République **A**lgérienne **D**émocratique et **P**opulaire

Ministère de l'**E**nseignement **S**upérieur et de la **R**echerche **S**cientifique

UNIVERSITE **M**ouloud **M**AMMERI DE **T**IZI- **O**UZOU

FACULTE DE **G**ENIE **E**LECTRIQUE ET D'**I**NFORMATIQUE

DEPARTEMENT D'ELECTRONIQUE

Mémoire de Fin d'Etudes

De MASTER ACADEMIQUE

Filière : Télécommunications

Spécialité : Réseaux et Télécommunications

Présenté par :

M^{r} AICHAOUI Anis

M^{r} AITBELKACEM Yanis

Thème

Etude et implémentation d'une architecture SDN LAN

Soutenu publiquement devant :

M^{r} M. LAZRI	Maitre de conférences	A, UMMTO,	Président
M^{r} F. OUALLOUCHE	Maitre de conférences	B, UMMTO,	Encadreur
M^{r} D. ALLOUACHE	Maitre de conférences	B, UMMTO,	Examinateur

Soutenu le : 04/07/2018

Remerciements

Avant tout, nous tenons à exprimer notre profonde gratitude à notre promoteur Mr Ouallouche Fethi pour la confiance qu'il nous a accordé en acceptant de nous encadrer dans ce mémoire. Nous le remercions pour son implication, ses conseils et l'intérêt qu'il a porté à notre travail.

Nous adressons nos vifs remerciements aux membres des jurys pour avoir accepté d'examiner et juger ce travail.

Nous tenons aussi à remercier nos chères familles pour leurs soutient, encouragements et leurs bienveillance pour notre bien-être et notre succès.

On tient à remercier nos amis(e)s pour leur sincère amitié et confiance. Nous leur devons toute notre reconnaissance et notre attachement.

A tous ces intervenants, on présente nos sincères remerciements, notre respect et notre gratitude.

AICHAOUI. A & AITBELKACEM.Y

Dédicaces

A la mémoire de ma grand-mère

Je dédie ce modeste travail à ma famille

En particulier mon cher grand père Arezki

À ma chère mère

Mes chers oncles

A mon binôme Anis ainsi que toute sa famille

Mes cher(e)s ami(e)s

Yanis

Dédicaces

Je dédie ce modeste travail à ma chère mère

A mon cher père

A mes chères petites sœurs

A mes chers grands parents

Ainsi qu'à mes tantes et mes oncles

A mon binôme Yanis et toute sa famille

Mes cher(e)s ami(e)s

Anis

Sommaire

Dédicaces

Remerciements

Liste des figures

Liste des tableaux

Liste des acronymes

Introduction ……..1

Chapitre 1 : Software Defined Network

1. Préambule ……...4

2. Contexte de genèse du SDN 4

2.1.Architecture des réseaux traditionnels ……..4

2.2.Le besoin de faire évoluer les réseaux ……4

3. Principes du SDN ……4

3.1.Définition ……...6

3.2.Architecture 7

3.2.1. Le switch SDN 8

3.2.2. Le contrôleur …… 10

3.2.3. Les interfaces de communication ……..12

3.2.3.1.L'interface SUD ……..12

3.2.3.2.L'interface NORD ……..13

3.2.3.3.L'interface EST/OUEST ……14

3.2.4. Les applications ……14

3.2.5. L'orchestrateur ……15

4. Openflow 16
4.1.Architecture du protocole openflow 16
4.1.1. Table de flux 17
4.1.1.1.Champs de correspondance (Match fields) 17
4.1.1.2.Compteurs 18
4.1.1.3.Actions 18
4.2.Les messages openflow 19
4.2.1. Message controleur-commutateur 19
4.2.2. Messages asynchrones 20
4.2.3. Messages symetriques 20
4.3.Les spécifications Openflow 21
5. Cas d'utilisations du SDN 22
5.1.QoS sur internet 22
5.2.Data center 22
5.3.Réseaux mobiles 22
5.4.Sécurité 23
5.5.Réseaux de campus et d'entreprise 23
6. Discussion 24

Chapitre 2 : Design et implémentation d'une solution SDN

1. Préambule 26
2. Infrastructure 26
2.1.Presentation du scénario 26
2.2.Open vswitch 27
2.2.1. Anatomie d'OVS 27
2.2.2. Fonctionnement d'Open vswitch 28
2.3.Mininet 29
2.3.1. La ligne de commande Mininet 29
2.3.2. Topologies personalisées via l'API Mininet 30

3. Le contrôleur SDN Opendaylight 31
3.1. Presentation du projet Opendaylight 31
3.2. Distribution d'ODL 32
3.3. Architecture Opendaylight 33
3.3.1. Couche d'abstraction de service(SAL) 34
3.4. Distribution Karaf du contrôleur ODL 34
3.4.1. RESTCONF 36
3.4.2. POSTMAN 38
4. L'application Openflow manager 40
4.1. Architecture de l'application OFM 40
4.2. Fonctionnalités d'OFM 41
5. Discussion 44

Chapitre 3 : Réalisation d'une solution SDN

1. Préambule 46
2. Scenario1 : Administration d'un réseau SDN 46
2.1. Topologie proposée 46
2.2. Opération d'administration 48
2.2.1. Commutation et Routage 48
2.2.2. Pare-feu 48
2.3. Implémentation de l'architecture 48
2.3.1. Création de la topologie 49
2.3.2. Planification du comportement du réseau 51
2.3.3. Installation des flux 52
2.3.4. Tests de fonctionnement 54
2.3.4.1. Test d'accessibilité 54
2.3.4.2. Test d'orientation du trafic 55
2.3.4.3. Test de redondance 56
3. Scenario2 : Routage sensible aux applications 57

3.1. Etude du scénario 57
3.2. Approche d'indentification des applications 59
3.3. Implémentation de l'identification par protocole 59
3.4. Résultats 60
3.4.1. http 60
3.4.2. FTP 60
3.4.3. UDP 61
4. Discussion 62
4.1. Scenario 1 62
4.2. Scenario 2 62
Conclusion 63
Bibliographie 66
Annexes

Liste des figures :

Figure 1 : Consternation du plan de contrôle dans un routeur 05
Figure 2: Architecture du réseau SDN 07
Figure 3 : Anatomie logique d'un switch SDN interprété de (Goransson et al ; 2016). 09
Figure 4 : Contrôleur SDN : éléments, services, et interfaces 11
Figure 5 : Processus de communication entre le Switch et le contrôleur.....13
Figure 6: Communication inter-contrôleur dans une architecture distribuée..... 14
Figure 7 : Architecture SDN avec un orchestrateur 15
Figure 8: Architecture Openflow 16
Figure 9 : Pipeline Openflow v1.1 17
Figure 10 : Entrée de flux.....17
Figure 11 : Champs de correspondance Openflow 18
Figure 12 : Schématisation du modèle de flux 19
Figure 13 : Message Symétrique..... 21
Figure 14 : Composants d'OVS 27
Figure 15 : Affichage de la commande show d'OVS 28
Figure 16 : Exemple du script Python d'une topologie personnalisée 31
Figure 17 : Membres du projet ODL..... 32
Figure 18 : Architecture simplifiée d'ODL.....34
Figure 19 : Distribution Karaf d'OpenDaylight.....35
Figure 20 : UI d'OpenDaylight 35
Figure 21 :Vue globale d'une topologie sur l'UI ODL.....36
Figure 22 : Documentation du REST API 37
Figure 23 : Types de requêtes REST API 37
Figure 24 : Vue d'ensemble de POSTMAN 38
Figure 25 : En tête d'une requête PUT.....39

Figure 26 : Exemple de script XML.....39
Figure 27 : Résultat de la commande dump-flows s2 40
Figure 28 : Résultat du test ping.....40
Figure 29 : Architecture d'OFM 41
Figure 30 : Interface utilisateur d'OFM 42
Figure 31 : Accès à la gestion de flux 42
Figure 32 : Programmation de flux sur OFM.....43
Figure 33 : Résultat de la requête GET 43
Figure 34 : Croquis de la topologie du réseau..... 47
Figure 35 : Configuration de la machine ODL 49
Figure 36 : Création des éléments du réseau dans Mininet 50
Figure 37 : Topologie du scénario 1 affichée sur ODL 50
Figure 38 : Vue globale de la topologie du scénario 1 affichée par OFM 51
Figure 39 : Installation de l'entrée flux 305 sur OFM 54
Figure 40 : Test d'accessibilité des hôtes 54
Figure 41 : Capture wireshark d'un ping entre H1 et H2 55
Figure 42 : Capture wireshark entre H2 et H4 55
Figure 43 : Capture wireshark d'un ping entre H4 et H5 56
Figure 44 : Détection de la perte de liaison s1 s5 sur ODL 56
Figure 45 : Capture wireshark test de redondance 57
Figure 46 : Latence du chemin S3 58
Figure 47 : Latence du chemin S4 58
Figure 48 : Latence du chemin S5 58
Figure 49 : Orientation du trafic HTTP sur OFM 59

Figure 50 : Test de l'application HTTP 60

Figure 51 : Test de l'application FTP 61

Figure 52 : Test de l'application UDP 61

Liste des tableaux

Tableau 1 : Caractéristiques des contrôleurs SDN 12

Tableau 2 : Tableau de comparaison des différentes versions Openflow 21

Tableau 3 : Versions d²Opendaylight.....33

Tableau 4 : Informations des hôtes.....47

Tableau 5 : Table deFlux s1 52

Tableau 6 : Table de flux s2 52

Tableau 7 : Table de flux de s3 53

Tableau 8 : Table de flux de s4 53

Liste des Acronymes :

AD-SAL : API Driven SAL

API : Application programing interface (Interface de programmation d'application)

BGP : BorderGatewayProtocol

Br : Bridge

BYOD : Bring Your Own Device (Apporter son propre matériel)

CLI : Command Line Interface (Ligne de commande)

CSDN : Cellular SDN SDN cellulaire

DDoS : Distributed Denial of Service (Déni de service distribué)

DPI :Deep Packet Inspection (Inspection approfondie des paquets)

DS : Differentiated Services (Services différenciés)

FTP : File Transfer Protocol (Protocole de transfert de fichier)

Gb : Giga Bit

HTTP : Hyper Text Transfer protocol (Protocole de Transfer hyper texte)

I2RS : Interface to Routing System (Interface vers système de routage)

IoT : Internet of Things (Internet des objets)

IP : Internet Protocol

IPv4 : Internet Protocol version 4

ISO : International Standardization Organisation

LAN : Local Area Network (Réseau local)

MAC : Media Access Control

MD-SAL : Model Driven SAL

MPLS : Multiprotocol Label Switching

NaaS : Network as a Service (Réseau en tant que service)

NETCONF : NETwork CONFiguration protocol (Protocole de configuration réseau)

NIB : Network Information Base (Base d'information réseau)

ODL : Opendaylight

OF : Openflow

OFM : OpenFlow Manager

ONF : Open Networking Foundation

ONOS : Open Network Operating System

OSGI : OpenServicesGateway Initiative

OVF : Open View Finder

OVS : Open Virtual Switch

Ovsdb : open virtual switch data base

OXM : OpenFlow eXtensible Match (Openflow à correspondance extensible)

QOS : Quality of Service (Qualité de service)

RAM : Read Access memory

REST : REpresentational State Transfer (Transfert d'état représentatif)

SAL : Service Abstraction Layer (couche d'abstraction de service)

SDN : Software Defined Network (Réseau défini par logiciel)

SNMP : Simple Network Management Protocol (Simple protocole de gestion de réseau)

SSH : Secure Shell

STP : SpanningTree Protocol

TCAM :Tenary contenant addressables memory

TCP : Transmission Control Protocol (Protocole de contrôle de transmission)

TLS : Transport Layer Security (Sécurité de la couche de transport)

TLV : Type Length Value (Valeur de type de longueur)

TTL : Time To Live

UDP : User Datagram Protocol

VLAN : Virtual Local Area Network (Réseau local virtuel)

VM : Virtual Machine (Machine virtuelle)

VoD : Video on Demand (Vidéo à la demande)

VoIP : Voice over Internet Protocol (Téléphonie IP)

VSDN : Video over SDN (Vidéo par SDN)

WAN : Wide Area Network

Introduction

Les technologies de la communication évoluent à grande vitesse, à tel point qu'une tendance peut se retrouver dans les archives en l'intervalle de quelques mois. Cette houle de créativité n'a laissé personne indifférent et affecte le quotidien de toute la société. Les services proposés aujourd'hui n'étaient même pas envisageables quelques années auparavant. L'objectif étant de tout virtualiser pour atteindre le « Everything as a service », un pas en plus vers l'internet des objets (IoT).

Le processus de virtualisation est à un stade très avancé au niveau du computing et du stockage, grâce notamment à l'avènement du cloud et des dockers, des techniques qui changent entièrement la perspective d'exploitation des ressources, introduisant une mobilité et une dynamique sans précédent.

La structure actuelle de routage et de commutation est rigide et ne peut répondre aux attentes de virtualisation et d'automatisation indispensables à une gestion optimale de la communication entre l'utilisateur, les machines et les applications régissant celles-ci. En deux décennies, l'architecture statique des réseaux traditionnels n'a connue aucun changement majeur, seules les technologies hardware ont évolués pour proposer de meilleurs débits de transmission (Giga Ethernet, Fibre optique ...). Cette vision simpliste est dépassée et n'est pas adéquate avec la plupart des déploiements de centres de données.

C'est dans ce contexte qu'a émergé l'idée du Software Defined Networking (SDN), ou réseau défini à base de logiciels. Un modèle qui a vu le jour ces dernières années, avec la promesse d'un réseau programmable et une préface très attirante à première vue. A cet effet, la majeure partie de la communauté réseau y compris des géants de l'industrie (Cisco, Citrix, Juniper, HP…), y ont concentré leur intérêt. Signe que la transition vers ce type d'architecture est inévitable.

L'idée de base est née du besoin d'interopérabilité libre entre les équipements. Les développeurs se sont ensuite inspirés du modèle open source de Linux pour unir les plans de contrôle de tous les équipements réseaux dans une seule et même plateforme appelée contrôleur. Ceci a l'avantage d'offrir une vue globale de ce qui se passe dans la couche infrastructure, et permet de récupérer toutes les informations liées à celle-ci. Ceci dit, une application positionnée à l'interface nord du contrôleur permet de programmer

dynamiquement le réseau de telle sorte à changer de comportement en fonction des événements occurrents en temps réel.

Cette perspective d'un renouveau dans la conception même du réseau, nous a motivés à tenter d'étudier les concepts régissant cette nouvelle vision. Par une étude bibliographique approfondie nous présenterons les différentes normes régissant le paradigme SDN ainsi que ses cas d'usage les plus en vue. Nous implémenterons ensuite son architecture dans un environnement virtuel pour répondre aux problématiques suivantes :

- Quelles sont les changements majeurs apportés aux tâches d'administration d'un réseau ?
- Comment exploiter les différentes fonctionnalités apportées par les éléments du SDN pour réaliser une solution facilitant l'exploitation des ressources déployées dans le réseau ?

Ce mémoire sera réparti en 3 chapitres :

Dans le chapitre 1, nous nous étalerons sur les principes fondamentaux du paradigme SDN. Il nous permettra d'avoir une idée globale sur son fonctionnement.

Le 2éme chapitre a pour objectifs de découvrir comment bâtir une architecture SDN.

Dans le 3éme chapitre, nous présenterons l'implémentation de la solution SDN. Pour cela, nous proposerons une topologie généralement utilisée au sein des entreprises puis nous expliquerons les configurations à suivre pour son administration. Des tests seront effectués afin de vérifier la fiabilité de cette implémentation.

Nous terminerons ce mémoire par une conclusion et une bibliographie.

Chapitre 1:
Software Defined Network

1. Préambule

Récemment, les réseaux programmés par logiciel (SDN, Software-Defined Networking) ont gagné en popularité dans les milieux industriels et de recherches, Un paradigme qui vient bouleverser la sphère réseau avec la promesse de changer notre perception de l'infrastructure. Dans ce chapitre nous allons revenir à la source de cette évolution, comprendre les facteurs qui ont poussé à l'apparition de ce paradigme. Ensuite nous nous étalerons sur les principes de fonctionnement du SDN, d'Openflow et de leurs applications les plus significatives.

2. Contexte de genèse du SDN

2.1. Architecture des réseaux traditionnels

À l'intérieur de chaque périphérique réseau et de sécurité, à savoir chaque commutateur, routeur et pare-feu, vous pouvez séparer le logiciel en quatre couches ou plans (Juniper, 2013).

- **Transfert :** A pour rôle d'acheminer les paquets réseau. Il est optimisé afin de déplacer les données aussi rapidement que possible.
- **Contrôle :** Ce plan décide de la direction des flux réseau, il décode les protocoles pour assurer la fluidité du trafic. Celui-ci se trouve dans un logiciel dédié intimement relié au hardware.
- **Services :** Cette couche n'existe pas dans les switchs. Un exemple de son utilisation est le pare-feu, elle permet d'exploiter et de déployer des services plus complexes sur l'appareil en question.
- **Gestion :** Le plan Gestion fournit les instructions de base indiquant comment le périphérique réseau doit interagir avec le reste du réseau. On y accède par l'interface de ligne de commande (CLI) pour insérer la configuration du périphérique.

Dans la figure ci-dessous on expose le plan de contrôle d'un routeur constamment bombardé de messages par le nombre interminable de protocoles dont dépend le réseau.

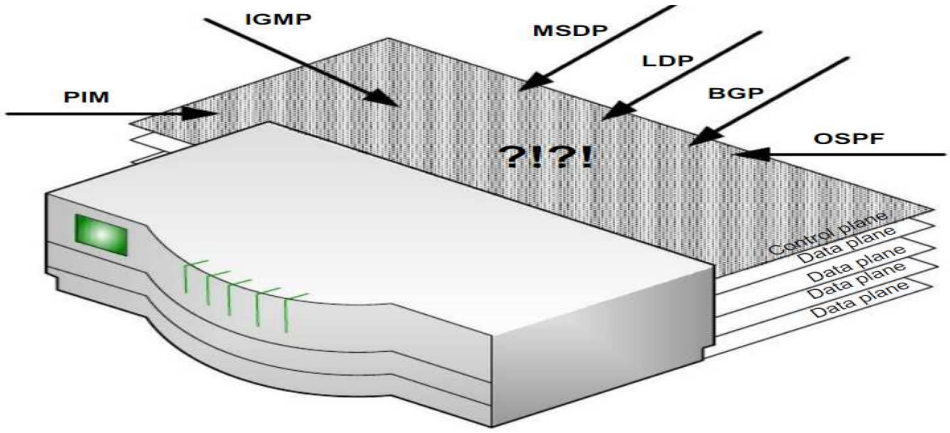

Figure 1 : Consternation du plan de contrôle dans un routeur (Goransson et al, 2016).

2.2.Le besoin de faire évoluer les réseaux

D'après Cisco Visual Networking (Cisco VNI, 2017), à l'horizon 2021 le nombre d'appareils mobiles connectés dépassera le nombre de personne sur terre à hauteur de 1,5 mobile par personne. Ce chiffre est représentatif de l'ampleur exponentielle que prennent les technologies de télécommunication dans la société moderne, et avec l'avènement d'internet des objets (Internet of Things IoT) tout sera connecté, ce qui implique une infrastructure réseau d'une complexité ingérable et la nécessité d'un réseau programmable, réduisant ainsi l'intervention humaine qui est la source principale des failles dans les réseaux.

L'architecture fondamentale des réseaux traditionnels n'a connue aucun changement majeur depuis plus de 20 ans. Ce statut quo est en partie dû à l'architecture des appareils en eux même, le fait est que la coexistence du plan de contrôle et du plan de données dans chaque équipement physique ce qui rend très difficile tout déploiement de nouvelle fonctionnalité ou protocole car ceux-ci doivent être incorporées dans l'équipement physique, 5 à 10 ans pour concevoir et déployer un protocole de routage (Benamrane, 2017). Cet environnement n'encourage pas le développement, car seuls les fabricants ont accès au hardware. Mais ce système a aussi créé une dépendance des particuliers envers leurs fabricants, faute de problèmes d'interopérabilité entre les différentes marques, les utilisateurs se trouvent comme liés à leurs fabricants et ne peuvent en aucun cas changer de fournisseur, une atmosphère sans compétitivité et qui encourage la flambée des prix des appareils réseaux. Ces systèmes renfermés ont instaurés une inertie dans le réseau en comparaison avec l'informatique et les technologies de stockage, et si on devait s'inspirer de celles-ci et d'un

des pionniers du développement de l'informatique qu'est Linux, L'open source est la meilleure initiative.

Mais le point qui force le networking à sortir de sa bulle est le contraste de sa nature rigide et statique avec les nouvelles techniques dans les data center que ce soit la virtualisation des machines ou le cloud computing (Dubey, 2016). Ces dernières apportent une flexibilité sans précédent au déploiement des services et l'exploitation des ressources, Or l'infrastructure réseau opère comme un frein dans ce modèle qui requiert une agilité particulière, car la ou il suffit de quelques minutes pour créer une nouvelle instance d'une VM, l'ajout d'une instance quelconque dans le réseau pourrait durer des jours. Ceci crée des problémes d'extensibilité dans les data center qui sont censés croitre au besoin (Goransson et al, 2016). C'est la virtualisation qui fait toute la différence. La tendance du cloud est le « everything as a service » (Tout en service : Ressources RAM, CPU, Stockage, Applications …), et la derniére piéce manquante pour le tout virtuel est le « Network as a service (NaaS) » (Le réseau en service).

Toutes ces limitations ont menés à l'émergence du Software Defined Networking.

3. Principes du SDN

3.1. Définition

Selon le RFC7149 (Boucadair et jaquenet, 2014), Le SDN (*Software-Defined Networking*) est un ensemble de techniques visant à faciliter l'architecture, la livraison et l'opération de services réseaux de manière déterministe, dynamique et pouvant être déployé à grande échelle.

L'ONF (Open Networking Foundation, 2013) définit quant à elle le SDN comme étant une architecture qui sépare le plan de contrôle du plan de données, et unifie les plans de contrôle de plusieurs périphériques dans un seul software de contrôle externe appelé « Contrôleur », qui voit le réseau dans sa totalité pour gérer l'infrastructure via des interfaces de communications appelées APIs. Le contrôleur en question fait abstraction de la couche physique pour les applications qui communiquent en langage développeur, permettant la programmation du réseau.

Globalement un réseau est dit SDN en considération des 5 caractéristiques suivantes :

- Séparation du plan de données et du plan de contrôle.
- Périphériques simplifiés.
- Contrôle centralisé.
- Automatisation du réseau et virtualisation.
- Open source.

3.2.Architecture

Le réseau SDN est composé de 3 couches communiquant entre elles par le biais d'interfaces APIs. De la plus basse à la plus haute nous avons :

- **La couche infrastructure :** Ici se trouve les périphériques qui contiennent les plans de données du réseau. Autrement dit, les switchs SDN responsables de l'acheminement du trafic.
- **La couche contrôle :** Elle se base sur le contrôleur SDN, il contrôle le plan de donnée de façon réactive ou proactive en insérant les différentes politiques de transfert. Cette partie est le cerveau du réseau et elle englobe la plupart des opérations de calcul.
- **La couche application :** Héberge les applications qui permettant d'exploiter la panoplie d'avantages qu'apporte l'architecture, en introduisant de nouvelles fonctionnalités réseaux.

Dans cette disposition l'application communique les décisions et politiques de routage au contrôleur, qui a son tour traduit ces décisions pour programmer les équipements appropriés. Les dispositifs de transmission comparent ensuite l'entête des paquets avec les tables de flux pour ensuite effectuer les actions prédéfinies dans les tables (Nadeau et Gray, 2013).

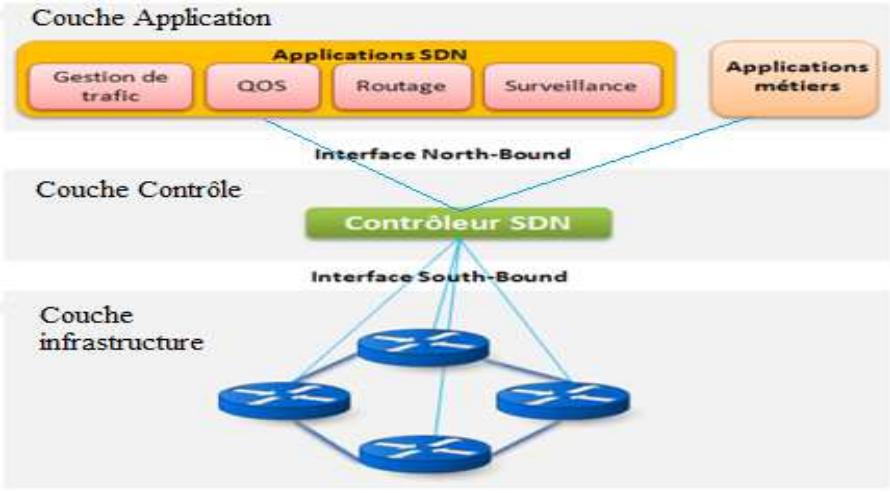

Figure 2 : Architecture du réseau SDN

L'architecture SDN est donc définie par trois abstractions (Kreutz, 2014) :

- L'abstraction de transmission : Permet aux applications de programmer et de prendre les décisions concernant le réseau sans pour autant connaitre les détails de l'infrastructure physique. Ceci est achevé à travers l'usage de protocoles ouvert et standardisé pour la communication avec les équipements réseau.
- L'abstraction de distribution : Elle est implémentée par le contrôleur et est essentiellement responsable de deux taches. La première est d'insérer les règles de transfert dans les switchs. La seconde est de récupérer les informations de la couche inférieure, pour informer les applications de l'état de celle-ci, pour pouvoir former une vue globale du réseau.
- L'abstraction de spécification : Permet aux applications d'exprimer le comportement désiré du réseau sans en être directement responsable.

Nous allons par la suite nous intéresser aux spécifications des principaux acteurs de cette architecture.

3.2.1. Le switch SDN

Par abus de langage, tout les équipements de transmission SDN sont appelés switch. Dans ce jargon le terme switch est attribué à tout équipement de transmission qui compare l'entête des paquets aux tables de flux, que la comparaison se fasse à base d'adresses MAC (Couche 2), d'adresses IP (Couche 3) ou une combinaison de plusieurs champs. La terminologie switch a été adoptée en référence à l'unique tache de ces équipements qui est la transmission (Kreutz et al, 2014).

Un switch SDN est composé d'une API pour communiquer avec le contrôleur, une couche d'abstraction qui consiste en un pipeline de tables de flux et une fonction de traitement de paquets (Goransson et al, 2016).

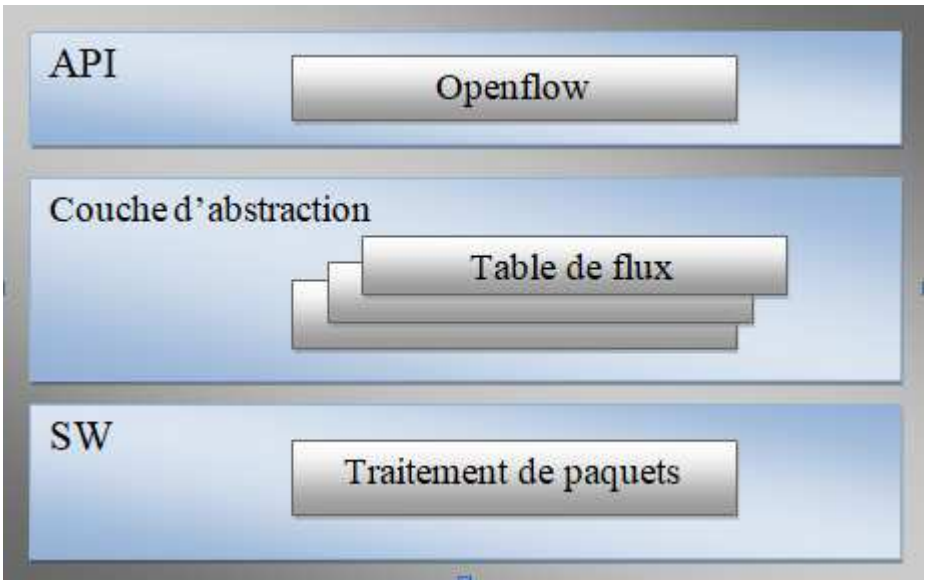

Figure 3 : Anatomie logique d'un switch SDN interprété de (Goransson et al, 2016).

Les tables de flux sont un élément fondamental du fonctionnement d'un switch SDN, elles sont composées d'un nombre d'entrées de flux, les quelles consistent en deux composantes principales :

- Champs de correspondance : Ils sont utilisés par ordre de priorités pour être comparés aux entêtes des paquets entrants, le premier champ correspondant est directement sélectionné.
- Les Actions : Ce sont les instructions qui doivent être exécutés si une correspondance entre un paquet entrant et l'entrée pour laquelle ces actions sont spécifiées.

La logique du traitement de paquets consiste en un nombre de mécanismes qui se mettent en actions en fonction du résultat de l'évaluation de l'entête du paquet et la correspondance de priorité la plus haute. Lorsqu'une correspondance est trouvée, le paquet est traité localement dans le switch, à moins qu'il soit explicitement envoyé au contrôleur. Si aucune correspondance n'est trouvée, une copie du paquet est envoyée au contrôleur qui devra décider quoi en faire, et éventuellement mettre à jour la table de flux pour l'y introduire comme entrée (Goransson et al, 2016 ; Nadeau et al, 2013).

On distingue deux types de switchs SDN :

- **Switch SDN logiciel :** C'est le moyen le plus simple pour créer un équipement SDN. Les tables de flux, les entrées et les champs de correspondance sont facilement implémentés dans les structures de données d'un software. Ce model est plus lent et moins efficace car il ne bénéficie pas d'une accélération hardware, mais présente l'avantage d'être plus flexible, plus riche dans les actions disponibles, et supporte un nombre d'entrées beaucoup plus important. Les switchs logiciels sont très présents dans les environnements virtualités et sont généralement open source. Les deux principales alternatives sont Open vSwitch (OVS) de Nicira et Indigo de Big Switch (Goransson et al, 2016).
- **Switch SDN matériel :** Ces implémentations sont plus rapides, et sont la seul possibilité pour un environnement très haut débit (100Gbs) étant sensible aux performances. Pour transcrire les différentes entrées de flux et leur composantes dans les switchs on a opté pour l'utilisation de hardware spécialisé, pour le traitement de couche 2 (MAC) on utilise les CAMs (Content-Addressable Memories) et pour la couche 3 les TCAMs (Ternary Content-addressable Memories). Ces switchs sont adaptés aux data center et au cœur du réseau, les politique de flux n'y sont pas centrés sur les utilisateurs ce qui fait que le nombre d'entrée y est moins important que dans les switchs qui sont plus prés de l'accès (Goransson et al, 2016).

3.2.2. Le contrôleur

Le rôle du plan de contrôle est de contrôler et de gérer les équipements de l'infrastructure, et de les relier avec les applications. Il est composé d'un ou de plusieurs contrôleurs et il est considéré comme système d'exploitation du réseau. Les premières versions du SDN présentent un plan de contrôle composé d'un seul contrôleur centralisé. Par la suite des architectures distribuées utilisant plusieurs contrôleurs ont été proposées pour améliorer les performances et la scalabilité du réseau. Les performances des contrôleurs SDN sont caractérisées par le débit, qui est la quantité de flux traités par seconde et la latence, qui est le temps d'installation d'une nouvelle règle (Benamrane, 2017).

Afin de pouvoir interagir avec le réseau, le contrôleur a besoin d'une vue précise de ce dernier. C'est ainsi que le concept de NIB (Network Information Base) a vu le jour. Cette NIB est construite au niveau du contrôleur et permet à ce dernier de savoir comment implémenter

chaque ordre abstrait, trouver les équipements qui doivent être reconfigurés, s'assurer de la capacité de ces derniers à implémenter une directive et les API supportées par l'équipement (Durand, 2015).

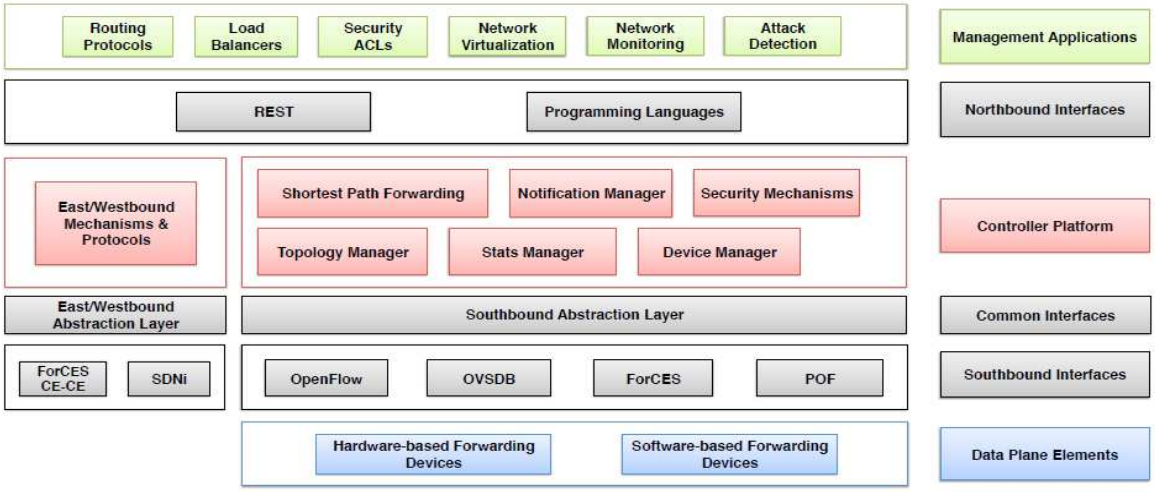

Figure 4 : Contrôleur SDN : éléments, services et interfaces (Kreutz et al, 2014)

Le contrôleur se base sur deux modes opérationnels : réactif et proactif.

- L'approche réactive fait tout transiter par le contrôleur. Lorsqu'il y a un paquet entrant sur le Switch, celui-ci est directement redirigé vers le contrôleur pour que ce dernier décide du comportement à adopter vis-à-vis du paquet. Ce model peut causer des temps de latences considérable, en fonction des ressources à disposition du contrôleur et la distance Switch-contrôleur.
- Dans l'approche proactive, le contrôleur introduit préalablement des règles dans les switchs pour qu'ils puissent traiter les paquets localement. Ici il le nombre de paquets envoyés au contrôleur diminue considérablement ce qui fait gagner en efficacité et en débit à l'architecture.

Plusieurs contrôleurs SDN sont développés, commerciales ou Open source. Ils différent par leur architecture (Centralisé ou Distribuée), leurs langages de programmation, les API qu'ils supportent, les techniques utilisés et leurs performances comme le débit et la latence.

Le tableau ci-dessous résume certains des contrôleurs les plus en vigueur actuellement (Kreutz et al, 2014 ; Project Floodlight, 2016) :

Contrôleur	Architecture	Openflow	Langage	Tolérance pannes	API Nord
Beacon	Centralisé multi-thread	V 1.0	Java	NON	Ad-hoc API
Floodlight	Centralisé multi-thread	V 1.1 1.2 1.3 1.4	Java	NON	RESTful API
ONOS	Distribué	V 1.0	Java	OUI	RESTful API
OpenDaylight	Distribué	V 1.0 1.3	Java	NON	REST/RESTCONF
Ryu	Centralisé	V 1.1 1.2 1.3 1.4 1.5	Python	NON	Ad-hoc API
NOX	Centralisé	V 1.0 1.1 1.2 1.3	C++	NON	Ad-hoc API

Tableau 1 : Caractéristiques des contrôleurs SDN

3.2.3. Les interfaces de communication

Les interfaces de communication ou API permettent au contrôleur d'interagir avec les autres couches du réseau SDN. On leur attribue la notation Nord/Sud/Est/Ouest en fonction de la position de la couche avec laquelle communique le contrôleur dans la hiérarchie de l'architecture. Pour communiquer avec une couche basse (Underlay), l'interface utilisée est une interface Sud. Inversement pour les couches hautes c'est une interface Nord. La communication avec d'autres contrôleurs se fait via les interfaces Est/Ouest.

3.2.3.1. L'interface Sud

Ce sont les interfaces (South-bound) qui permettent le processus de communication entre le contrôleur et les switchs/routeurs et autres éléments de la couche infrastructure réseau. C'est par le biais de cette interface et notamment le protocole Openflow dans le cas du standard Open SDN (Open Networking Foundation, 2012), que le contrôleur injecte les différente politique aux équipements, et récupère les informations permettant aux applications de construire une vue globale du réseau (Goransson et al, 2016).

La figure 5 illustre le type de messages transitant par l'interface sud (ex : Openflow), lors de l'instauration d'une nouvelle règle sur un switch.

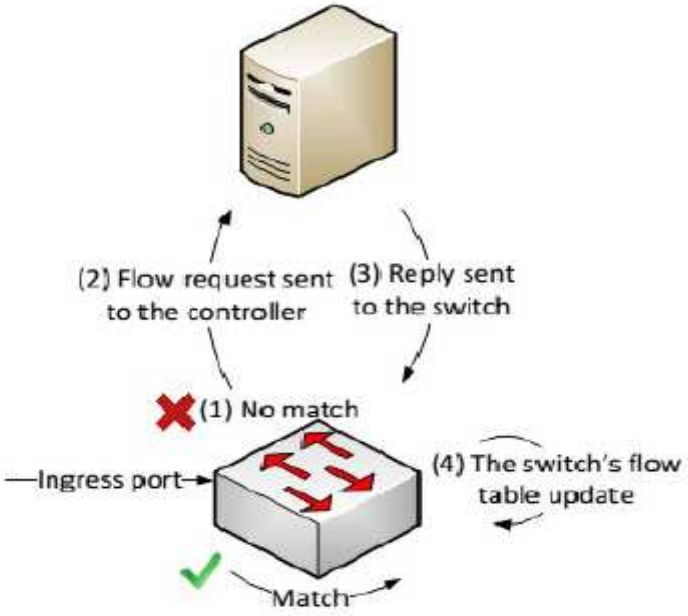

Figure 5 : Processus de communication entre le Switch et le contrôleur (Benamrane, 2017)

Hormis le standard Openflow, plusieurs protocoles faisant office d'interface sud ont été développés et sont utilisés dans certaines implémentations appelées SDN à base d'API. On peut en citer (Goransson et al, 2016) :

- Path Computation Element communication Protocol (PCEP)
- Interface to Routing System (I2RS)
- Simple Network Management Protocol (SNMP)
- NETwork CONFiguration protocol (NETCONF)
- BGP flow-spec

3.2.3.2. Interface Nord

Les interfaces Nord (North-bound) servent à programmer les éléments de la transmission en exploitant l'abstraction du réseau fourni par le plan de contrôle. En d'autres termes elles permettent la communication entre le contrôleur la couche applicative. Elles sont considérées davantage comme des API que comme protocole de programmation et de gestion de réseau. Il n'existe aucun standard intervenant entre la couche de contrôle et celle d'application. Selon l'ONF (Open Networking Foundation, 2012), plusieurs niveaux d'abstraction et différents cas d'utilisation peuvent être caractérisés, ce qui signifie qu'il peut y avoir plusieurs interfaces Nord pour servir tous les cas d'utilisation. Parmi les propositions des industriels, nous

trouvons une API basée sur REST API (REpresentational State Transfer) pour fournir une interface programmable utilisable par les applications.

3.2.3.3. Interface Est/Ouest

Ce sont des interfaces inter-contrôleurs, on les trouve dans les architectures distribués (Multi-contrôleurs). Ils permettent la communication entre contrôleurs pour synchroniser les états du réseau. Aucun standard n'est encore disponible pour ce type d'interfaces.

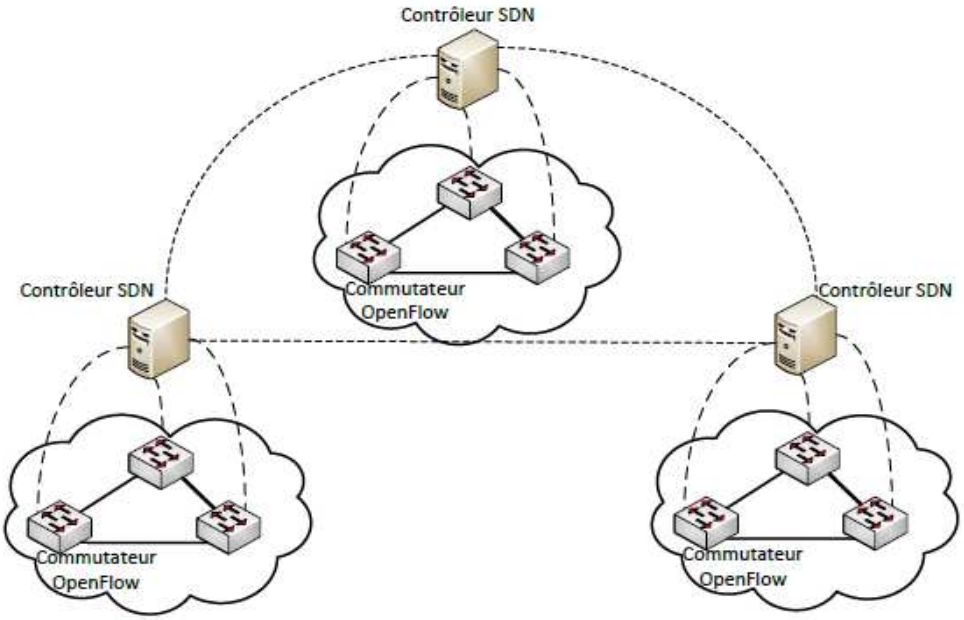

Figure 6 : Communication inter-contrôleur dans une architecture distribuée (Benamrane 2017)

3.2.4. Les applications

La partie applicative du SDN, représente toute l'ampleur du potentiel de ce paradigme. Ce sont les applications qui concrétisent tout les avantages qu'apporte le SDN et changent la façon dont est exploité le réseau. Opérant au dessus du contrôleur, ces applications sont responsables de la moindre tache accomplie par celui-ci.

Toute une panoplie d'applications a été conçue pour différentes fonctions réseau et de gestion :

- Configurer et maintenir les flux pour router les paquets par le meilleur chemin entre 2 points.
- Equilibrer la charger de trafic à travers le réseau.

- Réagir aux changements occurrents, comme la rupture d'une liaison ou l'addition d'un équipement ou d'un chemin.
- Redirection de trafic pour différentes raison de sécurité.

Les opérations du réseau ont toutes été traduites en application, allant de la découverte d'un simple switch à l'implémentation de services plus complexes. Le tout avec souvent une interface graphique permettant la gestion du contrôleur.

3.2.5. L'orchestrateur

L'orchestrateur vient répondre à la nécessité de pouvoir programmer le réseau de bout en bout. Si l'application est déployée dans un data center, les usagers peuvent être connectés en Wifi, connecté au LAN, derrière le WAN et des firewalls ou autres dispositifs réseau, ceux si sont souvent des domaines indépendants formant une chaine d'exécution, ce qui rend la tache pratiquement impossible pour le contrôleur (Durand, 2015).

Dans les architectures de fournisseur de services des opérateurs par exemple, ce n'est pas le contrôleur mais l'orchestrateur qui va offrir cette fonction de programmation de bout-en-bout. Ce dernier reçoit un ordre depuis une application et réalise ensuite la suite d'actions nécessaires pour mener à bien la tâche demandée. Pour réaliser sa mission, l'orchestrateur va pouvoir s'appuyer pour chaque élément de la chaîne sur le/les contrôleurs déployés (Durand, 2015).

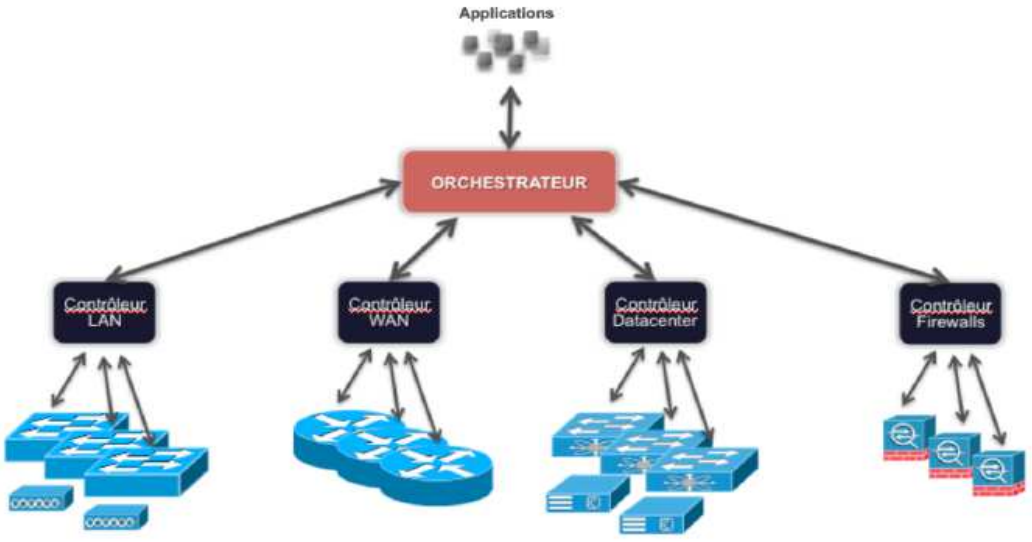

Figure 7 : Architecture SDN avec un orchestrateur

4. Openflow

L'histoire d'OpenFlow a commencé à l'Université de Stanford, par un groupe de chercheurs appelé le Clean Slate Program dont l'objectif était d'explorer comment on concevrait internet si on faisait table rase et qu'on avait 20 à 30 ans de recul. En 2008 l'Open Networking Foundation (ONF) a publié Openflow qui est un protocole réseau permettant la réalisation des architectures Software Defined Networking. Il permet d'accéder directement au plan de données des périphériques réseau (routeurs, switch), il s'agit d'une façon dynamique, centralisé et programmable d'interagir avec les différents équipements de l'infrastructure (Coker & Azodolmolky, 2017 ; Pujolle, 2014).

Ce protocole prend en compte les fonctions communes supportées par les différentes tables des switchs Ethernet et routeurs conçus par les constructeurs, ce qui permet l'utilisation des tables de flux par tous les dispositifs réseau. Openflow est utilisé comme une interface sud dans les réseaux définies par architectures SDN (Canceres, 2016).

4.1. Architecture du protocole Openflow

Le protocole Openflow définit la connexion entre un switch Openflow et un contrôleur. L'essence de ce protocole consiste d'un set de messages qui transitent entre le contrôleur et le switch dans les deux sens. Ce sont ces messages qui permettent au contrôleur de garder un contrôle sur les switchs et sur le trafic des utilisateurs (Noble, 2017 ; Kingston Smiler, 2013).

Quant à l'architecture du switch Openflow, comme nous en avons parlé dans la section précédente, celui-ci est régit par les principes des tables de flux.

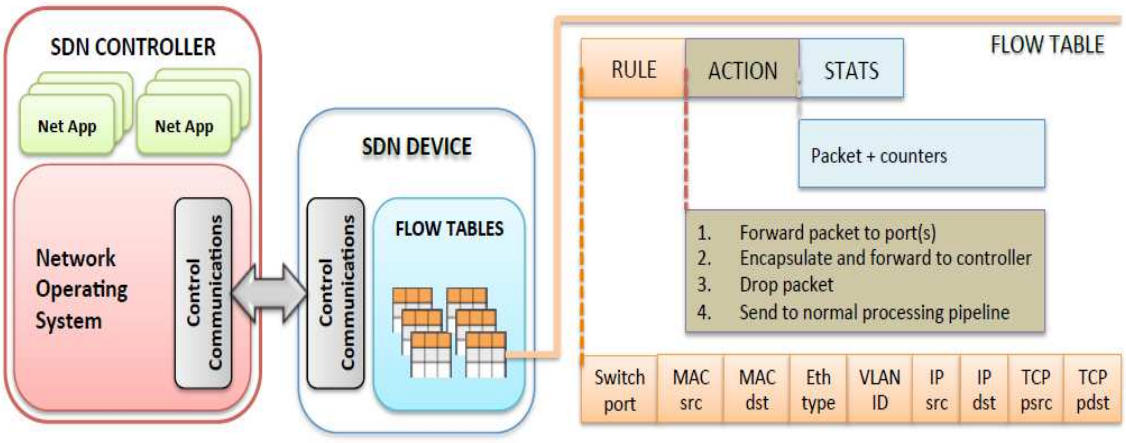

Figure 8 : Architecture Openflow (Kreutz et al, 2015)

4.1.1. Tables de Flux

Depuis la version 1.1 les commutateurs Openflow sont composés d'un pipeline de tables de flux. Celui-ci permet la gestion de l'acheminement des paquets, le mécanisme consiste à cumuler les actions de chaque table de flux pour les exécuter à la sortie du pipeline. La figure 9 ci-dessous décrit son fonctionnement.

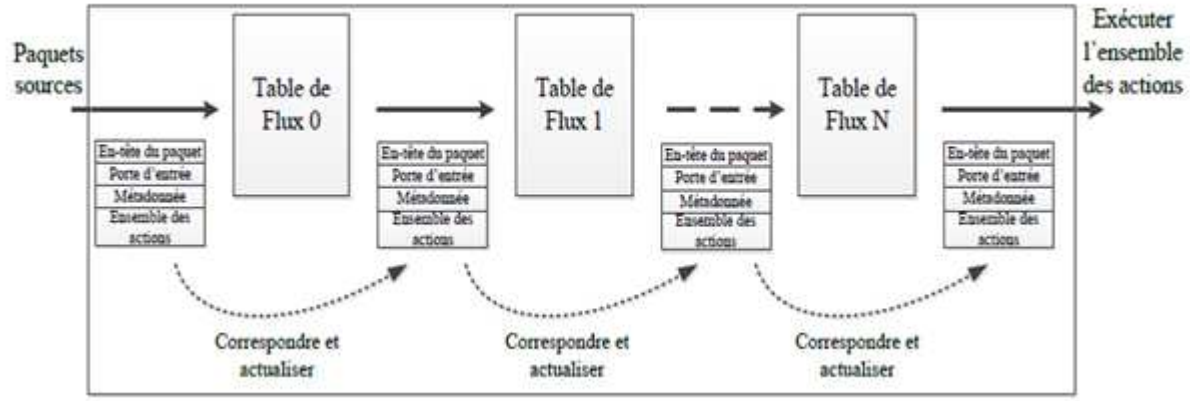

Figure 9 : Pipeline Openflow v1.1 (Benamrane, 2017)

Nous décrirons à présent les champs des entrées qui composent les tables de flux, globalement structurés comme suit.

Champ de Correspondance	Instructions	Compteurs

Figure 10 : Entrée de flux

4.1.1.1. Champs de correspondance (Match fields)

Utilisé lors de la recherche de l'entrée correspondante au paquet, Autrement dit l'identification des flux.

L'identification peut se faire à base d'une multitude d'informations contenues dans l'entête du paquet, allant de la couche 1 à la couche 4 du modèle ISO. La figure suivante montre quelques uns des champs proposés par le protocole, et depuis les versions les plus récentes d'autres possibilités d'identification ont été rajoutés comme les champs IPv6 et les étiquettes MPLS (Benamrane, 2017).

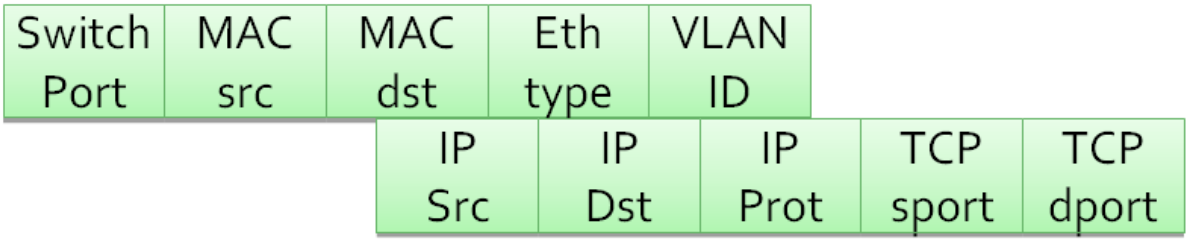

Figure 11 : Champs de correspondance Openflow v 1.0

4.1.1.2. Compteurs

Les compteurs sont utilisés pour collecter les statistiques de flux, l'enregistrement des paquets et la durée des flux et la gestion des entrées de table de flux, elles permettent entre autres de décider si une entrée de flux est active ou non. Pour chaque table, chaque flux, chaque port, des compteurs de statistiques sont maintenus (Open Networking Foundation, 2014).

4.1.1.3. Actions

À chaque entrée de la table de flux est associé un ensemble d'actions à exécuter sur les paquets, avant de les envoyer vers un port de sortie. Les actions doivent être exécutées dans le même ordre dans lequel elles sont présentées dans la table. Si une entrée ne contient aucune action, le paquet qui lui correspond est supprimé. Un switch OpenFlow peut ne pas supporter tous les types d'actions. Cependant il doit supporter les actions requises pour le fonctionnement minimal d'OpenFlow (Idoudi, 2014 ; Coker et al, 2017) :

- Transférer le paquet à travers un ou plusieurs ports de sortie vers le nœud suivant dans le réseau.
- Encapsuler et envoyer le paquet vers un contrôleur à travers le canal sécurisé. Cette action est principalement utilisée lorsqu'un paquet d'un flux est reçu pour la première fois. Le contrôleur peut décider si une nouvelle entrée doit être ajoutée dans la table, ou si le paquet doit être supprimé.
- Supprimer tous les paquets appartenant à un flux est aussi une action qui peut être utilisée pour une raison de sécurité pour mettre fin à une attaque par exemple.

Les actions optionnelles portent sur la modification de certains champs dans le paquet (Modification du VLAN ID, Réécriture de l'adresse MAC, recalcule du TTL ... etc.).

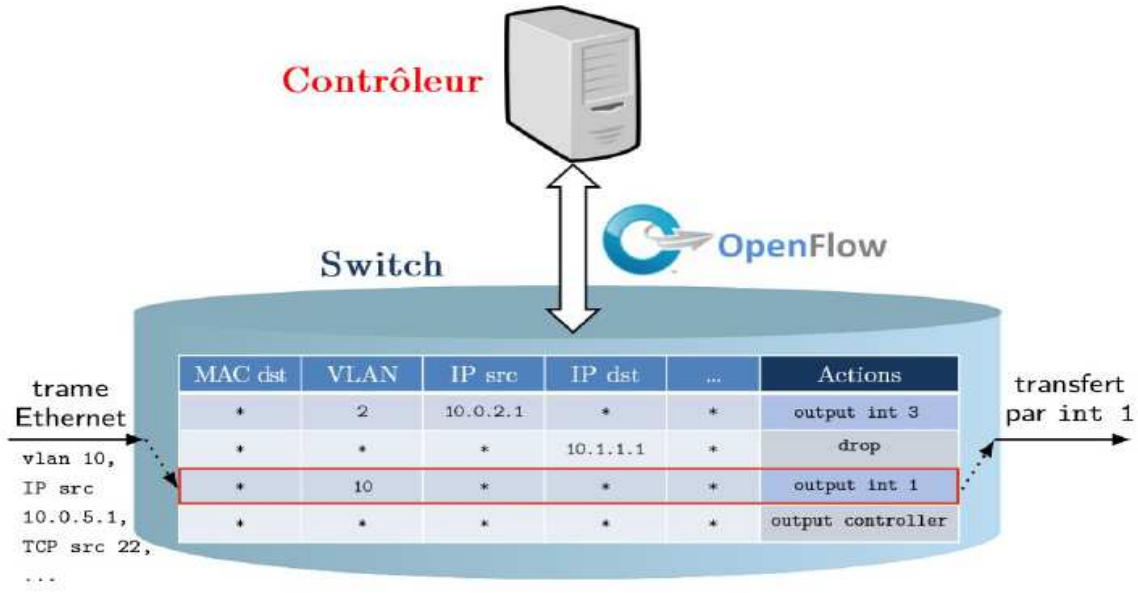

Figure 12 : Schématisation du modèle de flux (Tury, 2014)

4.2.Les messages Openflow

Tous les messages sont initiés par l'entête OpenFlow qui définit la longueur, l'identificateur de transaction et le type de message. Connaissant l'adresse IP du Contrôleur, le commutateur initie une connexion TLS avec le commutateur, ces message sont sécurisés et assurés par une connexion TLS sur TCP. On trouve trois catégories de message (symétrique, asymétrique, et Contrôleur-commutateur). (Open Networking Foundation, 2014)

4.2.1. Messages Contrôleur-Commutateur

Ces messages servent a vérifier l'état du switch et gérer et vérifier son état et peuvent demander des réponses de la part du switch (Kingston Smiler, 2013). Ces messages sont initiés par le Contrôleur.

- **Features** : le contrôleur peut demander l'identité et les capacités d'un switch cela se passe en envoyant une requête.
- **Modify-state** : ce type de message permet de gérer l'état des switchs ; ils permettent de modifier ou ajouter, effacer les entrées dans les tables openflow.
- **Configuration** : le contrôleur peut définir les paramètres de configuration du switch. Ce dernier répond aux requetés envoyés par le contrôleur.

- **Read-state** : le contrôleur utilise ces messages pour collecter différentes informations du switch, tel que les statistiques, les capacités et sa configuration actuelle.
- **Packet-out** : ces messages sont utilisés pour le transfert de paquets reçus par les messages Packet-in. Le message doit contenir un paquet entier ou l'ID du buffer faisant référence à un paquet stocké dans le switch. Si la liste d'actions du message est vide le paquet sera détruit.
- **Barrier** : utilisé par le contrôleur pour recevoir des notifications de l'opération terminée.

4.2.2. Messages asynchrones

Les messages asynchrones sont envoyés par le switch vers le contrôleur pour indiquer un changement d'état ou l'arrivée d'un paquet (Kingston Smiler, 2013). Ces messages sont :

- **Packet-in** : avec ce message le switch transfert le contrôle du paquet au contrôleur.
- **Flow-removed** : le switch informe le contrôleur de la suppression d'une entrée dans la table du flux.
- **Port-satus** : avec ce message le switch informe le contrôleur d'un changement sur un port.
- **Role-status** : quand un nouveau contrôleur est aux commandes, le switch envoi un Role-status a son contrôleur.

4.2.3. Messages Symétriques

Les messages symétriques sont envoyés sans aucune sollicitation ni du switch ni du contrôleur. Ces messages sont :

- **Hello** : envoyer pour vérifier la connectivité entre le contrôleur et le switch.
- **Echo** : une fois la connexion établie ces messages sont échangés entre le contrôleur et le switch. Chaque message ECHO_REQUEST doit être acquitté par un ECHO_REPLY.
- **Error** : utilisé par le contrôleur et le switch pour signaler un problème de connexion.

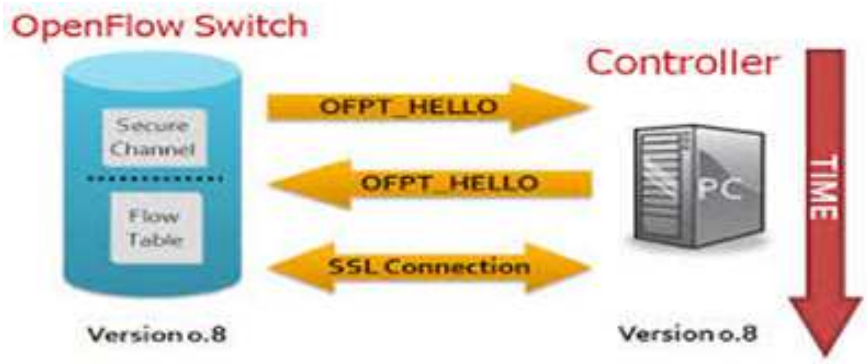

Figure 13 : Message Symétrique « Hello »

4.3.Les spécifications Openflow

Nous résumons les spécifications d'Openflow dans le tableau suivant, avec description des aditions apportées par chaque version.

Version Openflow	Additions apportées par la spécification
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, DS et ECN pour la couche 3
v 1.1	- Ajout du pipeline, de la table de groupe et des métadonnées - Ajout des champs d'identification MPLS
v 1.2	- Modèle OXM (Openflow Extensible Match) 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 assignés a 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.
v 1.5	- Notion des tables de sortie qui permet de traiter les paquets sur le port de sortie en même que sur les ports d'entrée.

Tableau 2 : Tableau de comparaison des différentes versions Openflow (Noble, 2017)

5. Cas d'utilisations du SDN

Le marché du SDN est en constante évolution depuis son apparition sur la scène mondiale en 2011, et ceci grâce aux nombreux avantages apportés par la technologie dans les différents domaines. Ce qui fait que les cas d'utilisations sont très variés et s'étendent sur toutes les parties du réseau.

5.1.QoS sur internet

Internet a toujours été une architecture qui ne garantie pas la stabilité, or les services nouvelle génération de streaming vidéo, VoD ou autre Visioconférence sont très peu tolérant des délais et erreurs de transmissions et nécessitent donc un réseau stable pour l'acheminement des paquets. En se basant sur la vue centralisée du réseau offerte par SDN, on peut sélectionner selon le débit, des chemins différents pour les divers flux de trafic. C'est à partir de là qu'est né le concept du VSDN (Video over SDN), une architecture qui détermine le chemin optimale en se basant sur la vue globale qu'offre la centralisation de contrôle (Benamrane, 2017).

5.2.Data center

Oracle SDN est un exemple de système qui fournit un réseau virtualisé du data center. Ce système relie dynamiquement des machines virtuelles avec les serveurs de réseau. À l'aide de l'interface du gestionnaire d'Oracle Fabric, la configuration et la surveillance de réseau virtuel est possible en tout lieu, et le déploiement de nouveaux services comme le pare-feu, l'équilibrage de charge et le routage devient à la demande. Selon Oracle, leur proposition augmente de 30 fois les performances des applications, et peut faire gagner un débit de serveur à serveur de 80 Gb/s (Oracle, 2015).

5.3.Réseaux mobiles :

Avec l'avènement imminent de la 5G, les opérateurs se doivent d'optimiser leur infrastructure de sorte à pouvoir gérer des masses de données de plus en plus importantes tout en assurant la qualité de service qui accompagne une telle technologie. Le SDN est une des pièces maitresses qui permettront aux opérateurs d'exploiter le potentiel de l'infrastructure et permettre d'optimiser les débits au maximum tout en gardant une importante visibilité sur le comportement de leurs équipements réseau. Une architecture appelée CSDN (Cellular SDN)

a été proposée (Bradai et al, 2015), c'est une approche qui utilise les concepts SDN pour une exploitation dynamique et centralisé des ressources, elle collecte les données des utilisateurs et les conditions du réseau, et les fait remonter au contrôleur afin d'optimiser la consommation d'énergie, l'utilisation des ressources et la personnalisation des services aux utilisateurs.

5.4.Sécurité

La nature dynamique de SDN a été exploitée par un groupe de chercheurs (Jafarian, Al-Shaer & Qi Duan 2012) qui ont réussi à réduire de 99% le risque de récupération d'information par un analyseur de paquet externe. Leur approche s'est basé sur une mutation aléatoire et assez fréquente de l'adresse IP de l'hôte

Des propositions ont été faites et notamment des mécanismes pour contrer les attaques de déni de service (DDoS), Dans l'architecture suggérée, deux éléments clés ont été exploités : Les IDS pour la détection d'intrusions intra-LAN et les switchs Openflow pour l'isolation dynamique des équipements infectés (Juba, Huang & Kawagoe, 2013).

5.5.Réseau de campus et d'entreprise

Les dernières années ont été marqué par un changement radical de la perception des équipements de travaille en entreprise et dans les campus universitaires. En effet la mode BYOD (Bring Your Own Device), ou chaque employée et étudiant utilise son propre matériel sur le campus, ce qui nous pousse a 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 a gérer manuellement et par les opérations manuelles qui pourrait permettre de connecter tout les utilisateurs à internet par exemple. Le SDN dont l'un des avantages est l'automatisation et la programmation des réseaux nous 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 (Application aware routing). Le nombre d'applications tournant sur les équipements des utilisateurs est considérable (Réseaux sociaux, streaming audio et vidéo …), 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.

6. Discussion

L'étude bibliographique du paradigme SDN, nous a démontré la nécessité du déploiement de celui-ci. Les avantages apportés par ce dernier transcendent les règles qui régissent les réseaux traditionnels et permettent d'anticiper les progrès du domaine informatique et autres percées dans le multimédia qui deviennent de plus en plus fréquentes et gourmande en ressources.

Dans la suite de notre mémoire nous allons investiguer les possibilités de mise en œuvre d'une solution SDN à partir d'un réseau classique.

Chapitre 2:
Design et implémentation d'une solution SDN

1. Préambule

Après nous être familiarisés avec les fondements du SDN, nous allons à présent nous intéresser à son implémentation dans le cas d'un réseau d'entreprise. Le but étant de comprendre ce qu'implique ce paradigme comme changements sur le design et l'administration d'un réseau, Nous allons entre autres nous pencher sur les méthodes d'interaction avec les éléments qui interviendront dans la solution proposée.

2. Infrastructure

L'avènement du SDN a apporté son lot de changements à la couche physique, outre la simplification du matériel de transmission, le design de la topologie est complètement transformé de façon à multiplier les routes, une telle approche permet aux applications se trouvant à l'interface nord du contrôleur de constamment comparer les différents chemins pour optimiser la transmission et éviter toute forme de congestion. Contrairement au protocole STP qui bloque les routes redondantes pour éviter les boucles, le SDN adopte le Multipathing et 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 (Jo, 2014), les auteurs présentent un algorithme exploitant les capacités du SDN dans un data center pour permettre une architecture Multipath à charge distribuée, leurs résultats ont démontré des performances largement supérieures à celles connues auparavant.

Ici nous allons implémenter une architecture qui exploitera les fonctionnalités les plus en vigueur du paradigme, offrant la possibilité d'une administration optimale du réseau.

2.1. Présentation du scénario

Dans le cadre de ce mémoire nous allons déployer un réseau SDN pour le cas d'une entreprise. Faute de moyens, le déploiement ne se fera pas avec des switchs matériels, nous exploiterons la nature portable et orientée logiciel du SDN pour implémenter l'infrastructure dans un environnement virtuel. Cette option est d'autant plus intéressante pour notre projet vu la richesse des switchs virtuels en possibilités (correspondances/actions) comparés à leurs équivalents hardware. Un second scénario sera ensuite développé, celui-ci proposera la configuration d'un routage conscient des applications du réseau (Application Aware Routing).

La couche infrastructure se composera de commutateurs Open vSwitch opérants sur une plateforme de l'émulateur mininet.

2.2. Open vSwitch

Open vSwitch est un commutateur logiciel multicouches conçu par Nicira et supporté par la fondation Linux (The Linux Foundation, s.d). Son implémentation principale étant dans les réseaux virtualisé et les data center, de par sa mobilité et sa réaction dynamique aux changements d'états, elle s'étend aux architectures physiques ou Open vSwitch peut être installé sur matériel open source.

2.2.1. Anatomie d'OVS

Le switch est principalement composé des éléments suivants (Open vSwitch, 2017):

- **ovsdb-server** : est un serveur base de donnée 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 base de données.
- **openvswitch.ko** (Module Kernel) : Le Kernel est le noyau du système d'exploitation il consiste en la partie qui s'occupe réellement de la commutation et du tunneling.

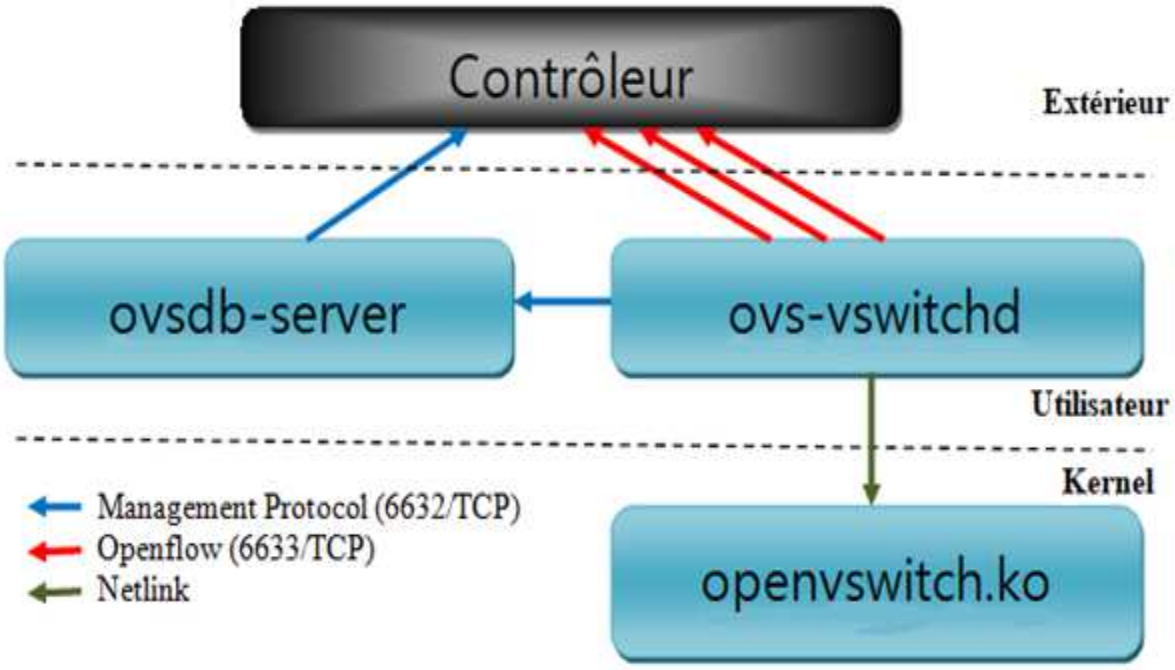

Figure 14 : Composants d'OVS

La communication de l'utilisateur avec ces composantes constitue une étape importante dans l'implantation de l'Open vSwitch. 3 outils d'espace utilisateur sont a disposition pour configurer et surveiller l'état du switch **ovs-vsctl**, **ovs-ofctl** et **ovs-dpctl**, chacun de ces outils offre une panoplie de commandes permettant d'interagir avec le switch.

2.2.2. 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 OpenFlow.

- La commande suivante `ovs-vsctl add-br br0` permet de créer et un pont et par défaut une interface du même nom (br0).
- En ce qui concerne la création de ports/interfaces il faut entrer l'instruction à soumettre est `ovs-vsctl add-port br0 eth1`. On peut éventuellement y attribuer un Vlan en ajoutant l'option `tag=10` ou 10 est le numéro du vlan
- Pour connaitre l'état global du switch `ovs-vsctl show`, l'affichage du résultat est illustré dans la figure 17.

```
918037ec-b307-45d7-a75a-f1ac4337d135
    Bridge "br0"
        Port "br0"
            Interface "br0"
                type: internal
        Port "eth1"
            tag: 10
            Interface "eth1"
    ovs_version: "2.0.2"
```

Figure 15 : Affichage de la commande show d'OVS

Pour l'architecture SDN, nous devons ensuite connecter le switch à un contrôleur, pour ce faire on introduit `ovs-vsctl set-controller br0 tcp:192.168.1.10:6633` ou

« 192.168.1.10 » est l'adresse IP du contrôleur et 6633 le port d'écoute du protocole OpenFlow. A partir de là pour communiquer avec OpenFlow l'outil privilégié est ovs-ofctl.

- `ovs-ofctl -O OpenFlow 13 dump-flows br0` Affichera les différents flux installés sur le switch.
- L'outil permet aussi d'ajouter des flux directement sur le switch sans passer par un contrôleur avec la commande **add-flow**. Dans l'exemple présent on ajoute un simple flux `ovs-ofctl -O OpenFlow13 add-flow br0 in_port=1,actions=output:2` ou tous les paquets reçus dans le port 1 sont renvoyés par le port 2. On peut vérifier que le flux a été bien ajouté en repassant par la commande dump-flows.

2.3.Mininet

Préalablement à tout déploiement, la conception du prototype est un processus qui permet d'anticiper le comportement du réseau. Dans ce contexte Mininet vient offrir une plateforme d'expérimentation et de développement avec la possibilité de créer un réseau virtuel réaliste, extensible et interactif (Lantz, Heller et McKeown, 2010). Un terrain fertile pour l'innovation dans le domaine du SDN, grâce à son interface programmable python (Python API) et sa ligne de commande (CLI), autrement dit une route fluide vers l'implémentation sur matériel (Mininet, s.d).

2.3.1. La ligne de commande Mininet :

Grâce à la commande `mn` et la multitude d'options offertes par l'émulateur, il est possible de réaliser maintes formes de topologies et de réseaux virtuels (Mininet, s.d) :

- L'option `--topo` permet de créer 3 types de topologies de base : **tree** (Arbre), **linear** (linéaire) et **single** (topologie centralisée d'un seul switch).
- Pour connecter notre topologie à un contrôleur l'option adéquate est `--controller` , dans l'exemple d'un contrôleur distant on ajoute `=remote,ip=192.168.1.10` .
- D'autres options sont disponibles sur Mininet pour n'en citer que certaines : `--switch` offre la possibilité de choisir le modèle de switch parmi ceux supportés par l'émulateur, et `--mac` qui simplifie l'adresse mac des nœuds réseau en les faisant correspondre à leur identifiant (exemple : `HWaddr 00:00:00:00:00:01`). L'option **–x** ouvre un terminal xterm pour chaque élément du réseau généré, et `--link tc` pour pouvoir personnaliser la bande passante et la latence des liaisons.

Après avoir inséré la commande mn suivie des options voulues et créé la topologie nous accédons à la ligne de commande de Mininet identifiable par l'expression `mininet>` au début de chaque ligne (Hors CLI les commandes sont précédées du symbole `$` comme toute plateforme sous linux). Les commandes principales de la CLI sont :

- `mininet> net` Affiche toute les liaisons du réseau, `mininet> nodes` qui cite tout les nœuds réseau et `mininet> dump` qui affiche les informations liées aux nœuds du réseau.
- Un simple test de connectivité peut se faire grâce au Ping et la commande `mininet> iperf` permet de mesurer la bande passante entre deux hôtes.
- Un terminal xterm peut être ouvert pour chaque hôte en utilisant `mininet> h1 xterm` ou h1 est le nom de l'hôte. On peut ainsi exécuter tout type de programme sur les différents hôtes (Tout ce qui est disponible sous linux) et ainsi tester le réseau à l'aide d'outils identiques à ceux de l'installation physique.
- Pour ce qui est la connectique avec OpenFlow `mininet> dpctl dump-flows` nous permet d'afficher les différents flux installés sur les nœuds à partir de la ligne de commande.

2.3.2. Topologies personnalisées via l'API Mininet:

L'émulateur Mininet fournit une interface de programmation Python, l'une de ces utilisations consiste en la création de topologies customisées en y définissant toutes les caractéristiques voulues des éléments du réseau, une approche pareille nous offre l'avantage d'exploiter le temps et les ressources en main de la meilleure façon possible.

Dans la figure ci-dessous on illustre avec un exemple la structure générale d'un script python comprenant les informations du réseau voulu. Dans le cas présent nous avons généré 3 switchs, 1 passerelle et 2 hôtes auxquelles nous avons alloué différents identifiants.

Le fichier **.py** doit être sauvegardé dans le répertoire `mininet/custom/` , de plus pour pouvoir appeler la topologie par l'option `--custom` la dernière ligne du script est primordiale, en gros nous y avons définis la topologie par le nom « exemple ». La commande complète pour générer cette topologie devra comporter le chemin d'accès du script et le nom de la topologie en question `sudo mn --custom mininet/custom/test.py --topo=exemple`

```
#!/usr/bin/python

from mininet.topo import Topo
from mininet.net import Mininet
from mininet.node import Node , Controller, RemoteController, OVSSwitch
from mininet.log import setLogLevel, info
from mininet.cli import CLI
from mininet.util import irange
from mininet.link import TCLink

class NetworkTopo( Topo ):
    "Exemple de topologie personnalisee Mininet"
    def build( self ):
        h1 = self.addHost( 'h1', ip='10.10.2.1/24', defaultRoute='via 10.10.2.254' )
        h2 = self.addHost( 'h2', ip='10.10.2.2/24', defaultRoute='via 10.10.2.254' )
        g1 = self.addHost( 'g1', ip='10.10.2.254/24')

        s1 = self.addSwitch( 's1', dpid='0000000000000001',protocols='OpenFlow13' )
        s2 = self.addSwitch( 's2', dpid='0000000000000002',protocols='OpenFlow13' )
        s3 = self.addSwitch( 's3', dpid='0000000000000003',protocols='OpenFlow13' )
        #core
        self.addLink ( s3, s1 )
        self.addLink ( s3, s2 )
        #acccess
        self.addLink( s1, h1 )
        self.addLink( s2, h2 )
        self.addLink( s3, g1 )
def run():
    topo = NetworkTopo()
    net = Mininet( topo=topo )
    net.start()

    CLI( net )
    net.stop()

if __name__ == '__main__':
    setLogLevel( 'info' )
    run()
topos = { 'exemple': NetworkTopo }
```

Figure 16 : Exemple du script Python d'une topologie personnalisée

3. Le contrôleur SDN Opendaylight

Opendaylight est un contrôleur basé sur JAVA, il est considéré comme un Framework facilitant l'exploitation des réseaux définis à base de logiciel (SDN), une formule complète et prête à être déployée (Toghraee, 2017).

3.1. Présentation du projet Opendaylight :

OpenDaylight fut créé en avril 2013 dans le but de développer des solutions SDN dans un cadre open source dirigé par la communauté et soutenu par le secteur industriel. Le consortium OpenDaylight, dont la fondation Linux (The Linux Foundation, 2018) est l'un des membres fondateurs, compte désormais environ 40 membres, pour beaucoup issus de la communauté du Networking, qui contribuent au programme de diverses manières (ressources

financières, développement, logiciels, etc.). L'objectif consiste à accélérer le développement d'une robuste plateforme SDN commune, utilisable par tous (OpenDaylight Project, s.d). La figure 19 montre l'ampleur du projet à travers ses membres en fonction de leur implication financière ou autre.

Figure 17 : Membres du projet ODL (The Linux Foundation, 2018)

L'approche traditionnelle communément adoptée par les comités de normalisation n'est adaptée qu'aux projets de faible ampleur et clairement définis. Pour les concepts complexes tel que le SDN, il est important d'adopter une approche pragmatique et de valider le concept préalablement, avant d'établir des normes autour de ce qui a été développé, ou d'utiliser des normes apparues de facto. Dans le cadre de son approche pragmatique, le consortium OpenDaylight travaille sur plusieurs démonstrations de faisabilité axées sur la résilience et l'ingénierie du trafic. Ce travail implique le chaînage de services, la fédération de contrôleurs SDN a mis en œuvre de nouvelles options de virtualisation de réseau et l'intégration d'un plus grand nombre de fonctionnalités des couches 4 à 7. Citrix est au cœur de cette initiative et s'occupe plus particulièrement des fonctionnalités des couches 4 à 7 et de chaînage des services. Elles constituent un composant essentiel de l'intégration des services ADC au SDN.

3.2.Distributions d'ODL

Depuis la première version du contrôleur sortie en 2014, 7 autres versions ont suivies chacune avec son lot d'améliorations, le tableau suivant représente une liste non exhaustives des différentes versions du d'OpenDaylight (OpenDaylight Project, 2018).

Nom de la version	**Date de** sortie
HYDROGEN	Février 2014
HELIUM	Octobre 2014
LITHIUM	Juin 2015
BERYLIUM	Février 2016
BORON	Novembre 2016
CARBON	Juin 2017
NITROGEN	Septembre 2017
OXYGEN	Mars 2018

Tableau 3 : Versions d'OpenDaylight

Lithium est la version que nous utiliserons pour projet, sa plateforme de services est développée sur Apache Karaf, ce qui permet la facilité du déploiement et la gestion dynamique des différents modules. A partir de la base du contrôleur l'utilisateur peut installer les différentes fonctions voulues (Apache, s.d). Cette version est surtout axée sur la programmablilitée des réseaux (Badotra et Singh, 2017).

3.3.Architecture Opendaylight

La structure d'OpenDaylight s'inspire de la nature de l'environnement SDN, la communication vers l'extérieur est basée sur des API et le fonctionnement du contrôleur en interne est géré par la plateforme OSGi (Open Service Gateway initiative) montée sur un conteneur Karaf, qui implémente un modèle de composants complètement dynamique. Autrement dit les l'OSGi gère les applications internes alors que l'api gère les applications externes (Hemid, 2017).

La couche d'abstraction de service (SAL) quant à elle communique les différents services générés par la plateforme aux différents équipements réseau, indépendamment du protocole sous-jacent. La SAL expose l'infrastructure aux applications situés au nord de celle-ci.

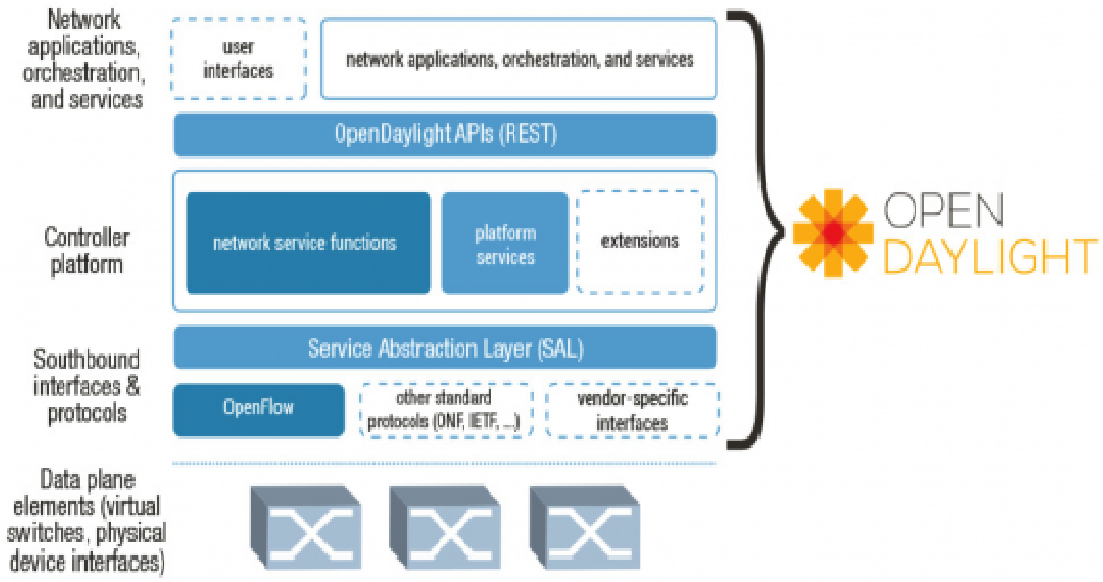

Figure 18 : Architecture simplifiée d'ODL (Meyer, 2013)

3.3.1. Couche d'abstraction de service (SAL)

Il existe deux approches différentes du SAL, chacune se démarquant par un comportement particulier vis-à-vis de la topologie et des composants du contrôleur (Hemid, 2017) :

AD-SAL (API Driven) : ou déterminé par API est un concept qui dépend des événements du réseau, le réseau y est programmé de manière réactive et les applications sont implémentées à l'intérieur du contrôleur sous forme de répertoires OSGi (Toghraee, 2017).

MD-SAL (Model Driven) : ou déterminé par modèles est l'approche proactive du SAL. Les applications sont programmées en dehors du contrôleur et communiquent avec celui-ci à travers une REST API commune à tous les modules. Les modèles en question peuvent être stockés dans le data store et peut supporter n'importe quel type d'équipement ou de service. Ce sera l'approche adoptée pour le projet et nous allons nous y intéresser par la suite à travers les différentes méthodes d'exploitation de ces modèles et de l'interface RESTCONF.

3.4.Distribution Karaf du contrôleur ODL

Cette distribution est la plus adaptée à l'approche MD-SAL, elle est facile à mettre en place, à implémenter et à développer. Après l'installation du contrôleur sur la machine Linux (voir Annexe 1), il suffit d'accéder au répertoire ou a été installé ce dernier et introduire la commande `./bin/karaf` pour pouvoir démarrer le serveur ODL. A partir de là l'utilisateur est confronté à la ligne de commande du contrôleur.

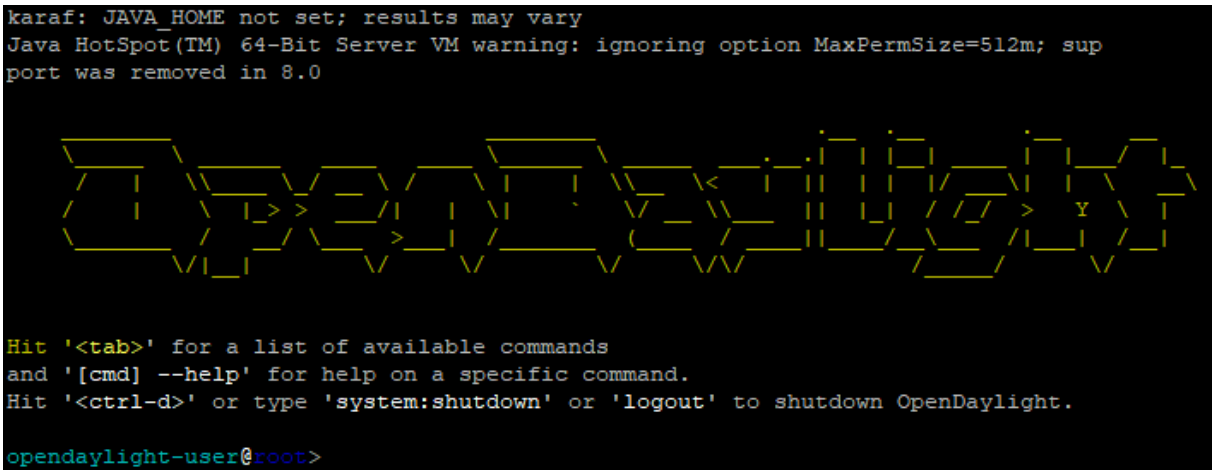

Figure 19 : Distribution Karaf d'OpenDaylight

Le serveur étant désormais fonctionnel, il faut installer certaines fonctionnalités essentielles pour le fonctionnement d'ODL dont l'interface graphique qui peut être accessible à partir de n'importe quel navigateur web. La commande adéquate est `opendaylight-user@root>feature:install odl-dlux-all`.

L'url qui renvoie vers l'interface graphique est <IP du serveur ODL> :8181/index.html. Après identification (voir Annexe 1), on accède à l'espace utilisateur qui permet d'avoir une vue globale sur les différentes topologies connectées au contrôleur.

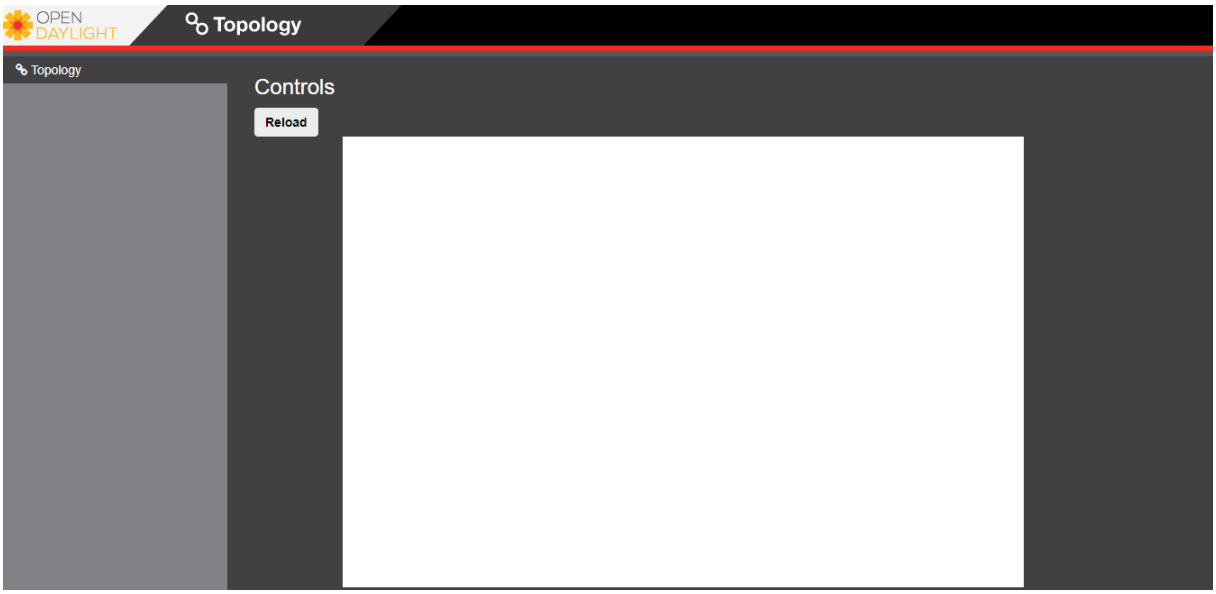

Figure 20 : UI d'OpenDaylight

Ici la zone blanche qui devrait afficher la topologie est vide, vu qu'aucun équipement n'est connecté au contrôleur. Toutes les fonctionnalités liées à l'interface sud (Openflow) devront d'abord être installés (voir Annexe 1). (Azodomolky, 2013).

Une fois ces tâches accomplies, toute topologie générée sur Mininet apparaitra graphiquement sur le contrôleur. Dans un premier temps seul les switchs apparaissent, pour ce qui est des hôtes il suffit de faire circuler un flux de données (un ping) entre eux pour qu'ils soient détectés par le contrôleur. Exemple d'une topologie linéaire : En entrant la commande `sudo mn --controller=remote,ip=192.168.1.4 --topo=linear,2 --mac` le résultat est illustré dans la figure ci-dessous.

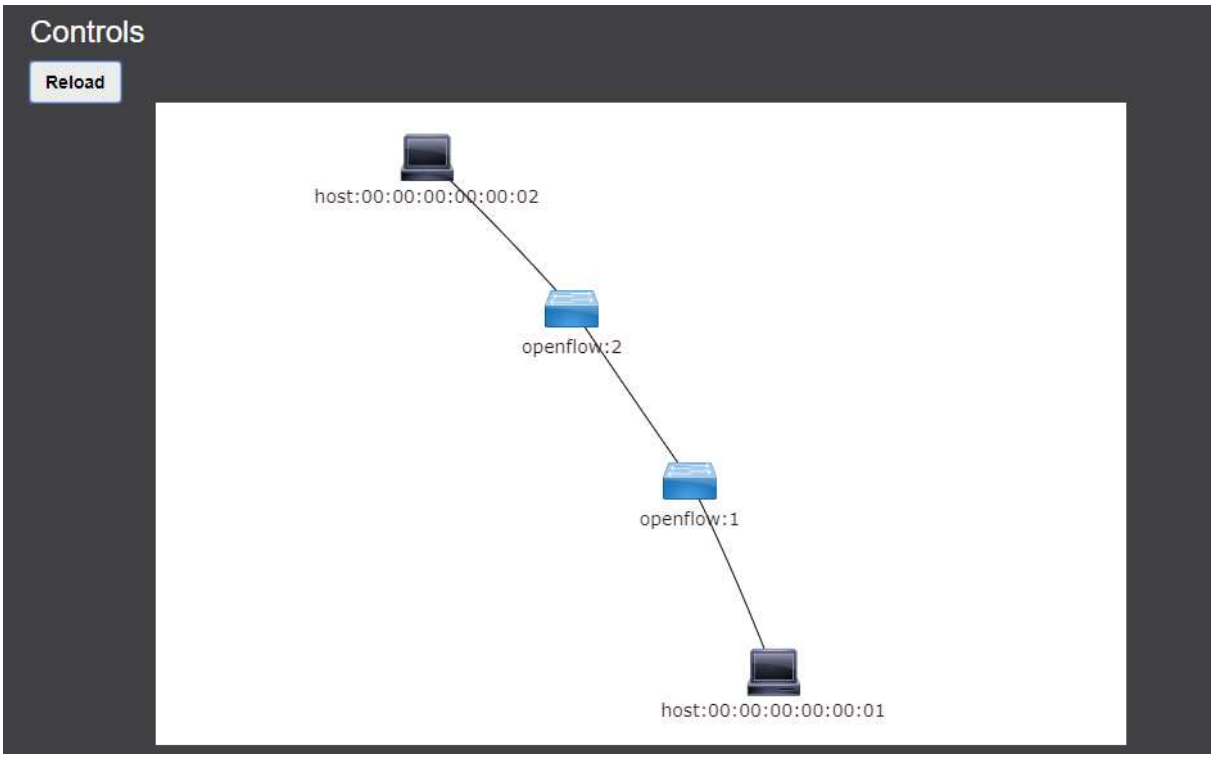

Figure 21 : Vue globale d'une topologie sur l'UI ODL

3.4.1. RESTCONF

La configuration et la programmation du réseau peut se faire directement en communiquant avec l'API RESTCONF. Pour accéder à l'explorateur REST à partir d'un navigateur et y poster des requêtes il faut installer les fonctionnalités suivantes :

```
opendaylight-user@root>feature:install odl-mdsal-all odl-restconf-all
```

Ensuite introduire l'url <AdresseIP> :8181/apidoc/explorer/index.html, qui nous réorientera vers la documentation complète de REST et toutes les interactions possibles avec l'API.

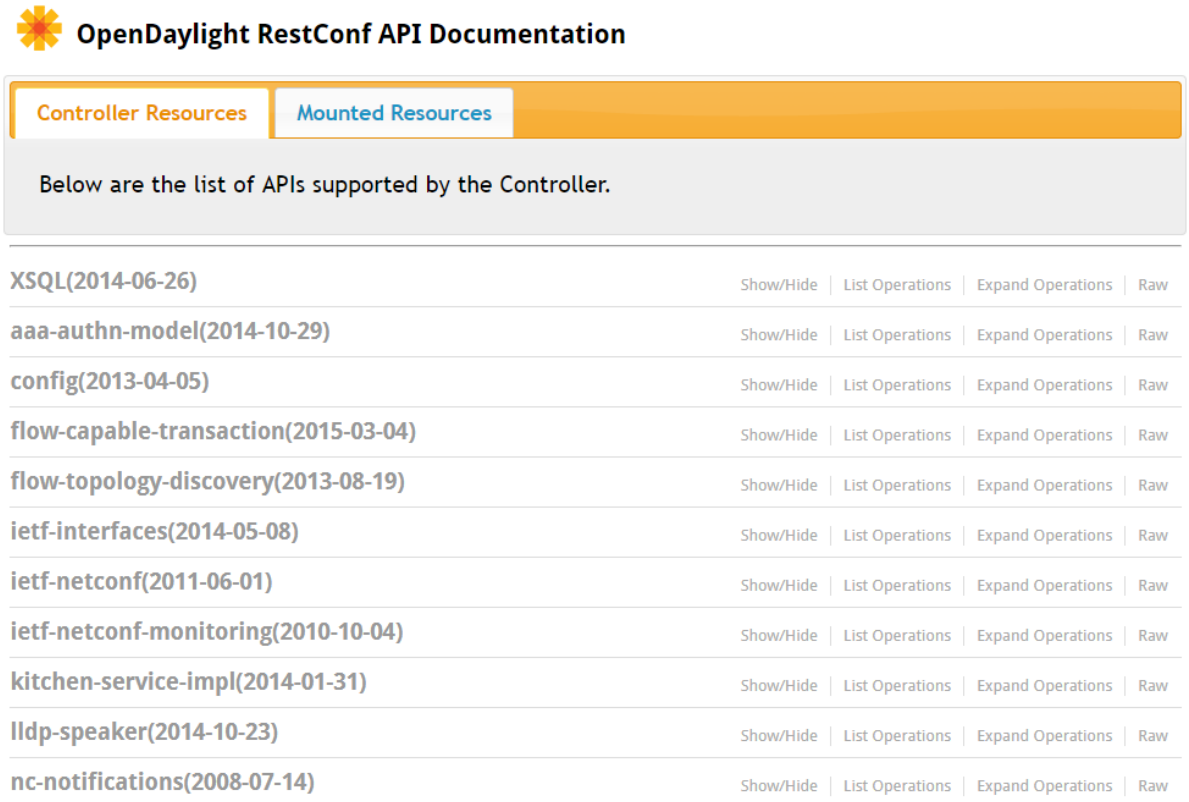

Figure 22 : Documentation du REST API

Chaque élément de cette liste dispose d'une panoplie d'instructions qui peut être explorée est utilisé par le biais de ce même explorateur.

4 types de requêtes sont supportés par l'API :

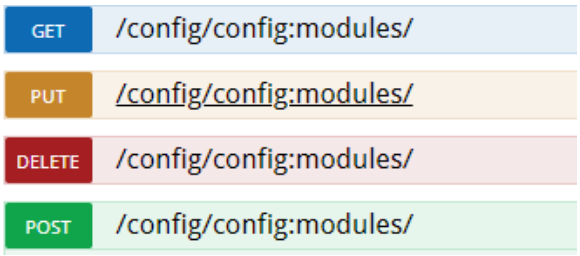

Figure 23 : Types de requêtes REST API

3.4.2. POSTMAN

Il est possible d'utiliser l'interface REST à partir d'un explorateur, mais on a trouvé plus pratique de l'exploiter avec l'outil POSTMAN disponible en extension du navigateur Google Chrome. Il offre une interface graphique intuitive et permet de sauvegarder les scripts dans des collections (Eseye, 2017). Voici une vue d'ensemble de cet outil dans la figure 26.

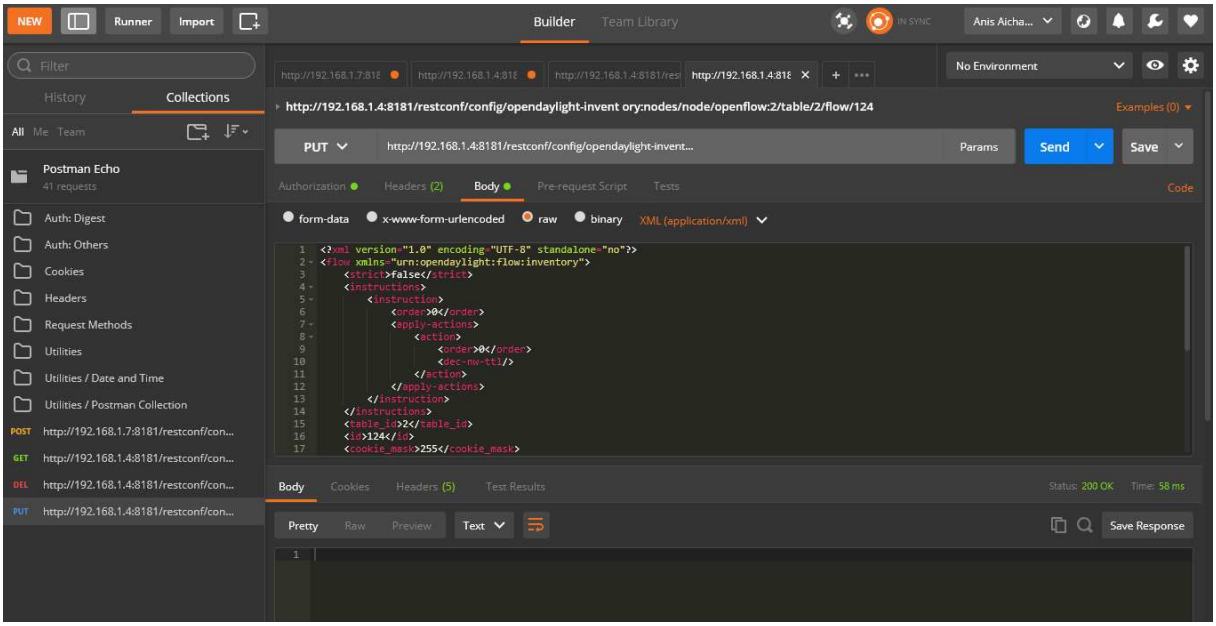

Figure 24 : Vue d'ensemble de POSTMAN

Pour interagir avec n'importe quelle interface REST 3 éléments doivent être fournis :

- **Les en têtes :** Pour déclarer le langage du script introduit (json ou xml) et pour l'authentification.
- **L'Adresse :** Celle-ci dépend de l'opération voulue (POST, PUT, DELETE, GET), de l'identifiant de l'équipement et du flux.
- **Le Corps du script :** Pour les requêtes POST et PUT on doit introduire les données sous forme de code.

Aussi après chaque requête nous recevrons une réponse HTTP. Un code 200 veut dire que la requête a été effectuée avec succès, 400 et 500 parcontre décrivent une erreur qui sera spécifié au pied de la page.

A présent nous allons envoyer une requête PUT avec l'en tête suivant :

	Key	Value
☑	Content-Type	application/xml
☑	Authorization	Basic YWRtaW46YWRtaW4=
☑	Accept	application/json

Figure 25 : En tête d'une requête PUT

Ensuite nous allons introduire le code suivant écrit en XML, qui consiste en une simple opération de firewalling de couche 2, à l'adresse *http://192.168.1.4:8181/restconf/config /opendaylight-inventory:nodes/node/openflow:2/table/0/flow/100* ou 'openflow:2' est l'identifiant du switch, 0 le numéro de la table et 100 l'ID de l'entrée flux. La topologie du réseau est celle illustrée précédemment dans l'interface graphique du contrôleur.

```xml
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<flow xmlns="urn:opendaylight:flow:inventory">
    <strict>false</strict>
    <instructions>
        <instruction>
            <order>0</order>
            <apply-actions>
                <action>
                    <order>0</order>
                    <drop-action/>
                </action>
            </apply-actions>
        </instruction>
    </instructions>
    <table_id>2</table_id>
    <id>126</id>
    <cookie_mask>255</cookie_mask>
    <installHw>false</installHw>
    <match>
        <ethernet-match>
            <ethernet-source>
                <address>00:00:00:00:00:01</address>
            </ethernet-source>
        </ethernet-match>
    </match>
    <hard-timeout>12</hard-timeout>
    <cookie>3</cookie>
    <idle-timeout>34</idle-timeout>
    <flow-name>UMMTO</flow-name>
    <priority>1000</priority>
    <barrier>false</barrier>
</flow>
```

Figure 26 : Exemple de script XML

Les tests pour vérifier que le flux est bien opérationnel sont multiples. D'abord vérifions que le flux a bien été introduit dans le switch avec la commande dump-flows :

```
OFPST_FLOW reply (OF1.3) (xid=0x2):
 cookie=0x3, duration=997.337s, table=0, n_packets=20, n_bytes=1288, priority=1000,
dl_src=00:00:00:00:00:01 actions=drop
 cookie=0x2b00000000000007, duration=5847.263s, table=0, n_packets=39, n_bytes=3486
, priority=2,in_port=1 actions=output:4,output:2
 cookie=0x2b00000000000005, duration=5847.263s, table=0, n_packets=59, n_bytes=4774
, priority=2,in_port=4 actions=output:2,output:1,CONTROLLER:65535
 cookie=0x2b00000000000006, duration=5847.263s, table=0, n_packets=0, n_bytes=0, pr
iority=2,in_port=2 actions=output:4,output:1
 cookie=0x2b00000000000004, duration=5852.877s, table=0, n_packets=3521, n_bytes=29
9285, priority=100,dl_type=0x88cc actions=CONTROLLER:65535
 cookie=0x2b00000000000000, duration=5852.877s, table=0, n_packets=0, n_bytes=0, pr
iority=0 actions=drop
```

Figure 27 : Resultat de la commande dump-flows s2

Ensuite à partir de h1 dont l'adresse mac est 00:00:00:00:00:01 nous allons ping h2 pour verifier que les paquets en provenance de l'adresse mac en question sont bel et bien bloqués :

```
mininet> h1 ping h2
PING 192.168.1.200 (192.168.1.200) 56(84) bytes of data.
^C
--- 192.168.1.200 ping statistics ---
7 packets transmitted, 0 received, 100% packet loss, time 6000ms
```

Figure 28 : Résultat du test ping

4. L'Application Openflow Manager

L'un des objectifs clés du SDN en général et d'ODL en particulier, est l'abstraction du réseau. Il s'agit de simplifier les opérations réseau gérés par l'administrateur, de telle sorte à ce qu'il n'ait pas à savoir ce qui se passe entre le contrôleur et le switch. L'utilisateur programmera l'infrastructure en dépit du type d'équipements ou des protocoles utilisés, en langage développeur.

4.1.Architecture de l'application OFM

Openflow Manager est une application développée par Cisco pour son contrôleur commercialisé basé sur OpenDaylight, ce qui fait que l'application est compatible avec ODL. Elle est faite pour une architecture MD-SAL, offrant la possibilité de créer des flux aisément grâce à une interface graphique accueillante. L'application communique avec les modèles YANG se trouvant dans le contrôleur à travers l'interface RESTCONF (Les mêmes concepts que la section antérieure). L'approche est simple, en cliquant sur une case pour choisir l'action ou les champs de correspondance par exemple, on génère une ligne de code en Json,

une fois l'opération validée, l'application envoie la requête via RESTCONF au contrôleur où le SAL traduira le modèle au langage de l'équipement en question (Cisco Dev Net, 2014).

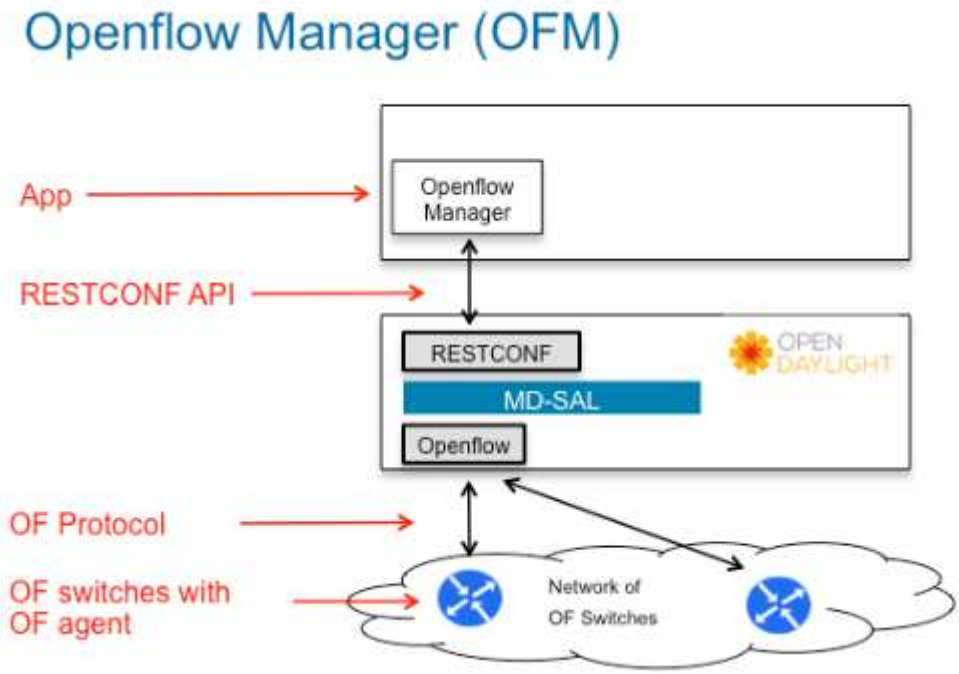

Figure 29 : Architecture d'OFM

4.2.Fonctionnalités d'OFM

Les fonctions de bases proposées par l'application affichée en haut de l'écran interface sont subdivisées en 4 onglets :

- **Basic view :** Vue globale de la topologie sous-jacente. OFM schématise la structure du réseau en affichant les switchs OpenFlow et les hôtes qui leurs sont connectés.
- **Flow management :** Ou gestion de flux. Permet de visualiser, ajouter, modifier ou supprimer les différents flux.
- **Statistics :** Fournit les statistiques liées aux flux et aux ports des switchs.
- **Hosts :** Résume les informations concernant les hôtes gérés par OFM.

Une fois l'application installée et configurée (voir Annexe 2), l'url qui nous renvoie l'interface graphique est le suivant : **<ADRESSE IP> :9000**

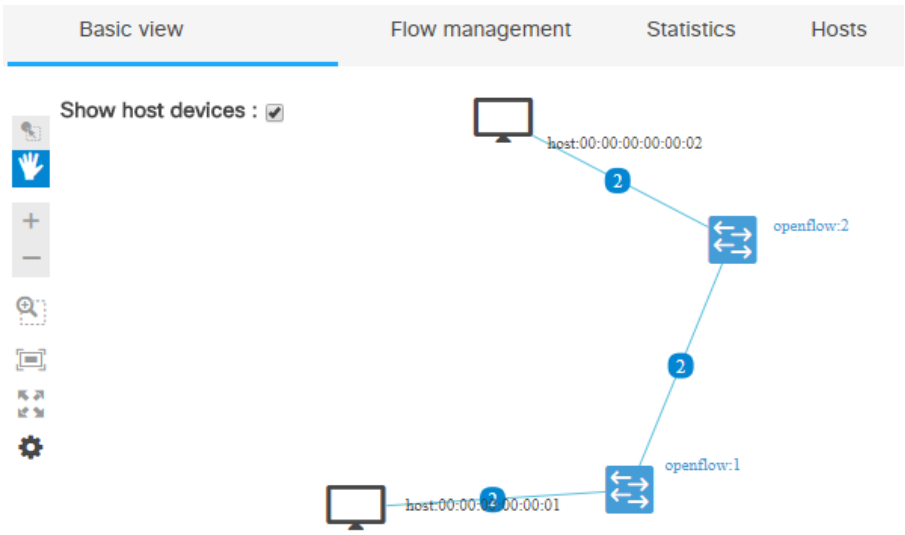

Figure 30 : Interface utilisateur d'OFM

En ce qui concerne la programmation de flux, OFM supporte toutes les actions et identifications de paquets disponibles sur OpenFlow 1.3.

Exemple, si on veut recréer le même flux que celui écrit précédemment sur POSTMAN, il suffit d'aller sur l'onglet Flow Management, et de cliquer sur le bouton 'show window for managing flows' comme suit :

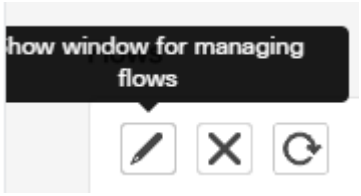

Figure 31 : Accès à la gestion de flux

Ce qui fera apparaitre la fenêtre illustrée dans la **figure 35,** ensuite il ne reste plus qu'à remplir les cases sélectionnées pour générer l'entrée de flux sur le switch choisi, et à envoyer la requête en cliquant sur 'send request'.

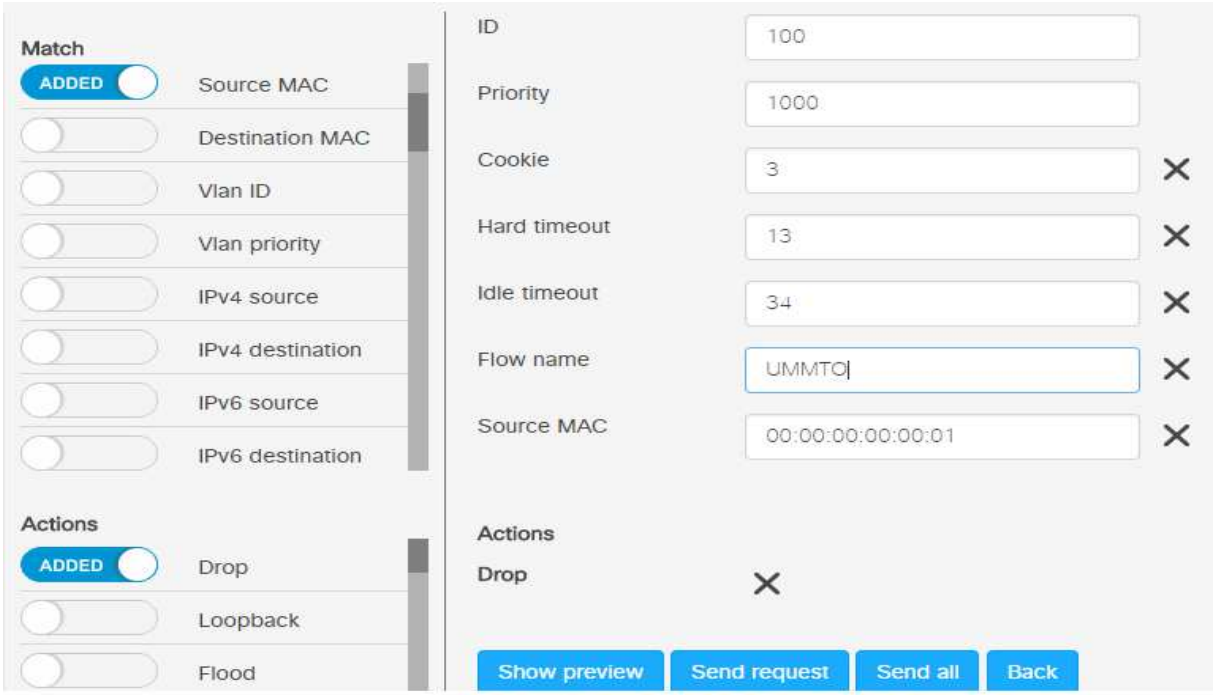

Figure 32 : Programmation de flux sur OFM

Comme test de fonctionnement nous allons envoyer une requête GET avec POSTMAN à l'adresse *http://192.168.1.4:8181/restconf/config/opendaylightInventory:nodes/node/openflow:2 /table/0/flow/100.* Qui nous renverra un script json dont le contenue est identique à la requête PUT écrite en XML dans la partie POSTMAN.


```
{
    "flow-node-inventory:flow": [
        {
            "id": "100",
            "strict": false,
            "table_id": 0,
            "priority": 1000,
            "idle-timeout": 34,
            "cookie": 3,
            "installHw": false,
            "instructions": {
                "instruction": [
                    {
                        "order": 0,
                        "apply-actions": {
                            "action": [
                                {
                                    "order": 0,
                                    "drop-action": {}
                                }
                            ]
                        }
                    }
                ]
            },
            "flow-name": "UMMTO",
            "match": {
                "ethernet-match": {
                    "ethernet-source": {
                        "address": "00:00:00:00:00:01"
                    }
```


Figure 33 : Resultat de la requête GET

5. Discussion

Dans ce chapitre nous avons fait le tour des principaux outils qui seront exploités dans l'accomplissement des deux scénarios sur les quelles sera axé le prochain chapitre. Nous avons étudié et testé chaque élément de l'architecture SDN, pour exposer les différentes fonctionnalités mais surtout pour démontrer leur fiabilité avant d'en faire usage par la suite.

Chapitre 3:
Réalisation d'une solution SDN

1. Préambule

Dans ce chapitre nous présenterons notre contribution principale qui consiste en la programmation et l'administration d'un réseau d'entreprise d'une part et la proposition d'une solution de routage sensible aux applications du réseau d'autre part. Le routage dans ce cas, fonctionnera en multipathing, et permet de choisir les routes des différents flux en fonction du type d'application et du type de protocole de niveau 4.

Les deux solutions que nous décrirons dans ce chapitre nécessitent deux scénarios. L'objectif du premier est d'expliquer comment administrer un réseau de type SDN. Par contre dans le deuxième scénario, nous allons tester les fonctionnalités de routage offert par le SDN. Pour cela, nous avons pris un segment du même réseau.

2. Scénario 1 : Administration d'un réseau SDN

Le paradigme SDN a révolutionné les tâches de l'administrateur réseaux de maints aspects. Les tâches rugueuses qui nécessitaient des connaissances accrues des protocoles et du langage de l'équipementier sont à présent réalisées par des applications, ce qui permet la programmation du comportement d'un réseau de manière plus fluide et automatisée en précisant uniquement la fonction souhaitée.

Ces changements sont accompagnés de possibilités de plus en plus riches en ingénierie de trafic. La capacité du contrôleur à récupérer les informations et les statistiques du réseau, combiné aux libertés considérables dans l'orientation des flux de données offrent des solutions d'optimisations sans précédant.

L'administration du réseau n'est plus limitée par la complexité du matériel. En effet, les plateformes utilisées pour l'administration du réseau sont basées exclusivement sur l'open source Linux. Par conséquent, les longues journées passés à configurer les appareils ligne par ligne, où à chercher la source d'une panne ou d'une latence, ne sont désormais plus d'actualité.

2.1. Topologie proposée

La topologie à configurer est adaptée à la structure d'une entreprise. Pour notre exemple, nous avons pris le cas d'une entreprise composée de 4 départements. Cette architecture réseau nécessite l'utilisation de 4 switchs d'accès sur lesquelles seront branchés les différents hôtes.

Le passage vers le cœur du réseau pour avoir accès à la ressource commune (application, cloud, internet …) du réseau devra se faire via l'un de 3 switchs de distribution. Ceci fournit aux paquets une multitude de routes et offre une redondance considérable. De plus, nous avons la possibilité d'équilibrer la charge sur le réseau, en isolant par exemple les flux éléphant, leur allouant un chemin dédié. Le Core switch *S8* devra être relié aux ressources externes par un lien de très large bande passante, vu que tout le trafic nord sud du réseau (hôte vers ressources) y sera acheminé.

Les commutateurs seront connectés au contrôleur en hors bande à travers S8. Ce dernier sera le seul à être relié physiquement au contrôleur.

L'architecture ciblée est illustrée dans la figure suivante.

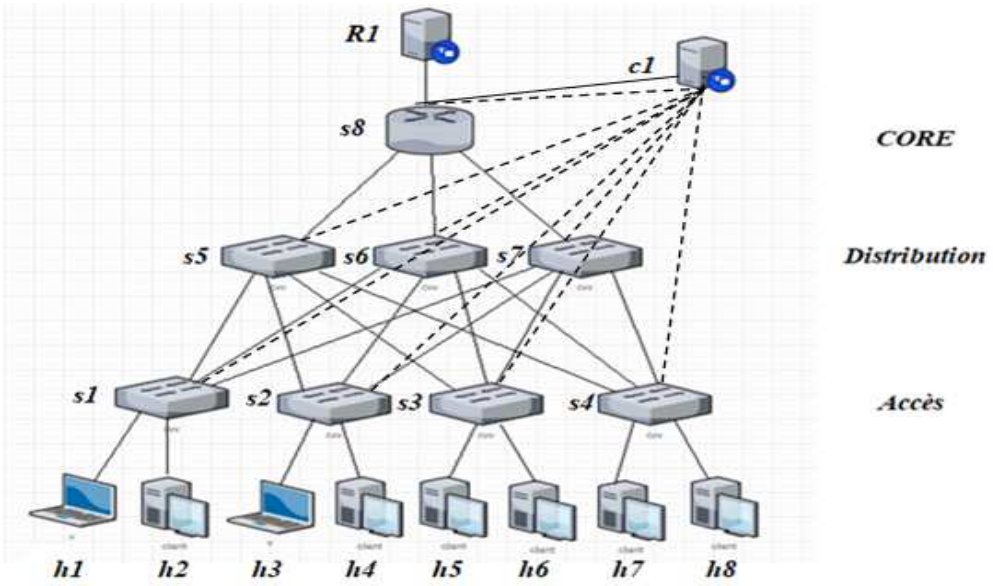

Figure 34 : Croquis de la topologie du réseau

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

Hôtes	Adresse MAC	Adresse IP
H1	00 :00 :00 :00 :00 :01	192.168.1.11
H2	00 :00 :00 :00 :00 :02	192.168.1.12
H3	00 :00 :00 :00 :00 :03	192.168.1.23
H4	00 :00 :00 :00 :00 :04	192.168.1.24
H5	00 :00 :00 :00 :00 :05	192.168.1.35
H6	00 :00 :00 :00 :00 :06	192.168.1.36
H7	00 :00 :00 :00 :00 :07	192.168.1.47
H8	00 :00 :00 :00 :00 :08	192.168.1.48

Tableau 4 : Informations des hôtes

2.2. Opérations d'administration

2.2.1. Commutation et Routage

En ce qui concerne le SDN, nous ne parlons plus de routage où de commutation, la terminologie utilisée est simplement la transmission, les décisions n'étant plus prises par l'équipement lui-même et les protocoles de routage ne sont plus nécessaires. Toutefois, leur implémentation reste possible en permettant le déploiement de réseaux hybrides, avec à la fois des switchs OpenFlow et des équipements classiques.

Les switchs étant multicouches, combinés au contrôleur cela permet d'avoir une configuration individuelle de l'orientation des flux et un choix granulaire du comportement du réseau.

Nous allons entre autres explorer la possibilité de faire communiquer des hôtes, en usant à la fois de politiques de couche 2 et de couche 3.

2.2.2. Pare-feu

Les tables de flux peuvent être exploitées pour générer des politiques de pare-feu. En utilisant l'action drop d'OpenFlow et la capacité du contrôleur à instaurer des règles pour les 4 premières couches du modèle iso, nous pouvons bloquer un trafic en fonction de son port d'entrée, des adresses mac et IP de source et de destination, des tags Vlan ou MPLS, et du type de protocole de transport voire le port d'écoute de source et de destination. Cependant pour vraiment exploiter le potentiel du SDN en termes de sécurité, une application permettant d'inspecter les paquets (DPI) est nécessaire. L'identification des paquets pouvant aller jusqu'à la couche 7, fait de ce genre d'applications une plateforme fiable de détection de menaces.

Dans ce scénario nous appliquerons un nombre de politiques de restrictions à notre topologie, et discuterons l'efficacité de notre méthode.

2.3.Implémentation de l'architecture

Pour installer les machines virtuelles nous avons opté pour l'hyperviseur Workstation de la marque VMware.

Deux VM ont été requises pour la réalisation du scénario, leur réseau est configuré en bridge pour qu'une adresse du même réseau que le PC leur soit allouée, permettant la communication entre la machine locale et les machines virtuelles. Ceci nous permettra

d'ouvrir des terminaux SSH pour une meilleure manipulation des VM avec surtout la possibilité d'exécuter toutes sortes de programmes linux sur la plateforme X11 et de pouvoir lancer les interfaces graphiques d'ODL et OFM sur un navigateur local.

La configuration des machines est la suivante :

- Mininet : Toutes les versions de l'émulateur sont disponibles sur (Github, 2017), sous formes de VM préparées, il suffit d'importer le fichier .OVF sur l'hyperviseur. La version installée dans notre cas est la 2.2.2, sur un OS ***ubuntu serveur 14.04.4 amd 64***. Par défaut la machine est configurée avec 1Gb de RAM, 1 Processeur et 8Gb de stockage max. Nous n'avons pas jugé utile de changer cette configuration.
- Opendaylight : Nous avons opté pour la version lithium du contrôleur, disponible dans le dépôt d'Opendaylight. Le téléchargement est possible dans 3 formats compressés (.tar .zip et .pom). Le contrôleur a été installé sur une plateforme ubuntu serveur 12.04 amd64. La configuration de la machine est illustrée dans la figure 35.

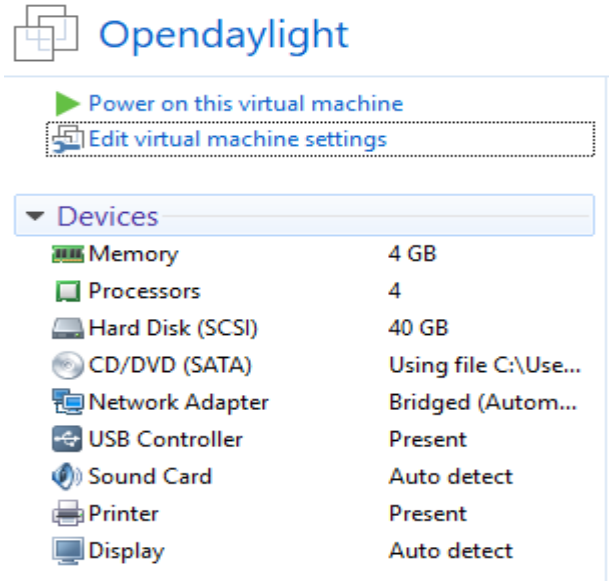

Figure 35 : Configuration de la machine ODL

2.3.1. Création de la topologie

La partie infrastructure de notre architecture sera customisée et générée sur la plateforme Mininet. La topologie a été créée par le biais d'un script Python (voir Annexe 3) sauvegardé dans le répertorie `mininet/custom/`.

La commande pour créer le réseau personnalisé cité ci-dessus est la suivante :

```
sudo mn --controller=remote,ip=192.168.1.9 --custom mininet/custom/admintopo.py --topo mytopo --mac
```

Celle-ci générera les différents éléments de la couche physique qui sera connectée au contrôleur OpenDaylight à distance.

```
*** Creating network
*** Adding controller
Connecting to remote controller at 192.168.1.9:6653
*** Adding hosts:
h1 h2 h3 h4 h5 h6 h7 h8
*** Adding switches:
s1 s2 s3 s4 s5 s6 s7 s8
*** Adding links:
(s1, h1) (s1, h2) (s1, s5) (s1, s6) (s1, s7) (s2, h3) (s2, h4) (s2, s5) (s2, s6)
 (s2, s7) (s3, h5) (s3, h6) (s3, s5) (s3, s6) (s3, s7) (s4, h7) (s4, h8) (s4, s5
) (s4, s6) (s4, s7) (s8, s5) (s8, s6) (s8, s7)
*** Configuring hosts
h1 h2 h3 h4 h5 h6 h7 h8
*** Starting controller
c0
*** Starting 8 switches
s1 s2 s3 s4 s5 s6 s7 s8 ...
*** Starting CLI:
```

Figure 36 : Création des éléments du réseau dans Mininet

Le contrôleur détecte les switchs Openflow, et après un ping global détecte les hôtes et parvient à afficher la stucture du réseau sous-jacent comme l'atteste la figure 36.

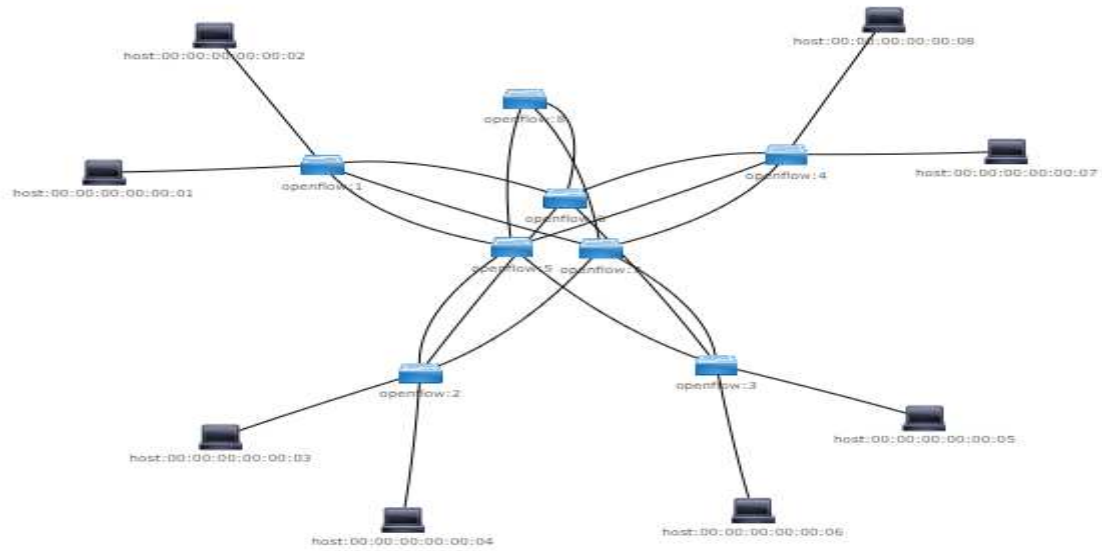

Figure 37 : Topologie du scénario 1 affichée sur ODL

L'application OpenFlow Manager écrite en *Node.js*, est déployée à l'interface nord du contrôleur sur un serveur web écoutant le PORT 9000. La topologie est tout de suite récupérée par celui-ci, et sera affichée sur l'interface graphique comme suit.

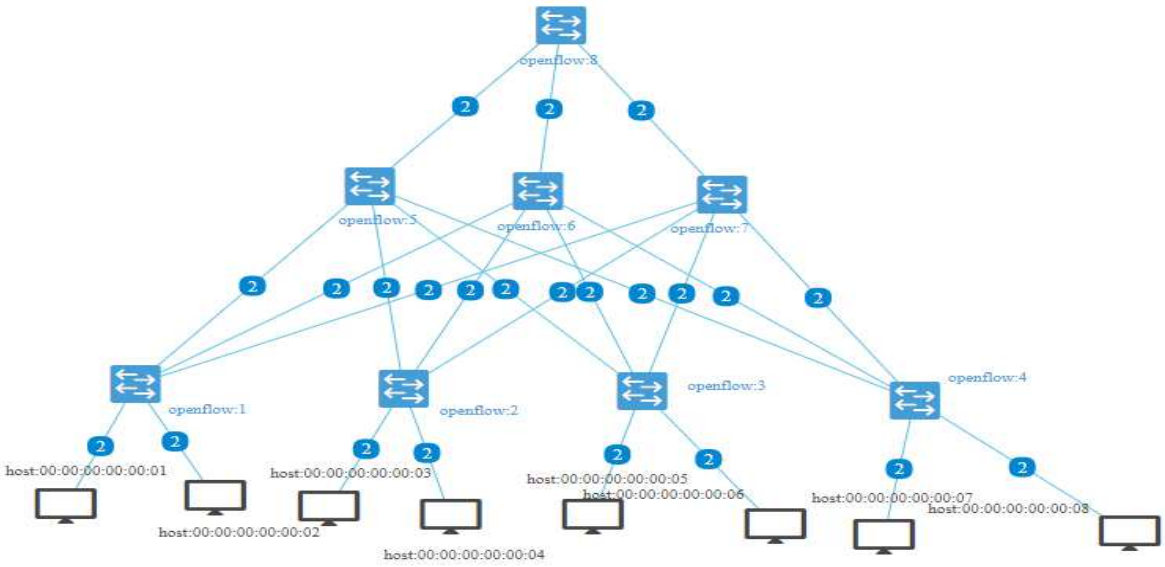

Figure 38 : Vue globale de la topologie du scénario 1 affichée par OFM

2.3.2. Planification du comportement du réseau

Ici nous allons résumer le comportement du réseau en précisant, les trafics bloqués et les chemins empruntés par les paquets qui seront transmis :

- Le switch S4 est isolé du reste du réseau, les hôtes qui y sont connectées ne communiquent qu'entre eux. Ceci peut être achevé par la simple assignation d'un VLAN, mais dans notre cas nous avons opté pour des régles pare-feu de couche 3, bloquant tout les paquets allant et venant du sous réseau auxquelles appartient les hôtes h8 et h7.
- h3 n'a pas les privilèges nécessaires pour communiquer avec h5. Les entrées pour bloquer ces paquets seront installés sur les deux switchs concernés sous formes de régles pare-feu de couche 3 ciblant les deux adresses IP spécifiques aux hôtes. Il est important de bloquer le trafic dans les deux sens, pour ne pas consommer de la bande passante inutilement.
- h1 et h2 ne communiquent pas, un pare-feu de couche 2 leur sera appliqué.

Hormis les restrictions citées préalablement, les chemins empruntés par les différents flux qui atteignent les switchs de distribution sont les suivants :

- Le trafic provenant ou allant vers un Laptop (h1 et h3), est distribué par s5. Les liens vers les deux autres switchs de distribution serviront de redondance. Si une panne

survient sur cette liaison, les paquets passeront par s7. Ceci peut être réalisé en instaurant un délai d'inactivité. Si aucune correspondance n'est détectée durant un temps donné, les entrées inactives sont retirées des tables de flux.

- Le reste du trafic entre les switchs s1 et s2 transitera par s6. s5 servira de redondance.
- La communication des tours avec h5 et h6 devra se faire à travers s7. s6 est le premier choix de redondance, si cette liaison est inutilisable s5 sera utilisée.

2.3.3. Installation des flux

✓ Synthèse des entrées de flux

Dans cette section les règles introduites dans les différents switchs sont résumés sous forme de tables de flux. Concernant les adresses MAC seul le dernier octet sera mentionné.

Openflow : 1							
ID Flux	Eth_type	Src_MAC	Dst-MAC	Src-IP	Dst-IP	Action	Priorité
102	*	01	02	*	*	Drop	2000
201	*	02	02	*	*	Drop	2000
1203	*	*	03	*	*	Output : 1	1500
12033	*	*	03	*	*	Output : 3	1400
1456	*	01	*	*	*	Output : 1	1500
11456	*	01	*	*	*	Output : 3	1400
204	*	02	04	*	*	Output : 2	1500
2204	*	02	04	*	*	Output : 1	1400
2056	2048	*	*	*	192.168.1.0	Output : 3	1000
256	2048	*	*	*	192.168.1.0	Output : 2	900
01	*	*	*	*	*	Contrôleur	100

Tableau 5 : Table de Flux s1

Openflow : 2							
ID Flux	Eth_type	Src_MAC	Dst-MAC	Src-IP	Dst-IP	Action	Priorité
305	2048	*	*	192.168.1 .23	192.168.1.35	Drop	2000
3401	*	*	01	*	*	Output : 1	1500
302	*	03	02	*	*	Output : 1	1500
402	*	04	02	*	*	Output : 2	1500
4056	2048	*	*	*	192.168.1.0	Output : 3	1000
02	*	*	*	*	*	Contrôleur	100

Tableau 6 : Table de flux s2

Openflow : 3							
ID Flux	Eth_type	Src_MAC	Dst-MAC	Src-IP	Dst-IP	Action	Priorité
503	2048	*	*	192.168.1.35	192.168.1.23	Drop	2000
5601	*	*	01	*	*	Output : 1	1500
5603	*	*	03	*	*	Output : 1	1500
5602	*	*	02	*	*	Output : 3	1500
5604	*	*	04	*	*	Output : 3	1500
03	*	*	*	*	*	Controller	100

Tableau 7 : Table de flux de s3

Openflow : 4							
ID Flux	Eth_type	Src_MAC	Dst-MAC	Src-IP	Dst-IP	Action	Priorité
807	*	08	07	*	*	Output : 4	2000
708	*	07	08	*	*	Output : 5	2000
087	2048	*	*	192.168.1.0	192.168.1.0	Drop	1500
04	*	*	*	*	*	Controller	100

Tableau 8 : Table de flux de s4

Pour les switchs d'agrégation, les flux ne sont plus insérés de façon granulaire, la table de flux en est moins détaillée. Un réseau subdivisé en Vlan ou en sous réseaux est encore plus simple à mettre en place.

Ici on s'est seulement basé sur l'adresse mac de destination pour orienter les flux vers un port de sortie.

✓ Méthode d'implémentation des flux

Les différentes méthodes ont été présentées dans le chapitre précédent. Pour la suite nous allons nous baser sur l'application OFM. La raison principale de ce choix, est la visibilité qu'offre l'interface graphique de l'application, ce qui simplifie l'édition et la suppression de flux installés auparavant. Il serait encombrant d'illustrer l'introduction de tout les flux dans l'application, et d'autant plus répétitif. Nous allons seulement afficher ci-dessous, l'un des flux introduits. A noter qu'il est impératif d'informer le contrôleur du type Ethernet qui sera utilisé pour l'identification des paquets, dans l'exemple suivant la valeur 2048 (0x0800) représente le type Ethernet IPv4.

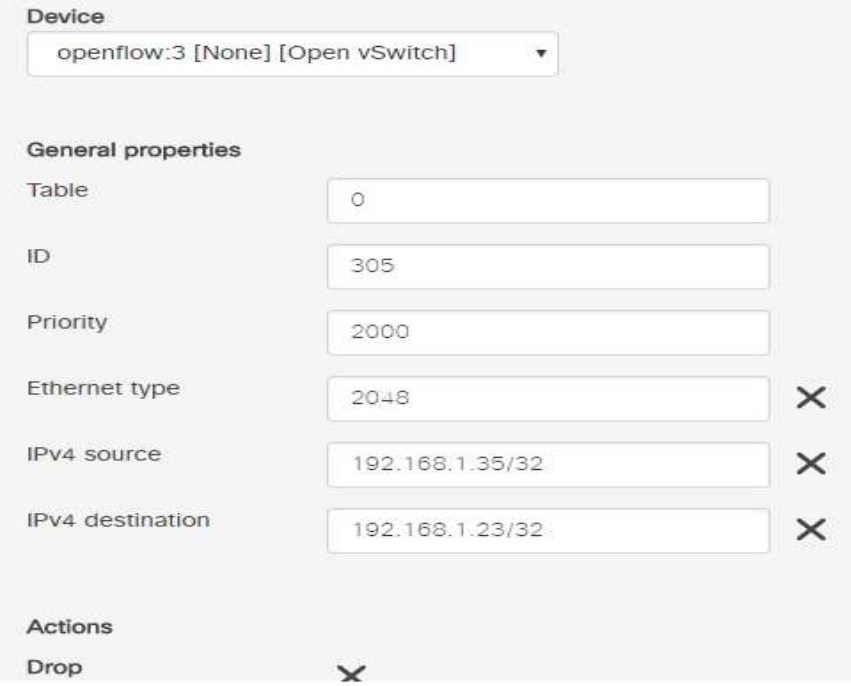

Figure 39 : Installation de l'entrée flux 305 sur OFM

2.4.Tests de fonctionnement

Un nombre de test a été effectué pour confirmer que l'objectif du scénario a bien été atteint. Nous avons subdivisé ces tests en 3 catégories.

2.4.1. Test d'accessibilité

Comme l'atteste la figure 39, le réseau se comporte comme prévu en termes d'accessibilité, ce qui confirme la fiabilité des restrictions appliquées.

```
*** Ping: testing ping reachability
h1 -> X h3 h4 h5 h6 X X
h2 -> X h3 h4 h5 h6 X X
h3 -> h1 h2 h4 X h6 X X
h4 -> h1 h2 h3 h5 h6 X X
h5 -> h1 h2 X h4 h6 X X
h6 -> h1 h2 h3 h4 h5 X X
h7 -> X X X X X X h8
h8 -> X X X X X X h7
*** Results: 50% dropped (28/56 received)
```

Figure 40 : test d'accessibilité des hôtes

2.4.2. Test d'orientation du trafic

Pour ce test nous avons utilisé l'analyseur de paquets wireshark. Celui-ci nous permettra de visualiser le trafic traversant des interfaces prédéfinis.

Les 3 figures suivantes confirment que les flux empruntent les chemins précédemment définis dans les tables de flux.

- H1 et H3 communiquent à travers S5 (Figure 40).
- H2 et H4 communiquent à travers S6 (Figure 41)
- H4 et H5 à travers S7 (Figure 42)

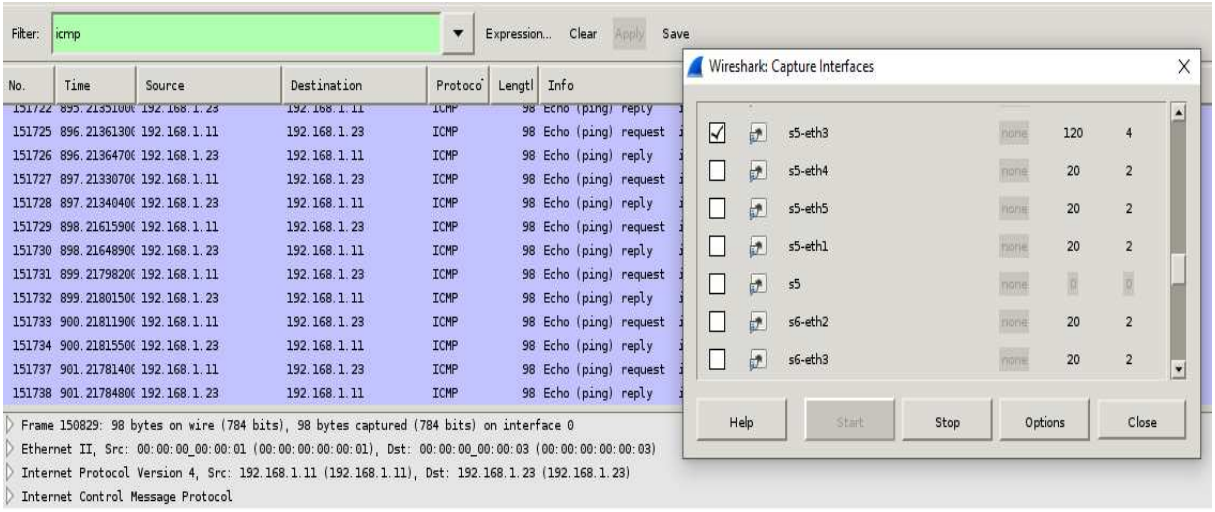

Figure 41 : Capture wireshark d'un ping entre H1 et H2

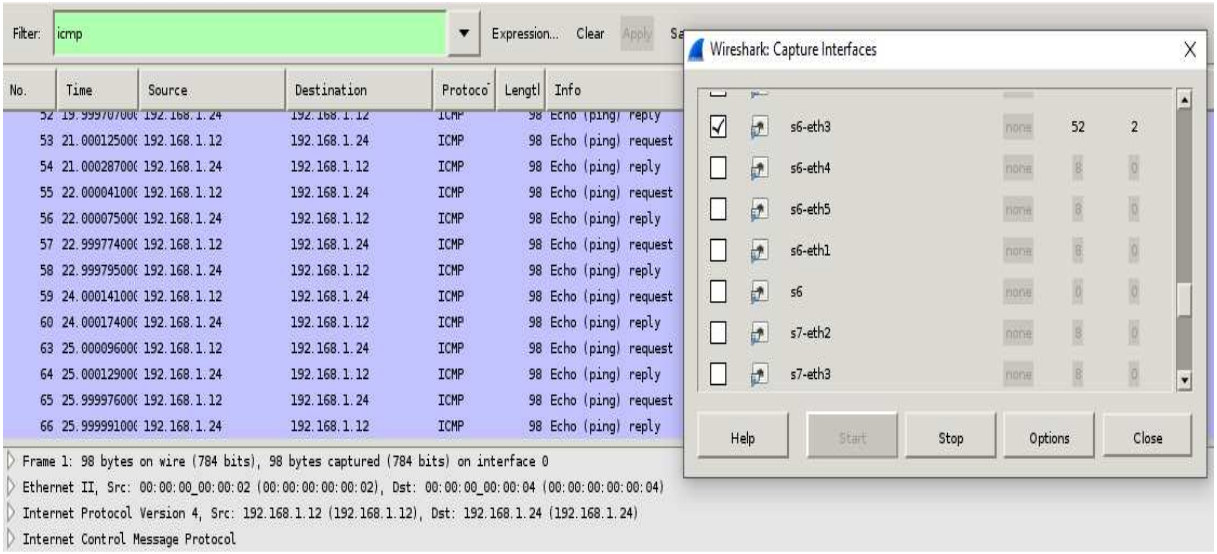

Figure 42 : Capture wireshark entre H2 et H4

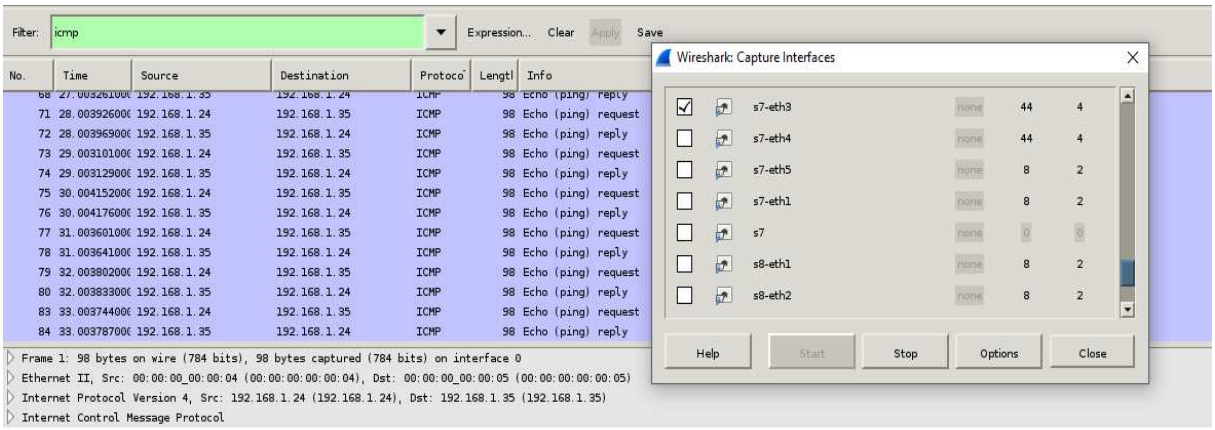

Figure 43 : Capture wireshark d'un ping entre H4 et H5

2.4.3. Test de redondance

Pour tester la redondance des liens nous supprimerons la lisaison entre s1 et s5 avec la commande `link s1 s5 down` sur Mininet.

Le contrôleur détecte que le lien est défectueux comme l'atteste la figure suivante. Cependant il faut d'abord rafraichir la page pour voir la liaison entre s1 et s5 disparaître ce qui n'est pas pratique en milieu de production.

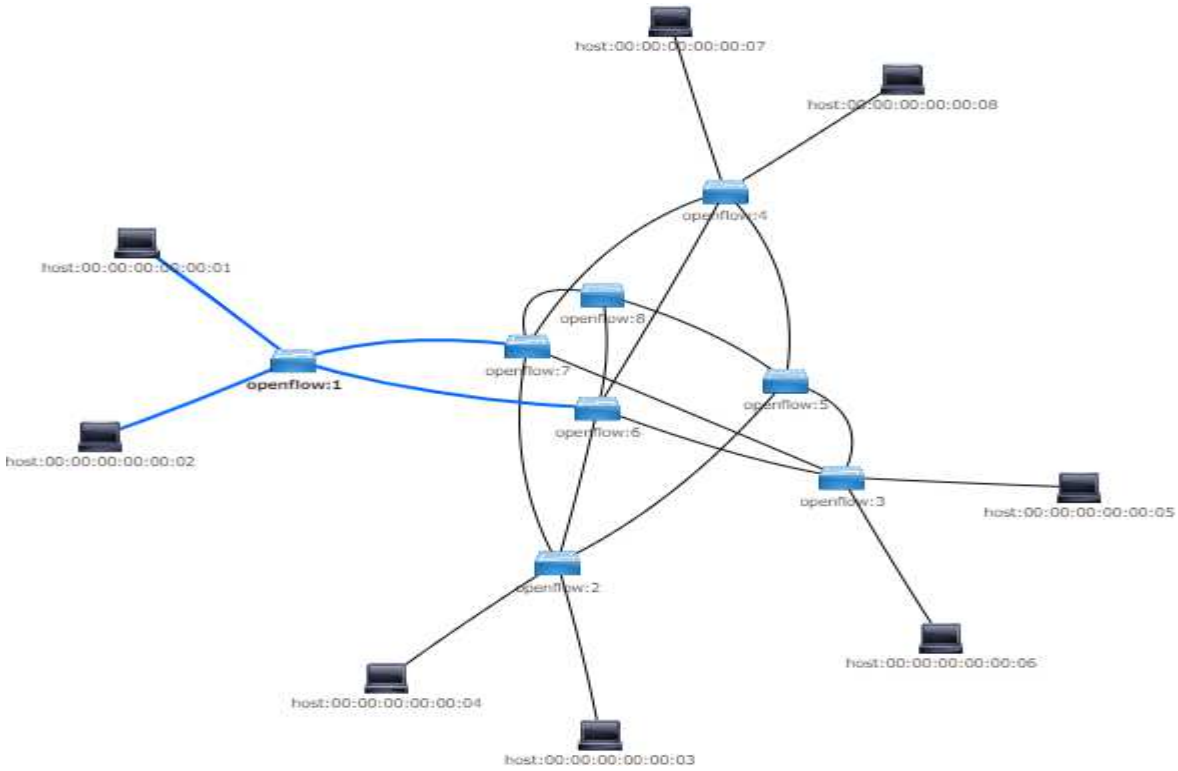

Figure 44 : Détection de la perte de liaison s1 s5 sur ODL

Ensuite une capture wireshark nous permettra de voir que le trafic entre h1 et h3 a été réorienté vers la liaison secondaire s1 s7.

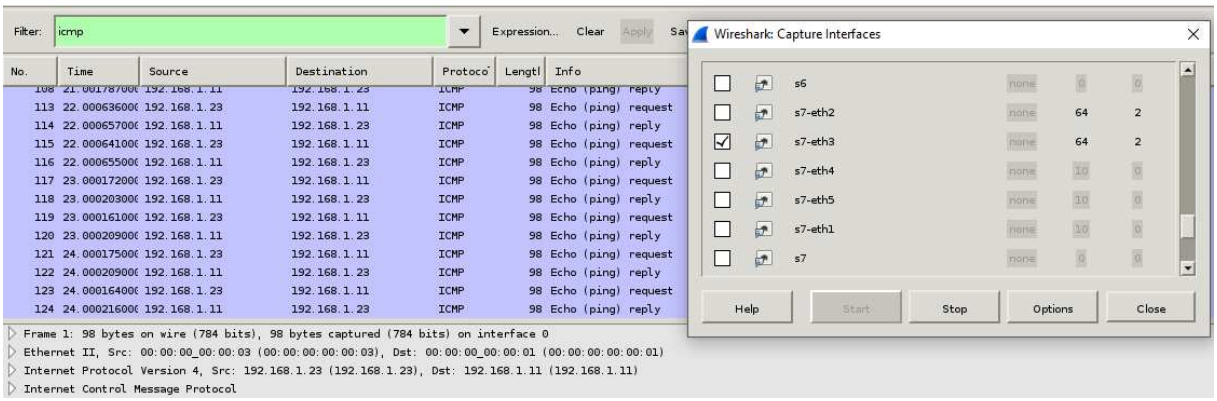

Figure 45 : Capture wireshark test de redondance

3. Scénario 2 : Routage sensible aux applications

3.1. Etude du scénario

Selon l'index de Cisco Global Cloud (Cisco, 2013), le trafic nord sud dans un campus représente plus de 90% des flux de données du réseau. Ceci inclut la connexion entre l'utilisateur et les différentes applications déployés. Certaines de ces ressources étant primordiale pour la plus part des entreprises, une congestion ou une perte de données peut s'avérer gênante voir problématique

Pour cette partie nous nous intéresserons à la communication entre un hôte et une ressource, en gardant la même topologie que celle du premier scénario. Mais en ignorant les hôtes et les switchs qui ne seront pas nécessaires à notre simulation.

L'approche en vue considère aussi que la bande passante et la latence des liaisons diffère. Ceci dit les caractéristiques idéales d'une liaison dépendent surtout de l'application. Pour illustrer cette idée prenons l'exemple d'une visioconférence ou de n'importe quelle application fonctionnant en temps réel. Celle-ci peut ne pas être gourmande en bande passante, mais la présence d'une latence importante peut la rendre inutilisable. Inversement une application ftp est plus tolérante à la latence, mais reste très gourmande en bande passante.

Basé sur cette idée nous proposerons une solution pour orienter le trafic en fonction de l'application utilisée. Notre ressource sera composée de 3 serveurs. Un serveur Web, un serveur de stockage de données et un serveur VoIP.

Notre choix des routes pour ces applications dépend principalement de la latence des 3 chemins. Pour calculer les lags le test le plus fiable reste le ping.

Voici illustré par les figures ci-dessous, les valeurs de latence pour les 3 chemins disponibles.

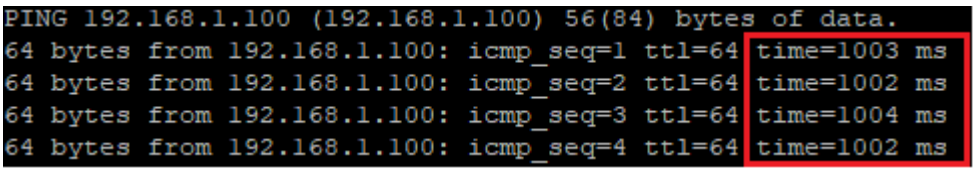

Figure 46 : Latence du chemin S3

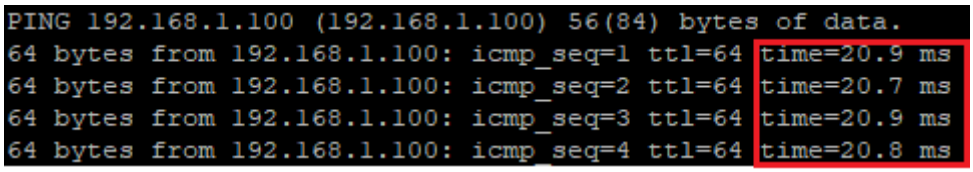

Figure 47 : Latence du chemin S4

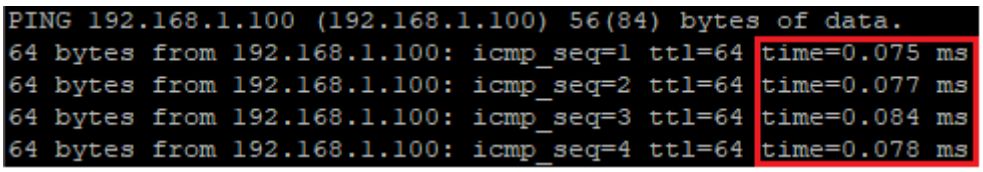

Figure 48 : Latence du chemin S5

Comme on peut le voir la route s3 présente une latence inacceptable pour une application temps réel, malgré une bande passante plus élevée que les deux autres (prédéfinis à 100Mbit/s).

L'objectif du scénario sera d'orienter le trafic comme suit :

- Application streaming ou VoIP : Distribuée par S5
- Application FTP : Distribuée par S3
- Application HTTP : Distribuée par S4

3.2.Approche d'identification des applications

Le cas idéal serait de pouvoir inspecter les paquets jusqu'à la couche 7 pour pouvoir appliquer des classes QoS à l'architecture en place.

Ici nous allons seulement exploiter les protocoles et ports d'écoute utilisés par les applications. Chaque protocole de transport est identifié par un numéro unique, et il existe un port d'écoute par défaut sur cette même couche de transport pour les protocoles d'application.

En ce qui nous concerne nous allons identifier les applications précédemment cités de la manière suivante :

- Application streaming ou VoIP : Protocole UDP (17)
- Application FTP : Protocle TCP (6) port d'écoute par défaut (21)
- Application http : Protocole TCP (6) port d'écoute par défaut (80)

3.3. Implémentation de l'identification par protocole

La méthode d'implémentation des flux reste la même. Le plus sur ce qu'on a déjà vu est l'introduction de l'identification par protocole et par port source et destination. Le flux introduit dans la figure 49 force le trafic HTTP de passer par s4

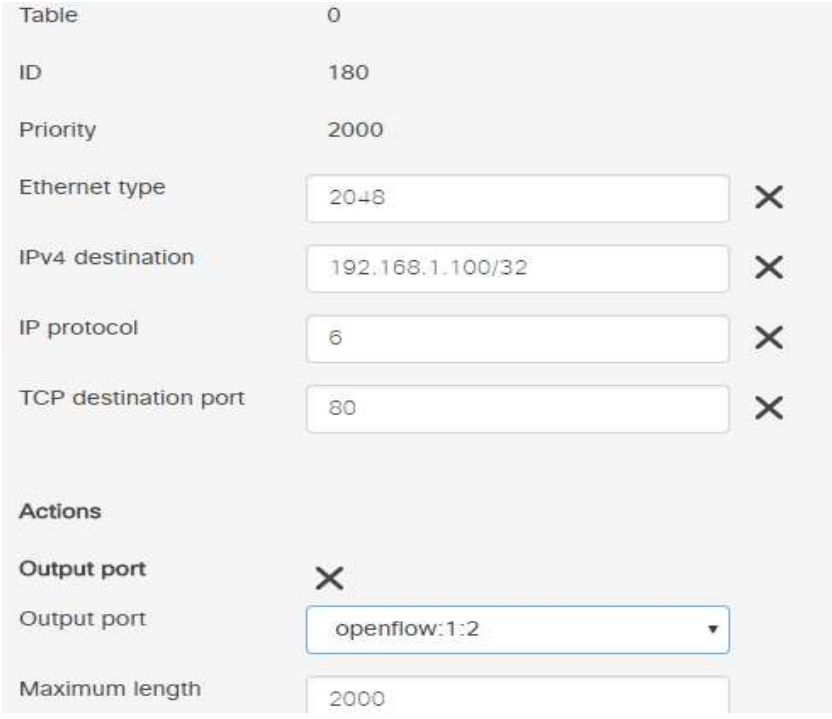

Figure 49 : Orientation du trafic HTTP sur OFM

3.4. Résultats

Pour confirmer le bon fonctionnement de notre solution, 3 tests ont été appliqués.

3.4.1. Le HTTP

Pour confirmer que le trafic traverse le commutateur S4, nous avons simulé un serveur HTTP sur R1 en utilisant un SimpleHTTPServer qui est contenu dans la bibliothèque standard de python.

Sur le client h1 une commande **wget** avec l'adresse du serveur est exécutée.

Sur le terminal du switch, l'outil tcpdump a été exploité pour afficher tout le trafic traversant l'interface choisie.

Le test affiche ceci :

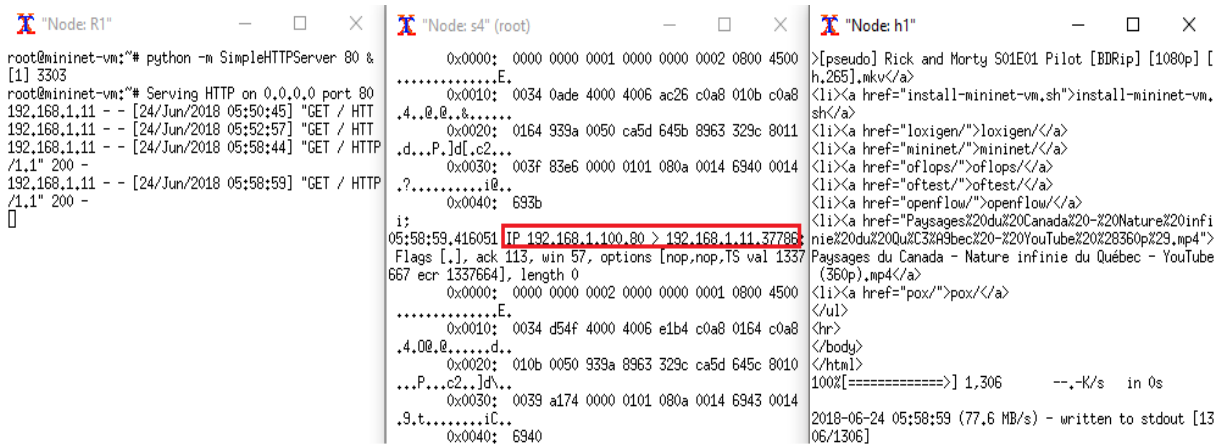

Figure 50 : Test de l'application HTTP

Les résultats sont explicites :

- La communication HTTP entre le client et le serveur est bien établie.
- Le trafic HTTP est distribué par s4.

3.4.2. FTP

L'outil utilisé pour le transfère FTP est Netcap. Pour ce faire la commande coté client est nc.

Sur le switch s3 la commande utilisée est : tcpdump –XX –n –i s3-eth1

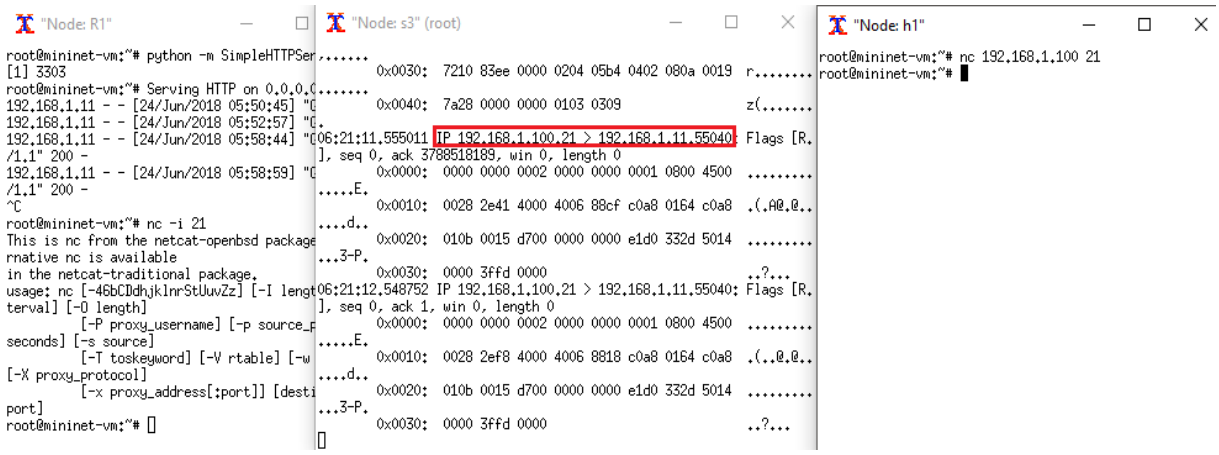

Figure 51 : Test de l'application FTP

Les résultats confirment que l'objectif a bien été atteint.

3.4.3. UDP

Pour l'application streaming ou VoIP, transporté par le protocole UDP (SIP avec asterisk par exemple), l'outil de test utilisé est iperf. Comme illustré dans la figure ci-dessous.

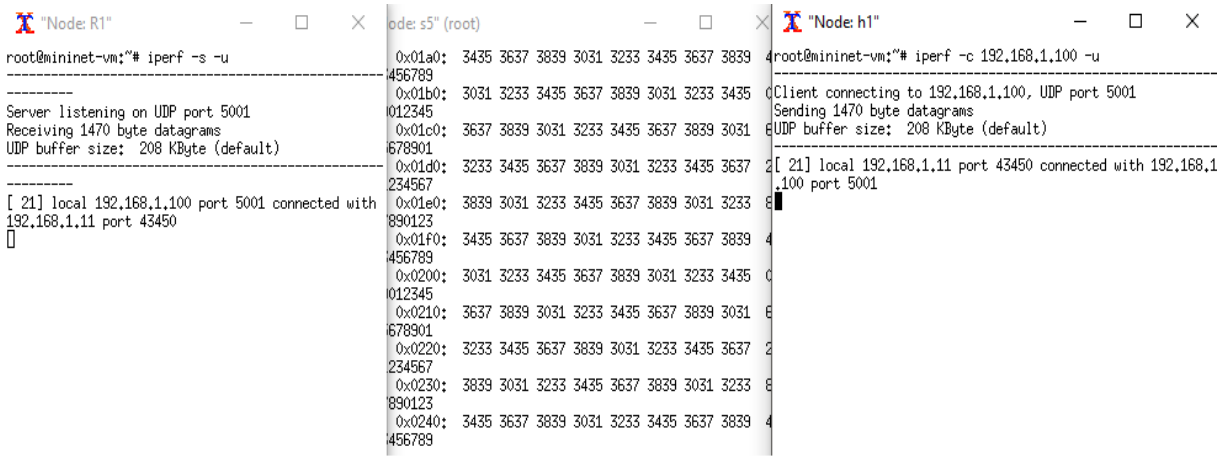

Figure 52 : Test de l'application UDP

Sur le commutateur s5, on voit l'affluence des données entre le client et le serveur, ce qui démontre le passage du trafic par s5.

4. Discussions

4.1.Le Scénario 1

Dans cette première partie nous avons décrit les opérations d'administration les plus fréquentes. En plus de la simplicité de l'implémentation et le gain de temps considérable en ce qui concerne la programmation du comportement des réseaux. Les résultats ont démontrés la fiabilité de l'architecture en termes du respect des politiques prédéfinis.

Un test d'accessibilité a montré le comportement du réseau en termes de restrictions, celui-ci a confirmé la conformité des communications envers ce qui a été préétabli.

Un test pour déterminer l'orientation du trafic a validé la méthode utilisée pour définir les chemins emprunté par les flux.

Du côté de la redondance, l'objectif a été atteint. Par contre nous sommes arrivés à la conclusion que OpenDaylight n'est pas le contrôleur idéal pour un environnement de production non tolérant aux erreurs. Le fait d'avoir à rafraichir l'interface pour voir la disparition d'une liaison défectueuse peut s'avérer handicapant, contrairement à certains contrôleurs comme ONOS qui a démontré une très grande fiabilité dans ce genre de situations.

4.2.Scénario 2

Nous avons proposés une méthode pour séparer le trafic du réseau en fonction du type d'application utilisée et la latence entre les différents nœuds, en nous intéressant uniquement aux flux nord sud de la topologie.

Les résultats des tests, ont démontré la fiabilité de cette méthode. En effet, nous avons pu réorienter le trafic d'applications, de sorte à ce que le serveur streaming qui était inutilisable avec une latence de plus d'une seconde, fonctionne parfaitement à travers la nouvelle route qui lui a été allouée.

Conclusion

Notre objectif dans ce travail est de comprendre ce que représentait le SDN, pour ensuite l'implémenter et en déduire son apport sur le comportement du réseau tout en testant sa fiabilité.

Le premier constat à prélever est le manque cruel de standardisation. Nous avons rencontré au court de notre étude bibliographique d'innombrables définitions sur ce qu'est ou ce que devrait être le SDN. Par contre l'objectif de celui-ci reste clair et unifié ; apporter une plateforme de contrôle et de programmation pour la communication et le déploiement de services automatisés, simplifier les périphériques réseau et accélérer le développement en adoptant le modèle Open Source de linux. Le seul standard à l'heure actuelle reste OpenFlow, qui a ouvert les discussions sur le SDN, avec l'idée de découplage du plan de contrôle et du plan de données. Malgré son déploiement massif par les plus grandes sociétés (Google, Microsoft, Facebook, Amazon …), il reste tout de même à l'état embryonnaire dans sa façon d'aborder les applications.

Nous avons choisis l'implémentation du modèle Openflow, en l'intégrant dans un environnement virtuel. L'un des points forts de celui-ci est la facilité de son déploiement grâce notamment à la portabilité du switch OVS, qui peut fonctionner en virtuel ou être installé sur du très simple matériel (Rasberry, whitebox …). L'étude s'est ensuite focalisée sur le contrôleur OpenDaylight, qui est le fruit d'un projet géré par la communauté réseau et financé par les plus grands industriels du domaine. Ce contrôleur qui supporte la version 1.3 d'OpenFlow, offre une plateforme d'abstraction et de programmablilitée très riche en diversité de fonctionnalités, elle exploite le modèle correspondance/Actions du protocole OF et offre entre autres une interface fluide aux administrateurs pour programmer les réseaux à travers l'API RESTCONF.

Nous avons exploités les différents éléments cités antérieurement pour réaliser un réseau SDN programmable grâce à l'interface OpenFlow Manager. A travers ce travail, nous avons utilisés de nombreux outils de test qui permettent de confirmer la fiabilité de cette architecture. Les tests que nous avons effectués nous ont confirmé la possibilité du déploiement d'un réseau SDN dans le réseau d'entreprise. Concernant le routage sensible aux applications, nous avons proposés une méthode permettant d'orienter les flux de données en

fonction des applications. Nous avons fait usage des ports d'écoute par défaut des applications et des fonctionnalités d'OpenFlow. La latence étant un élément perturbateur pour les applications en temps réel, notre approche a permis d'éviter les désagréments provoqués par ce facteur en réorientant le trafic des applications UDP et http de manière à éviter leur dysfonctionnement. Le fonctionnement d'une application FTP n'étant pas perturbé par les lags, nous n'avons pas jugé utile de la réorienter vers une autre liaison, ce qui a permis l'optimisation du trafic.

A partir des résultats de tests, nous pouvons dire que le SDN remet en question le modèle actuel des réseaux, et offre de nombreuses solutions quant à la programmation du comportement des flux de données. Il reste cependant du chemin à faire en termes de standardisation, de quoi préparer le terrain pour une transition globale inévitable vers ce type de réseaux.

Nous envisageons, comme perspectives de ce travail d'effectuer des études plus approfondis, concernant la sécurité et la qualité de service dans le SDN. Le point fort de celui-ci étant la couche application, et sa perméabilité au déploiement instantané de services, nous nous pencherons sur les méthodes qui feront un meilleur usage de cette capacité.

BIBLIOGRAPHIE

Bibliographie

1. Azodomolky, S. (2013). *Software Defined Networking with Openflow*. Packt Publishing. Premiére édition. Birmingham, Royaume uni.
2. Badotra, S., & Singh, J. (2017). OpenDaylight as a Controller for Software Defined Networking. *International Journal of Advanced Research in Computer Science*, 8(5), 1105-1111.
3. Benamrane, F. (2017) *Etude des Performances des Architectures du Plan de Contrôle des Réseaux 'Software-Defined Networks* (Thèse de doctorat). Université de Rabat.
4. Boucadair, M., & Jaquenet, C. (2014). *Software-Defined Networking: A Perspective from within a Service Provider Environment.* RFC 7149. Consulté sur : https://tools.ietf.org/html/rfc7149
5. Bradai, A., Singh, K., & Ahmed, T. (2015). Cellular software defined networking: a Framework. *IEEE Communications Magazine*, 53(6).36-43.
6. Canceres, E. (2016). *Le Protocole OpenFlow dans l'Architecture SDN.* Consulté sur : http://www.efort.com/r_tutoriels/OPENFLOW_EFORT.pdf
7. Cisco Dev Net. (2014). *OpenDaylight OpenFlow Manager (OFM) App.* Consulté sur : https://github.com/CiscoDevNet/OpenDaylight-Openflow-App
8. Cisco VNI. (2017). *Global Mobile Data Traffic Forecast Update, 2016–2021* (livre blanc). Consulté sur : https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/mobile-white-paper-c11-520862.html
9. Cisco. (2013). *Cisco Global Cloud Index: Forecast and Methodology, 2013–2018.* Livre blanc.
10. Cocker, O., & Azodomolky, S. (2017). *Software Defined Networking with Openflow*. Packt Publishing. Seconde édition. Birmingham, Royaume uni.
11. Dubey, N. (2016). From Static Network to software-defined networking. *Icasa journal*. vol4. disponible : https://www.isaca.org/Journal/archives/2016/volume-4/Documents/From-Static-Networks-to-SoftwaredefinedNetworking__joa_Eng_0716.pdf
12. Durand, J. (2015). *Le SDN pour les nuls.* Journées Réseaux de l'Enseignement et de la Recherche (JRES), Montpelier.
13. Eseye. (2017). *Postman User Guide.* Consulté sur : https://www.eseye.com/wp-content/uploads/8314-Postman-User-Guide.pdf

14. Github. (2017). Mininet VM images. Consulté sur : https://github.com/mininet/mininet/wiki/Mininet-VM-Images
15. Goransson, P., Black, C., & Culver, T. (2016). *Software Defined Networks A Comprehensive Approach*. 2éme édition. New York, USA. : Elsevier Science.
16. Hemid, A. (2017). Facilitation of The OpenDaylight Architecture. *The 4th Computer Science Conference for University of Bonn Students,* Bonn, Allemagne.
17. Idoudi, K. (2014) *Implémentation d'un plan de contrôle unifié pour les réseaux multicouches IP / DWDM* (Mémoire de Master). Université de Montréal.
18. Jafarian, J.H., Al-Shaer, E., & Duan, Q. (2012). OpenFlow Random Host Mutation: Transparent Moving Target Defense using Software Defined Networking. *Networking Department of Software and Information Systems* Université de Caroline du Nord, Charlotte USA.
19. Jo, E., Pan, D., Liu, J., & Butler, L. (2014). A simulation and emulation study of SDN-based multipath routing for fat-tree data center networks. *Proceedings of the Winter Simulation Conference, Savanah, GA, pp. 3072-3083.*
20. Juniper. (2013). *Décodage du SDN.* Livre Blanc. Juniper Networks, Inc.
21. Kingston Smiler, S. (2015). *OpenFlow Cookbook.* Packt Publishing. Premiére édition. Birmingham, Royaume uni.
22. Kreutz, D., Ramos, F. M. V., Veríssimo, P. E., Rothenberg, C. E., Azodolmolky, S. & Uhlig, S. (2015). Software-Defined Networking: A Comprehensive Survey. *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14-76.
23. Lantz, B., Heller, B. & McKeown, N. (2016) A Network in a Laptop: Rapid Prototyping for Software-Defined Networks. *9th ACM Workshop on Hot Topics in Networks,* Monterey, Californie, USA. Consulté sur : http://conferences.sigcomm.org/hotnets/2010/papers/a19-lantz.pdf
24. Meyer, D. (2013). *The OpenDaylight Project: Introduction and Overview.* Consulté sur : http://www.1-4-5.net/~dmm/talks/OpenDaylight_SDN_Workshop_AZ.pdf
25. Mininet. (s.d). *Mininet walkthrough.* Consulté sur : http://mininet.org/walkthrough/
26. Noble, S. (2017). *Building Modern Networks.* Packt Publishing. Première édition. Birmingham, Royaume uni.
27. Open Networking Foundation, (2014). *OpenFlow Switch Specification version 1.5.0.* Consulté sur : http://www.opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.0.noipr.pdf

28. Open Networking Foundation. (2012). *Software-Defined Networking: The New Norm for Networks.* Livre blanc.
29. Open Networking Foundation. (2013). *Software-Defined Networking (SDN) Definition*. Consulté sur : https://www.opennetworking.org/sdn-resources/sdn-definition
30. Oracle. (2015). *Oracle SDN Performance Acceleration with Software-Defined Networking.* Fiche technique. Consulté sur : http://www.oracle.com/us/products/networking/virtual-networking/sdn/oracle-sdn-ds-1873198.pdf
31. Project Floodlight, (2016). *Floodlight.* Consulté sur : http://www.projectfloodlight.org/floodlight/
32. Pujolle, G. (2014). *Les réseaux.* Editions Eyrolles. 8éme édition. Paris, France.
33. The Linux Foundation. (2018). *Membres du projet OpenDaylight*. Consulté sur : https://www.opendaylight.org/support/members
34. The Linux Foundation. (s.d). *Open vSwitch documentation.* Consulté sur : http://docs.openvswitch.org/en/latest/
35. Toghraee, R. (2017). *Learning OpenDaylight.* Packt Publishing. Première édition. Birmingham, Royaume uni.
36. Tury, M. (2014). *Les risques d'OpenFlow et du SDN*. ANSSI. France.

ANNEXES

Annexe 1 : Installation d'OpenDaylight

1. Prérequis

- Pour faire fonctionner ODL sur une plateforme Karaf, il faut télécharger Java et configurer la variable d'environnement JAVA_HOME.

```
# apt-get update

# apt-get install –y bash-completion software-properties-common python-software-properties curl git

#add-apt-repository ppa :webupt8team/java        //Ajout du dépôt de Java

# apt-get update

# apt-get install oracle-java8-installer oracle-java8-set-default

# nano ~/.bashrc

// Ajouter la ligne Export JAVA_HOME="/usr/lib/jvm/java-8-oracle"

#~/.bashrc

# echo $JAVA_HOME    // Vérification de l'exportation
```

2. Téléchargement d'OpenDaylight

- La commande wget suivie du lien suivant permettra de télécharger la version lithium

https://nexus.opendaylight.org/content/groups/public/org/opendaylight/integration/distribution-karaf/0.3.0-Lithium/distribution-karaf-0.3.0-Lithium.tar.gz

- Ensuite il suffit de décompresser le fichier .tar et accéder au répertoire pour démarrer ODL

```
# tar zxvf distribution-karaf-0.3.0-Lithium.tar.gz

# cd distribution-karaf-0.3.0-Lithium

~/distribution-karaf-0.3.0-Lithium# ./bin/karaf
```

3. Installation des fonctionnalités d'OpenDaylight

- Dans la ligne de commande d'ODL, la commande suivante permettra d'installer les interfaces (REST, Openflow, Yang), la couche d'abstraction (mdsal) et l'apprentissage réactif des nœuds de couche 2 (l2switch).

```
Opendaylight-user@root>feature:install odl-restconf-all odl-openflowplugin-all
odl-l2switch-all odl-mdsal-all odl-yangtools-common
```

4. Interface graphique d'ODL

- url=<IP de la VM> :8181/index.html
- username=admin
- password=admin

Annexe 2 : Installation d'OpenFlow Manager (OFM)

1. Prérequis

- Installation de Nodejs, un environnement d'exécution Javascript côté serveur. Ceci permettra de faire fonctionner grunt.

```
# curl –sl https://deb.nodesource.com/setup_4.x | sudo  -E bash –

#apt-get install nodejs
```

2. Téléchargement d'OFM

- Clonage du répertoire github de l'application.

git clone https://github.com/CiscoDevNet/opendaylight-openflow-App.git

- Configuration de l'adresse IP du serveur web:

```
# sed –i 's/localhost/<adresse IP de la machine>/g'
./opendaylightopenflowApp/ofm/src/common/config/env.module.js
```

- Installation de grunt

```
# npm install –g grunt-cli
```

3. Exécution du serveur web d'OFM

- La commande grunt démarrera une instance du serveur web, et affichera ceci sur la ligne de commande

```
Running "connect:def" (connect) task
Waiting forever...
Started connect web server on http://localhost:9000
```

Annexe 3 : Topologie du chapitre 3 (Script python)

```
#!/usr/bin/python

from mininet.topo import Topo

from mininet.net import Mininet

from mininet.node import Node , Controller, RemoteController, OVSSwitch

from mininet.log import setLogLevel, info

from mininet.cli import CLI

from mininet.util import irange

from mininet.link import TCLink


class NetworkTopo( Topo ):

        def build( self ):

                h1 = self.addHost( 'h1', ip='192.168.1.11/24')

                h2 = self.addHost( 'h2', ip='192.168.1.12/24')

                h3 = self.addHost( 'h3', ip='192.168.1.23/24')

                h4 = self.addHost( 'h4', ip='192.168.1.24/24')

                h5 = self.addHost( 'h5', ip='192.168.1.35/24')

                h6 = self.addHost( 'h6', ip='192.168.1.36/24')

                h7 = self.addHost( 'h7', ip='192.168.1.47/24')

                h8 = self.addHost( 'h8', ip='192.168.1.48/24')


                s1 = self.addSwitch( 's1', dpid='0000000000000001',protocols='OpenFlow13' )

                s2 = self.addSwitch( 's2', dpid='0000000000000002',protocols='OpenFlow13' )

                 s3 = self.addSwitch( 's3', dpid='0000000000000003',protocols='OpenFlow13' )
```

```
s4 = self.addSwitch( 's4', dpid='0000000000000004',protocols='OpenFlow13' )
s5 = self.addSwitch( 's5', dpid='0000000000000005',protocols='OpenFlow13' )
s6 = self.addSwitch( 's6', dpid='0000000000000006',protocols='OpenFlow13' )
s7 = self.addSwitch( 's7', dpid='0000000000000007',protocols='OpenFlow13' )
s8 = self.addSwitch( 's8', dpid='0000000000000008',protocols='OpenFlow13' )

#Coeur
self.addLink ( s8, s5 )
self.addLink ( s8, s6 )
self.addLink ( s8, s7 )
#distribution
self.addLink ( s1, s5 )
self.addLink ( s1, s6 )
self.addLink ( s1, s7 )
self.addLink ( s2, s5 )
self.addLink ( s2, s6 )
self.addLink ( s2, s7 )
self.addLink ( s3, s5 )
self.addLink ( s3, s6 )
self.addLink ( s3, s7 )
self.addLink ( s4, s5 )
self.addLink ( s4, s6 )
self.addLink ( s4, s7 )
```

```
        #acccess
        self.addLink( s1, h1 )
        self.addLink( s1, h2 )
        self.addLink( s2, h3 )
        self.addLink( s2, h4 )
        self.addLink( s3, h5 )
        self.addLink( s3, h6 )
        self.addLink( s4, h7 )
        self.addLink( s4, h8 )
def run():
topo = NetworkTopo()
net = Mininet( topo=topo )
net.start()

CLI( net )
net.stop()
if __name__ == '__main__':
setLogLevel( 'info' )
run()
topos = { 'mytopo': NetworkTopo }
```