



**Université
de Limoges**

**FACULTÉ
DES SCIENCES
ET TECHNIQUES**

Rapport de stage

Licence 3 : mention informatique

**Etude et mise en œuvre d'un réseau SDN
(Software Defined Network)**

Réalisé par :

❖ Thilleli Bibi

Encadré par :

❖ M. Emmanuel Conchon

Année universitaire : 2018/2019

Sommaire

Remerciements

Introduction

Chapitre 1 : software defined network

- I. Définition
- II. Architecture SDN
- III. Avantages du SDN

Le protocole openflow :

- I. Définition
- II. Genèse d'openflow
- III. Contrôleur
- IV. Commutateur openflow
- V. Messages openflow

Chapitre 2 : Mininet

- I. Définition
- II. Configuration utilisée
- III. Fonctionnement
- IV. Open vSwitch

Chapitre 3 : Mise en place d'un réseau SDN

- I. Introduction
- II. Définitions
- III. Configuration
- IV. Déroulement de tests

Remerciement

Avant de commencer le développement de ce rapport, il me paraît tout naturel de commencer par remercier les personnes qui m'ont permis d'effectuer ce travail.

Je tiens à exprimer ma gratitude et mes respects les plus sincères à mon encadrant, M. Emmanuel Conchon, pour avoir accepté si généreusement de m'encadrer et pour l'aide qu'il a bien voulu m'accorder tout au long de mon travail.

Je dédie ce modeste travail comme un témoignage d'affection, de respect et d'admiration à mes parents pour leur soutien et à toute ma famille.

Introduction

Les limites actuelles des architectures réseaux deviennent de plus en plus problématiques pour les administrateurs réseaux. En effet, les architectures IP traditionnelles sont, d'une part, complexes à configurer, d'autre part, difficiles à faire évoluer en raison du fort couplage qui existe entre le plan de contrôle et le plan de données des équipements du réseau.

Le plan de données est responsable de l'acheminement du trafic et du traitement des paquets selon des instructions précises. Ces instructions sont indiquées par le plan de contrôle, celui-ci est concerné par l'élaboration des informations d'une table de routage qui définit ce qu'il faut faire avec les paquets entrants.

Le manque d'innovation dans l'univers des réseaux a posé des contraintes importantes sur le développement des applications réseaux. Les principaux domaines d'innovation au cours des dernières années sont du côté de la programmabilité, du contrôle centralisé, et de la virtualisation. Ces innovations sont sous le nom commun SDN (ou software-defined network), et visent à apporter le niveau de flexibilité et de simplicité dont ont besoin les réseaux.

SDN a pour but de résoudre les problèmes des architectures réseaux IP actuels, notamment en les rendant plus programmables, et de permettre aux applications d'interagir directement avec les réseaux.

L'objectif de ce travail est de mettre en place et d'étudier un réseau SDN avec une zone démilitarisée (DMZ) en montrant leurs concepts et fonctionnements dans une plateforme simple qui permet de virtualiser un réseau, appelée Mininet.

Ce rapport comporte trois chapitres :

Le premier chapitre consiste à définir la technologie SDN et le protocole Openflow ainsi que leurs fonctionnements et leurs architectures.

Le deuxième chapitre décrit la plateforme Mininet, son architecture et son fonctionnement.

Enfin, le dernier chapitre sera réservé à mon propre travail qui consiste à mettre en place et de tester le fonctionnement d'un réseau SDN constitué d'une DMZ, de routeurs, firewall et d'hôtes appartenant à deux réseaux différents.

Chapitre 1 : Software Defined Network (SDN)

I. Définition :

Software Defined Network, ou réseau défini par logiciel, est un modèle d'architecture réseau qui sépare les fonctions de contrôle et celles de transfert de données du réseau. Dans ce modèle, les équipements réseau (switch, routeurs...) se contentent d'implémenter des règles de traitement des flux de données injectées par des applications. Plus besoin d'avoir sur ces équipements des protocoles de routage car le contrôle est dissocié du matériel et transféré à une application logicielle appelée « **contrôleur** » qui voit le réseau dans sa globalité et a pour objectif de contrôler le plan de données en exerçant des actions sur les équipements.

Le SDN est donc reconnu aujourd'hui comme une architecture permettant d'ouvrir le réseau aux applications. Cela intègre les deux volets suivants :

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

II. Architecture SDN :

I. Routage classique des réseaux :

Dans les réseaux classiques, le routeur travaille sur plusieurs plans décrits par la figure 1. Il contrôle le routage (recherche de la meilleure route...), il contrôle le flux de données, gère la configuration et la transmission. L'ajout ou la modification d'équipements et de règles dans le réseau est complexe, parfois longue, ce qui n'encourage pas l'évolution et la modification du réseau. Pour les équipements de réseaux traditionnels, le plan de contrôle et le plan de données sont localisés dans le même équipement.

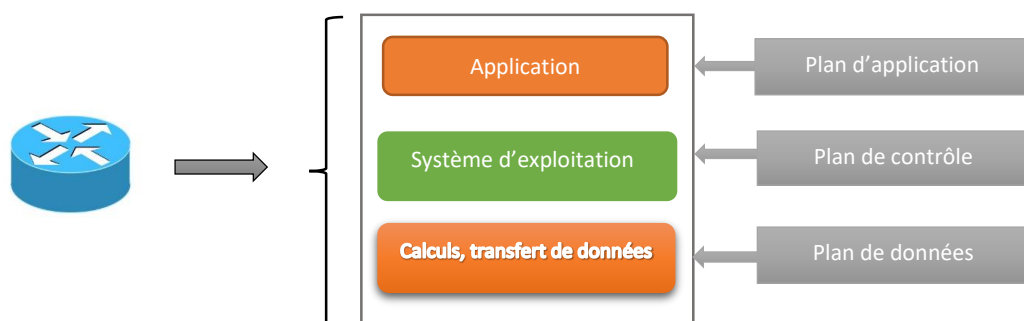


Figure 1 : Routage classique

II. Architecture SDN :

La figure suivante (Figure 2) représente une vue logique de l'architecture SDN : L'intelligence réseau est (logiquement) centralisée dans des logiciels placés dans des contrôleurs qui maintiennent une vue globale du réseau.

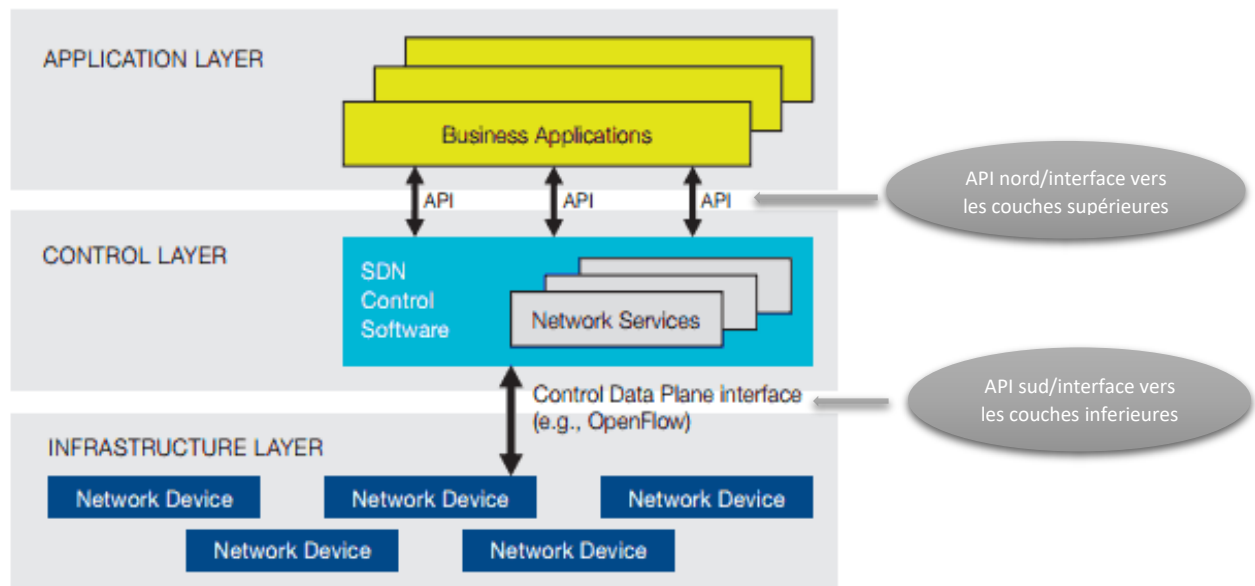


Figure 2 : Architecture SDN

L'architecture SDN est constituée de trois couches distinctes accessibles depuis des interfaces de programmation API :

- **Couche application** Il s'agit de la couche où les développeurs créent les applications qui donnent de la valeur ajoutée au SDN (automatisation, routage intelligent, sécurité et équilibrage de la charge...) en exploitant les informations collectées par le contrôleur.
- **Le plan de contrôle** (Control Plane) est logicielle basée sur le contrôleur SDN qui récolte des informations sur tout le réseau. Il offre une vue centralisée du réseau global et envoie des commandes à tous les équipements du réseau.
- **Le plan de données** contient les éléments de réseau responsables de l'acheminement du trafic et s'occupe du traitement des paquets selon les instructions du plan de contrôle.

Ainsi, dans une architecture SDN, la charge de calcul associée au contrôleur est retirée des routeurs.

Notion d'interface :

La programmation des équipements réseau nécessite sur ces derniers la capacité de recevoir des directives de l'extérieur. Pour cela des interfaces de programmation sont nécessaires : des API (Application Programming Interface).

Une interface représente un moyen pour une couche d'exposer des capacités à une autre couche, et donc d'assurer la communication entre les différentes couches du réseau SDN.

Il existe de nombreuses API pouvant agir sur différents éléments de l'équipement (plan de données, plan d'application...).

On distingue alors deux sortes d'interfaces :

- Les interfaces de programmation d'application (API) en direction du Sud (**SDN southbound API**) sont utilisées pour communiquer entre le contrôleur SDN et les commutateurs et les routeurs du réseau. OpenFlow en est l'exemple le plus connu.
- les interfaces de programmation d'application (API) en direction du nord (**SDN northbound API**) sont utilisées pour communiquer entre le contrôleur SDN et les applications intelligentes exécutées sur le réseau. Les API orientées vers le nord peuvent être utilisées pour faciliter l'innovation et permettre une automatisation efficace du réseau.

III. Avantages du SDN :

1. Réseaux programmables :

Avec SDN, il est plus simple de modifier les stratégies réseau car il suffit de changer une politique de haut niveau et non de multiples règles dans divers équipements de réseau. De plus, la centralisation du contrôle et la puissance de calcul élevée du contrôleur, simplifient le développement des fonctions les plus complexes. Cette capacité à programmer le réseau est l'élément clé du SDN.

2. Routage SDN :

SDN peut également être utilisé pour gérer les informations de routage de manière centralisée en déléguant le routage et en utilisant une interface pour le contrôleur.

3. Politique unifiée :

Avec son contrôleur, SDN garantit également une politique réseau unifiée et à jour. Puisque le contrôleur est responsable de l'ajout de règles dans les commutateurs, il n'y a aucun risque d'oublier un commutateur ou d'installer des règles incohérentes entre les dispositifs. En effet, on spécifie simplement une nouvelle règle et le contrôleur adaptera la configuration pour envoyer des règles cohérentes dans chaque dispositif.

4. Simplification matérielle :

SDN utilise des technologies standards pour contrôler les équipements du réseau tandis que la puissance de calcul n'est requise qu'au niveau du contrôleur. Ainsi, les équipements de réseau deviendront des produits à bas prix offrant des interfaces standards. Avec ce type de matériel, il serait également simple d'ajouter de nouveaux périphériques, puisqu'ils ne sont

pas spécialisés, de les connecter au réseau et de laisser le contrôleur les gérer conformément à la politique définie. Ainsi, le réseau devient facilement évolutif.

5. Flexibilité :

SDN apporte également une grande flexibilité dans la gestion du réseau. Il devient facile de rediriger le trafic, d'inspecter des flux particuliers ou de tester de nouvelles stratégies.

Le protocole Open Flow :

I. Définition :

OpenFlow est un protocole réseau qui permet de réaliser une architecture SDN. C'est le lien entre le plan de contrôle (contrôleur) et le plan de données (équipements réseau). Autrement dit, il permet de contrôler le comportement du plan de données d'une façon centralisée, dynamique et programmable.

L'échange de messages, entre ces deux plans, se fait au cours d'une session TCP établie via le port 6633 du serveur contrôleur.

Openflow est donc une composante du SDN. Son développement a commencé en 2007 et a été établie à l'origine par l'université de Stanford et l'université de Californie à Berkeley. Cette norme est maintenant définie par l'**Open Networking Foundation (ONF)**.

Selon Wikipédia, l'ONF est une organisation professionnelle à but non lucratif, financée par des sociétés telles que Deutsche Telekom , Facebook , Google , Microsoft , Verizon et Yahoo, visant la promotion de la programmation réseau avec le SDN et à standardiser le protocole OpenFlow et les technologies associées.

II. La genèse d'OpenFlow :

L'histoire d'OpenFlow est intéressante et permet de mieux comprendre son rôle fondamental dans la conception de l'architecture SDN et la virtualisation des fonctions réseau.

OpenFlow a été initié comme un projet à l'université de Stanford lorsqu'un groupe de chercheurs exploraient la manière de tester de nouveaux protocoles dans le monde IP mais sans arrêter le trafic du réseau de production lors des tests. C'est dans cet environnement que les chercheurs de Stanford ont trouvé un moyen de séparer le trafic de recherche du trafic du réseau de production qui utilisent le même réseau IP. Ils ont découvert que bien que les constructeurs de matériel réseau conçussent leurs produits différemment, tous utilisaient des tables de flux (flow table) afin d'implanter les services réseau tels que les NATs, les firewalls ...etc. Par ailleurs, bien que l'implantation des tables de flux différât entre ces constructeurs, les chercheurs ont découvert qu'ils pouvaient exploiter un ensemble de fonctions communes. Le résultat de l'équipe de cette recherche a été OpenFlow.

III. Contrôleur :

Les contrôleurs dans un réseau défini par logiciel (SDN) constituent le «cerveau» du réseau. C'est l'application qui agit en tant que point de contrôle stratégique dans le réseau SDN, gère le contrôle de flux vers les commutateurs et routeurs via les API sud « Southbound » et reçoit les ordres via les API en direction nord « Northbound ». Openflow se positionne comme une API sud agissant directement sur le plan de données, c'est-à-dire entre le contrôleur et les équipements réseaux.

Le contrôleur SDN permet donc d'implémenter rapidement un changement sur le réseau en traduisant une demande globale en une suite d'opérations sur les équipements réseau : ajout, suppression ou modification des entrées dans une table appelée table de flux, cela est illustré par la figure suivante :

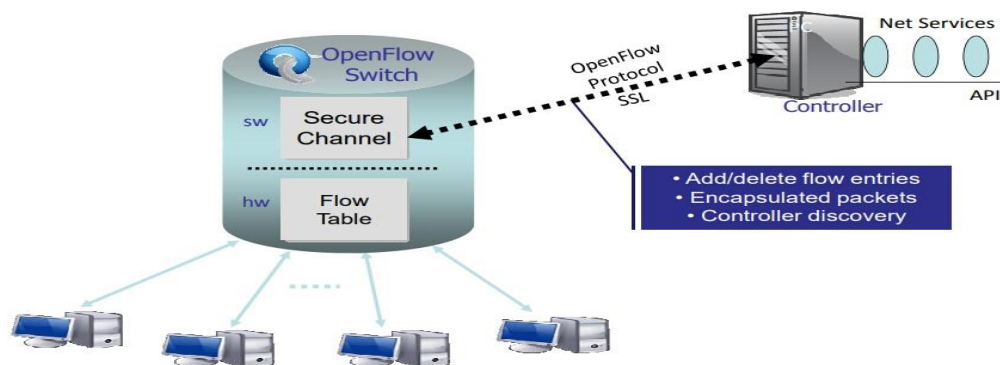


Figure 3 : contrôleur SDN

Parmi les contrôleurs existants :

- **NOX** : Initialement développé chez Nicira, NOX est le premier contrôleur OpenFlow. C'est Open-source et écrit en C ++. Il est actuellement à la baisse : il n'y a pas eu de changements majeurs depuis 2012.
- **POX** : C'est un contrôleur open-source écrit en Python, et comme NOX, fournit un cadre pour le développement et le test d'un contrôleur OpenFlow.
- **Floodlight** est un contrôleur open-source OpenFlow basé sur Java, Il est sous licence Apache. Il est facile à configurer et à montrer aussi de grandes performances. Avec toutes ses fonctionnalités, Floodlight est une solution complète.
- **OpenDaylight** est un projet de la Fondation Linux. C'est Un framework open source pour faciliter l'accès au logiciel de définition de réseau (SDN). Comme Floodlight, il peut également être considéré comme une solution complète.
- **Beacon** : est un contrôleur Java connu pour sa stabilité. Il a été créé en 2010 et est toujours maintenu. En raison de ses performances, c'est une solution fiable pour l'utilisation dans des conditions réelles.
- **Ryu** : contrôleur SDN capable de configurer les équipements réseaux en utilisant différents protocoles.

IV. Commutateur OpenFlow :

Un commutateur OpenFlow contient une ou plusieurs tables de Flux, qui traitent les paquets entrants et les commutent vers la destination, il s'appuie sur un Canal sécurisé pour communiquer avec le contrôleur.

Lorsqu'un paquet arrive au switch, les valeurs contenues dans ses en-têtes sont comparées aux différentes règles enregistrées dans la table de flux du switch et une de ces actions suivantes est exécutée par le commutateur :

- a. Relayer le paquet sur un port de sortie.
- b. Supprimer le paquet.
- c. Passer le paquet au contrôleur. Le paquet est encapsulé dans un message OpenFlow « **PACKET_IN** ».

Lorsque le contrôleur reçoit un message **PACKET_IN** du switch, lui demandant l'action à effectuer, le contrôleur analyse le paquet par l'intermédiaire des APIs du plan de contrôle. Ensuite, il donne les directives au switch grâce au message **PACKET_OUT**. Ceci est illustré par la figure 4.

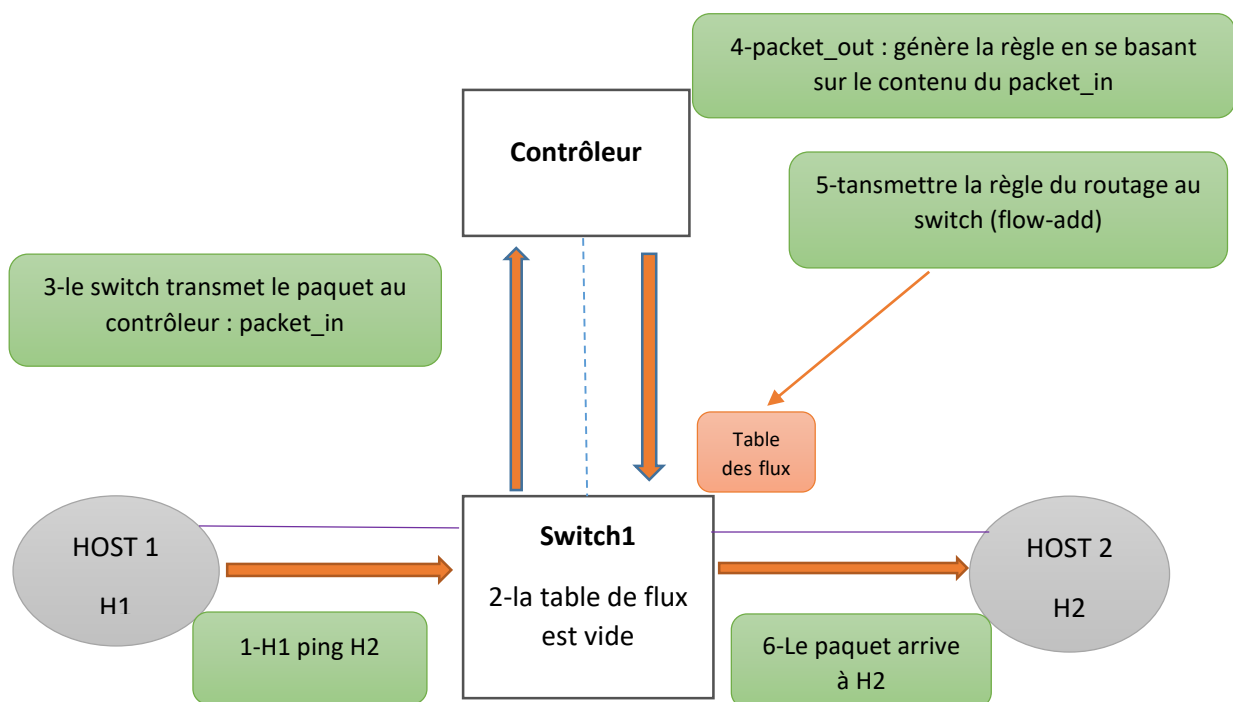


Figure 4 : Base d'échange entre le Switch et le contrôle via OpenFlow

Comportement du switch :

Recevoir un paquet ;

Vérifier la table de flux ;

Si la règle se trouvant dans le paquet existe dans la table des flux {

Envoyer le paquet suivant la règle ;

}

Sinon

Envoyer le paquet au contrôleur ;

Table de flux :

Chaque Table de Flux dans un Switch contient un ensemble d'entrées.

Champ de correspondance	instructions	compteurs
-------------------------	--------------	-----------

Les principaux éléments d'une entrée dans une table de Flux sont :

- **Champ de correspondance (Match fields)** : utilisé pour la recherche de l'entrée correspondante au paquet IP. Il est constitué des entêtes des paquets et des ports d'entrées.
- **Compteurs** : représentent des statistiques qui servent à la gestion des entrées de la table des flux. Pour ensuite décider si une entrée de flux est active ou non. Pour chaque table, chaque flux, chaque port, des compteurs de statistiques sont maintenus
- **Instructions** : actions à effectuer en cas de correspondance.

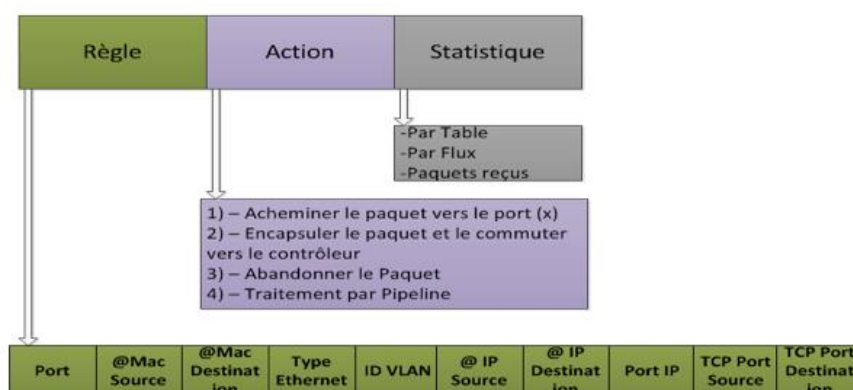


Figure 5 : table de flux

Parmi les instructions supportées on trouve :

- Apply-Action : appliquer l'action spécifique immédiatement, sans aucun changement de l'Action set. Cette instruction peut être utilisée pour modifier le paquet entre deux tables de flux ou pour exécuter plusieurs actions du même type.
- Clear-Actions : Effacer toutes les actions dans l'action set immédiatement.
- Write-Actions : Fusionner les actions dans l'action set actuel, si une des actions du même type existe déjà dans l'action set actuel, l'écraser, autrement l'ajouter.
- Write Metadata : Ecrire la valeur dans le champ Meta-data.
- Goto-Table : Indique la prochaine table de Flux dans le traitement pipe-line.

Action set :

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

Ce set est vide par défaut. Une entrée dans la table de Flux peut modifier l'action set en utilisant l'instruction Write-Action ou Clear-Action. Quand plusieurs actions du même type sont requises, l'instruction Apply-Actions peut être utilisée.

Traitement par Pipeline :

Pipeline OpenFlow : chaque Switch contient plusieurs tables de Flux, et chaque table contient plusieurs entrées. Le traitement pipeline OpenFlow décrit comment les paquets interagissent avec les Tables de Flux.

Les tables de Flux d'un Switch OpenFlow sont successivement énumérées, commençant par 0.

Le traitement par Pipeline commence toujours avec la première table de Flux : on fait correspondre le paquet selon les entrées de la table de Flux 0. D'autres tables de Flux peuvent être utilisées selon le résultat obtenu lors de la première correspondance.

Si le paquet correspond à une entrée dans la table de Flux, l'instruction sera exécutée. Les instructions dans les tables de Flux peuvent diriger le paquet vers une autre table de Flux en utilisant l'instruction **Goto**.

Une entrée x dans une Table de Flux n peut enchaîner le traitement du paquet en l'envoyant vers une autre table de Flux si seulement cette dernière dispose d'un numéro n supérieur à celui de la table où l'entrée x se trouve. Ainsi la dernière table

du pipeline n'inclue pas l'instruction **Goto**. Si l'entrée dans la table de Flux ne redirige pas le paquet à une autre table de Flux, le traitement pipeline s'arrête et le paquet est traité avec l'action correspondante.

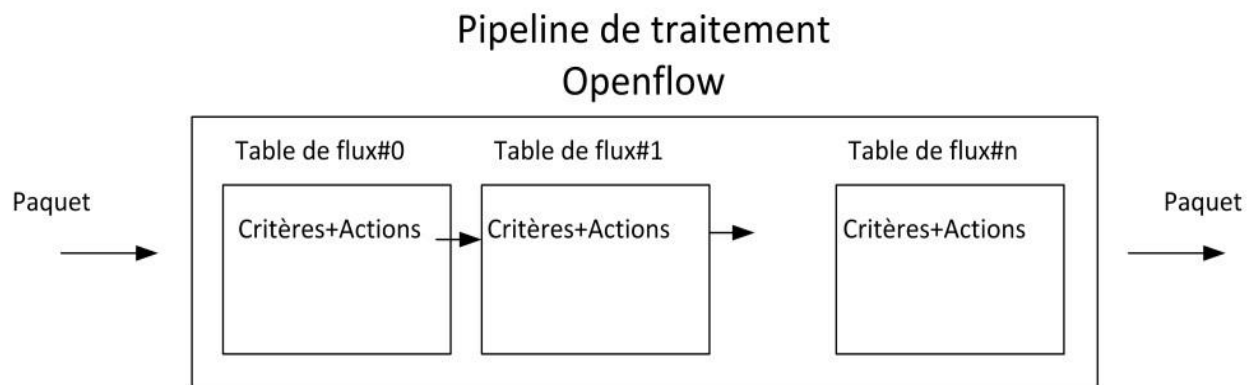


Figure 6 : traitement de flux par pipeline

IV. Messages openflow :

Le protocole OpenFlow supporte trois types de messages. Messages Contrôleur vers Switch, messages asynchrone et messages symétrique, chaque type a une sous-catégorie.

1. Les Messages depuis le Contrôleur vers Switch :

Sont initiés par le contrôleur, ils servent à gérer ou vérifier l'état du switch, ces types de messages peuvent ou non demander une réponse de la part du switch. Ils représentent la catégorie la plus importante de messages OpenFlow. Ils peuvent être représentés en cinq sous-catégories : *switch configuration*, *command from Controller*, *statistics*, *queue configuration* et *barrier* :

- a) **Les messages switch configuration** : consistent en un message unidirectionnel et deux paires de messages requête-réponse :
 - Le contrôleur émet le message unidirectionnel **SET_CONFIG** afin de positionner les paramètres de configuration du commutateur.
 - Le contrôleur utilise les messages **FEATURES_REQUEST** et **FEATURES_REPLY** afin d'interroger le commutateur au sujet des fonctionnalités qu'il supporte.
 - La paire de message **GET_CONFIG_REQUEST** et **GET_CONFIG_REPLY** est utilisée afin d'obtenir la configuration du commutateur.

b) Les messages **command from Controller** :

- Le contrôleur **utilise PACKET_OUT** afin d'émettre des paquets de données au commutateur pour leur acheminement. Soit, ils contiennent le paquet en entier, soit l'identificateur du buffer faisant référence au paquet stocké dans le switch. Ils doivent contenir aussi une liste d'actions à appliquer, s'il n'y a pas d'action définie le paquet sera détruit.
- Le contrôleur modifie les entrées de flux existantes dans le commutateur via le message **FLOW_MOD**.
- **PORT_MOD** est utilisé pour modifier l'état d'un port OpenFlow.

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

d) Queue configuration : La configuration de files d'attente associées à un port n'est pas spécifiée par OpenFlow. Un autre protocole de configuration doit être utilisé pour ce faire. Le contrôleur peut seulement interroger le commutateur via **QUEUE_GET_CONFIG_REQUEST** acquitté par **QUEUE_GET_CONFIG_REPLY** pour apprendre quelle est la configuration des files d'attente associées à un port afin de pouvoir acheminer des paquets sur des files d'attente spécifiques.

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

2. Les Messages asynchrones :

Les messages asynchrones sont envoyés par le switch vers le contrôleur pour signaler un changement d'état ou l'arrivée d'un paquet.

- **Packet-In** : Avec ce type de message le switch transfère le contrôle du paquet au contrôleur (lorsque aucune entrée de flux ne correspond au paquet entrant ou lorsque l'action de l'entrée correspondante spécifie que le paquet doit être relayé au contrôleur).
- **Flow-removed** : informe le contrôleur de la suppression d'une entrée dans la table de Flux. Cela arrive lorsque aucun paquet entrant n'a de correspondance avec cette entrée pendant un temporisateur spécifié par le contrôleur lors de la création de cette entrée au niveau de la table de flux du commutateur
- **Port-Status** : Informe le contrôleur d'un changement sur un port du switch.

3. Les Messages symétriques :

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

- **Echo** : pour la vérification de la connectivité entre switch et contrôleur.
- **Hello** : Ces messages sont échangés entre les deux une fois la connexion établie.
- **Error** : Utilisé pour (par le switch et par le contrôleur) signaler des problèmes de connexion.

4. Déroulement de l'établissement de connexion d'un switch OpenFlow au contrôleur :

Cas n°1 :

D'abord, il faut renseigner l'adresse IP du contrôleur au niveau du switch. Le switch envoie lors de son démarrage un paquet **OFPT_HELLO** avec le numéro de version d'OpenFlow supporté. Le contrôleur vérifie la version d'OpenFlow supporté par le switch et lui répond par un **OFPT_HELLO** en indiquant la version d'OpenFlow avec laquelle ils communiqueront. La connexion est alors établie.

Cas n°2 :

Dans ce cas le contrôleur ne supporte pas la version OpenFlow du switch. Il lui retourne donc un paquet **OFPT_ERROR** en indiquant que c'est le problème. La figure 8 illustre ce cas.

Cas n°3 :

Le switch envoie un paquet **OFPT_HELLO** à son contrôleur, si celui-ci ne répond pas il tente alors de joindre les autres contrôleurs qui lui ont été paramétrés. S'il n'arrive pas à les joindre, il utilise sa table de flux, si un paquet ne correspond à aucune entrée dans la table il le supprime.



Figure 7 : connexion switch-contrôleur : contrôleur(s) injoignable(s)

Chapitre 2 : Mininet

I. Définition :

Mininet est un émulateur de réseau, il permet de créer une topologie réseau qui se compose d'un ensemble d'hôtes, de switches, de contrôleurs et liens virtuels, sur la même machine. Il fournit la capacité de créer des hôtes, des commutateurs et contrôleurs via :

- Ligne de commande
- Script Python
- Interface interactive

Dans ce rapport on s'intéresse aux deux premières méthodes.

II. Configuration utilisée :

Afin de visualiser le fonctionnement du SDN, j'ai utilisé une machine virtuelle téléchargée depuis :

<https://github.com/mininet/mininet/wiki/Mininet-VM-Images>

Et accessible via VirtualBox, elle utilise le système d'exploitation Linux.

III. Architecture de Mininet dans VirtualBox :

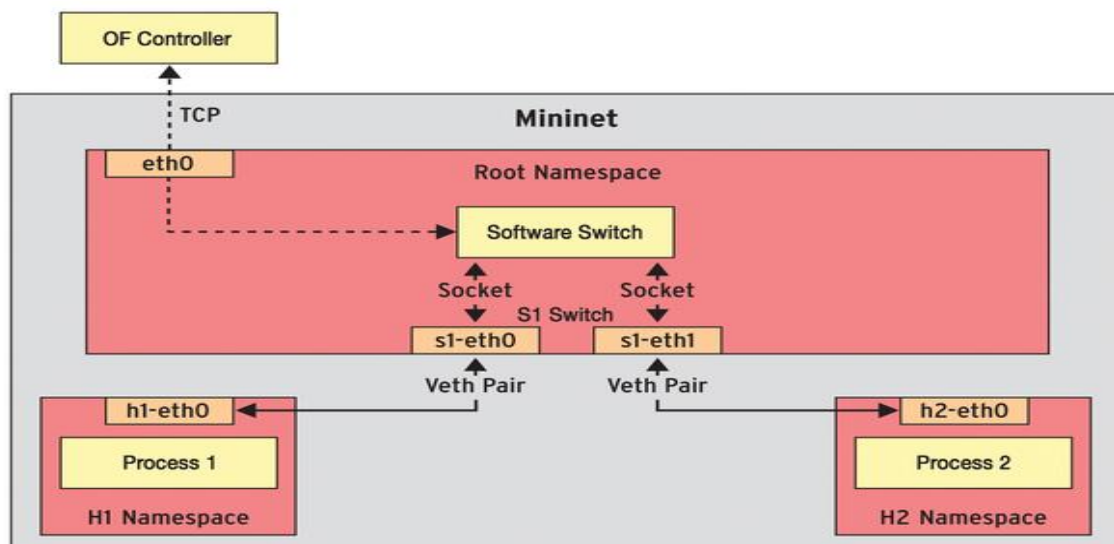


Figure 8 : architecture de mininet dans VirtualBox

Mininet émule switches et hôtes avec de simples processus.

Grace au principe de *network namespace*, le noyau permet de séparer ces processus avec des interfaces réseaux virtuelles individuelles.

La figure ci-dessus illustre un cas simple où Mininet utilise le principe de namespace :

Les hôtes H1 et H2 sont connectés à un seul switch (virtuel) S1 et communiquent entre eux grâce aux câbles Ethernet virtuels (Veth Pair).

Le switch est émulé dans le namespace racine, tandis que H1 et H2 ont leurs propre namespace.

IV. Fonctionnement :

Mininet permet de réaliser des réseaux virtuels en utilisant des commandes ou des scripts.

Création de topologies :

Une topologie **minimale** se crée avec la commande **mn** (`$ sudo mn`), elle est constituée d'un contrôleur, un switch et deux hôtes comme le montre la figure suivante :

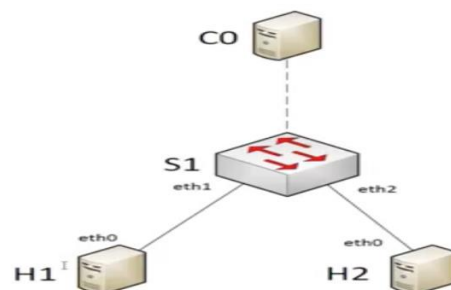


Figure 9 : topologie minimale.

Autres commandes :

En plus de cette topologie, Mininet fournit la topologie « **single** », « **linear** » et « **tree** » .

- Pour charger l'une de ces topologies on utilise l'option « **--topo** », par exemple :
`$ sudo mn --topo linear`
- On peut ajouter comme argument un chiffre qui indique le nombre d'hôtes à créer :
`$ sudo mn --topo single,4.`

Mininet Topology

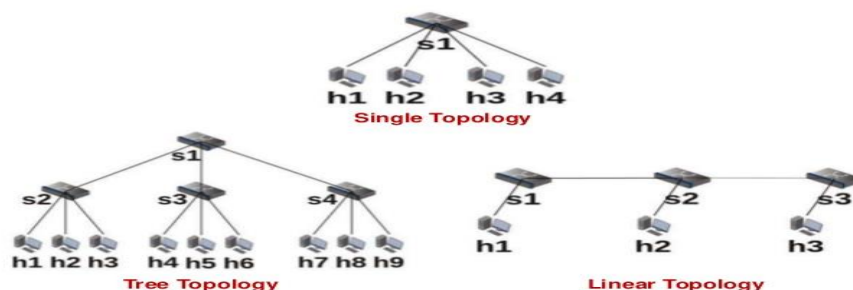
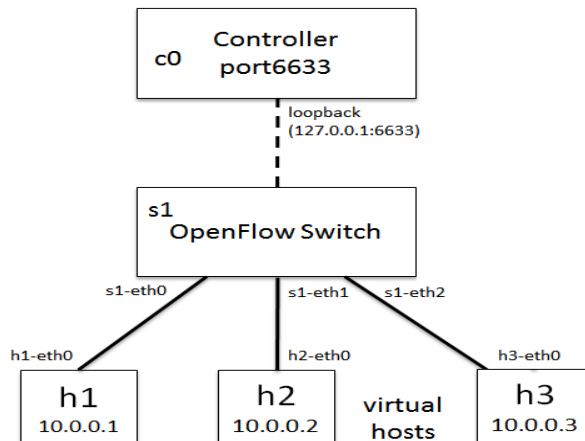


Figure 10 : topologie linear, tree ,single

- `$ sudo mn --topo single,3 --mac --switch ovsk --controller remote :`
 - Créer trois hosts, chacun avec une adresse IP différente.
 - Créer un seul OpenvSwitch avec trois ports.

- Connecter chaque host au switch via un câble Ethernet virtuel.
- L'adresse Mac de chaque host sera égale son adresse IP
(Exemple : @IP 10.0.0.2 @Mac 00 :00 :00 :00 :00 :02)
- Le switch est connecté à un contrôleur via une connexion SSH.

Cette commande crée la topologie suivante :



Si nous souhaitons personnaliser une topologie déjà créée, nous appliquons une des commandes suivantes :

- `self.addSwitch('s1')` : ajouter un switch s1 .
- `self.addHost('h1')` : ajouter un hôte h1 .
- `self.addLink(s1 ,h1)` : ajouter un lien entre s1 et h1.
- On peut tester si les paquets sont routés correctement avec la commande ping qui vérifie la connectivité : `Mininet> h1 ping h2` ou bien `Mininet> pingall`.
- Et pour analyser les règles insérées dans chaque commutateur, on utilise la commande `dpctl` : `Mininet> dpctl dump-flows`.

D'une façon générale, pour accéder à toutes les commandes utilisées sous Mininet :

`Mininet> help`.

Topologies personnalisées :

Pour créer une topologie personnalisée on utilise des scripts Python. Par exemple, si on veut créer trois hôtes h1, h2 et h3 tel que h1 et h2 sont connectés à un switch s1 et h3 connecté à un switch s2, voici le script Python correspondant :

```
class Test_Topo(Topo):
    def __init__(self):
        "Create topology"
        # Initialize topology
        Topo.__init__(self)
        # Add hosts and switches
        h1 = self.addHost('h1')
        h2 = self.addHost('h2')
        h3 = self.addHost('h3')
        s1 = self.addSwitch('s1')
        s2 = self.addSwitch('s2')
        # Add links
        self.addLink(h1,s1)
        self.addLink(h2,s1)
        self.addLink(h3,s2)
        self.addLink(s1,s2)

topos = { 'nomfichier': (lambda: Test_Topo()) }
```

Après, on sauvegarde le code dans un fichier nomfichier.py, et on exécute la commande suivante : `$ sudo mn --custom /chemin/vers/nomfichier.py --topo nomfichier`.

V. Open vSwitch :

Open vSwitch est adapté pour fonctionner comme un commutateur virtuel dans les environnements Virtuels. En outre, il supporte plusieurs technologies de virtualisation basées sur Linux, y VirtualBox. Open vSwitch est donc une implémentation logicielle d'un switch Ethernet. Il est constitué d'un service (ovs-vswitchd) et d'un module kernel (openvswitch_mod). Le service permet de commuter les paquets vers les bons ports virtuels, alors que le module kernel permet de capturer le trafic provenant des interfaces réseau.

Pour fonctionner comme n'importe quel switch, Open vSwitch utilise la notion de ports. Chaque port est constitué d'une ou plusieurs interfaces, qui correspondent à des interfaces (logiques ou physiques).

Interagir avec Open vSwitch :

- Création d'un switch : `ovs-vsctl add-br <nom du virtuel switch>`
- Pour connaître l'état global d'un switch : `ovs-vsctl show`
- Pour ajouter un flux directement dans la table de flux sans avoir à utiliser un contrôleur, nous utilisons la commande : `dpctl` :
 - `mininet> dpctl add-flow in_port=1,actions=output:2`
 - `mininet> dpctl add-flow in_port=2,actions=output:1`

Cette commande permet de transférer les paquets arrivant à port1 vers port 2 et vice versa.

Pour vérifier le contenu de la table de flux : `$ dpctl dump-flows`.

Chapitre 3 : Mise en place d'un réseau SDN

I. Introduction :

Les chapitres précédents présentent la technologie SDN et le protocole Openflow ainsi que leurs composantes et les outils utilisés pour réaliser un réseau SDN.

Dans ce chapitre, nous allons démontrer comment mettre en place un réseau SDN constitué d'une DMZ, de switches agissant comme routeurs et firewalls, et de deux hôtes de réseaux différents et cela à l'aide de la plateforme Mininet.

Les contrôleurs utilisés sont des contrôleurs POX.

Pour réaliser ce travail Nous avons suivi le « tutoriel Openflow ».

II. Définitions :

DMZ (demilitarized zone) : ou une zone démilitarisée est un sous réseau isolé séparant le réseau local (LAN) d'un autre réseau considéré comme moins sécurisé (comme internet). Cette séparation est faite par un firewall.

La DMZ contient les machines (ou services) du réseau interne susceptibles d'être accédées depuis l'extérieur, et qui n'ont pas besoin d'accéder au réseau local.

Tous les flux en provenance de l'extérieur sont redirigés vers la DMZ par le firewall. Le pare-feu bloquera donc les accès au réseau local à partir de la DMZ pour garantir la sécurité. Une zone démilitarisée va nous permettre donc d'échanger avec l'extérieur, sans mettre en danger notre réseau interne.

Firewall : ou pare-feu est un élément du réseau, sous forme d'un logiciel ou d'un composant matériel, qui a pour fonction de sécuriser un réseau en définissant les communications autorisées ou interdites. Il surveille et contrôle les applications et les flux de données.

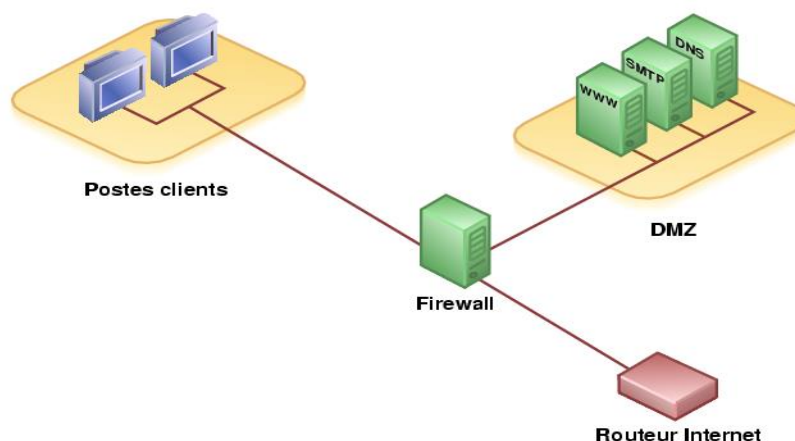
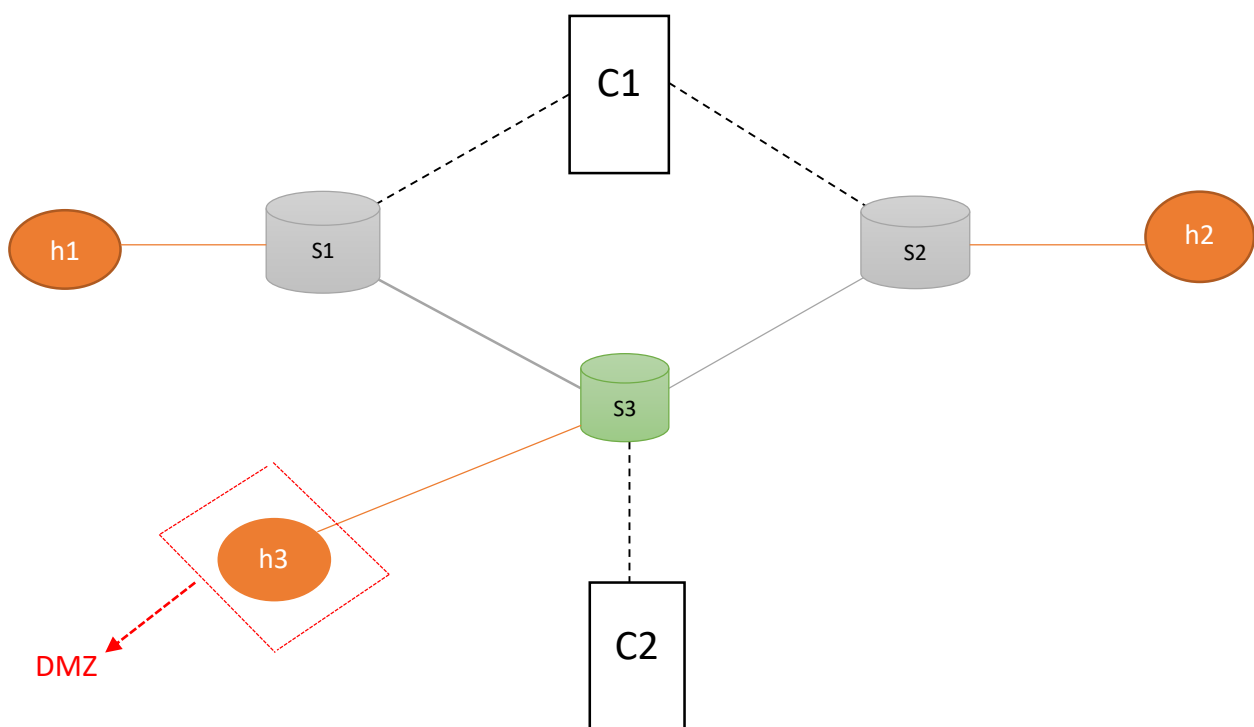


Figure 11 : zone démilitarisée

III. Configuration utilisée :

La configuration utilisée ; montrée dans la figure ci-dessous ; est composée de deux contrôleurs POX (c1 et c2), trois switches Openflow : s1 et s2 agissant comme routeurs et le dernier (s3) comme firewall, ainsi que trois hôtes (h1, h2, h3).

- h1, h2, h3 sont reliés respectivement aux switches s1, s2, s3, ces derniers sont reliés entre eux.
- h1 et h3 appartiennent au même réseau leurs adresses IP sont respectivement : 10.0.1.2 et 10.0.1.3. h3 représente la DMZ, il est isolé de h1 et h2 par un firewall (s3).
- l'adresse IP de h2 est : 10.0.2.2
- Le contrôleur c1 est connecté aux switches s1 et s2
- Le contrôleur c2 est connecté à s3.



IV. Déroulement des tests :

La topologie permettant de créer ce réseau est définie dans le fichier « topologie.py ».

Le fichier permettant de configurer s1 et s2 est « advancedrouter.py ».

Finalement la configuration de s3 est définie dans le fichier « firewall.py » et les règles de filtrage du firewall (s3) sont listées dans le fichier « configuration.config ».

Ces règles autorisent la connexion de h1 vers h3 (DMZ) sur n'importe quel port, elles autorisent aussi la connexion entre h2 et h3 dans les deux sens sur le port 5004.

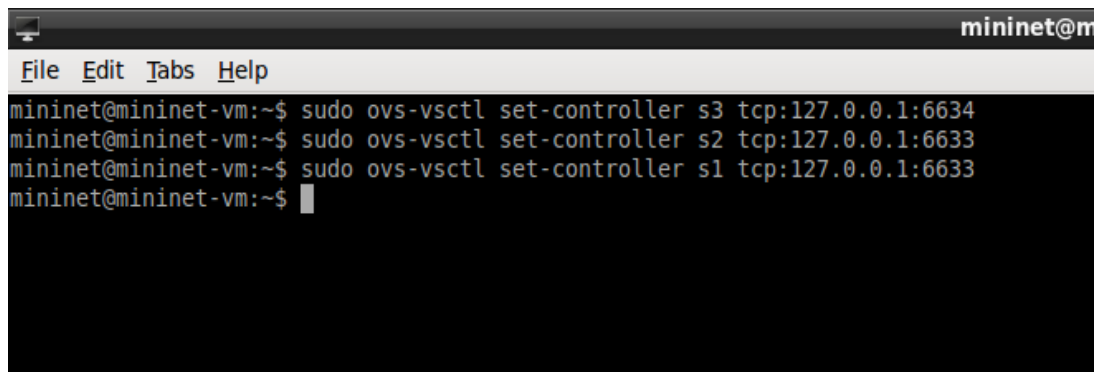
1. Dans un terminal 1 on exécute la commande suivante pour créer la topologie :

- `$ sudo python topologie.py .`

```
mininet@mininet-vm:~$ sudo python topologie.py
*** Creating (reference) controllers
*** Creating switches
*** Creating hosts
*** Creating links
*** Starting network
*** Configuring hosts
h1 h2 h3
*** Testing network
*** Ping: testing ping reachability
h1 -> X X
h2 -> X X
h3 -> X X
*** Results: 100% dropped (0/6 received)
*** Running CLI
*** Starting CLI:
```

2. Pour spécifier à chaque switch le port sur lequel le contrôleur correspondant est à l'écoute on exécute les commandes suivantes dans un deuxième terminal :

- `$ sudo ovs-vsctl set-controller s1 tcp :127.0.0.1 :6633`
- `$ sudo ovs-vsctl set-controller s2 tcp :127.0.0.1 :6633`
- `$ sudo ovs-vsctl set-controller s3 tcp :127.0.0.1 :6634`



```
mininet@mininet-vm:~$ sudo ovs-vsctl set-controller s3 tcp:127.0.0.1:6634
mininet@mininet-vm:~$ sudo ovs-vsctl set-controller s2 tcp:127.0.0.1:6633
mininet@mininet-vm:~$ sudo ovs-vsctl set-controller s1 tcp:127.0.0.1:6633
mininet@mininet-vm:~$
```

3. Pour lancer le contrôleur connecté à s1 et s2 (routeurs) sur le port 6633, on exécute la commande ci-après ; dans un terminal 3.

- `./pox.py openflow.of_01 --port=6633 log.level --DEBUG misc.advancedrouter`

4. Dans un terminal 4 on exécute la dernière commande qui nous permettra de lancer le contrôleur c2 sur le port 6634, et avec comme paramètre le nom de fichier de configuration contenant les règles définies par le firewall.

- `./pox.py openflow.of_01 --port=6633 log.level --DEBUG misc.advancedrouter --configuration='configuration.config'`

```

mininet@mininet-vm:~/pox$ ./pox.py openflow.of_01 --port=6634 log.level --DEB
UG misc.firewall --configuration='lab.config'
POX 0.2.0 (carp) / Copyright 2011-2013 James McCauley, et al.
DEBUG:core:POX 0.2.0 (carp) going up...
DEBUG:core:Running on CPython (2.7.6/Oct 26 2016 20:32:47)
DEBUG:core:Platform is Linux-4.2.0-27-generic-i686-with-Ubuntu-14.04-trusty
INFO:core:POX 0.2.0 (carp) is up.
DEBUG:openflow.of_01:Listening on 0.0.0.0:6634
INFO:openflow.of_01:[00-00-00-00-00-03 1] connected
DEBUG:misc.firewall:Controlling [00-00-00-00-00-03 1]
['10.0.1.2', '57182', '10.0.1.3', '5001']
['10.0.1.3', '52228', '10.0.2.2', '5004']

```

Après l'exécution de ces commandes, on aura finalement la topologie définie au départ, Et on peut visualiser le résultat et examiner le bon fonctionnement des équipements réseau utilisés et cela en testant la connectivité entre les hôtes.

Exemple : test de connectivité entre h2 et h3 :

```

root@mininet-vm:~# iperf -p 5004 -c 10.0.2.2
-----
Client connecting to 10.0.2.2, TCP port 5004
TCP window size: 85.3 KByte (default)
-----
[ 21] local 10.0.1.3 port 52412 connected with 10.0.2.2 port 5004

```

Demande de connexion de h2 vers h3

```

root@mininet-vm:~# iperf -s -p 5004
-----
Server listening on TCP port 5004
TCP window size: 85.3 KByte (default)
-----
[ 22] local 10.0.2.2 port 5004 connected with 10.0.1.3 port 52412

```

h3 est connecté avec h2

Test de connectivité entre h3 et h1 :

"Node: h1"	"Node: h3"
<pre> root@mininet-vm:~# iperf -s ----- Server listening on TCP port 5001 TCP window size: 85.3 KByte (default) ----- </pre>	<pre> root@mininet-vm:~# iperf -c 10.0.1.2 connect failed: Connection timed out root@mininet-vm:~# </pre>

Connexion de h3 à h1 refusée

CONCLUSION

Ce stage de deux mois, effectué à la faculté des sciences et techniques de l'université de Limoges, dans le cadre de la troisième année licence informatique, a été une chance et une formidable opportunité de découvrir le fonctionnement des réseaux SDN ainsi que leurs avantages.

Grâce au SDN, les réseaux sont maintenant programmables, et la mise à disposition d'interfaces de programmation d'applications va permettre de programmer les équipements du réseau en utilisant différents langages. En outre, L'automatisation ouvre beaucoup de nouvelles perspectives. C'est le pouvoir de renforcer l'agilité, la sécurité ou encore les capacités du réseau grâce à de nouvelles solutions logicielles.

La technologie SDN devrait révolutionner les architectures des réseaux, et permettre de déployer des nouveaux services de manière beaucoup plus rapides. Le protocole Openflow permet une programmation simplifiée via une interface standard. La facilité de programmation permet de concevoir une couche de contrôle afin de centraliser l'intelligence dans le réseau et de fournir la programmabilité promise par SDN.

Référence :

1. https://indico.wacren.net/event/20/contributions/146/attachments/87/105/article_wacren_2016_.pdf .
2. <https://www.sdxcentral.com/networking>.
3. <https://www.supinfo.com/articles/single/1010-protocole-openflow>.
4. https://www.ssi.gouv.fr/uploads/2015/06/SSTIC2015-Article-risques_openflow_et_sdn-tury.pdf.
5. www.efort.com/r_tutoriels/OPENFLOW_EFORT.pdf.
6. <https://www.slideshare.net/EdouardDEBERDT/introduction-au-software-defined-networking-sdn>.
7. <https://www.miscmag.com/sdn-ou-comment-le-reseau-sautomatise-a-grande-echelle-partie-1-2/>.
8. <https://wapiti.telecom-lille.fr/commun/ens/peda/options/st/rio/pub/exposes/exposesrio2009-ttnfa2010/cloarec-ferreira/controller.html>.
9. mininet.org.
10. <https://github.com/mininet/openflow-tutorial/wiki>.
11. www.linux-magazine.com/index.php/layout/set/print/Issues/2014/162/Mininet.
12. <https://www.slideshare.net/sdnrgitb/eksperimen-custom-topology>.
13. cookieconnecte.fr/2016/07/02/firewall-lessentiel-8-minutes/.