SOFTWARE DEFINED NETWORK

Travail de semestre

Réalisé par : Ouafae IFAKREN

Encadré par : Gérald Litzistorf professeur HES

Table des matières

Résumé 4

Introduction 5

Enoncée et cahier de charges 6

CHAPITRE I THEORIE 7

I- DEFINITIONS 8

1- SDN : SOFTWARE DIFINED NETWORK 8

1-1 Définition : 8

1-2 Différence entre SDN et les réseaux traditionnels : 9

2- OpenFlow 10

2-1 Définition : 10

2-2 Réseau OpenFlow : 10

2-3 Le contrôleur OpenFlow 10

2-4 le fonctionnement d'OpenFlow 11

2-4-1 Le comportement du Switch et du contrôleur 11

2-5 la table de flux : 12

2-6 Les messages Openflow : 13

CHAPITRE II MININET 15

I- DEFINITION : 16

II- CONFIGURATION REQUISE 16

III- ARCHITECTURE 17

1 Architecture de Mininet 17

2 Architecture de Mininet dans VirtualBox 18

3- Installation et configuration 18

IV- FONCTIONNEMENT DE MININET 20

1 La typologie des réseaux 20

1-1 Ligne de commande 20

1-2 Interface graphique 21

CHAPITRE III OF EN PRATIQUE 23

I- SCENARIO UTLISE 24

1- Configuration physique 24

2- Configuration virtualisée 24

II- DEROULEMENT DES TESTS 25

III- Statistiques de la table de flux 25

V- LA CAPTURE WIRESHARK 26
1- Syntaxe des captures wireshark 26
2- Les captures 26
Déroulement des messages 33
CHAPITRE IV FIREWALL 36
I- SCENARIO 37
II- ALGORITHME 37
III- LE CODE 37
IV- DEROULEMENT DES TESTS ET RESULTATS 38
CONCLUSION 40
Problèmes rencontrés 41
REFERENCES 42

Résumé

Ce rapport comporte trois parties

La première partie consiste à définir d'une façon la technologie SDN, le protocole OpenFlow ainsi que leurs fonctionnements et leurs architectures

La deuxième partie : décrit la plateforme MIninet, son architecture, son fonctionnement et les différentes messages Openflow.

La troisième partie : décrit le fonctionnement des messages Openflow en pratique.

La quatrième partie : contient un script qui ajoute la fonctionnalité de firewall au contrôleur. Pour cela Cette partie permet d'avoir un petit aperçu sur les fonctions que noua pourrions attribuer au contrôleur, en ajoutant des fonctionnalités par le biais des scripts.

Enfin, une conclusion mettant en question l'aspect sécurité dans le protocole openflow et la technologie SDN *(Software Defined Networking)*

Introduction

Ce projet a pour but d'étudier ce protocole et son fonctionnement dans une plateforme simple qui permet de virtualiser un réseau et de tester le protocole sus mentionné dans différentes configurations.

Nous avons utilisé la plateforme *Mininet* qui peut virtualiser tout un réseau sur un seul *Kernel* .

Pour cela, nous avons besoin seulement d'une machine, *Virtualbox* et l'image *Mininet .*

Des tests simples nous ont permis d'étudier OF et ses différents messages et composants ainsi que les fonctions qui peuvent être rajoutés par une simple programmation.

Pour cela, j'ai choisi le contrôleur *POX* qui m'a permet de découvrir la programmation *python.*

Enoncée et cahier de charges

h e p i a
Haute école du paysage, d'ingénierie et d'architecture de Genève

Année académique 2015-2016
Projet de semestre

VIRTUALISATION
SOFTWARE DEFINED NETWORK

Descriptif :

La virtualisation du réseau offre des perspectives intéressantes grâce à des modules logiciels (commutateur, routeur, ...) pilotés de manière centralisée par un contrôleur via le protocole OpenFlow.

Elle devient incontournable dans les architectures de type Cloud comme OpenNebula ou OpenStack.

Les premières mises en œuvre seront basées sur le projet Mininet

Travail demandé :

Cette étude comprend les étapes suivantes :

1) Apprentissage à partir des exemples disponibles sur http://mininet.org/

2) Etude du protocole OpenFlow et des commandes OpenvSwitch utilisés au point 1)

3) Configuration et tests d'un dispositif de sécurité
Choix du contrôleur

4) Rapport comprenant :
- le descriptif de l'architecture utilisée,
- la marche à suivre des démonstrations choisies,
- l'analyse des commandes réseau (OpenFlow - OpenvSwitch)
- les avantages de SDN,
- les principales difficultées rencontrées,
- une conclusion.

Sous réserve de modification en cours du projet de semestre

Candidat :
Mme IFAKREN OUAFAE
Filière d'études : ITI
Département : ITI

Professeur(s) responsable(s) :
Litzistorf Gérald

En collaboration avec :
Projet de semestre soumis à une convention de stage en entreprise : non
Projet de semestre soumis à un contrat de confidentialité : non

Timbre de la direction

CHAPITRE I THEORIE

I- DEFINITIONS

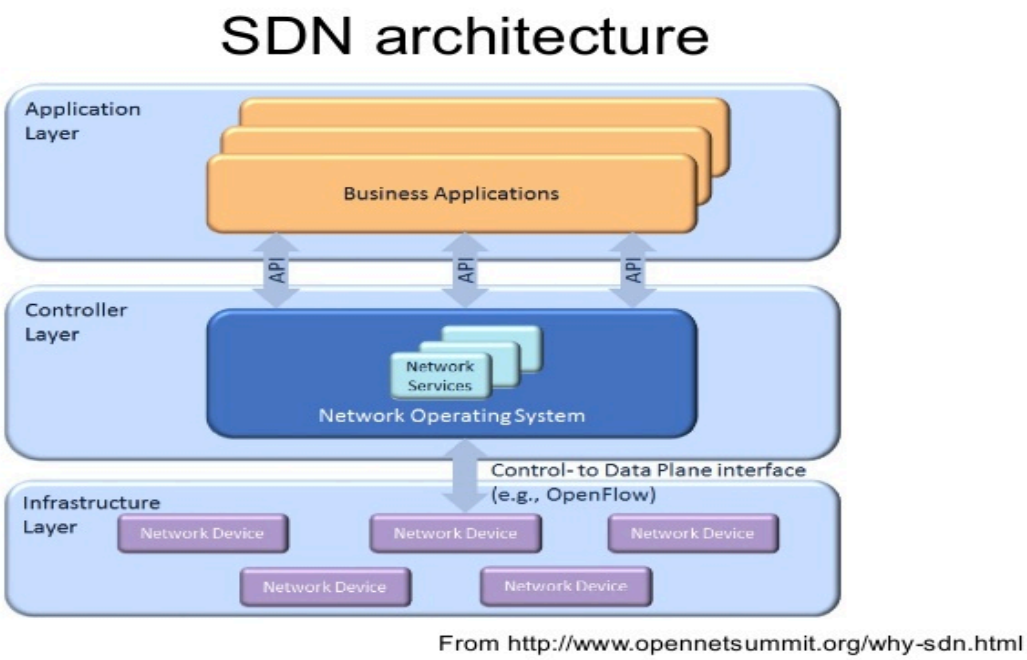
1- SDN : SOFTWARE DIFINED NETWORK

1-1 Définition :

C'est un modèle d'architecture réseau qui ajoute un niveau d'abstraction aux fonctionnalités des équipements réseau afin de pouvoir les gérer de façon globale. La partie décisionnelle des équipements est séparée de leur partie opérationnelle et déportée vers un unique point de contrôle qui dirige l'ensemble de façon cohérente. Ce découplage permet de déployer le plan de contrôle sur des plateformes dont les capacités sont plus grandes que celles des commutateurs réseaux classiques. Enfin, cette abstraction à travers une API réseau standard permet un développement de services réseaux à forte valeur ajoutée affranchi des spécificités des équipementiers. [1]

Il y a trois composantes importantes qui définissent l'architecture de SDN

- La décorréllation du plan de contrôle et du plan de données
- L'abstraction du réseau physique
- La programmabilité du réseau



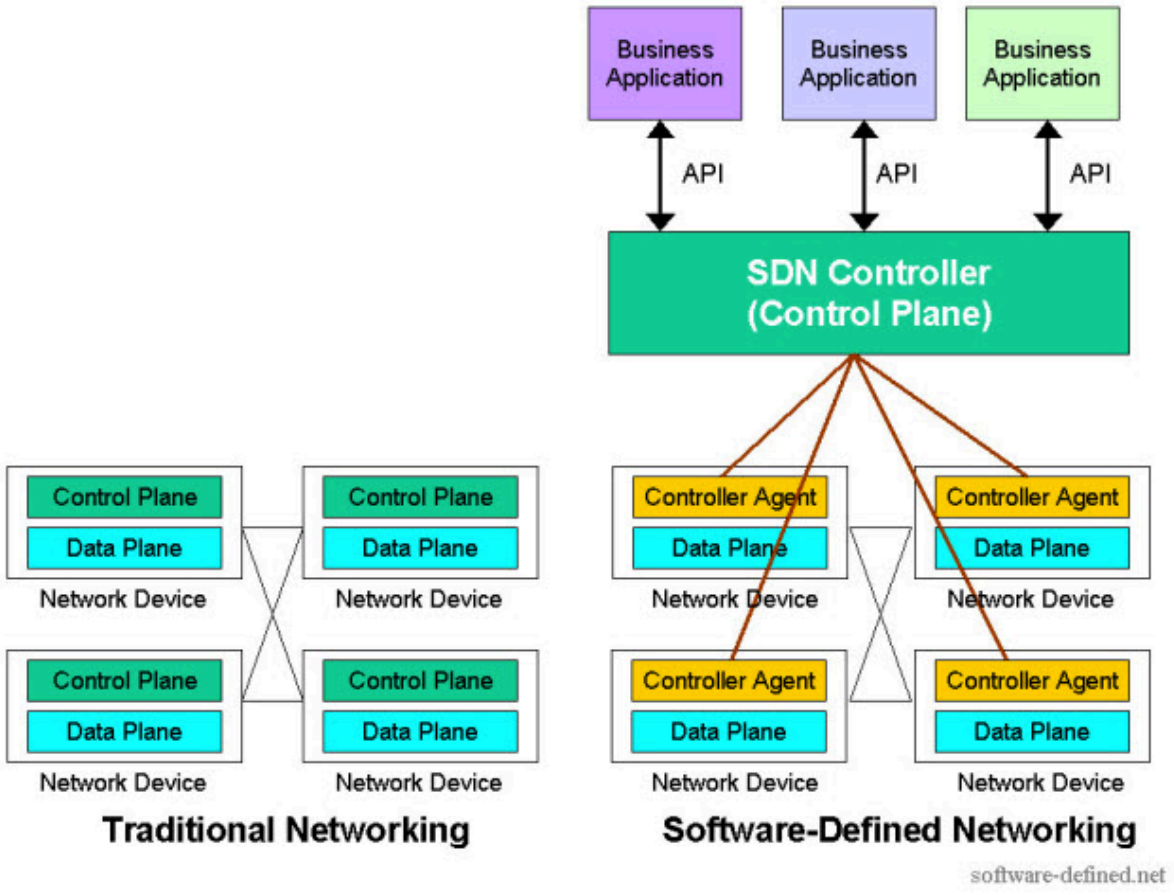
From http://www.opennetsummit.org/why-sdn.html

[1] https://fr.wikipedia.org/wiki/Software_defined_networking

1-2 Différence entre SDN et les réseaux traditionnels :

En déplaçant le plan de contrôle dans la partie logicielle, SDN permet un accès et une administration dynamique du réseau.

- Modèle de réseau basé sur un contrôleur.
- Le contrôle du réseau est directement programmable.
- Management centralisé : l'intelligence du réseau est centralisée dans un logiciel appelé SDN controller, qui maintient une vue globale du réseau.
- Configuration automatique : SDN permet aux administrateurs réseaux de configurer, administrer, sécuriser et optimiser les réseaux rapidement grâce à des programmes SDN dynamiques et automatisés.
- Séparation entre le plan de contrôle et le plan de données
- L'abstraction du contrôle du relayage permet aux administrateurs d'ajuster dynamiquement le réseau au trafic.



2- OpenFlow

2-1 Définition :

Openflow est le protocole utilisé par SDN (Software-Defined Networking) .

Le développement d'*OpenFlow* a commencé en 2007 dans le cadre d'une collaboration entre les mondes de l'université et des affaires. Etablie 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)*. HP a été un leader de la technologie *OpenFlow* depuis ses débuts et est un membre fondateur de *l'ONF*.[2]

2-2 Réseau OpenFlow :

Les principaux composants basés sur un contrôleur OpenFlow sont

- Des switchs compatibles avec le protocole Openflow
- Des serveurs capables d'exécuter le processus du contrôleur
- Une base de données contenant une vue générale de la topologie du réseau

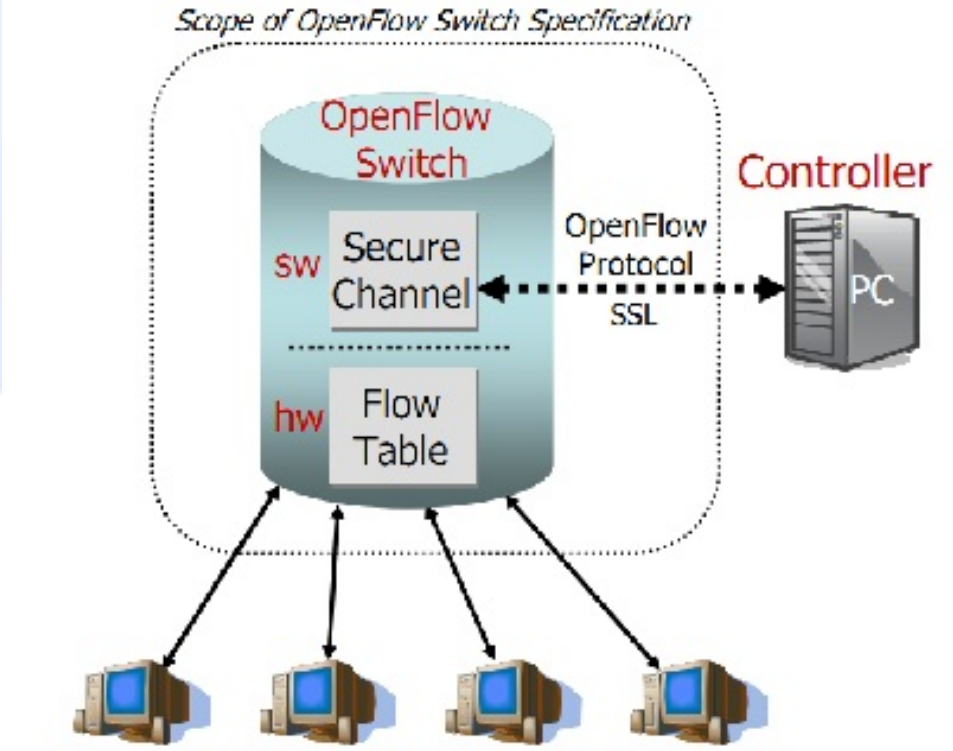


Figure : http://keepingitclassless.net/2013/10/introduction-to-open-vswitch/

2-3 Le contrôleur OpenFlow

Un contrôleur OF[3] ajoute ou supprime les entrées de flux dans une table dédiée à cela, appelée la table de flux.

---

[2] http://h17007.www1.hp.com/ch/fr/solutions/technology/openflow/index.aspx

[3] OpenFlow

Chaque entrée se compose :

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

Parmi les contrôleurs existants :

- **NOX et POX** : sont les deux premiers contrôleurs OpenFlow. NOX -> C++ et POX -> Python
- **Beacon** : est un contrôleur OpenFlow rapide, multi-plateforme, modulaire qui est basé sur Java.
- **Floodlight :** est un contrôleur supporté par BigSwitch, Il est sous licence Apache et écrit en java
- **Ryu** : est un contrôleur SDN capable de configurer les équipements réseaux en utilisant OpenFlow, NetConf et OF-config.

2-4 le fonctionnement d'OpenFlow

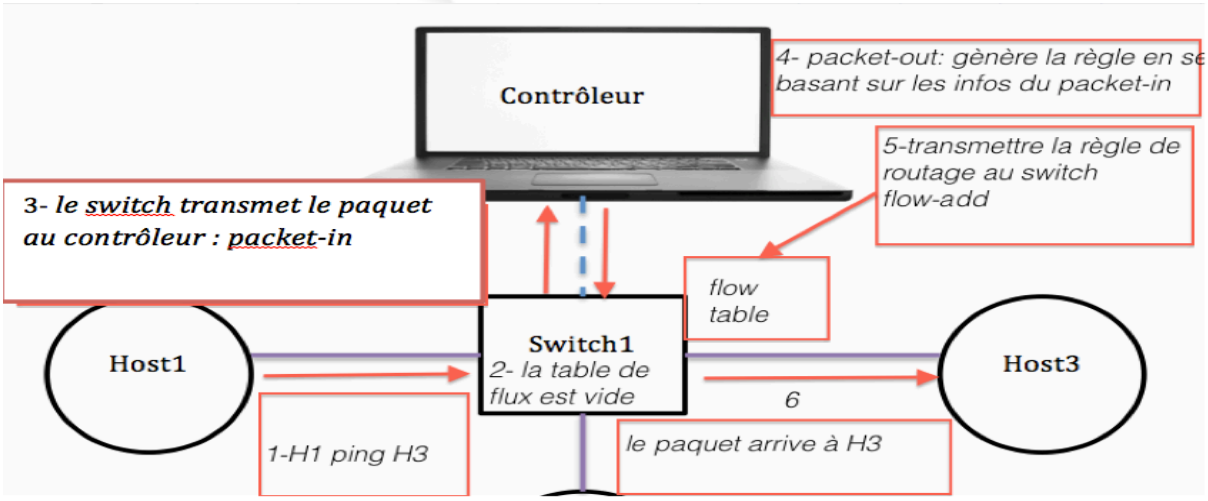
2-4-1 Le comportement du Switch et du contrôleur

```
recieve packet ;
check flow table ;
if {
    the packet rule is in the flow table ;
    send out the packet following the rule ;
}
else
   send packet to the coltroller
```

```
Receive packets from the switch
If
Non Ethernet packet  // from the switch
        Packet dropped ;
Else {
        Check Mac and Ip adress

Send rule to the switch  // flow-mod
        (exemple : source : 10.0.0.1- Destination : 10.0.0.3-
Port :1)
}
Switch can send the packet to the destination
```

2-4-3 Diagramme de flux des messages OF



2-5 la table de flux :

2-5-1 Définition

La table de flux est extrêmement simple dans le protocole OpenFlow. Chaque entrée de la table de flux correspond au tuple suivant :

Une définition d'un champ d'en-tête

La correspondance avec le champ d'en-tête offre plusieurs possibilités. En effet, il est tout à fait possible de rechercher les correspondances avec les en-têtes classiques des paquets IP (adresses mac, ip source, ip destination, type, ...).

Il est également possible d'effectuer des correspondances au bit près. Ainsi, il est possible pour un chercheur qui souhaite développer un nouveau protocole (autre qu'IP) de définir lui-même ses en-têtes et de pouvoir effectuer ensuite des correspondances sur le switch OpenFlow.

Une action à effectuer en cas de correspondance
* Transférer le paquet sur le port n°X
* Envoi du paquet au contrôleur
* Destruction du paquet
* Faire suivre le processus normal identique au trafic de production

Un certain nombre de statistiques
Il est possible de disposer d'un certain nombre de statistiques. Une partie d'entre-elles servent à la gestion des entrées de la table de flux. Voici une liste des statistiques que l'on peut retrouver dans les entrées de la table de flux :

* *PerTable*
* *ActiveEntries 32*
* *PacketLookups 64*
* *PacketMatches 64*
* *PerFlow*
* *ReceivedPackets 64*
* *ReceivedBytes 64*
* *Duration(seconds) 32*
* *Duration(nanoseconds) 32*

IP src	IP dest	IP port	TCP s-port	TCP d-port

Switch Port	MAC src	MAC dest	ETH type	VLAN Id

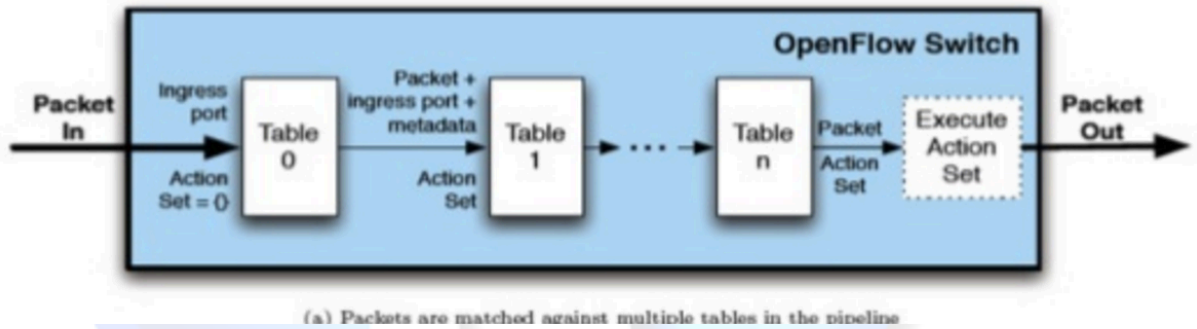
2-5-2 fonctionnement

packet in start at flow table 0
for *i =0 ; i= n {*
if

match table i {

update counters
execute instruction set
update action set ;
update packet match fields ;
update metadata ;

}
else
{
based on table configuration , do one
send to controller
drop
contunue to next
}



(a) Packets are matched against multiple tables in the pipeline

2-6 Les messages Openflow :

https://www.osapublishing.org/jocn/abstract.cfm?uri=jocn-5-10-A57

OFPT_HELLO: Switch <--> controller
le switch initie la connexion avec le contrôleur en TCP
Lorsque la connexion est établie, le switch et le contrôleur envoient , tous les deux, un message OFPT_HELLO avec la version Openflow négociée.

Feature Echo- Request : **Switch <--> controller**
Est utilisé pour échanger des informations sur la latence et la bande passante. Echo Request Timeout indique la déconnexion. Si la mise en œuvre OpenFlow est multi-couches, Echo_request devrait être mis en œuvre dans le niveau le plus bas (le plus proche du noyau), car il pourrait être utilisé pour le débogage dans certaines situations.

Feature Echo_ Reply : Switch <--> controller
Le switch répond avec ce message en indiquant l'ID de datapath et les capacités du switch , que le contrôleur pourrait accepter ou refuser .

OFPT_SET_CONFIG : Controller-->Switsh
Message envoyé par le contrôleur. il contient la longueur du paquet maximale que le switch devrait envoyer au contrôleur.

***Packet-in :* Switch --> controller**
Ce message se compose par un entête, suivi d'un buffer-id, une valeur unique indiquant le numéro du buffer ou le paquet est stocké. La longueur du paquet capturé est indiquée par total-len.
In-port est le port par lequel le paquet a été reçu.
Reason dans ce cas indique table de flow non trouvé
Enfin, le paquet capturé commence juste après le champ padding

Flow Mod : Controller-->Switsh
Ceci est l'un des principaux messages, il permet au contrôleur de modifier l'état d'un switch OpenFlow. Tous les messages FlowMod commencent avec l'en-tête de OpenFlow standard, contenant la version et le type des valeurs appropriées, suivis par la structure FlowMod.

Packet_out : Controller-->Switsh
Message envoyé par le contrôleur au switch .
Ce message permet au contrôleur d'injecter le paquet reçu dans le plan de données du switch
Buffer_id : est le numéro du buffer ou se trouve le paquet. la même valeur que dans le message précédent
In_port : est utilisé comme le port d'arrivée

CHAPITRE II MININET

Ce chapitre décrit

- L'architecture de Mininet et son fonctionnement
- Les messages OF et leurs structures
- Les différents types de controller et leur rôle dans la gestion de ces messages

I- DEFINITION :

Mininet est un émulateur de réseau qui permet de créer des *hosts, switch*, contrôleurs et liens virtuels sur un même noyau Linux.

Il fournit la capacité de créer des hôtes, les commutateurs et contrôleurs via :

- Ligne de commande
- Interface utilisateur interactive
- Application Python

II- CONFIGURATION REQUISE [4]

Laptop MAC (OS X Lion (10.7)
Mininet 2.2.1 on Ubuntu 14.04 – 64 bits
https://github.com/mininet/mininet/wiki/Mininet-VM-Images
VirtualBox 5.0.6 for OS X hosts amd64
https://www.virtualbox.org/wiki/Downloads

J'ai choisi cette configuration parce que c'est la configuration la plus simple pour étudier le protocole OF et son comportement.

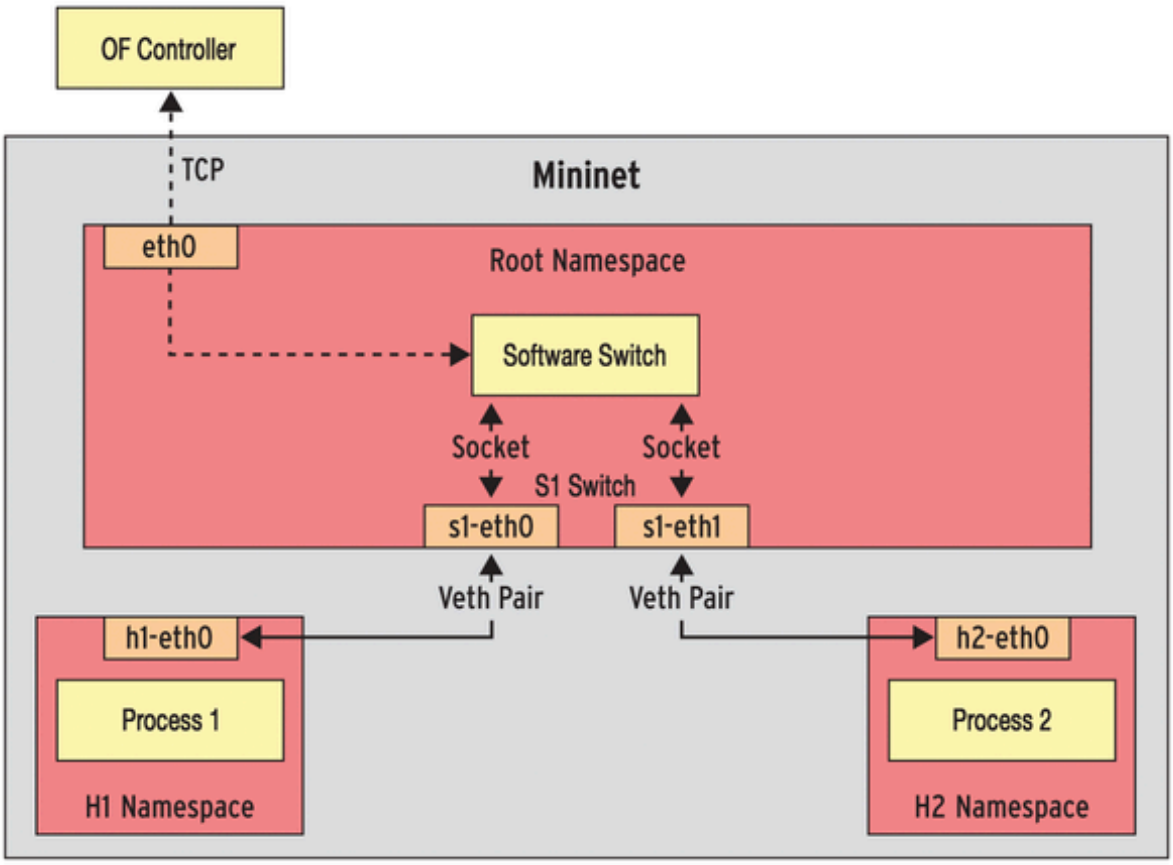
Le fait de faire les tests sur une seule machine permet de faire les tests rapidement, et d'avoir une meilleure performance sachant que les mêmes tests peuvent être réalisés sur un réseau physique, ou logique. Sur une ou plusieurs machines.

La seule chose qui devrait, à priori changer, est la vitesse avec laquelle les messages OF seront transmis.

[4] http://mininet.org/download/

III- ARCHITECTURE

1 Architecture de Mininet [5]



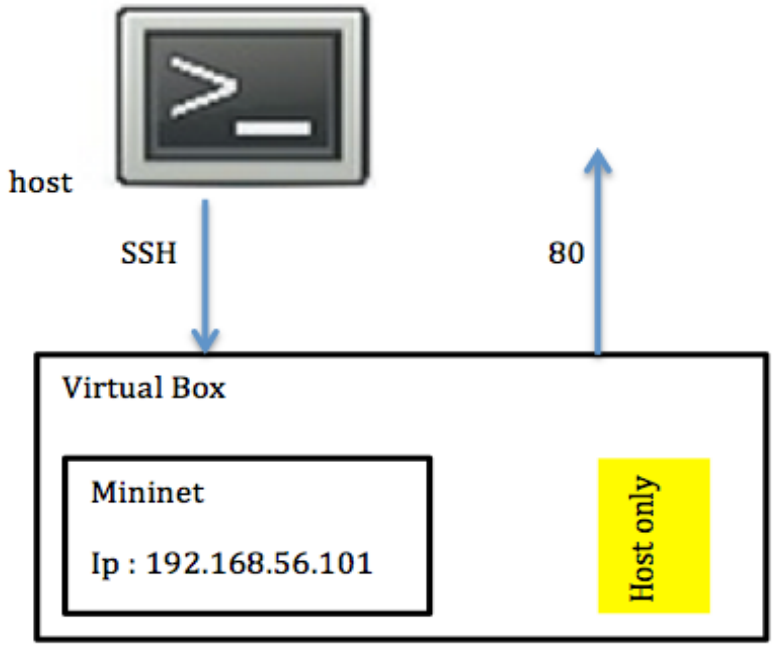
Mininet est une plateforme qui permet de virtualiser un réseau sur un seul Kernel
Il émule switchs et hôtes avec de simples processus.
Grâce au principe de *network namespace* introduit par avec Linux 2.2.24, le Kernel permet de séparer ces processus avec des interfaces réseaux virtuelles individuelles.

La figure ci-dessus illustre un cas simple ou Mininet utilise le principe de *namespace* .
Les deux hôtes H1 et H2 communiquent entre eux grâce aux câbles ethernet virtuels (Veth Pair)
H1 et H2 sont connectés à un seul (virtuel) switch S1 .
Le switch est émulé dans le namespace racine, tandis que les deux hôtes ont leur propre namespace .

[5] http://www.linux-magazine.com/Issues/2014/162/Mininet

2 Architecture de Mininet dans VirtualBox

Le PC communique avec la VM Mininet via le protocole SSH



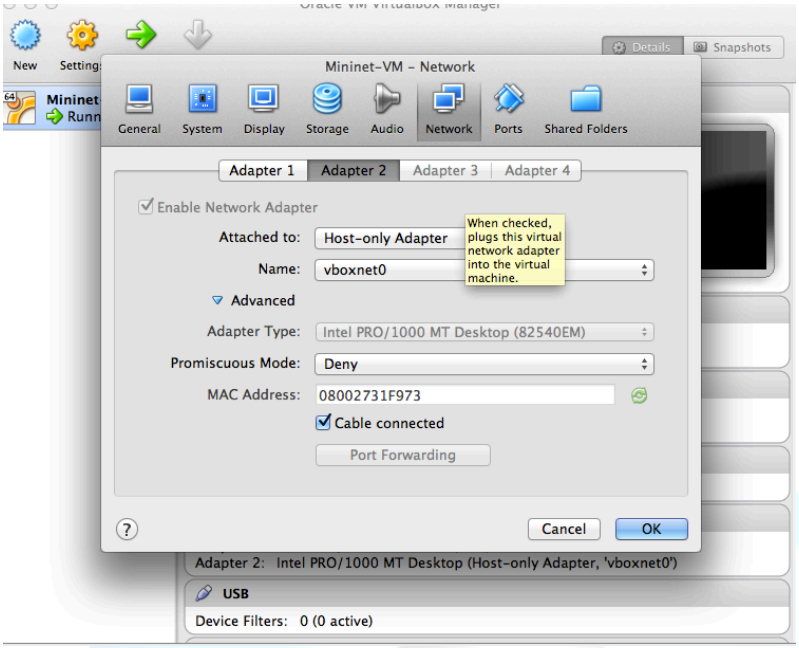
3- Installation et configuration [6]

Après le téléchargement de l'image Mininet,

1-Double click sur le fichier.ovf

2-Sélectionner Settings, -> Network-> Adapter 2, et choisir **host only**

[6] http://mininet.org/vm-setup-notes/



3-Ouvrir la VM avec un login/password : mininet/mininet

Pour configurer le SSH

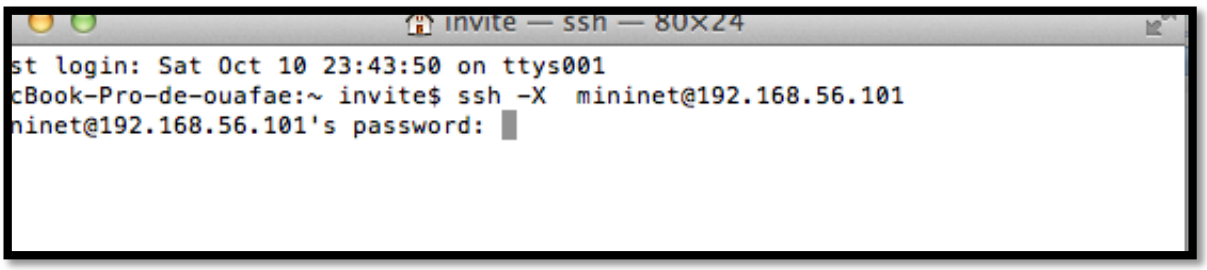
Configurer l'interface eth1

```
sudo dhclient eth1   # make sure that eth1 has an IP address ifconfig eth1
```

Chercher l'adresse ip de la VM fournie par Virtualbox

```
ifconfig eth0
```

Pour se connecter via SSH, ouvrez un terminal sur votre machine et exécutez la commande suivante :



IV- FONCTIONNEMENT DE MININET

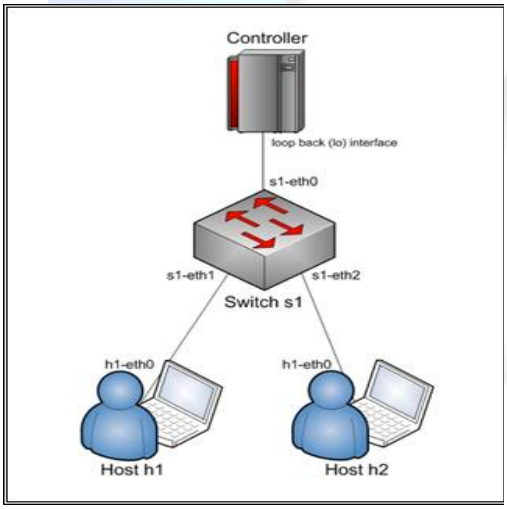
1 La typologie des réseaux

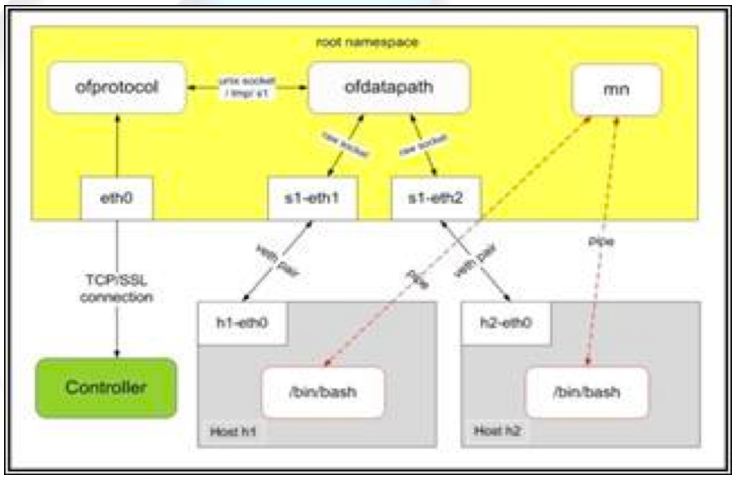
1-1 Ligne de commande

Mininet permet avec un simple jeu de commande réaliser des réseaux virtuels, utilisant le même kernel.

La commande : mn

Cette simple commande permet de réaliser un réseau avec deux hôtes, un vswitch et un contrôleur.





http://article.sciencepublishinggroup.com/html/10.11648.j.com.20150305.18.html

Les autres commandes :

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 égal à son adresse ip (exemple 10.0.0. 1 ➔ 00:00:00:00:00:01)
- Le switch est connecté à un contrôleur via une connexion SSH

```
$ sudo mn –top  linear,4  --mac  –-switch ovsk –-
controller=remote
```

```
$ sudo mn –top  tree,3   --mac  –-switch ovsk  –-
controller=remote
```

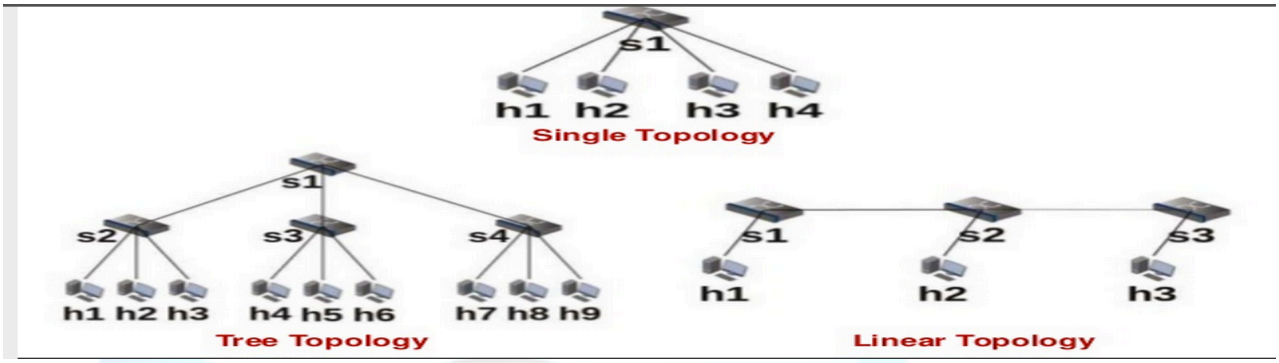


Figure : http://fr.slideshare.net/sdnrgitb/eksperimen-custom-topology

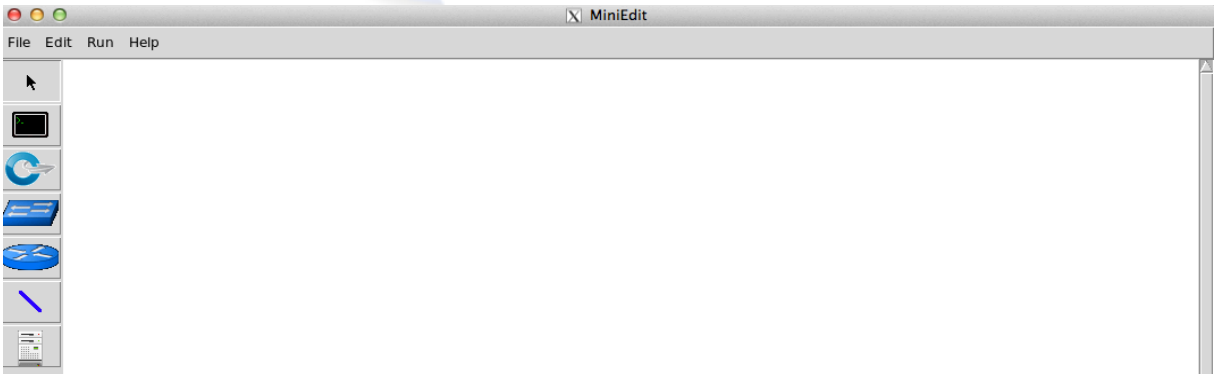
1-2 Interface graphique [7]

MIniEdit est une interface graphique qui permet de créer des réseaux virtuels sans passer par les commandes.

C'est un outil qui est encore en phase d'expérimentation, avec très peu de fonctionnalités.

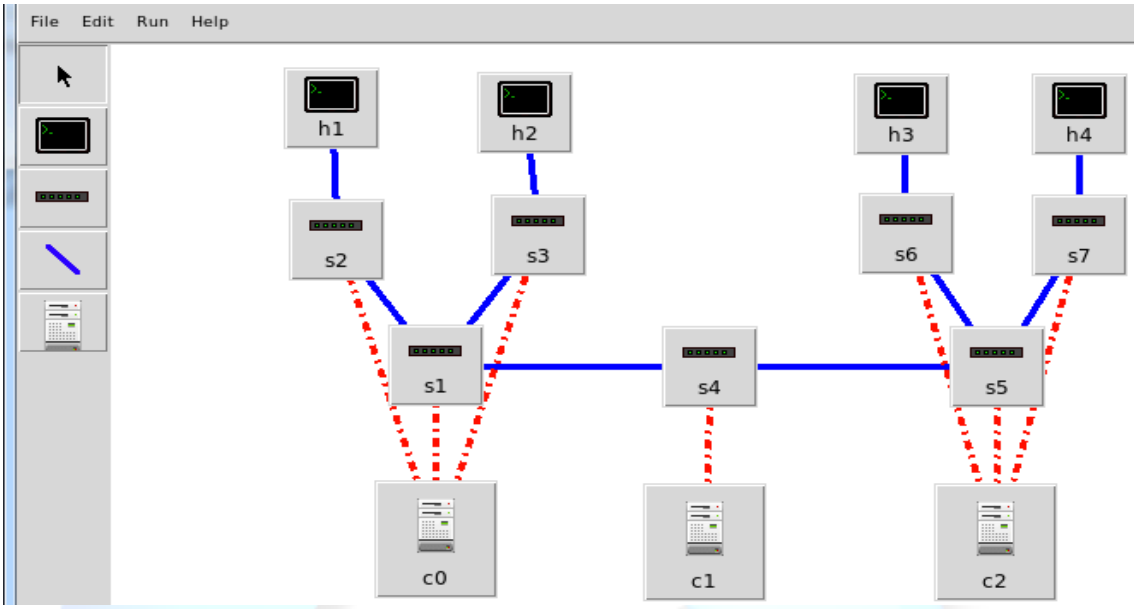
Fonctionnement :
Pour ouvrir l'interface graphique, exécuter la commande suivante :

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



[7] http://www.brianlinkletter.com/how-to-use-miniedit-mininets-graphical-user-interface/

Pour réaliser un réseau virtuel, il suffit de glisser le dessin situé à gauche et le déposer sur la fenêtre.



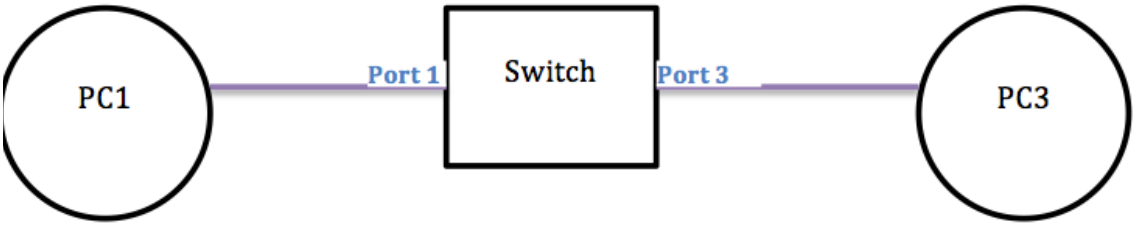
CHAPITRE III OF EN PRATIQUE

Dans ce chapitre nus allons voir les messages OF en détail, grâce à la capture wireshark et les différentes commandes de Mininet

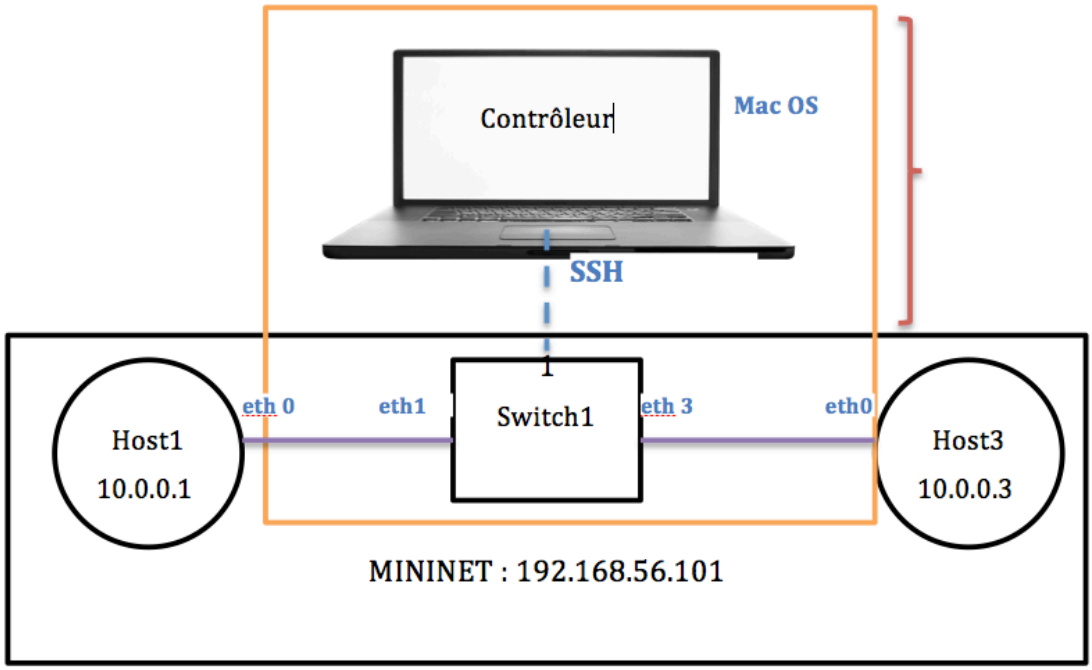
I- SCENARIO UTLISE

La configuration utilisée est simple composée d'un contrôleur, un switch et trois hosts. Comme H2 n'est pas utilisé dans le test, il ne sera pas dessiné par souci de simplicité.

1- Configuration physique



2- Configuration virtualisée



Légende : le fonctionnement d'un switch traditionnel est reproduit grâce à deux types d'outil distincts. Un contrôleur, d'une part, est chargé du calcul et de la mise à jour des tables de flux, d'autre part. un switch, qui est responsable du transfert des paquets d'une interface à une autre en fonction des règles établies par le contrôleur

II- DEROULEMENT DES TESTS

```
sudo mn –topo single,3 –mac –switch ovsk –controller
remote
```

[8]

Cette commande permet d'émuler un contrôleur qui écoutera sur le port par défaut 6633 et se connecte au switch via SSH

```
mininet@mininet-vm:~$ controller ptcp:
```

On fait un simple ping entre H1 et H3

```
Mininet>h1 ping h3
```

III- Statistiques de la table de flux

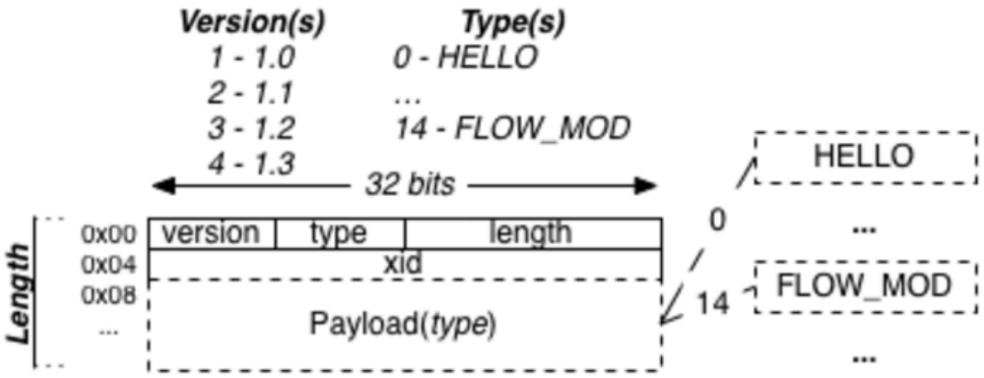
```
root@mininet-vm:~# sudo ovs-ofctl show s1
OFPT_FEATURES_REPLY (xid=0x2): dpid:0000000000000001
n_tables:254, n_buffers:256
capabilities: FLOW_STATS TABLE_STATS PORT_STATS QUEUE_STATS ARP_MATCH_IP
actions: OUTPUT SET_VLAN_VID SET_VLAN_PCP STRIP_VLAN SET_DL_SRC SET_DL_DST SET_N
W_SRC SET_NW_DST SET_NW_TOS SET_TP_SRC SET_TP_DST ENQUEUE
 1(s1-eth1): addr:92:d7:8a:1d:c5:2e
     config:     0
     state:      0
     current:    10GB-FD COPPER
     speed: 10000 Mbps now, 0 Mbps max
 2(s1-eth2): addr:3a:75:f6:a7:f6:68
     config:     0
     state:      0
     current:    10GB-FD COPPER
     speed: 10000 Mbps now, 0 Mbps max
 3(s1-eth3): addr:3a:34:ce:8f:8d:ea
     config:     0
     state:      0
     current:    10GB-FD COPPER
     speed: 10000 Mbps now, 0 Mbps max
```

H1, H2 et H3 sont connectés au switch, respectivement, via les interfaces eth1, eth2 et eth3

[8] Explication de la commande page 19 paragraphe 1-1

V- LA CAPTURE WIRESHARK

1- Syntaxe des captures wireshark



2- Les captures

- 6 packets-in
- 1 packet-out
- 5 packets flow-add

Packet-in : h1 envoie un ARP request en broadcast

Le switch encapsule le paquet et le renvoie au contrôleur quand il ne trouve pas de correspondance dans sa table de flux

No. Time Source Destination Protocol Length Info
1357 51.729002000 00:00:00_00:00:01 Broadcast OF 1.0 126 of_packet_in

Frame 1357: 126 bytes on wire (1008 bits), 126 bytes captured (1008 bits) on interface 0
Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
Internet Protocol Version 4, Src: 127.0.0.1 (127.0.0.1), Dst: 127.0.0.1 (127.0.0.1)
Transmission Control Protocol, Src Port: 33236 (33236), Dst Port: 6633 (6633), Seq: 89, Ack: 89, Len: 60
OpenFlow
version: 1// OF 1.0.0
type: OFPT_PACKET_IN (10)
length: 60
xid: 0
buffer_id: 256
total_len: 42
in_port: 1 //port logique du switch
reason: OFPR_NO_MATCH (0)
Ethernet packet
Ethernet II, Src: 00:00:00_00:00:01 (00:00:00:00:00:01), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
Address Resolution Protocol (request)

Packet_out : le contrôleur envoie le paquet à travers le chemin des données du switch(Data path). Il utilise le même numéro du buffer-id

No. Time Source Destination Protocol Length Info
1358 51.729163000 127.0.0.1 127.0.0.1 OF 1.0 90 of_packet_out

Frame 1358: 90 bytes on wire (720 bits), 90 bytes captured (720 bits) on interface 0

Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
Internet Protocol Version 4, Src: 127.0.0.1 (127.0.0.1), Dst: 127.0.0.1 (127.0.0.1)
Transmission Control Protocol, Src Port: 6633 (6633), Dst Port: 33236 (33236), Seq: 89, Ack: 149, Len: 24
OpenFlow
  version: 1
  type: OFPT_PACKET_OUT (13)
  length: 24
  xid: 0
  buffer_id: 256
  in_port: 1
  actions_len: 8
  of_action list
    of_action_output
      type: OFPAT_OUTPUT (0)
      len: 8
      port: 65531
      max_len: 0

No. Time Source Destination Protocol Length Info
1360 51.729392000 00:00:00_00:00:03 00:00:00_00:00:01 OF 1.0 126 of_packet_in
h3 envoie un ARP reply

Frame 1360: 126 bytes on wire (1008 bits), 126 bytes captured (1008 bits) on interface 0
Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
Internet Protocol Version 4, Src: 127.0.0.1 (127.0.0.1), Dst: 127.0.0.1 (127.0.0.1)
Transmission Control Protocol, Src Port: 33236 (33236), Dst Port: 6633 (6633), Seq: 149, Ack: 113, Len: 60
OpenFlow
  version: 1
  type: OFPT_PACKET_IN (10)
  length: 60
  xid: 0
  buffer_id: 257// buffer-id pour le paquet envoyé par H3
  total_len: 42
  in_port: 3// le port logique su switch que h3 utilise
  reason: OFPR_NO_MATCH (0)
  Ethernet packet
Ethernet II, Src: 00:00:00_00:00:03 (00:00:00:00:00:03), Dst: 00:00:00_00:00:01 (00:00:00:00:00:01)
    Address Resolution Protocol (reply)

No. Time Source Destination Protocol Length Info
1361 51.729478000 127.0.0.1 127.0.0.1 OF 1.0 146 of_flow_add
ce message permet d'introduire la première entrée dans la table de flux
les champs de la table sont marqués en jaune

Frame 1361: 146 bytes on wire (1168 bits), 146 bytes captured (1168 bits) on interface 0
Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
Internet Protocol Version 4, Src: 127.0.0.1 (127.0.0.1), Dst: 127.0.0.1 (127.0.0.1)
Transmission Control Protocol, Src Port: 6633 (6633), Dst Port: 33236 (33236), Seq: 113, Ack: 209, Len: 80
OpenFlow
  version: 1
  type: OFPT_FLOW_MOD (14)
  length: 80
  xid: 0
  of_match
    wildcards: 0x0000000000000000
    in_port: 3
    eth_src: 00:00:00_00:00:03 (00:00:00:00:00:03)

eth_dst: 00:00:00_00:00:01 (00:00:00:00:00:01)
vlan_vid: 65535
vlan_pcp: 0
eth_type: 2054
ip_dscp: 0
ip_proto: 2
ipv4_src: 10.0.0.3 (10.0.0.3)
ipv4_dst: 10.0.0.1 (10.0.0.1)
tcp_src: 0
tcp_dst: 0
cookie: 0
_command: 0
idle_timeout: 60
hard_timeout: 0
priority: 0
buffer_id: 257
out_port: 0
flags: Unknown (0x00000000)
of_action list
of_action_output
type: OFPAT_OUTPUT (0)
len: 8
port: 1
max_len: 0

No.	Time	Source	Destination	Protocol	Length	Info
1362	51.729855000	10.0.0.1	10.0.0.3	OF 1.0	182	of_packet_in

H1 envoie un ICMP request
Frame 1362: 182 bytes on wire (1456 bits), 182 bytes captured (1456 bits) on interface 0
Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
Internet Protocol Version 4, Src: 127.0.0.1 (127.0.0.1), Dst: 127.0.0.1 (127.0.0.1)
Transmission Control Protocol, Src Port: 33236 (33236), Dst Port: 6633 (6633), Seq: 209, Ack: 193, Len: 116
OpenFlow
version: 1
type: OFPT_PACKET_IN (10)
length: 116
xid: 0
buffer_id: 258
total_len: 98
in_port: 1
reason: OFPR_NO_MATCH (0)
Ethernet packet
Ethernet II, Src: 00:00:00_00:00:01 (00:00:00:00:00:01), Dst: 00:00:00_00:00:03 (00:00:00:00:00:03)
Destination: 00:00:00_00:00:03 (00:00:00:00:00:03)
Source: 00:00:00_00:00:01 (00:00:00:00:00:01)
Type: IP (0x0800)
Internet Protocol Version 4, Src: 10.0.0.1 (10.0.0.1), Dst: 10.0.0.3 (10.0.0.3)
Internet Control Message Protocol
Type: 8 (Echo (ping) request)
Code: 0
Checksum: 0x65cb [correct]
Identifier (BE): 1727 (0x06bf)
Identifier (LE): 48902 (0xbf06)
Sequence number (BE): 1 (0x0001)
Sequence number (LE): 256 (0x0100)
[Response frame: 1364]
Timestamp from icmp data: Nov 9, 2015 11:30:07.000000000 PST

[Timestamp from icmp data (relative): 0.088964000 seconds]
Data (48 bytes)

No. Time Source Destination Protocol Length Info
1363 51.729969000 127.0.0.1 127.0.0.1 OF 1.0 146 of_flow_add
la 2ème entrée dans la table de flux

Frame 1363: 146 bytes on wire (1168 bits), 146 bytes captured (1168 bits) on interface 0
Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
Internet Protocol Version 4, Src: 127.0.0.1 (127.0.0.1), Dst: 127.0.0.1 (127.0.0.1)
Transmission Control Protocol, Src Port: 6633 (6633), Dst Port: 33236 (33236), Seq: 193, Ack: 325, Len: 80
OpenFlow
version: 1
type: OFPT_FLOW_MOD (14)
length: 80
xid: 0
of_match
wildcards: 0x0000000000000000
in_port: 1//port d'entrée
eth_src: 00:00:00_00:00:01 (00:00:00:00:00:01)// MAC adress
eth_dst: 00:00:00_00:00:03 (00:00:00:00:00:03)
vlan_vid: 65535
vlan_pcp: 0
eth_type: 2048
ip_dscp: 0
ip_proto: 1
ipv4_src: 10.0.0.1 (10.0.0.1)
ipv4_dst: 10.0.0.3 (10.0.0.3)
tcp_src: 8
tcp_dst: 0
cookie: 0
_command: 0
idle_timeout: 60
hard_timeout: 0
priority: 0
buffer_id: 258
out_port: 0
flags: Unknown (0x00000000)
of_action list
of_action_output
type: OFPAT_OUTPUT (0)
len: 8
port: 3
max_len: 0

No. Time Source Destination Protocol Length Info
1364 51.730255000 10.0.0.3 10.0.0.1 OF 1.0 182 of_packet_in

H3 envoie un ICMP reply
Frame 1364: 182 bytes on wire (1456 bits), 182 bytes captured (1456 bits) on interface 0
Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
Internet Protocol Version 4, Src: 127.0.0.1 (127.0.0.1), Dst: 127.0.0.1 (127.0.0.1)
Transmission Control Protocol, Src Port: 33236 (33236), Dst Port: 6633 (6633), Seq: 325, Ack: 273, Len: 116
OpenFlow
version: 1
type: OFPT_PACKET_IN (10)
length: 116

xid: 0
buffer_id: 259
total_len: 98
in_port: 3
reason: OFPR_NO_MATCH (0)
Ethernet packet
Src: 00:00:00_00:00:03 (00:00:00:00:00:03), Dst: 00:00:00_00:00:01 (00:00:00:00:00:01)
Destination: 00:00:00_00:00:01 (00:00:00:00:00:01)
Source: 00:00:00_00:00:03 (00:00:00:00:00:03)
Type: IP (0x0800)
Internet Protocol Version 4, Src: 10.0.0.3 (10.0.0.3), Dst: 10.0.0.1 (10.0.0.1)
Internet Control Message Protocol
Type: 0 (Echo (ping) reply)
Code: 0
Checksum: 0x6dcb [correct]
Identifier (BE): 1727 (0x06bf)
Identifier (LE): 48902 (0xbf06)
Sequence number (BE): 1 (0x0001)
Sequence number (LE): 256 (0x0100)
[Request frame: 1362]
[Response time: 0.400 ms]
Timestamp from icmp data: Nov 9, 2015 11:30:07.000000000 PST
[Timestamp from icmp data (relative): 0.089364000 seconds]
Data (48 bytes)

No.	Time	Source	Destination	Protocol	Length	Info
1365	51.730338000	127.0.0.1	127.0.0.1	OF 1.0	146	of_flow_add

la 3ème entrée dans la table de flux

Frame 1365: 146 bytes on wire (1168 bits), 146 bytes captured (1168 bits) on interface 0
Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
Internet Protocol Version 4, Src: 127.0.0.1 (127.0.0.1), Dst: 127.0.0.1 (127.0.0.1)
Transmission Control Protocol, Src Port: 6633 (6633), Dst Port: 33236 (33236), Seq: 273, Ack: 441, Len: 80
OpenFlow
version: 1
type: OFPT_FLOW_MOD (14)
length: 80
xid: 0
of_match
wildcards: 0x0000000000000000
in_port: 3//port d'entrée
eth_src: 00:00:00_00:00:03 (00:00:00:00:00:03)
eth_dst: 00:00:00_00:00:01 (00:00:00:00:00:01)
vlan_vid: 65535
vlan_pcp: 0
eth_type: 2048
ip_dscp: 0
ip_proto: 1
ipv4_src: 10.0.0.3 (10.0.0.3)
ipv4_dst: 10.0.0.1 (10.0.0.1)
tcp_src: 0
tcp_dst: 0
cookie: 0
_command: 0
idle_timeout: 60
hard_timeout: 0
priority: 0
buffer_id: 259

out_port: 0
flags: Unknown (0x00000000)
of_action list
of_action_output
type: OFPAT_OUTPUT (0)
len: 8
port: 1 //port de sortie
max_len: 0

No. Time Source Destination Protocol Length Info
1529 56.732023000 00:00:00_00:00:03 00:00:00_00:00:01 OF 1.0 126 of_packet_in
H3 envoie un ARP request à H1

Frame 1529: 126 bytes on wire (1008 bits), 126 bytes captured (1008 bits) on interface 0
Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
Internet Protocol Version 4, Src: 127.0.0.1 (127.0.0.1), Dst: 127.0.0.1 (127.0.0.1)
Transmission Control Protocol, Src Port: 33236 (33236), Dst Port: 6633 (6633), Seq: 449, Ack: 361, Len: 60
OpenFlow
version: 1
type: OFPT_PACKET_IN (10)
length: 60
xid: 0
buffer_id: 260
total_len: 42
in_port: 3
reason: OFPR_NO_MATCH (0)
Ethernet packet
Ethernet II, Src: 00:00:00_00:00:03 (00:00:00:00:00:03), Dst: 00:00:00_00:00:01 (00:00:00:00:00:01)
Destination: 00:00:00_00:00:01 (00:00:00:00:00:01)
Source: 00:00:00_00:00:03 (00:00:00:00:00:03)
Type: ARP (0x0806)
Address Resolution Protocol (request)

No. Time Source Destination Protocol Length Info
1530 56.732295000 127.0.0.1 127.0.0.1 OF 1.0 146 of_flow_add
la 4ème entrée de la table de flux
Frame 1530: 146 bytes on wire (1168 bits), 146 bytes captured (1168 bits) on interface 0
Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
Internet Protocol Version 4, Src: 127.0.0.1 (127.0.0.1), Dst: 127.0.0.1 (127.0.0.1)
Transmission Control Protocol, Src Port: 6633 (6633), Dst Port: 33236 (33236), Seq: 361, Ack: 509, Len: 80
OpenFlow
version: 1
type: OFPT_FLOW_MOD (14)
length: 80
xid: 0
of_match
wildcards: 0x0000000000000000
in_port: 3
eth_src: 00:00:00_00:00:03 (00:00:00:00:00:03)
eth_dst: 00:00:00_00:00:01 (00:00:00:00:00:01)
vlan_vid: 65535
vlan_pcp: 0
eth_type: 2054
ip_dscp: 0
ip_proto: 1
ipv4_src: 10.0.0.3 (10.0.0.3) // dans ce cas l'adresse ip est marqué
ipv4_dst: 10.0.0.1 (10.0.0.1)

tcp_src: 0
tcp_dst: 0
cookie: 0
_command: 0
idle_timeout: 60
hard_timeout: 0
priority: 0
buffer_id: 260
out_port: 0
flags: Unknown (0x00000000)
of_action list
of_action_output
type: OFPAT_OUTPUT (0)
len: 8
port: 1

No. Time Source Destination Protocol Length Info
1532 56.732846000 00:00:00_00:00:01 00:00:00_00:00:03 OF 1.0 126 of_packet_in
H1 envoie un ARP reply à H3

Frame 1532: 126 bytes on wire (1008 bits), 126 bytes captured (1008 bits) on interface 0
Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
Internet Protocol Version 4, Src: 127.0.0.1 (127.0.0.1), Dst: 127.0.0.1 (127.0.0.1)
Transmission Control Protocol, Src Port: 33236 (33236), Dst Port: 6633 (6633), Seq: 509, Ack: 441, Len: 60
OpenFlow
version: 1
type: OFPT_PACKET_IN (10)
length: 60
xid: 0
buffer_id: 261
total_len: 42
in_port: 1
reason: OFPR_NO_MATCH (0)
Ethernet packet
Ethernet II, Src: 00:00:00_00:00:01 (00:00:00:00:00:01), Dst: 00:00:00_00:00:03 (00:00:00:00:00:03)
Destination: 00:00:00_00:00:03 (00:00:00:00:00:03)
Source: 00:00:00_00:00:01 (00:00:00:00:00:01)
Type: ARP (0x0806)
Address Resolution Protocol (reply)

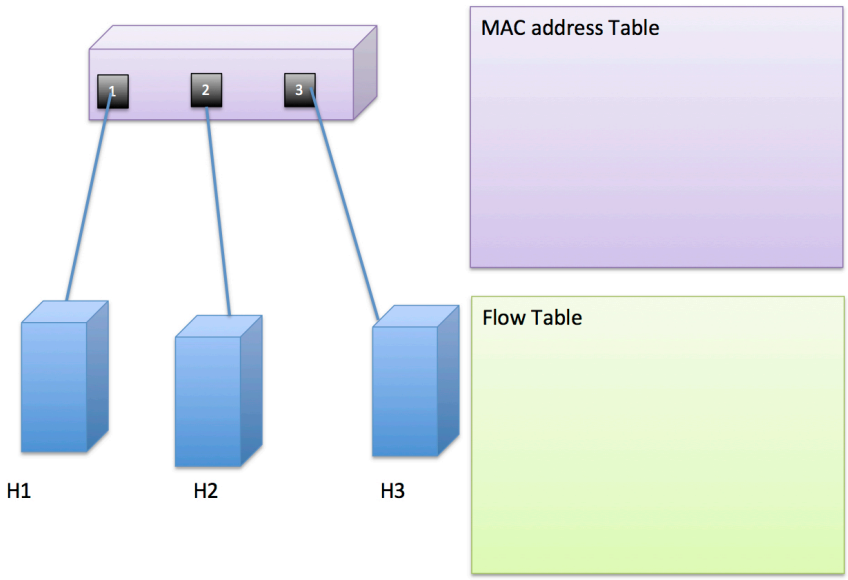
No. Time Source Destination Protocol Length Info
1533 56.732996000 127.0.0.1 127.0.0.1 OF 1.0 146 of_flow_add
la 5ème entrée de la table de flux
Frame 1533: 146 bytes on wire (1168 bits), 146 bytes captured (1168 bits) on interface 0
Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
Internet Protocol Version 4, Src: 127.0.0.1 (127.0.0.1), Dst: 127.0.0.1 (127.0.0.1)
Transmission Control Protocol, Src Port: 6633 (6633), Dst Port: 33236 (33236), Seq: 441, Ack: 569, Len: 80
OpenFlow
version: 1
type: OFPT_FLOW_MOD (14)
length: 80
xid: 0
of_match
wildcards: 0x0000000000000000
in_port: 1
eth_src: 00:00:00_00:00:01 (00:00:00:00:00:01)
eth_dst: 00:00:00_00:00:03 (00:00:00:00:00:03)

vlan_vid: 65535
vlan_pcp: 0
eth_type: 2054
ip_dscp: 0
ip_proto: 2
ipv4_src: 10.0.0.1 (10.0.0.1)
ipv4_dst: 10.0.0.3 (10.0.0.3)
tcp_src: 0
tcp_dst: 0
cookie: 0
_command: 0
idle_timeout: 60
hard_timeout: 0
priority: 0
buffer_id: 261
out_port: 0
flags: Unknown (0x00000000)
of_action list
of_action_output
type: OFPAT_OUTPUT (0)
len: 8
port: 3
max_len: 0

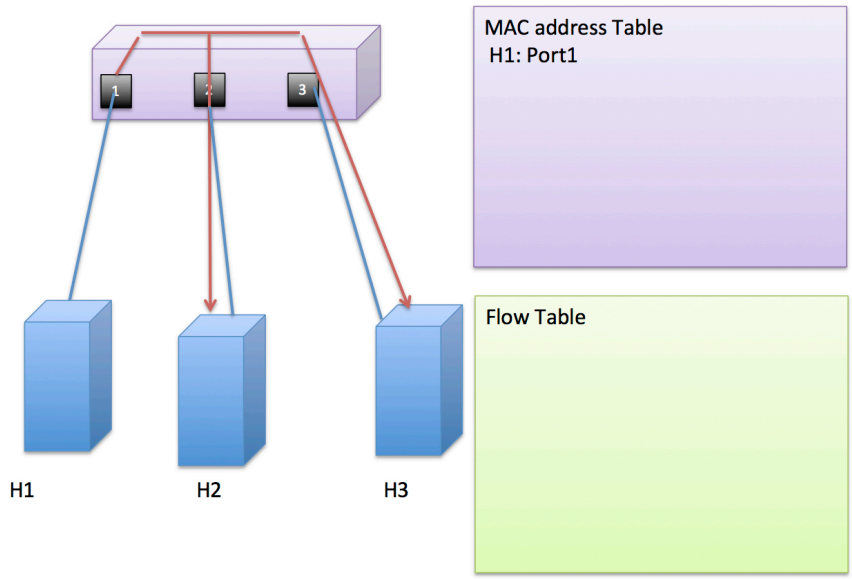
Déroulement des messages

L'état initial : H1 est connecté au port 1 et H3 au port 3

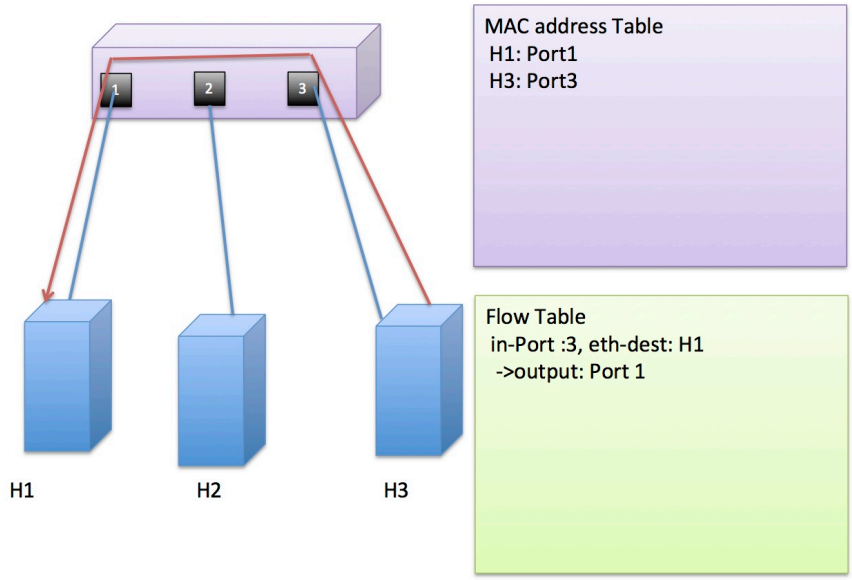
La table de flux est vide



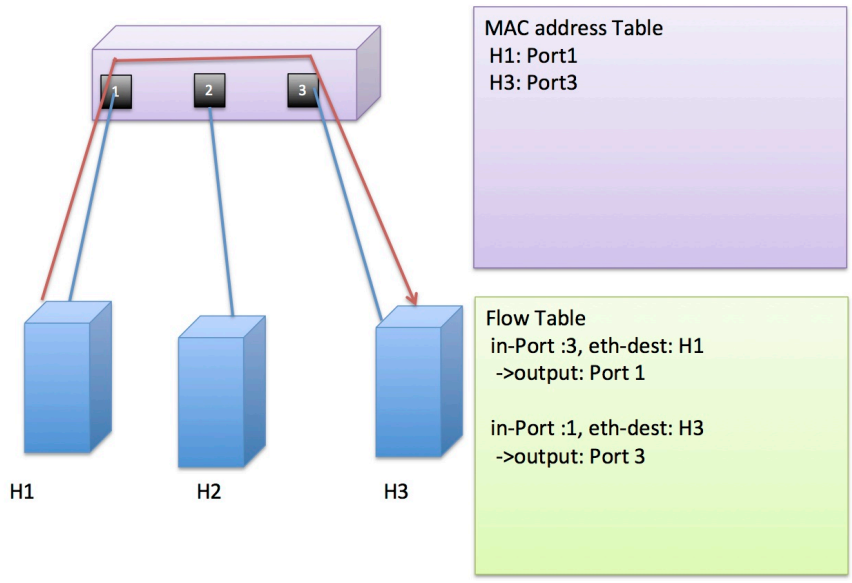
H1 ping H3 : paquet 1357



Paquet 1360



Paquet 1532


1
2
3
H1
H2
H3
MAC address Table
H1: Port1
H3: Port3
Flow Table
in-Port :3, eth-dest: H1
->output: Port 1
in-Port :1, eth-dest: H3
->output: Port 3

CHAPITRE IV FIREWALL

I- SCENARIO

Par curiosité de découvrir la programmation python , J'ai choisi le contrôleur *POX* pour réaliser un firewall.

Ce contrôleur est fourni avec *Mininet*, comme le contrôleur *NOX* qui lui, est basé sur la programmation C++.

Il suffit donc d'écrire le code dans éditeur de texte, l'enregistrer avec l'extention.py dans le Répertoire *pox,* puis l'exécuter comme cela est indiqué dans le quatrième paragraphe.

II- ALGORITHME

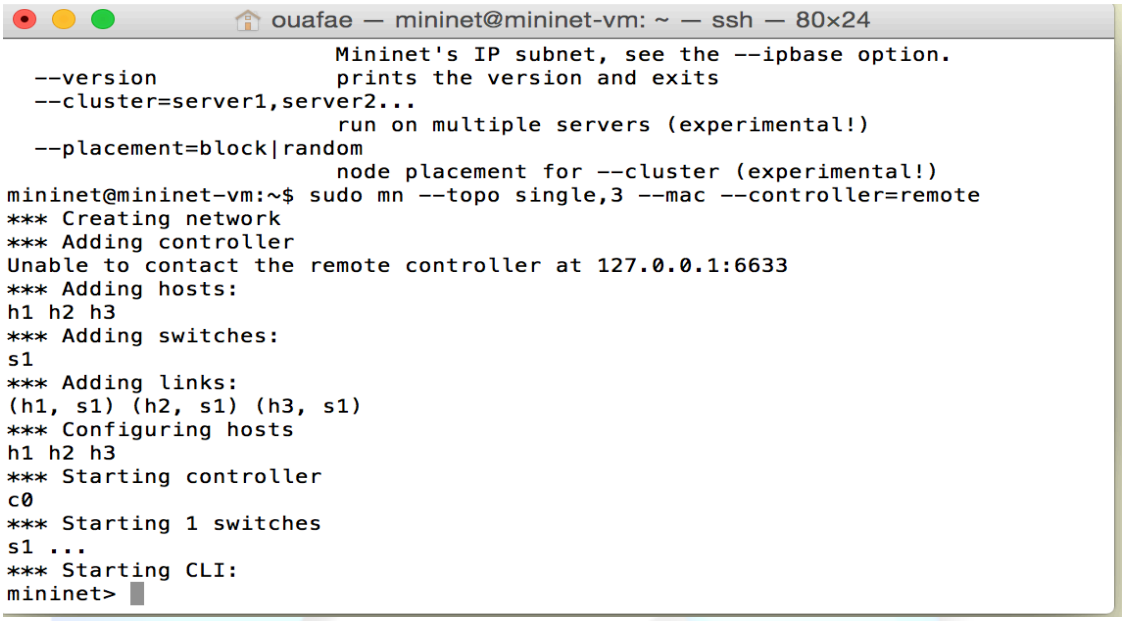
For each packet from the switch:
1) Use source address and switch port to update address/port table
2) Is transparent = False and either Ethertype is LLDP or the packet's destination address is a Bridge Filtered address?
Yes:
2a) Drop packet -- don't forward link-local traffic (LLDP, 802.1x)
DONE
3) Is destination multicast?
Yes:
3a) Flood the packet
DONE
4) Port for destination address in our address/port table?
No:
4a) Flood the packet
DONE
5) Is output port the same as input port?
Yes:
5a) Drop packet and similar ones for a while
6) Install flow table entry in the switch so that this flow goes out the appopriate port
6a) Send the packet out appropriate port

III- LE CODE

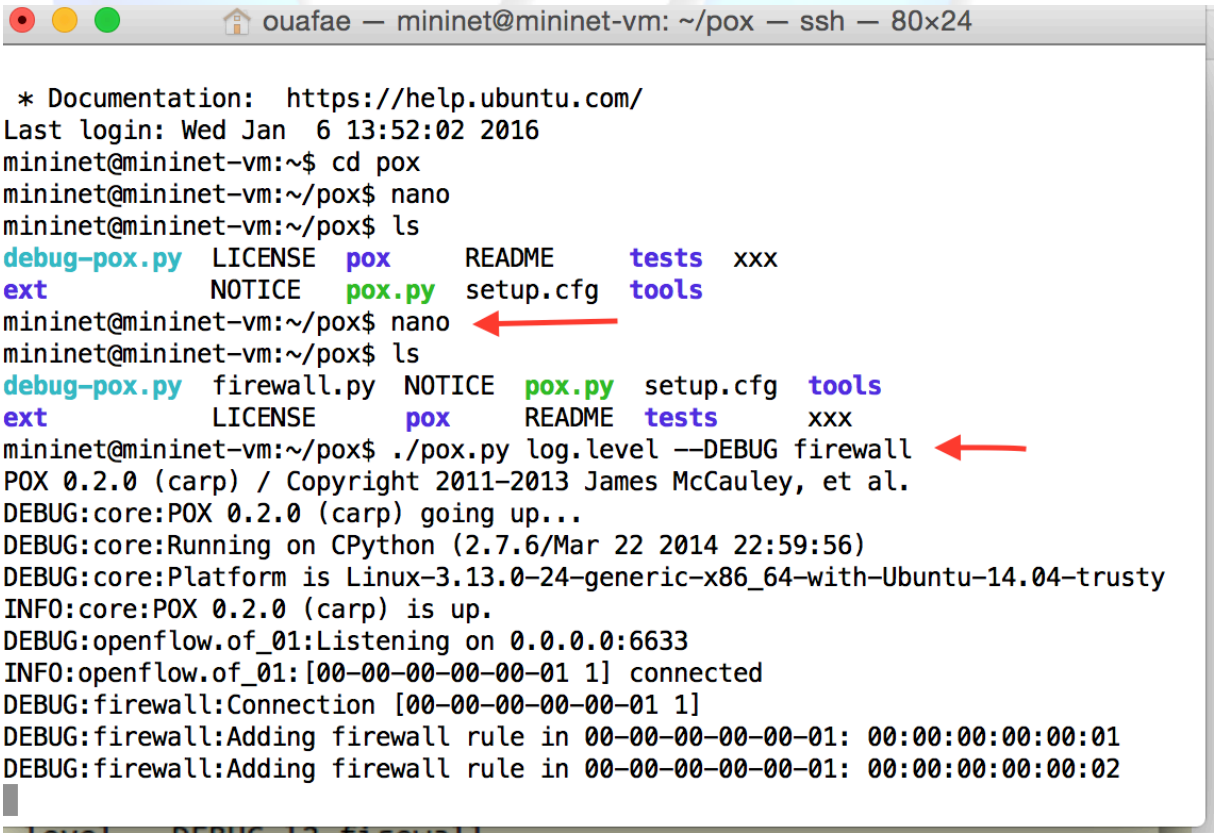
Voir ANNEXES

IV- DEROULEMENT DES TESTS ET RESULTATS

Créer une typologie simple avec in contrôleur, un switch et 3 hosts

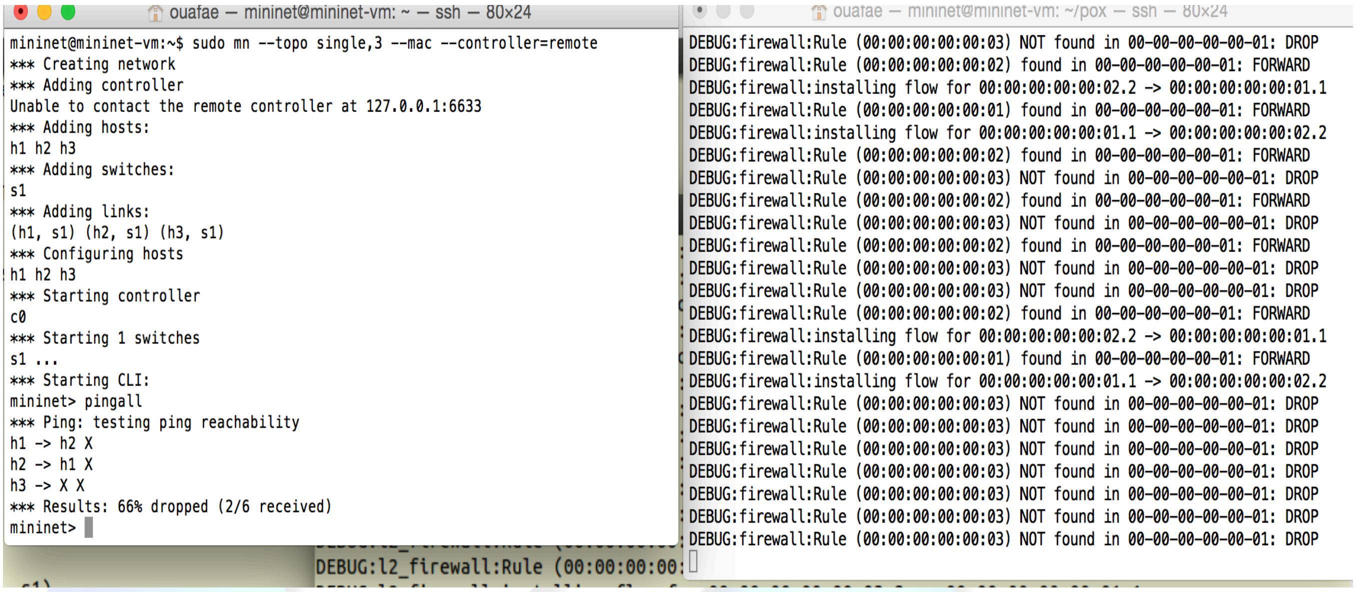


Ouvrir *nano* , copier le code et l'enregistrer sous le nom de firewall.py dans le répertoire *pox* puis l'exécuter en mode debug

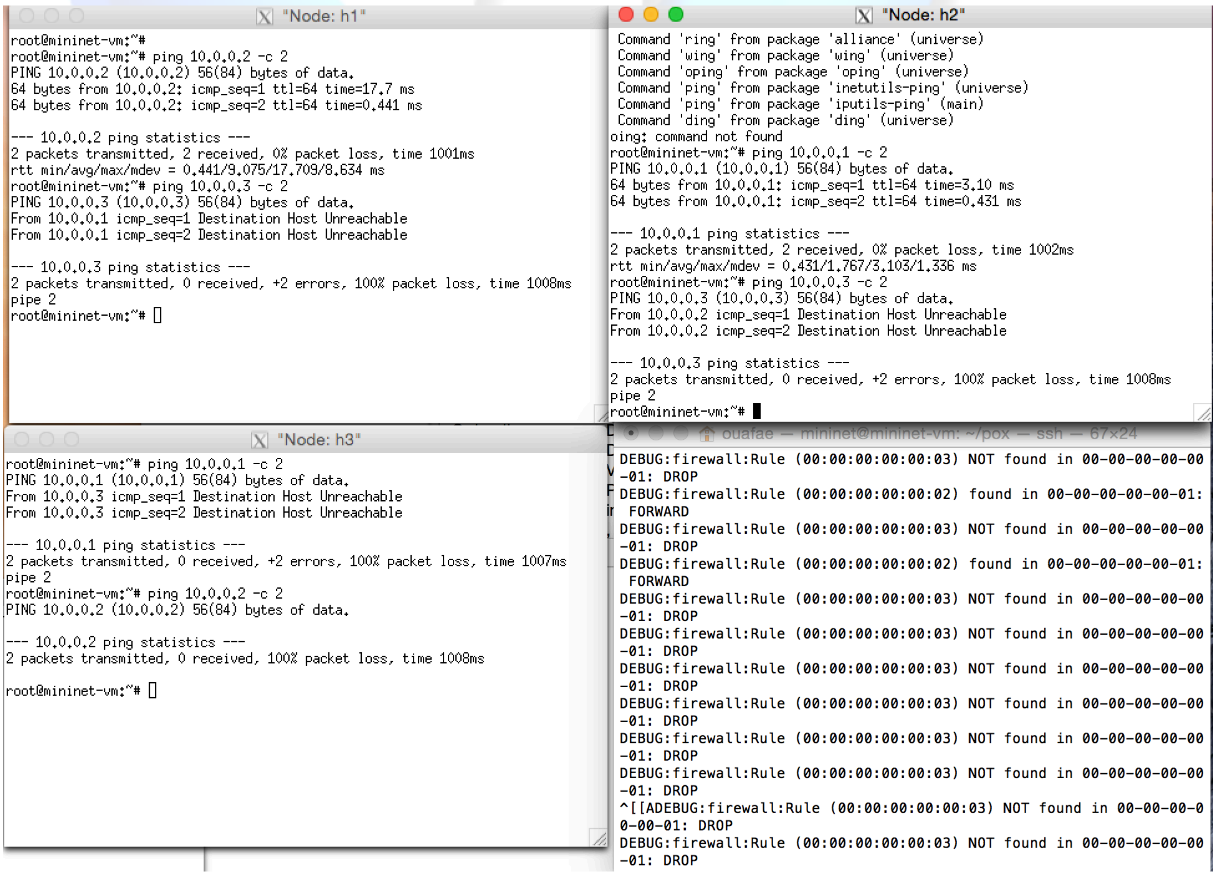


Tous les paquets provenant de la *MAC address* **00:00:00:00:00:01** et **00:00:00:00:00:02** seront transmis. Les autres (host3) seront supprimés.

Nous pouvons observer les résultats à l'aide de la commande **pingall**



ou en faisant un *ping* entre chaque deux hosts



CONCLUSION

La technologie SDN devrait révolutionner à terme les architectures des réseaux des opérateurs, et permettre de déployer des nouveaux services de manière beaucoup plus rapide et avec des coûts significativement réduits. Elle changerait complètement l'écosystème des infrastructures Télécoms dans les années à venir.

OpenFlow permet une programmation simplifiée via une interface standard pour les périphériques réseau. La facilité de programmation permet de concevoir une couche de contrôle robuste, afin de centraliser l'intelligence dans le réseau et de fournir la programmable promise par *SDN*.

Bien que ce protocole soit fonctionnel, il reste néanmoins plusieurs anomalies à améliorer, surtout au niveau sécurité.

La perméabilité de la liaison entre le contrôleur et le switch, permet à tout attaquant ayant accès au réseau d'administration d'agir sur les le paramétrage du *switch* comme il le souhaite.

L'écoute passive des messages non chiffrés envoyés par le contrôleur permet à l'attaquant d'obtenir plusieurs renseignements sur le réseau qui peuvent compromettre la sécurité de celui-ci.

Problèmes rencontrés

1- Au début. J'ai trouvé le sujet un peu vaste , et la difficulté était de trouver une structure à mon projet
2- En ci concerne Mininet, le fait d'avoir très peu de référence qui explique le fonctionnement et surtout l'architecture de cette plateforme , rend la compréhension un peu difficile
3- Des bugs dans le fonctionnement de Mininet . qui ne sont pas expliquées dans MIninet.org
    - Par exemple, si on un a un réseau qui comporte plusieurs hosts , et on veut ouvrir un Xterm à H1, mais cela ne fonctionne pas , passer par les étapes suivantes

>mininet> h1 ifconfig >h1-eth0 *Link encap:Ethernet HWaddr 36:f0:a4:2f:1f:8c > inet addr:10.0.0.1 Bcast:10.255.255.255 Mask:255.0.0.0 > UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1 > RX packets:0 errors:0 dropped:0 overruns:0 frame:0 > TX packets:0 errors:0 dropped:0 overruns:0 carrier:0 > collisions:0 txqueuelen:1000 > RX bytes:0 (0.0 B) TX bytes:0 (0.0 B) > > >lo Link encap:Local Loopback > inet addr:127.0.0.1 Mask:255.0.0.0 > UP LOOPBACK RUNNING MTU:16436 Metric:1 > RX packets:0 errors:0 dropped:0 overruns:0 frame:0 > TX packets:0 errors:0 dropped:0 overruns:0 carrier:0 > collisions:0 txqueuelen:0 > RX bytes:0 (0.0 B) TX bytes:0 (0.0 B) > > >*

mininet> h1 echo $DISPLAY *>localhost:0.0*

mininet> h1 whoami >root *>*

mininet> h1 xterm *>Warning: This program is an suid-root program or is being run by the root user. >The full text of the error or warning message cannot be safely formatted >in this environment. You may get a more descriptive message by running the >program as a non-root user or by removing the suid bit on the executable. >xterm Xt error: Can't open display: %s >*

mininet> h1 xterm -display localhost:0.0 *>Warning: This program is an suid-root program or is being run by the root user. >The full text of the error or warning message cannot be safely formatted >in this environment. You may get a more descriptive message by running the >program as a non-root user or by removing the suid bit on the executable. >xterm Xt error: Can't open display: %s >*

mininet> > > >-- > >Li > > >_____

4- Une clé USB, même si elle est détectée par l'OS de Mininet, elle ne l'est pas par wireshark Donc si on veut enregistrer une capture sur notre machine, passer d'abords par la commande scp , qui permet d'envoyer un fichier ou un répertoire de la machine virtuelle à notre machine physique .
5- L'apprentissage du langage de programmation python.

REFERENCES

http://fr.slideshare.net/kingstonsmiler/tutorial-on-sdn-and-openflow

http://mininet.org/sample-workflow/

https://mailman.stanford.edu/pipermail/mininet-discuss/2014-January/003763.html

https://www.cs.princeton.edu/courses/archive/fall10/cos561/assignments/ps2-tut.pdf

http://archive.openflow.org/wk/index.php/OpenFlow_Tutorial

https://www.safaribooksonline.com/library/view/sdn-software-defined/9781449342425/ch04.html

http://goc.pragma-grid.net/pragma-doc/pragma24/Cloud-SDN-Workshop/ViNe-Tsugawa.pdf

https://openflow.stanford.edu/display/ONL/POX+Wiki

http://web.univ-pau.fr/~puiseux/enseignement/python/python.pdf

http://www.coyotus.com/repo/Apprendre%20Python/bases-python-by-LaSourisVerte.pdf

ANNEXES

CODE SWITCH FIREWALL

```
from pox.core import core
import pox.openflow.libopenflow_01 as of
from pox.lib.util import dpid_to_str
from pox.lib.util import str_to_bool
import time

from pox.lib.addresses import EthAddr

log = core.getLogger()

# attendre avant d'inonder le switch
_flood_delay = 0

class fSwitch (object):
   def __init__ (self, connection, transparent):
   # connecter le switch
   self.connection = connection
   self.transparent = transparent

   #Notre table
   self.macToPort = {}

   # Notre firewall table
   self.firewall = {}

   # Ajouter les deux règles
   self.AddRule('00-00-00-00-00-01',EthAddr('00:00:00:00:00:01'))
   self.AddRule('00-00-00-00-00-01',EthAddr('00:00:00:00:00:02'))

   # attendre le message packetin en écoutant simplement la connection
   connection.addListeners(self)

   #self.hold_down_expired = _flood_delay == 0

 # ajouter les règles dans firewall table
   def AddRule (self, dpidstr, src=0,value=True):
   self.firewall[(dpidstr,src)]=value
   log.debug("Adding firewall rule in %s: %s", dpidstr, src)

 # fonction permet la suppression des règle de filtrage
 def DeleteRule (self, dpidstr, src=0):
    try:
      del self.firewall[(dpidstr,src)]
      log.debug("Deleting firewall rule in %s: %s",
             dpidstr, src)
    except KeyError:
      log.error("Cannot find in %s: %s",
             dpidstr, src)
```

```
# vérifier si le paquet est conforme aux règles
 def CheckRule (self, dpidstr, src=0):
  try:
   entry = self.firewall[(dpidstr, src)]
   if (entry == True):
    log.debug("Rule (%s) found in %s: FORWARD",
           src, dpidstr)
   else:
    log.debug("Rule (%s) found in %s: DROP",
           src, dpidstr)
   return entry
  except KeyError:
   log.debug("Rule (%s) NOT found in %s: DROP",
          src, dpidstr)
   return False

 def _handle_PacketIn (self, event):
  """
  implémenter notre algorithme
  """

  packet = event.parsed

  def flood (message = None):
   """ Innondation des paquets """
   msg = of.ofp_packet_out()
   if time.time() - self.connection.connect_time >= _flood_delay:
    # attendre un petit moment après la connexion avant l'inondation
    if self.hold_down_expired is False:
        self.hold_down_expired = True
        log.info("%s: Flood hold-down expired -- flooding",
        dpid_to_str(event.dpid))

    if message is not None: log.debug(message)
      msg.actions.append(of.ofp_action_output(port = of.OFPP_FLOOD))
    else:
    pass
    msg.data = event.ofp
    msg.in_port = event.port
    self.connection.send(msg)

    def drop (duration = None):
   """
   supprimer ce paquet et les paquets semblables pendant une durée de temps
   """
   if duration is not None:
    if not isinstance(duration, tuple):
     duration = (duration,duration)
    msg = of.ofp_flow_mod()
    msg.match = of.ofp_match.from_packet(packet)
```

```
    msg.idle_timeout = duration[0]
    msg.hard_timeout = duration[1]
    msg.buffer_id = event.ofp.buffer_id
    self.connection.send(msg)
  elif event.ofp.buffer_id is not None:
    msg = of.ofp_packet_out()
    msg.buffer_id = event.ofp.buffer_id
    msg.in_port = event.port
    self.connection.send(msg)

self.macToPort[packet.src] = event.port # 1

# Obtenir DPID du switch connecté
dpidstr = dpid_to_str(event.connection.dpid)

# vérifier les règles
if self.CheckRule(dpidstr, packet.src) == False:
  drop()
  return

if not self.transparent: # 2
  if packet.type == packet.LLDP_TYPE or packet.dst.isBridgeFiltered():
    drop() # 2a
    return

if packet.dst.is_multicast:
  flood() # 3a
else:
  if packet.dst not in self.macToPort: # 4
    flood("Port for %s unknown -- flooding" % (packet.dst,)) # 4a
  else:
    port = self.macToPort[packet.dst]
    if port == event.port: # 5
      # 5a
      log.warning("Same port for packet from %s -> %s on %s.%s.  Drop."
          % (packet.src, packet.dst, dpid_to_str(event.dpid), port))
      drop(10)
      return
    # 6
    log.debug("installing flow for %s.%i -> %s.%i" %
          (packet.src, event.port, packet.dst, port))
    msg = of.ofp_flow_mod()
    msg.match = of.ofp_match.from_packet(packet, event.port)
    msg.idle_timeout = 10
    msg.hard_timeout = 30
    msg.actions.append(of.ofp_action_output(port = port))
    msg.data = event.ofp # 6a
    self.connection.send(msg)
```

Tutorial

Setup

Reference
http://www.openflowswitch.org/foswiki/bin/view/OpenFlow/MininetGettingStarted for more thorough Mininet walkthrough if desired

Necessary Downloads

1. Download VM at http://www.cs.princeton.edu/courses/archive/fall10/cos561/assignments/COS561Tutorial.zip
2. Download VirtualBox at http://www.virtualbox.org/wiki/Downloads (VMWare will work too but VirtualBox is free)

Installing VM image (In VirtualBox)
The image we provide already has Mininet and Nox installed on it so we need to load it into VirtualBox.

1. **Open** VirtualBox.
2. Click **New** then **Next**
3. Enter desired name for VM (Ex: COS561Tutorial). OS: Linux. Version: Ubuntu. Click **Next**
4. Choose desired memory (Default 512 MB is fine). Click **Next**
5. Check **Use existing hard disk**. The folder button will open a pop-up. Click **add** and locate OpenFlowTutorial.vmdk. Click **Open**. Click **Select**.
6. Click **Continue**
7. You have installed your VM image!

Configuring VirtualBox for SSH
http://www.linuxjournal.com/content/tech-tip-port-forwarding-virtualbox-vboxmanage
The VM image we provided is only command line. We will need to SSH and use X Forwarding in order to load certain graphic application. There are subtle differences in this step between Mac/Linux and Windows, please follow the specific instructions for your machine.

Mac/Linux Instructions
Enable VM for SSH (In VirtualBox)

1. Select your VM and click **Settings**.
2. Go to the **Network** tab and click on **Advanced**
3. Check **Enable Network Adapter**. Attached to: **NAT**. Adapter type: PCnet-FAST III (Am79C973). Check **Cable Connected**.

Configure VM for SSH (through Terminal)

1. Open up Terminal and find the Virtualbox Application directory where VirtualBox is installed (the VM image should not be running)
2. Enter the following commands substituting *"VM Name Here"* with your VM name from above (Ex: COS561Tutorial)

```
$ VBoxManage setextradata "VM Name Here" \
      "VBoxInternal/Devices/pcnet/0/LUN#0/Config/guestssh/Protocol" TCP

$ VBoxManage setextradata "VM Name Here" \
      "VBoxInternal/Devices/pcnet/0/LUN#0/Config/guestssh/GuestPort" 22

$ VBoxManage setextradata "VM Name Here" \
     "VBoxInternal/Devices/pcnet/0/LUN#0/Config/guestssh/HostPort" 2222
```

Starting SSH Session

1. Open VirtualBox
2. **Start** the VM that we created (Ex: COS561Tutorial) and let it proceed until it asks for username. We will be using SSH to login to the VM. Login with usersname/pw: mininet/mininet
3. In VM image, *sudo dhclient eth1* (This is to "seed" the ip address. It's a strange issue that we found with Macs)
4. Open Terminal and SSH to VM image
   ssh -Y -l mininet -p 2222 localhost

If everything has proceeded correctly, you should see a couple folders (Ex: mininet, openflow, noxcore etc.) indicating that you have logged into the VM session. We can now proceed with developing inside the VM.

Windows Instructions

1. Select your VM and click **Settings**.
2. Go to the **Network** tab and click on **Advanced**
3. Check **Enable Network Adapter**. Attached to: **NAT**. Adapter type: PCnet-FAST III (Am79C973). Check **Cable Connected**.

Configuring VM for SSH (through CMD)

Open up command prompt and find your Virtualbox application directory where it VirtualBox is installed. (Be sure that the VM image is not running)

3. Enter the following commands substituting *"VM Name Here"* with your VM name from above (Ex: COS561Tutorial)

```
$ VBoxManage setextradata "VM Name Here" \
      "VBoxInternal/Devices/pcnet/0/LUN#0/Config/guestssh/Protocol" TCP

$ VBoxManage setextradata "VM Name Here" \
      "VBoxInternal/Devices/pcnet/0/LUN#0/Config/guestssh/GuestPort" 22

$ VBoxManage setextradata "VM Name Here" \
     "VBoxInternal/Devices/pcnet/0/LUN#0/Config/guestssh/HostPort" 2222
```

Enable X-Forwarding (In SSH application)

We will be using SSH to connect to the VM image when it is running

1. Choose your favorite SSH client and make sure that X Forwarding is **enabled**

ARCHITECTURE LOGICIELLE DE MININET