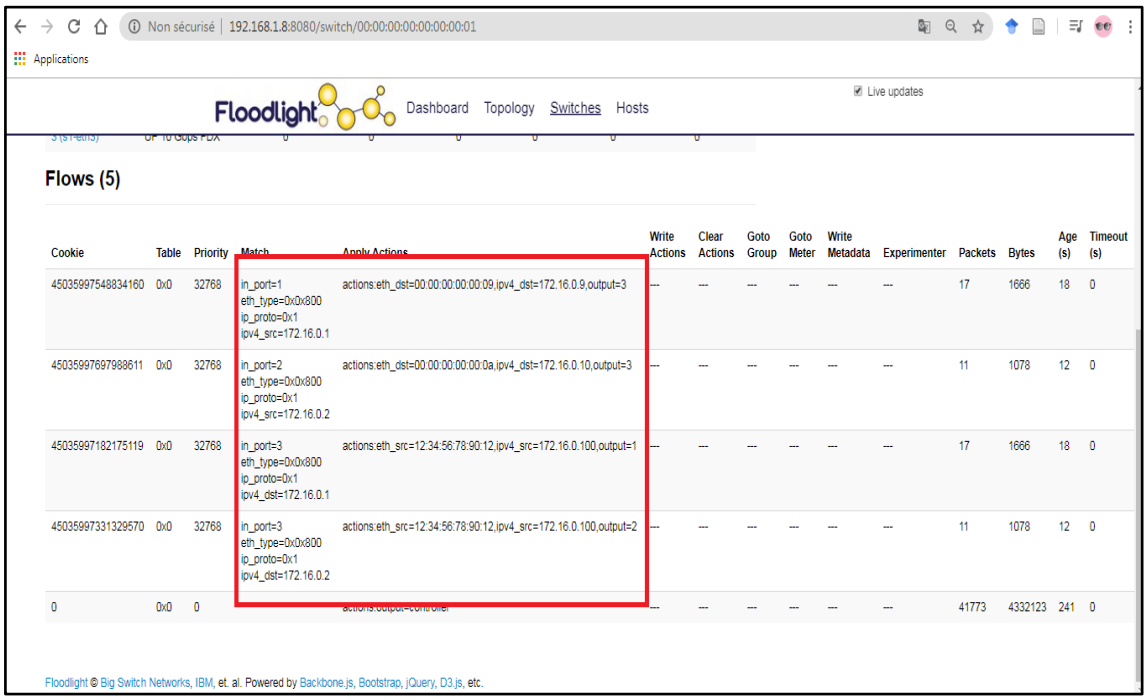


Figure III-20 : Affichage des flux créés dans S1 dans GUI du Floodlight après le ping

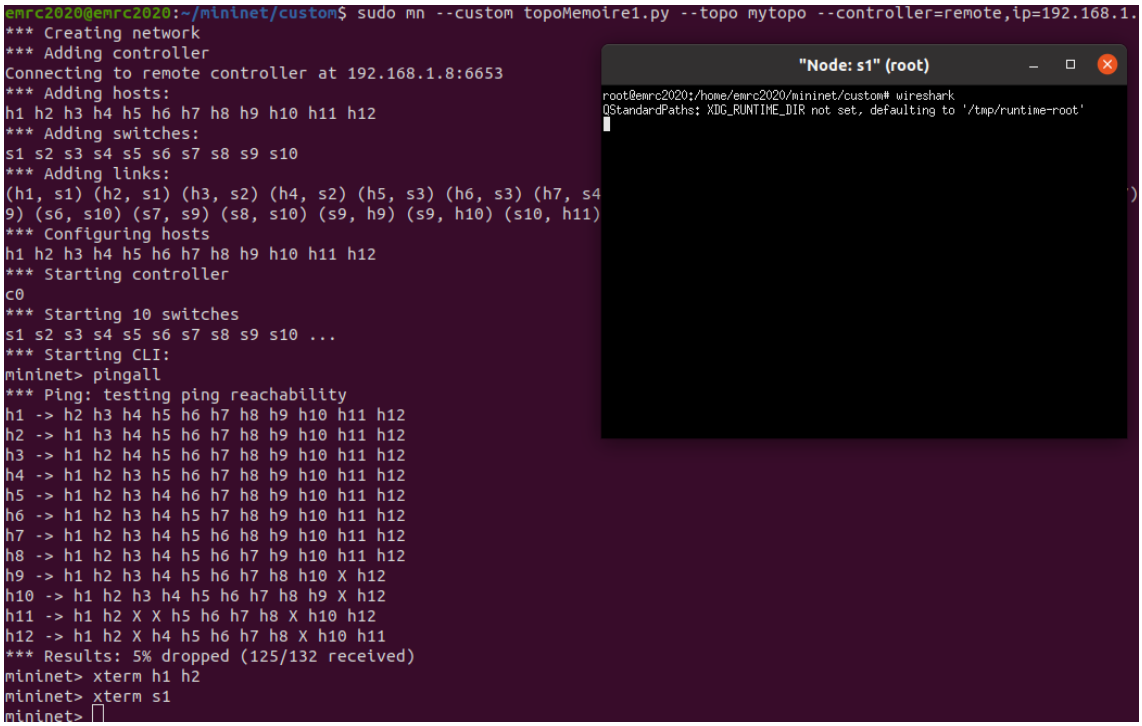


Figure III-21 : Lancement de wireshark dans S1

Pour voir les résultats dans wireshark, Nous avons capturer le trafic au niveau du switch S1 en sélectionnant les interfaces s1-eth1, s1-eth2 et s1-eth3 avec un filtrage des protocoles ARP et ICMP.

Lors du lancement du ping, h1 et h2 cherchent la résolution de l'adresse 172.16.0.100 donc ils envoient des requêtes ARP à S1 qui vont être reçu par le contrôleur Floodlight. A son tour, Floodlight génère des messages de sortie de paquets de réponse ARP avec une fausse adresse MAC.

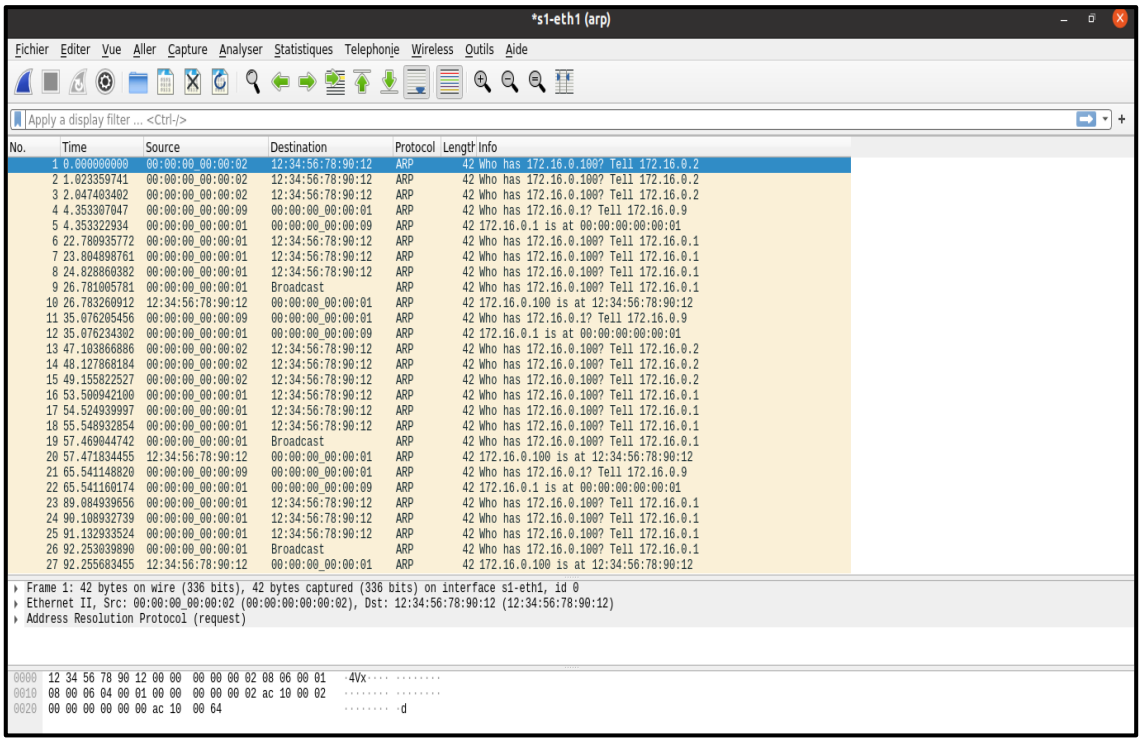


Figure III-22 : Capture des requêtes ARP des hôtes h1 et h2

Après la réponse de la requête ARP(MAC 12:34:56:78:90:12) correspondant à l'adresse IP 172.16.0.100, h1 et h2 commencent à envoyer la requête IPv4 (dans notre cas ICMP).

Chaque requête arrive à S1, ensuite elle est envoyée au contrôleur Floodlight car aucune entrée de flux ne correspond à cette en-tête. A ce moment-là, le module du loadbalancing va examiner ce message, il va trouver que cette adresse IP de destination est la même que la VIP, l'algorithme d'équilibrage de charge (dans notre cas round robin) est exécuté pour sélectionner le serveur approprié qui sera le h9 pour l'hôte h1 et le h10 pour l'hôte h2. Et par conséquent le contrôleur envoie des message pour la construction des tables des flux au niveau des switches participants dans le trajectoire vers de h1 vers 172.16.0.9 et de h2 vers 172.16.0.10.

En capturant le trafic au niveau du S1 dans le port eth1 et eth2, les paquets ICMP request sont destinées vers l'adresse virtuelle avec la fausse adresse MAC délivré par le contrôleur.

- Pour hôte H1, Port Eth1 In :

IP Source	MAC Source	IP Destination	Mac Destination
172.16.0.1	00 :00 :00 :00 :00 :01	172.16.0.100	12 :34 :56 :78 :90 :12

- Pour hôte H2, Port Eth2 In :

IP Source	MAC Source	IP Destination	Mac Destination
172.16.0.2	00 :00 :00 :00 :00 :02	172.16.0.100	12 :34 :56 :78 :90 :12

Ensuite les paquets sont forwardés vers le port eth3 selon la table des flux établis par Floodlight et les destinations sont remplacés par les adresses réelles des serveurs en partageant le trafic vers 172.16.0.9 et 172.16.0.10

- Port Eth3 Out :

IP Source	MAC Source	IP Destination	Mac Destination
172.16.0.1	00 :00 :00 :00 :00 :01	172.16.0.9	00 :00 :00 :00 :00 :09
172.16.0.2	00 :00 :00 :00 :00 :02	172.16.0.10	00 :00 :00 :00 :00 :0a

Et en retour les paquets reply au niveau du S1 sont reçus dans le port uplink eth3 (la partie source) par les adresses réelles des serveurs puis rediriger vers les ports access eth1 et eth2 des hôtes h1 et h2 en modifiant les entêtes (partie source) par les adresses VIP 172.16.0.100 et MAC 12 :34 :56 :78 :90 :12

- Port Eth1 Out :

IP Source	MAC Source	IP Destination	Mac Destination
172.16.0.100	12 :34 :56 :78 :90 :12	172.16.0.1	00 :00 :00 :00 :00 :01

- Port Eth2 Out :

IP Source	MAC Source	IP Destination	Mac Destination
172.16.0.100	12 :34 :56 :78 :90 :12	172.16.0.2	00 :00 :00 :00 :00 :02

Ce mécanisme est présenté en détail dans Les captures ci-dessous.

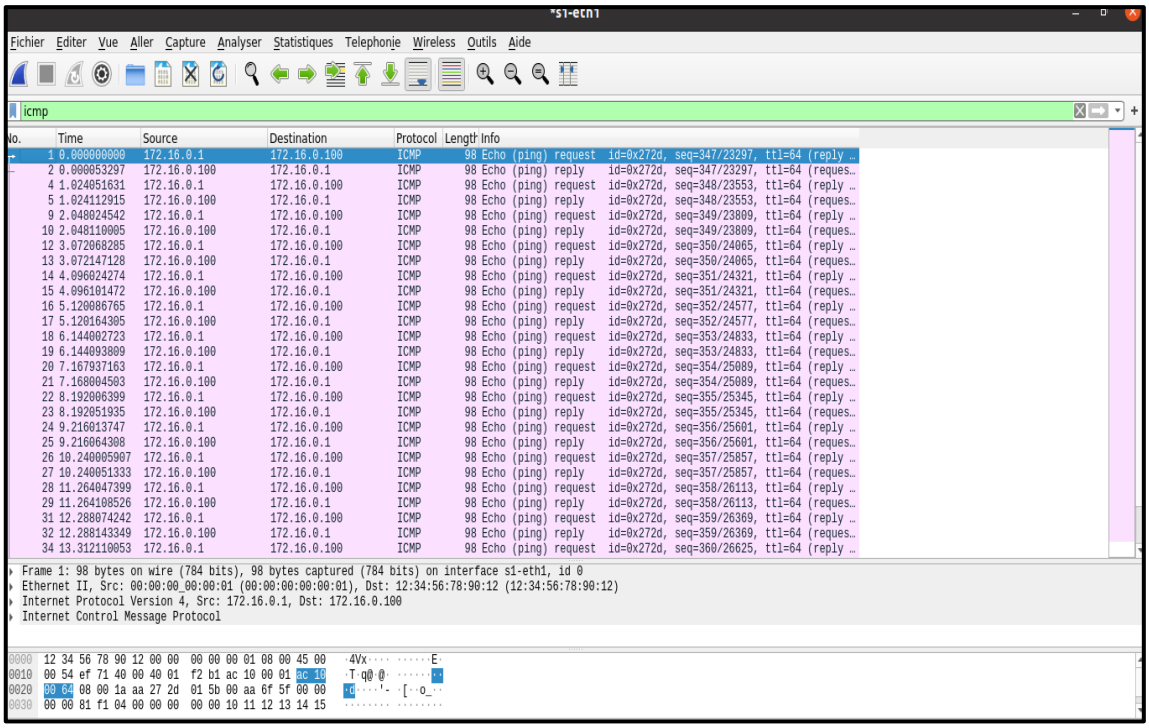


Figure III-23 : Capture du trafic ICMP de h1 au niveau du port s1-eth1

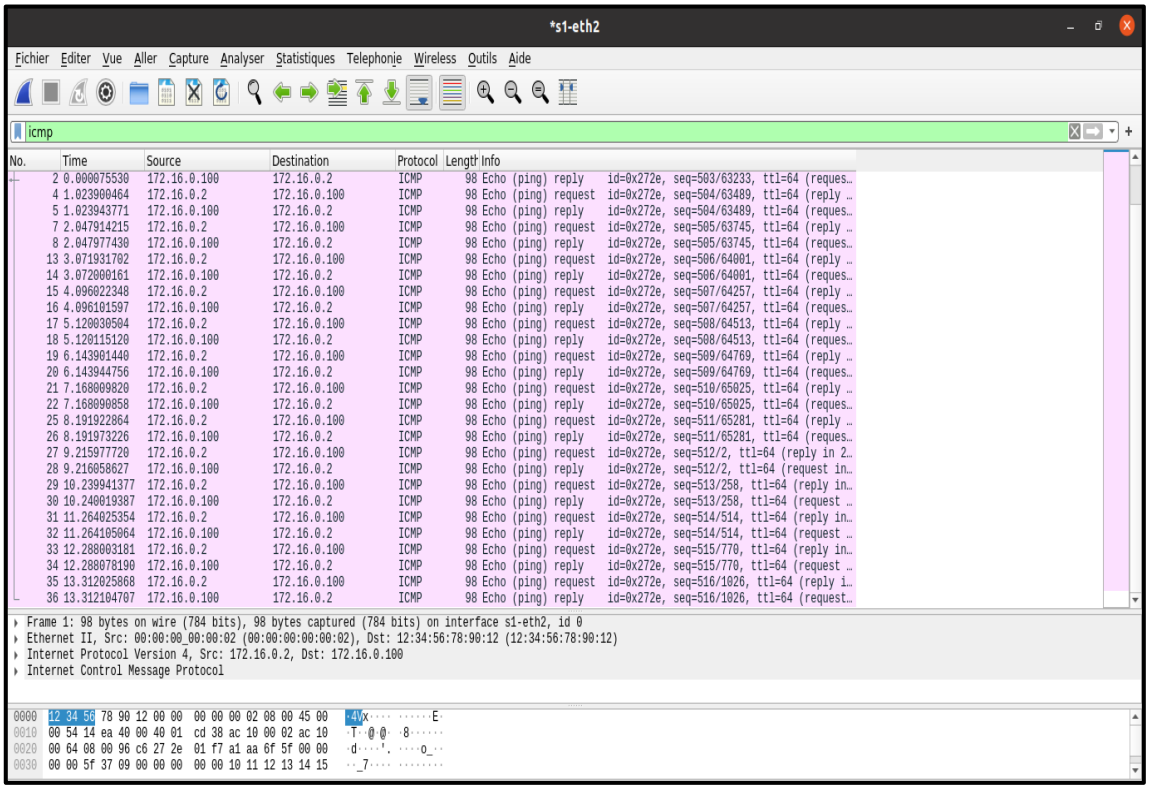


Figure III-24 : Capture du trafic ICMP du h2 au niveau du port s1-eth2

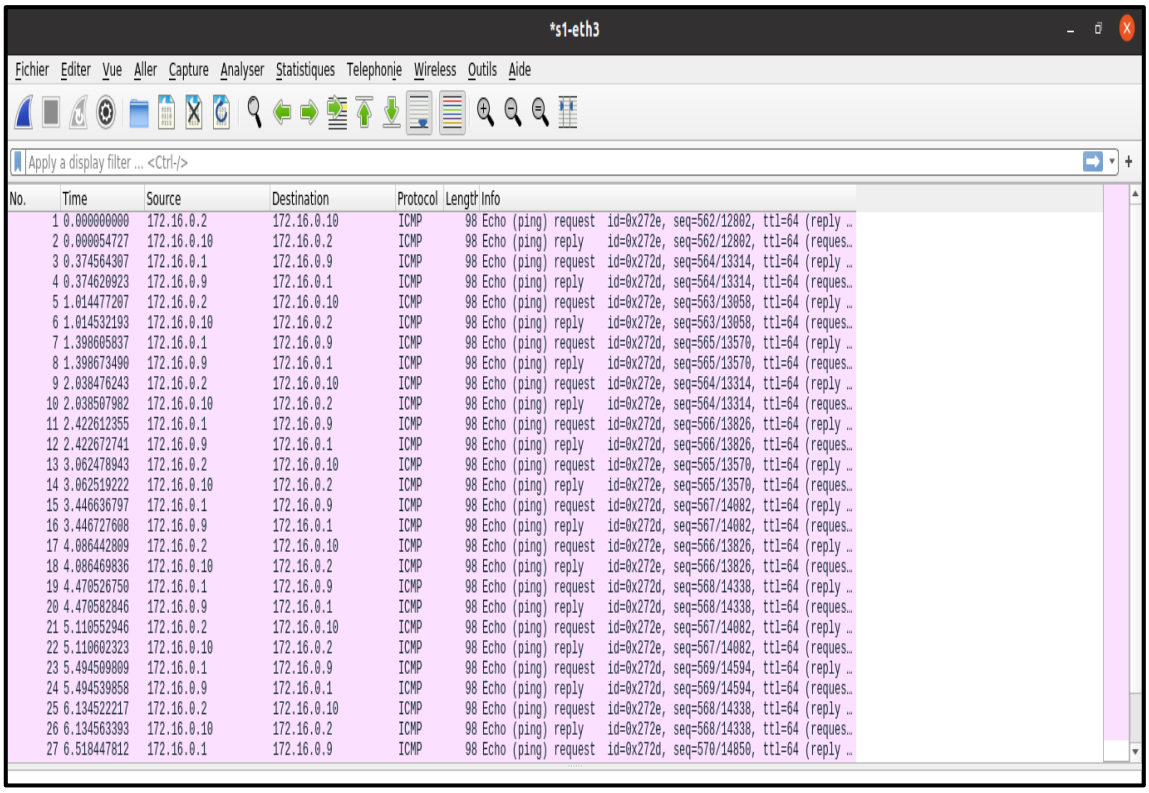


Figure III-25 : Capture du trafic ICMP du h1 et h2 en niveau du port s1-eth3

Et par ce principe les modules loadbalancing et la fonctionnalité de forwarding du contrôleur assume le partage de charge en appliquant l'algorithme de Round Robin, en prenant en compte toutes les requêtes des autres hôtes vers les serveurs.

Afin d'évaluer les performances de cette algorithme, nous avons lancer un autre ping à partir de l'hôte h4 avec une taille de paquet standard (64 bytes), nous constatons que les requêtes de h4 sont redirigées vers 172.16.0.9 malgré que la bande passante du serveur 172.16.0.10 est moins occupée que 172.16.0.9 comme le démontre les figures et captures ci-dessous.

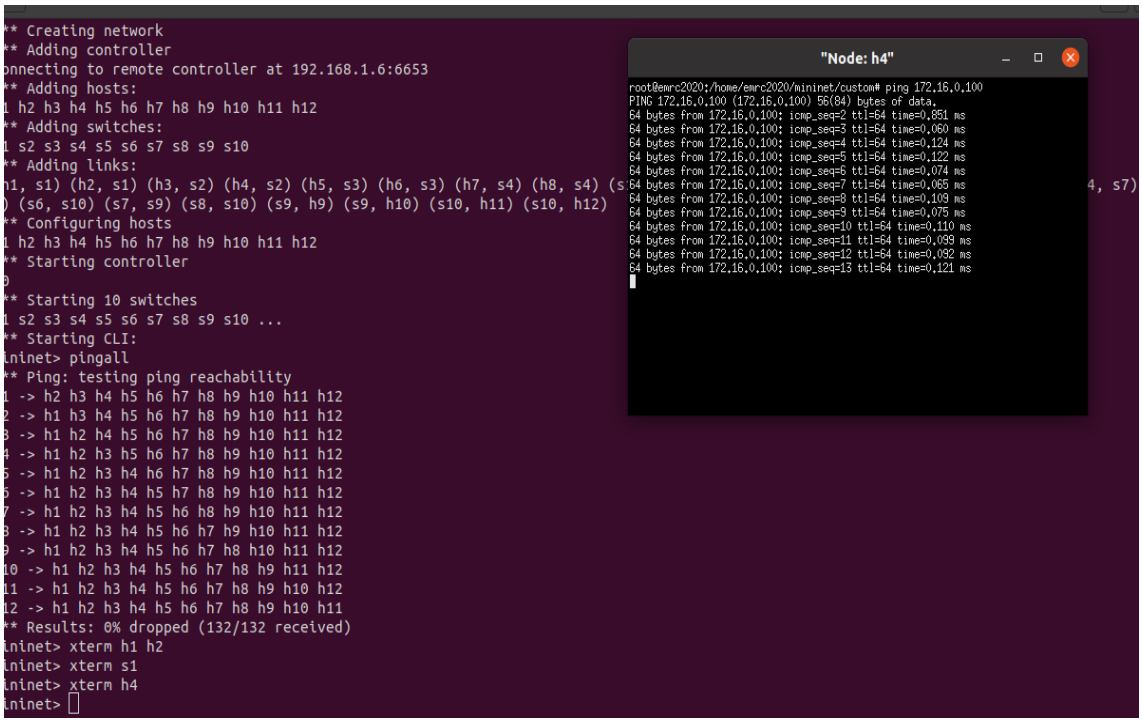


Figure III-26 : Lancement du ping de h4 vers 172.16.0.100

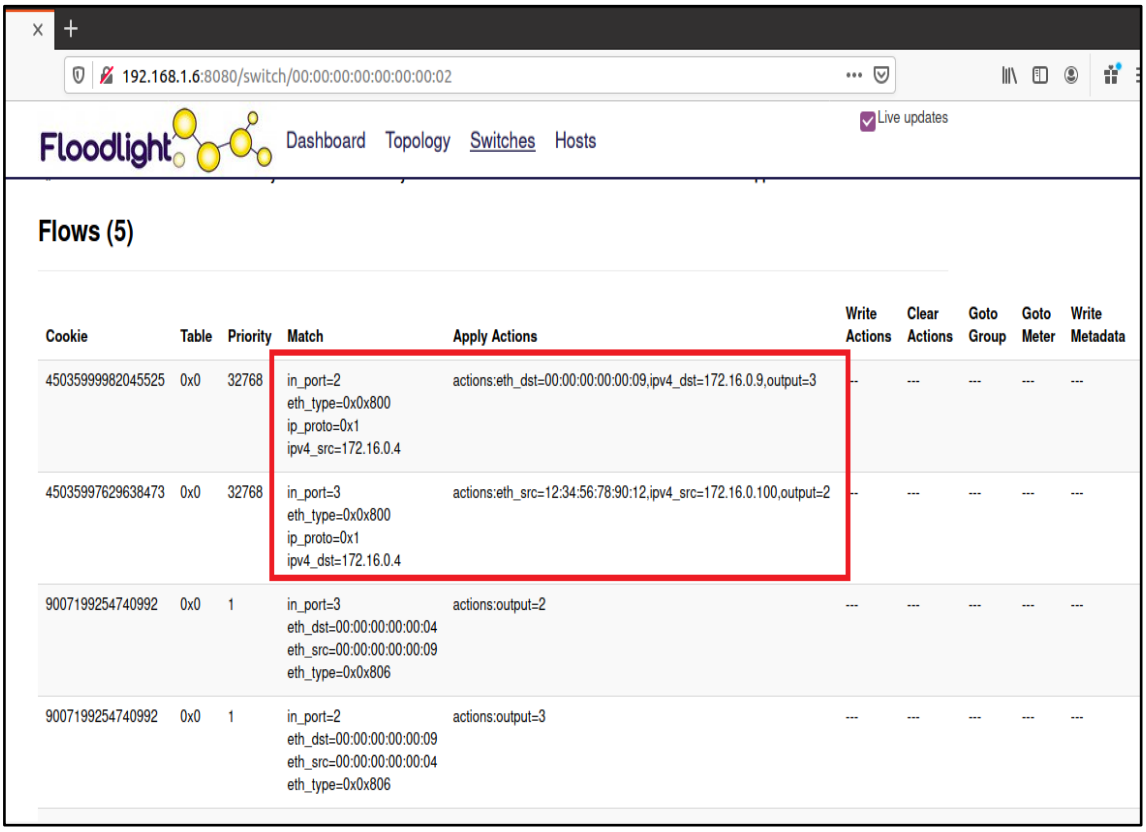
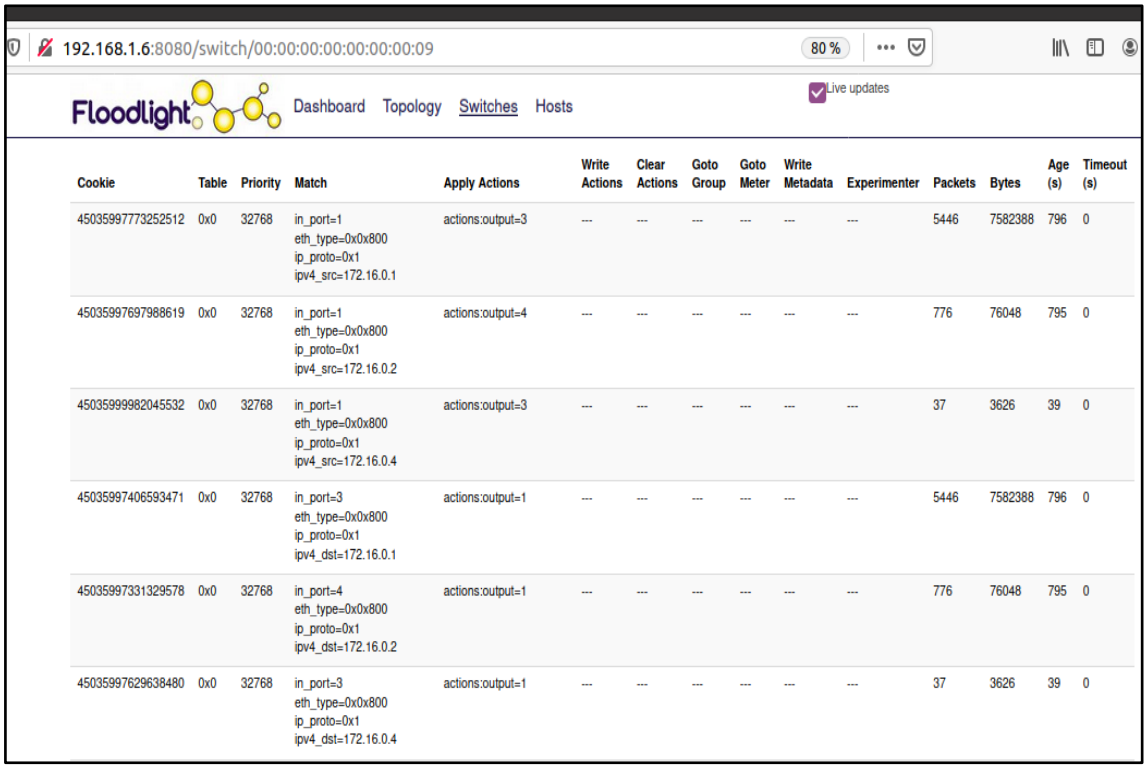


Figure III-27 : Affichage des flux créés dans S2 dans GUI du Floodlight après le ping de h4



Cookie	Table	Priority	Match	Apply Actions	Write Actions	Clear Actions	Goto Group	Goto Meter	Write Metadata	Experimenter	Packets	Bytes	Age (s)	Timeout (s)
45035997773252512	0x0	32768	in_port=1 eth_type=0x0x800 ip_proto=0x1 ipv4_src=172.16.0.1	actions:output=3	---	---	---	---	---	---	5446	7582388	796	0
45035997697988619	0x0	32768	in_port=1 eth_type=0x0x800 ip_proto=0x1 ipv4_src=172.16.0.2	actions:output=4	---	---	---	---	---	---	776	76048	795	0
45035999982045532	0x0	32768	in_port=1 eth_type=0x0x800 ip_proto=0x1 ipv4_src=172.16.0.4	actions:output=3	---	---	---	---	---	---	37	3626	39	0
45035997406593471	0x0	32768	in_port=3 eth_type=0x0x800 ip_proto=0x1 ipv4_dst=172.16.0.1	actions:output=1	---	---	---	---	---	---	5446	7582388	796	0
45035997331329578	0x0	32768	in_port=4 eth_type=0x0x800 ip_proto=0x1 ipv4_dst=172.16.0.2	actions:output=1	---	---	---	---	---	---	776	76048	795	0
45035997629638480	0x0	32768	in_port=3 eth_type=0x0x800 ip_proto=0x1 ipv4_dst=172.16.0.4	actions:output=1	---	---	---	---	---	---	37	3626	39	0

Figure III-28 : Affichage des flux créés dans S9 dans GUI du Floodlight après le ping de h4

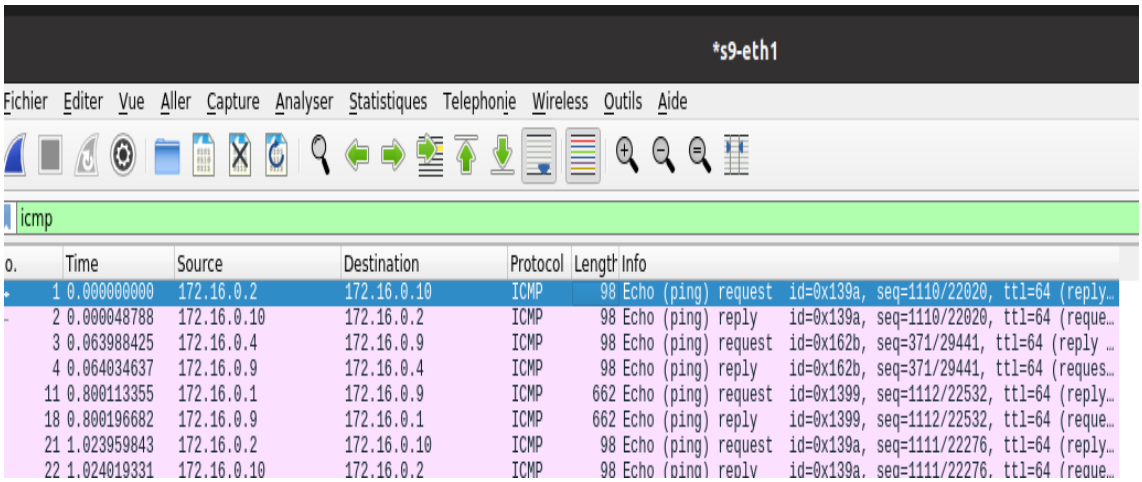


Figure III-29 : Capture du trafic ICMP du S9 au niveau du port s9-eth1

III.4 Conclusion

Dans notre plateforme nous avons implémenté la fonctionnalité du module loadbalancer concernant les trafics icmp, udp et tcp en utilisant un routage intelligent et l'algorithme Round Robin pour éviter la saturation des serveurs. En évaluant cette méthode, nous

constatons que le partage entre les hôtes et les serveurs se fait sur la base des connexions et non pas sur le volume du trafic demandé par les hôtes.

Cette approche et adaptable quand les réponses des serveurs au hôtes sont égaux en termes de volume par exemple les serveurs de VOD (Video à la demande), IPTV (IP sur TV) etc….

Par contre il y aura un partage non équitable dans le cas où on aura des volumes de trafic différents (video,data, voix,…)

CONCLUSION GENERALE

Le SDN est reconnu aujourd'hui comme une architecture permettant d'ouvrir le réseau aux applications. On est donc loin de la définition rigide initiale dans laquelle il était seulement question de dissocier les plans de contrôle et de données par exemple le forwarding de paquets orchestré par un contrôleur mais d'offrir d'autres fonctionnalités avancées par rapport aux réseaux traditionnels tel que le partage de charge dans un environnement de fermes de serveurs.

A cet effet nous avons choisi ce projet afin de mettre en œuvre et évaluer les performances de la solution du loadbalancing sous le contrôleur Floodlight. Nous avons implémenté une infrastructure constituée d'un réseau reliant plusieurs hôtes et clusters de serveur sous mininet. En appliquant la fonctionnalité de partage de charge et en faisant les tests nécessaires et captures de trafic nous avons constaté que le module API de loadbalancer de Floodlight utilise l'algorithme Roundrobin en fonction des connexions des hôtes et ne prend pas en considération les volumes des flux demandés. Cette approche est applicable dans certaines solutions où les volumes des trafics demandés sont égaux.

A cet effet et en perspective de futurs travaux nous prévoyons de développer une autre application de loadbalancing qui sera intégré au contrôleurs en utilisant d'autres algorithmes plus avancées qui prend en considération la bande passante, les ressources des serveurs et les volumes du trafic et surtout dans un environnement virtualisé Cloud.

Références Webographiques

[1] « fournisseur de services : définition de fournisseur de services et synonymes de fournisseur de services (français) ». http://dictionnaire.sensagent.leparisien.fr/fournisseur%20de%20services/fr-fr/ (consulté le juill. 20, 2020).

[2] « Seine et marne - Les réseaux Fibre optique ». http://www.seine-et-marne-numerique.fr/Les-Technologies/Technologies-filaires/Les-reseaux-Fibre-optique (consulté le juill. 29, 2020).

[3] « Comprendre ce qu'est le peering et le transit IP ». https://lafibre.info/peering/peering-transit/ (consulté le août 06, 2020).

[4] admin, « Southbound interfaces vs Northbound Interfaces - SNMPcenter », *the SNMP center*, avr. 05, 2013. https://snmpcenter.com/south-bound-interfaces/ (consulté le août 15, 2020).

[5] « OpenFlow », *Wikipédia*. déc. 23, 2019, Consulté le: août 16, 2020. [En ligne]. Disponible sur: https://fr.wikipedia.org/w/index.php?title=OpenFlow&oldid=165644129.

[6] « OPENFLOW_EFORT.pdf ». Consulté le: août 16, 2020. [En ligne]. Disponible sur: http://www.efort.com/r_tutoriels/OPENFLOW_EFORT.pdf.

[7] « SD-LAN vs LAN: What Are The Key Differences? », *Extreme Networks*, juin 17, 2017. https://www.extremenetworks.com/extreme-networks-blog/sd-lan-vs-lan-what-are-the-key-differences/ (consulté le août 14, 2020).

[8] « Open vSwitch Documentation — Open vSwitch 2.14.90 documentation ». https://docs.openvswitch.org/en/latest/ (consulté le août 22, 2020).

[9] M. Qassim, M. N. Abdullah, et A. Tariq, « Floodlight Controller onto Load Balancing of SDN Management », vol. 04, nº 08, p. 8.

[10] R. Eweka et S. Elango, « IMPLEMENTATION OF ADDRESS LEARNING/PACKET FORWARDING, FIREWALL AND LOAD BALANCING IN FLOODLIGHT CONTROLLER FOR SDN NETWORK MANAGEMENT », *Int. J. Inf. Syst. Eng.*, vol. 3, nº 1, p. 160‑170, avr. 2015, doi: 10.24924/ijise/2015.11/v3.iss1/160.170.

[11] « Load balancer : pour un meilleur temps d'accès au serveur », *IONOS Digitalguide*. https://www.ionos.fr/digitalguide/serveur/know-how/load-balancer-repartition-de-charge-sur-un-serveur/ (consulté le sept. 29, 2020).

[12] « loadbalancing-des-les-reseaux-SDN.pdf ».

Références Bibliographiques

[13] BOUIDA Hafida née SAIDI, **« Etude et mise en oeuvre d'une solution SDN : Application de Gestion de VLANs »**, Mémoire de Master en Informatique Option: Réseaux et Systèmes distribués (R.S.D), Université Abou Bekr Belkaid Tlemcen , Soutenu le 12 Juin 2017.

[14] CHEKHI Mehdi, DJEMIL Adel, **« Etude et Implémentation de l'approche Software Defined Network dans un réseau local »**, Mémoire de master en informatique Spécialité: Systèmes Informatiques et Réseaux, Université Saad Dahleb Blida, Promotion 2018/2019.

[15] AICHAOUI Anis, AITBELKACEM Yanis, **« Etude et implémentation d'une architecture SDN LAN »**, Mémoire de MASTER ACADEMIQUE Filière : Télécommunications Spécialité: Réseaux et Télécommunications, Soutenu le : 04/07/2018.

[16] Mohamed Said Seddiki, **« Allocation dynamique des ressources et gestion de la qualité de service dans la virtualisation des réseaux »**, Mémoire de Doctorat Mention Informatique, Université de Lorraine, Mémoire de Doctorat mention Technologies de l'Information et de la Communication, Ecole Supérieure des Communications de Tunis, soutenu le : 14 Avril 2015.

Annexe 1 : *Installation de Floodlight*

❖ Prés-requis :

- Il nous faut un système de virtualisation. Le système le plus recommandé et que on l'avait utilisé est Oracle VM VirtualBox, il est gratuit et il peut fonctionner sous Windows et Linux.

- Pour faire fonctionner Floodlight, Il faut installer n'importe quelle distribution Linux (exemple utilisé dans notre cas : Ubuntu 14.04.6)

- Il faut télécharger Les dépendances pour floodlight comme Java, ant, maven, python-dev par la commande suivante :

sudo apt-get install build-essential openjdk-7-jdk ant maven python-dev

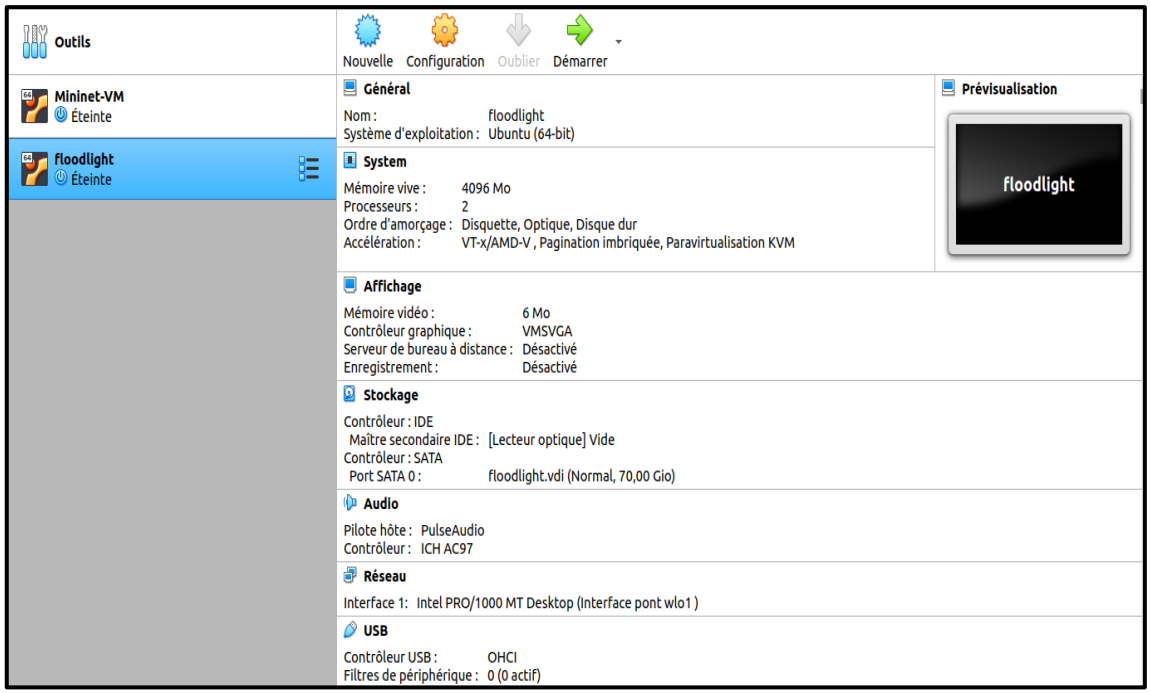
❖ Téléchargement du Floodlight :

- On peut télécharger floodlight à partir du site github .

- On a utilisé la version 1.2 en suivant les commande ci-dessous , pour spécifier la branche de la version , dans l'étape "git clone" en ajoutant "-b <nom de la version>", par exemple "-b v1.2",

- Après on lance la commande ant (qui est un projet du groupe Apache-Jakarta dont but de fournir un outil écrit en Java pour construire des applications (compilation, exécution de tâches post et pré compilation, ...).

```
git clone git://github.com/floodlight/floodlight.git -b v1.2
cd floodlight
git submodule init
git submodule update
ant

sudo mkdir /var/lib/floodlight
sudo chmod 777 /var/lib/floodlight
```



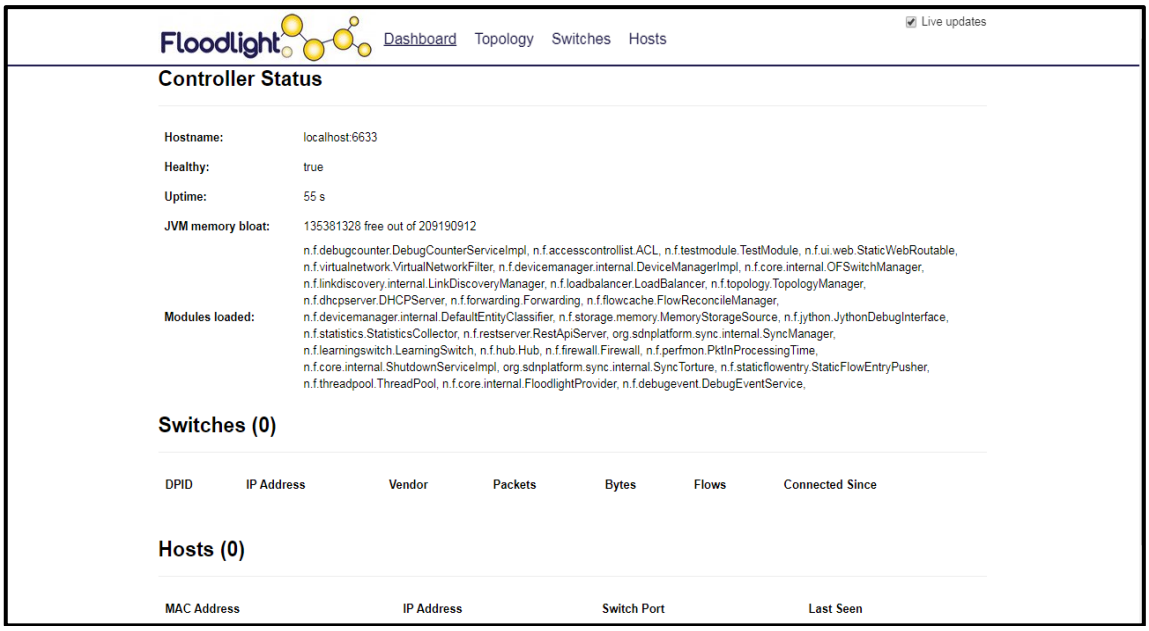
❖ Exécution du Floodlight

Une fois satisfaire ces exigences, on procède à la reconstruction du contrôleur pour pouvoir le lancer comme montre la capture ci-dessus, comme ça on peut exécuter le fichier floodlight.jar généré par ant.

```
floodlight@floodlight:~$ cd floodlight/
floodlight@floodlight:~/floodlight$ export JAVA_HOME=/usr/lib/jvm/java-7-openjdk-amd64
floodlight@floodlight:~/floodlight$ ant
Buildfile: /home/floodlight/floodlight/build.xml

init:

compile:
    [javac] Compiling 1 source file to /home/floodlight/floodlight/target/bin

compile-test:

dist:
     [echo] Setting Floodlight version: 1.2
     [echo] Setting Floodlight name: floodlight
      [jar] Building jar: /home/floodlight/floodlight/target/floodlight.jar
      [jar] Building jar: /home/floodlight/floodlight/target/floodlight-test.jar

BUILD SUCCESSFUL
Total time: 13 seconds
floodlight@floodlight:~/floodlight$ java -jar target/floodlight.jar
```

❖ Interface graphique de Floodlight

On ouvre un navigateur web (chrome par exemple) et on saisit l'url : http://ip-controlleur-floodlight :8080/ui/index.html


Floodlight
Dashboard
Topology
Switches
Hosts
Live updates
Controller Status
Hostname:
localhost:6633
Healthy:
true
Uptime:
55 s
JVM memory bloat:
135381328 free out of 209190912
Modules loaded:
n.f.debugcounter.DebugCounterServiceImpl, n.f.accesscontrollist.ACL, n.f.testmodule.TestModule, n.f.ui.web.StaticWebRoutable,
n.f.virtualnetwork.VirtualNetworkFilter, n.f.devicemanager.internal.DeviceManagerImpl, n.f.core.internal.OFSwitchManager,
n.f.linkdiscovery.internal.LinkDiscoveryManager, n.f.loadbalancer.LoadBalancer, n.f.topology.TopologyManager,
n.f.dhcpserver.DHCPServer, n.f.forwarding.Forwarding, n.f.flowcache.FlowReconcileManager,
n.f.devicemanager.internal.DefaultEntityClassifier, n.f.storage.memory.MemoryStorageSource, n.f.jython.JythonDebugInterface,
n.f.statistics.StatisticsCollector, n.f.restserver.RestApiServer, org.sdnplatform.sync.internal.SyncManager,
n.f.learningswitch.LearningSwitch, n.f.hub.Hub, n.f.firewall.Firewall, n.f.perfmon.PktInProcessingTime,
n.f.core.internal.ShutdownServiceImpl, org.sdnplatform.sync.internal.SyncTorture, n.f.staticflowentry.StaticFlowEntryPusher,
n.f.threadpool.ThreadPool, n.f.core.internal.FloodlightProvider, n.f.debugevent.DebugEventService,
Switches (0)
DPID
IP Address
Vendor
Packets
Bytes
Flows
Connected Since
Hosts (0)
MAC Address
IP Address
Switch Port
Last Seen

Listes des figures

Listes des tableaux

Listes des abréviations

	Anglais	Français
API	Application Programing Interface	Interface de programmation d'application
BDD		Base De Donnée
BYOD	Bring Your Own Device	Apportez votre propre matériel
CLI	Commande Line Interface	Interface de ligne de commande
FAI		Fournisseur d'Accées à Internet
FTP	File Tranfer Protocol	Protocole de transfert de fichiers
FttH	Fibe to the Home	Fibe à la maison
GAE	Google Apps Engine	Moteur d'applications Google
GUI	Graphic User Interface	Interface utilisateur graphique
IaaS	Infrastructure as a Service	Infrastructure en tant que service
IXP	Internet eXchange Point	Point d'échange Internet
JVM	Java Virtuel Machine	Machine virtuelle Java
LAN	Local Area Network	Réseau local
MAN	Metropolitain Area Network	Réseau régional métropolitain
NBI	North Bound Interface	Interface de liaison nord
NE	Network Element	Élément de réseau
NMS	Network Management System	Système de gestion de réseau
ONF	Open Networking Fondation	Fondation de réseau ouvert
OVS	Open Virtuel Switch	Switch virtuel
OXM	Openflow eXentisible Match	
PaaS	Platforme as a Service	Plateforme en tant que service
PAN	Personal Area Network	Réseau personnel
SaaS	Software as a Service	Logiciel en tant que service
SAL	Service Application Layer	Couche d'application de service
SBI	Sounth Bound Interface	Interface de liaison sud
SD-LAN	Software Defined LAN	LAN défini par logiciel
SDN	Software Defined Network	Réseau défini par logiciel
SD-WAN	Software Defined WAN	WAN défini par logiciel
SFTP	Secure File Tranfer Protocol	Protocole de transfert de fichiers sécurisé
SNMP	Simple Network Management Protocol	Protocole de gestion de réseau simple
SSH	Sucure SHell	
STP	Spanning Tree Protocol	Protocole Spaning Tree
TLS	Transport Layer Security	Couche de transport sécurisé
VE	Virtual Environment	Environnement virtuel
VPS	Virtual Private Server	Serveur privé virtuel
WAN	Wide Area Network	Réseau à grande distance
XML	Extensible Markup Language	Langage de balisage extensible