

Réseaux et systèmes autonomes

Rapport de projet :

« Mise en Place d'une Architecture SD-WAN avec Open vSwitch et WireGuard »

Réalisée par :

Idriss KOURBANHOUSSEN (22218639)

Encadrant : Sami SOUIHI

SOMMAIRE

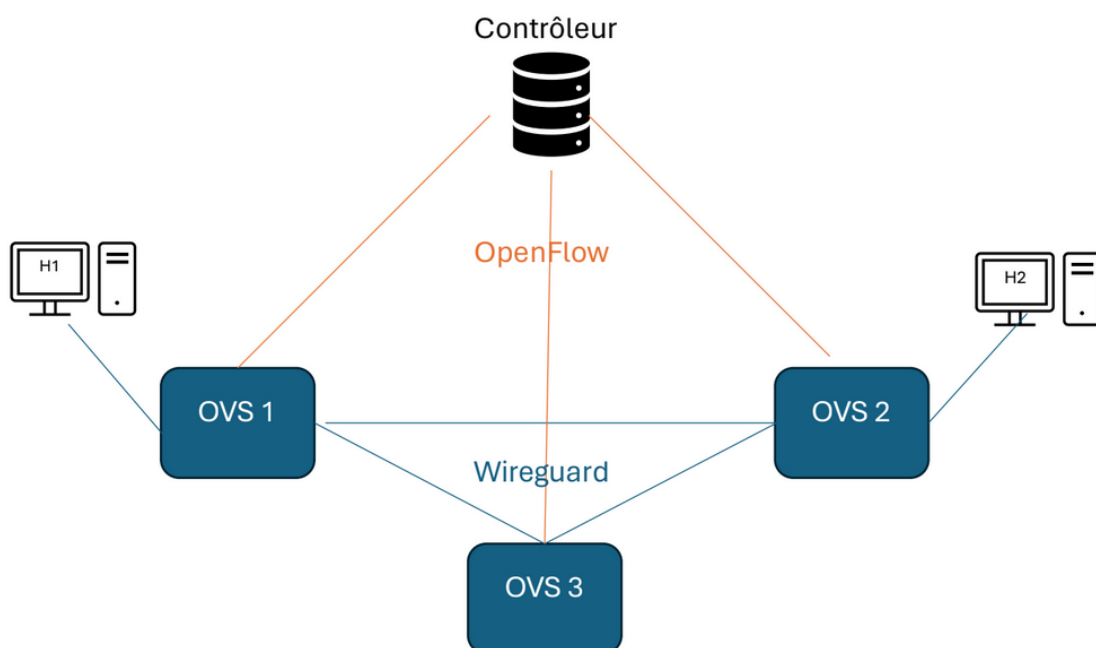
- I. Présentation du Projet**
- II. Création des Instances Open vSwitch (OVS)**
- III. Interconnexion des OVS en Graphe Complet avec Wireguard**
 - a. Établissement des connexions**
 - b. Mise en place des tunnels Wireguard**
 - c. Vérifications des tunnels Wireguard**
- IV. Contrôle SDN avec un Contrôleur (POX)**
- V. Démonstration de Connectivité**
 - a. Configuration des hôtes virtuelles**
 - b. Démonstration de connectivité entre les 2 hôtes**
- VI. Références**

I. Présentation du Projet

Ce projet a pour objectif de mettre en place une architecture SD-WAN avec Open vSwitch et Wireguard.

Comme on peut le voir sur la figure ci-après, notre architecture SD-WAN est composée de trois instances d'Open vSwitch (OVS1, OVS2 et OVS3), et elles sont interconnectées en graphe complet grâce à des tunnels Wireguard, ce qui leur permettra de communiquer de façon sécurisée entre elles. Afin d'avoir une architecture SD-WAN, j'ai mis en place un contrôleur POX sur ma machine virtuelle hôte qui va gérer les OVS, en utilisant une configuration inbound qui était par défaut. De plus, j'ai ajouté deux hôtes virtuels, h1 et h2, qui sont respectivement connectés à l'instance OVS1 et OVS2.

L'objectif à la fin de ce projet est donc de démontrer la connectivité entre les deux hôtes, h1 et h2, grâce à un ping. Ce qui démontrera que l'architecture mise en place fonctionne correctement.



II. Création des Instances Open vSwitch (OVS)

N'ayant pas assez de mémoire sur mon ordinateur pour mettre en place les instances Open vSwitch sous différentes machines virtuelles, j'ai choisi de les héberger dans des conteneurs Docker.

Pour précision, l'ensemble du projet a été réalisé sur une machine virtuelle Ubuntu 22.04 sous VMware.

Voici les étapes que j'ai suivies pour la création des instances Open vSwitch :

- Installation de Docker en suivant les étapes précisées dans la documentation Docker ci-joint : <https://docs.docker.com/engine/install/ubuntu/>

- Création et démarrage du conteneur Docker **OVS1** avec Open vSwitch :
 - `sudo docker run --cap-add=NET_ADMIN -p 6633:6633 -itd --name ovs1 ubuntu:latest`
 - Dans cette commande, j'ai ajouté la capacité `NET_ADMIN` au conteneur car elle va nous donner les privilèges pour la configuration des interfaces réseaux. Cela nous sera utile pour le reste du projet.
 - J'ai aussi précisé le mappage du port 6633, car le contrôleur Pox va écouter sur le port 6633. En effet, lors de mon premier essai, je n'ai pas pu établir une connexion avec le contrôleur sans ce mappage.
 - `Sudo docker exec -it --privileged ovs1 bash`
 - Pour créer une session interactive (bash) à l'intérieur du conteneur ovs.
 - J'ai lancé la session avec les privilèges, ils nous seront utiles pour le lancement du service `openvswitch-switch`.
 - `apt-get install --reinstall linux-modules-$(uname -r)`
 - `apt update`
 - `apt-get install -y openvswitch-switch`
 - `service openvswitch-switch start`

- Création et démarrage du conteneur Docker **OVS2** avec Open vSwitch :
 - `sudo docker run --cap-add=NET_ADMIN -p 6633:6633 -itd --name ovs2 ubuntu:latest`
 - `Sudo docker exec -it --privileged ovs2 bash`
 - `apt-get install --reinstall linux-modules-$(uname -r)`
 - `apt update`
 - `apt-get install -y openvswitch-switch`
 - `service openvswitch-switch start`

- Création et démarrage du conteneur Docker **OVS3** avec Open vSwitch :
 - `sudo docker run --cap-add=NET_ADMIN -p 6633:6633 -itd --name ovs3 ubuntu:latest`
 - `Sudo docker exec -it --privileged ovs3 bash`
 - `apt-get install --reinstall linux-modules-$(uname -r)`
 - `apt update`
 - `apt-get install -y openvswitch-switch`
 - `service openvswitch-switch start`

Comme on peut le voir sur la capture ci-après, nous avons bien les 3 conteneurs OVS1, OVS2 et OVS3. Et avec Openvswitch qui fonctionne parfaitement, sans afficher d'erreurs.

```

idriss@idriss-virtual-machine: ~
idriss@idriss-virtual-machine:~$ sudo docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS        NAMES
ed56cfb1312f   ubuntu:latest  "/bin/bash"             20 hours ago   Up 4 minutes   ovs3
b73c16dc75dc   ubuntu:latest  "/bin/bash"             20 hours ago   Up 4 minutes   ovs2
a06404897adc   ubuntu:latest  "/bin/bash"             21 hours ago   Up 4 minutes   ovs1
idriss@idriss-virtual-machine:~$ sudo docker exec --privileged -it ovs1 service openvswitch-switch status
ovsdb-server is running with pid 79
ovs-vswitchd is running with pid 95
idriss@idriss-virtual-machine:~$ sudo docker exec --privileged -it ovs2 service openvswitch-switch status
ovsdb-server is running with pid 76
ovs-vswitchd is running with pid 91
idriss@idriss-virtual-machine:~$ sudo docker exec --privileged -it ovs3 service openvswitch-switch status
ovsdb-server is running with pid 55
ovs-vswitchd is running with pid 70
idriss@idriss-virtual-machine:~$
  
```

III. Interconnexion des OVS en Graphe Complet avec Wireguard

a. Établissement des connexions

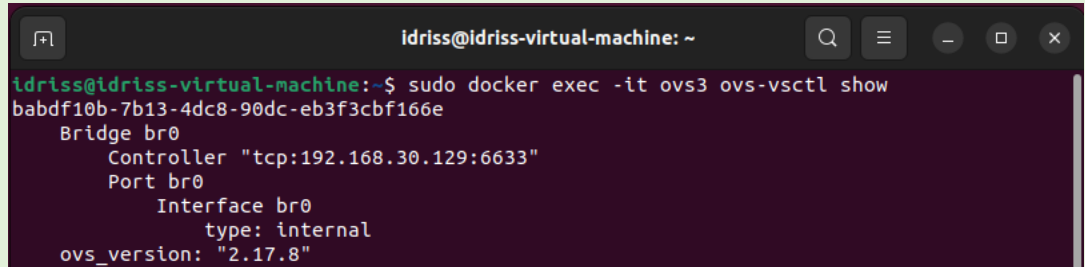
- Pour établir la connexion, j'ai tout d'abord commencé par créer des ponts (bridges) sur chaque conteneur OVS, c'est grâce à ce pont que les OVS pourront établir une connexion avec le contrôleur. Voici les commandes d'**OVS1** :
 - `docker exec -it ovs1 ovs-vsctl add-br br0`
 - Le pont sur lequel OVS1 va communiquer avec le contrôleur : br0
 - `docker exec -it ovs1 ovs-vsctl set-controller br0 tcp:192.168.30.129:6633`
 - Grâce à cette commande, je précise au conteneur qu'il doit communiquer avec le contrôleur sur le pont br0, à l'adresse 192.168.30.129 et sur le port 6633. Cette adresse est celui de ma machine virtuelle, car je vais lancer mon contrôleur POX sur cette dernière.
 - On peut voir sur la capture ci-après que la configuration du pont à bien été prise en compte.

```
idriiss@idriiss-virtual-machine: ~  
idriiss@idriiss-virtual-machine:~$ sudo docker exec -it ovs1 ovs-vsctl show  
dbc8e585-e4a9-4b33-8ecd-66c678cc2a50  
    Bridge br0  
        Controller "tcp:192.168.30.129:6633"  
        Port br0  
            Interface br0  
                type: internal
```

- J'ai réalisé la même manipulation sur le conteneur **OVS2** :
 - `docker exec -it ovs2 ovs-vsctl add-br br0`
 - `docker exec -it ovs2 ovs-vsctl set-controller br0 tcp:192.168.30.129:6633`
 - On peut voir sur la capture, ci-après que la configuration du pont à bien été prise en compte.

```
idriiss@idriiss-virtual-machine: ~  
idriiss@idriiss-virtual-machine:~$ sudo docker exec -it ovs2 ovs-vsctl show  
dd01a840-1428-4a72-b4fa-22df34c69799  
    Bridge br0  
        Controller "tcp:192.168.30.129:6633"  
        Port br0  
            Interface br0  
                type: internal
```

- J'ai réalisé la même manipulation sur le conteneur **OVS3** :
 - `docker exec -it ovs3 ovs-vsctl add-br br0`
 - `docker exec -it ovs3 ovs-vsctl set-controller br0 tcp:192.168.30.129:6633`
 - On peut voir sur la capture, ci-après que la configuration du pont à bien été réalisé.



```

idriss@idriss-virtual-machine: ~
idriss@idriss-virtual-machine:~$ sudo docker exec -it ovs3 ovs-vsctl show
babdf10b-7b13-4dc8-90dc-eb3f3cbf166e
    Bridge br0
        Controller "tcp:192.168.30.129:6633"
        Port br0
            Interface br0
                type: internal
    ovs_version: "2.17.8"
  
```

b. Mise en place des tunnels Wireguard

- Pour réaliser cette partie, je me suis inspiré de cette vidéo qui détail la mise en œuvre d'un tunnel Wireguard entre 2 hosts :

<https://www.youtube.com/watch?v=x5INmYtrBK4>

- Dans notre réseau SD-WAN nous avons fait le choix d'utiliser un protocole de réseau virtuel privé (VPN) afin de fournir une solution sécurisée, rapide et flexible pour créer une connexion entre les différents OVS. Nous allons donc créer des tunnels Wireguard entre les différents OVS. Commençons par **l'OVS1** :
 - `apt install wireguard`
 - `wg genkey > private.key`
 - Ici on génère une clé privée, et on l'enregistre dans un fichier `private.key`, on pourra donc l'avoir directement avec la commande `cat`.
 - `wg pubkey > public.key < private.key`
 - Ici on génère la clé publique, et on l'enregistre dans un fichier `public.key`, on pourra donc l'avoir directement avec la commande `cat`.
 - Nous aurons besoin de ces clés privées et publiques pour la configuration des tunnels Wireguard, cela va permettre aux OVS d'échanger des informations de façon sécurisée.
 - `sudo nano /etc/wireguard/wg0.conf`
 - J'ai créé ici un fichier de configuration Wiregard pour définir les paramètres spécifiques de la connexion Wireguard.

- Comme on peut le voir sur la capture ci-après, nous avons précisé la clé privée du conteneur, son port d'écoute (par défaut le Port d'écoute de Wireguard est le 51820), et j'ai attribué une adresse IP à l'interface. J'ai ensuite précisé les deux pairs distants en spécifiant leurs clés publiques, adresses IP et ports.

Allowed IPs spécifie les adresses IP qui sont autorisées à travers la connexion et l'endpoint spécifie l'adresse IP et le port du serveur (ou du pair) distant auquel le client doit se connecter.

```

root@a06404897adc: /
idriiss@idriiss-vir... x root@a0640489... x root@b73c16dc... x root@ed56cfb1... x
GNU nano 6.2 /etc/wireguard/wg0.conf
#OV5i configuration
[Interface]
Address = 10.0.20.1/24
PrivateKey = iD6TTHK1Z55fEfLnMv/h7xKbDH9xyJdgSp3GRWtQSEw=
ListenPort = 51820

[Peer]
PublicKey = aSL52e59WnlQ/0YQJHCqd5uu8eDlvBXlEccdeBjiGWE=
AllowedIPs = 10.0.20.2/32
Endpoint = 172.17.0.3:51820

[Peer]
PublicKey = PxRpBuAtrzezQlEysUCXtfX0ESYqaK4q1XAIRgEeTWY=
AllowedIPs = 10.0.20.3/32
Endpoint = 172.17.0.4:51820

```

- sudo wg-quick up wg0
 - Pour démarrer le service Wireguard avec la configuration spécifiée.
- sudo iptables -A INPUT -p udp --dport 51820 -j ACCEPT
 - J'ai précisé ici les règles de pare-feu pour autoriser le trafic sur le port 51820.
- On peut voir sur la capture ci-après, que les tunnels Wireguard ont bien été établis avec les bonnes interfaces et les bonnes connexions

```

root@a06404897adc: /
idriiss@idriiss-vir... x root@a0640489... x root@b73c16dc... x root@ed56cfb1... x
root@a06404897adc: /# wg show
Interface: wg0
public key: gsdWdCBuzby3o0wNBj35jcNXAFLr7wCASd2jzWbgHM=
private key: (hidden)
listening port: 51820

peer: aSL52e59WnlQ/0YQJHCqd5uu8eDlvBXlEccdeBjiGWE=
endpoint: 172.17.0.3:51820
allowed ips: 10.0.20.2/32
latest handshake: 9 minutes, 32 seconds ago
transfer: 348 B received, 436 B sent

peer: PxRpBuAtrzezQlEysUCXtfX0ESYqaK4q1XAIRgEeTWY=
endpoint: 172.17.0.4:51820
allowed ips: 10.0.20.3/32
root@a06404897adc: /#

```


- Voici les configurations de l'OVS2 :
 - apt install wireguard
 - wg genkey > private.key
 - wg pubkey > public.key < private.key
 - sudo nano /etc/wireguard/wg0.conf
 - Comme on peut le voir sur la capture ci-après, nous avons précisé la clé privée du conteneur, son port d'écoute (par défaut le Port d'écoute de Wireguard est le 51820), et j'ai attribué une adresse IP à l'interface. J'ai ensuite précisé les deux pairs distants en spécifiant leurs clés publiques, adresses IP et ports.

```

root@b73c16dc75dc: /
GNU nano 6.2 /etc/wireguard/wg0.conf
#OVS2 configuration
[Interface]
Address = 10.0.20.2/24
PrivateKey = EI/M073AzhX7y dq2benyeq6ExSDE+RiRlu5uAFAGXVE=
ListenPort = 51820

[Peer]
PublicKey = gsdWdCBuzby3o00wNBj35jcNXAfLr7wCASd2jzWbgHM=
AllowedIPs = 10.0.20.1/32
Endpoint = 172.17.0.2:51820

[Peer]
PublicKey = PxRpBuAtrzezQiEysUCXtfX0ESYqaK4q1XAIRgEeTWY=
AllowedIPs = 10.0.20.3/32
Endpoint = 172.17.0.4:51820

```

- sudo wg-quick up wg0
- sudo iptables -A INPUT -p udp --dport 51820 -j ACCEPT
- On peut voir sur la capture ci-après, que les tunnels Wireguard ont bien été établis avec les bonnes interfaces et les bonnes connexions.

```

root@b73c16dc75dc: /
idriiss@idriiss-vir... x root@a0640489... x root@b73c16dc... x root@ed56cfb1... x
root@b73c16dc75dc: /# wg show
interface: wg0
  public key: aSL52e59WnlQ/0YQJHCqd5uu8eDInvBXLEccdeBjiGWE=
  private key: (hidden)
  listening port: 51820

peer: gsdWdCBuzby3o00wNBj35jcNXAfLr7wCASd2jzWbgHM=
  endpoint: 172.17.0.2:51820
  allowed ips: 10.0.20.1/32
  latest handshake: 10 minutes ago
  transfer: 436 B received, 348 B sent

peer: PxRpBuAtrzezQiEysUCXtfX0ESYqaK4q1XAIRgEeTWY=
  endpoint: 172.17.0.4:51820
  allowed ips: 10.0.20.3/32
root@b73c16dc75dc: /#

```

- Voici les configurations de l'OVS3 :
 - apt install wireguard
 - wg genkey > private.key
 - wg pubkey > public.key < private.key
 - sudo nano /etc/wireguard/wg0.conf
 - Comme on peut le voir sur la capture ci-après, nous avons précisé la clé privée du conteneur, son port d'écoute (par défaut le Port d'écoute de Wireguard est le 51820), et j'ai attribué une adresse IP à l'interface. J'ai ensuite précisé les deux pairs distants en spécifiant leurs clés publiques, adresses IP et ports.

```

root@ed56cfb1312f: /
idriiss@idriiss-vir... x root@a0640489... x root@b73c16dc... x root@ed56cfb1... x
GNU nano 6.2 /etc/wireguard/wg0.conf
#OVS2 configuration
[Interface]
Address = 10.0.20.3/24
PrivateKey = YKWuom+Wnd+WZWuQeRJUWNsbnXE7BKTEg1fxaXFhD2Q=
ListenPort = 51820

[Peer]
PublicKey = aSL52eS9WnlQ/0YQJHCqd5uu8eDIvBXlEccdeBjiGWE=
AllowedIPs = 10.0.20.2/32
Endpoint = 172.17.0.3:51820

[Peer]
PublicKey = gsdWdCBuzby3o00wNBj35jcNXAfLr7wCASd2jzWbgHM=
AllowedIPs = 10.0.20.1/32
Endpoint = 172.17.0.2:51820

```

- sudo wg-quick up wg0
- sudo iptables -A INPUT -p udp --dport 51820 -j ACCEPT
- On peut voir sur la capture ci-après, que les tunnels Wireguard ont bien été établis avec les bonnes interfaces et les bonnes connexions.

```

root@ed56cfb1312f: /
idriiss@idriiss-vir... x root@a0640489... x root@b73c16dc... x root@ed56cfb1... x
root@ed56cfb1312f:/# wg show
interface: wg0
  public key: PxRpBuAtrzezQiEysUCxtfX0ESYqaK4q1XAIRgEeTWY=
  private key: (hidden)
  listening port: 51820

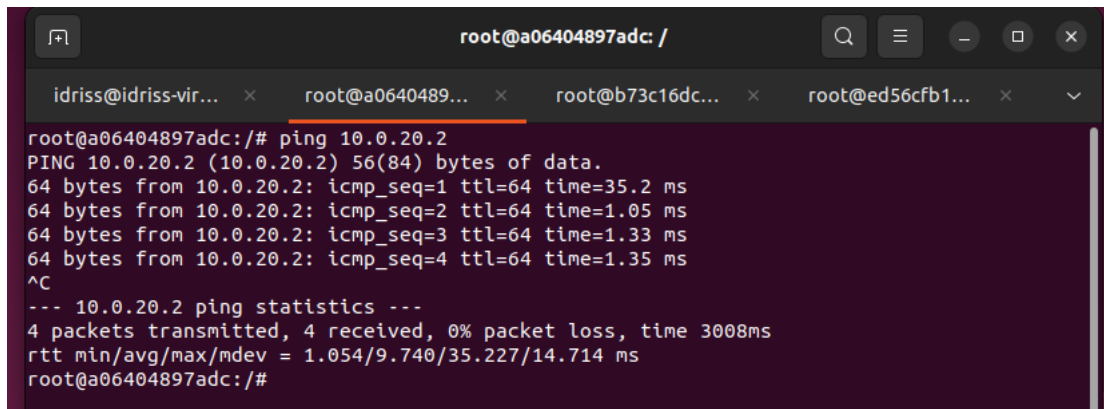
peer: aSL52eS9WnlQ/0YQJHCqd5uu8eDIvBXlEccdeBjiGWE=
  endpoint: 172.17.0.3:51820
  allowed ips: 10.0.20.2/32

peer: gsdWdCBuzby3o00wNBj35jcNXAfLr7wCASd2jzWbgHM=
  endpoint: 172.17.0.2:51820
  allowed ips: 10.0.20.1/32
root@ed56cfb1312f:/#

```

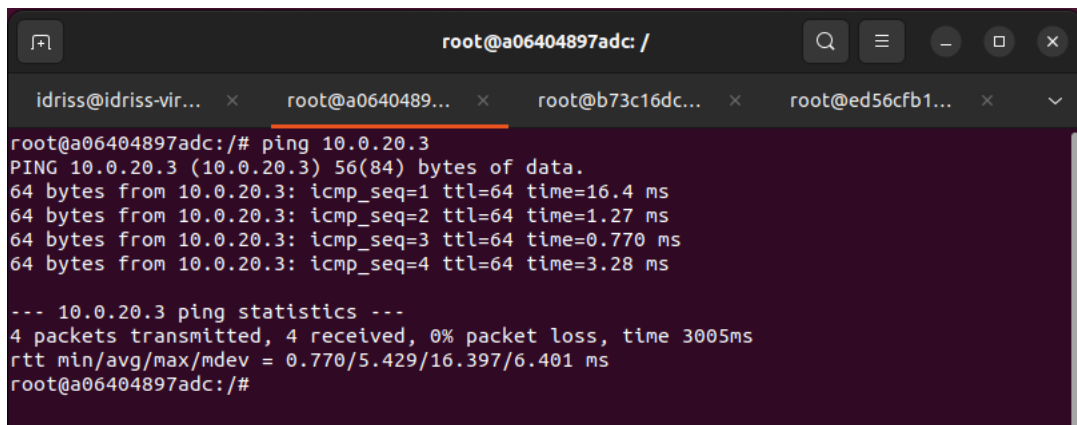
c. Vérifications des tunnels Wireguard

- Ping depuis OVS1 vers OVS2 :
 - apt-get install -y iputils-ping



```
root@a06404897adc: /
idriiss@idriiss-vir... x root@a0640489... x root@b73c16dc... x root@ed56cfb1... x
root@a06404897adc: /# ping 10.0.20.2
PING 10.0.20.2 (10.0.20.2) 56(84) bytes of data.
64 bytes from 10.0.20.2: icmp_seq=1 ttl=64 time=35.2 ms
64 bytes from 10.0.20.2: icmp_seq=2 ttl=64 time=1.05 ms
64 bytes from 10.0.20.2: icmp_seq=3 ttl=64 time=1.33 ms
64 bytes from 10.0.20.2: icmp_seq=4 ttl=64 time=1.35 ms
^C
--- 10.0.20.2 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3008ms
rtt min/avg/max/mdev = 1.054/9.740/35.227/14.714 ms
root@a06404897adc: /#
```

- Ping depuis OVS1 vers OVS3 :



```
root@a06404897adc: /
idriiss@idriiss-vir... x root@a0640489... x root@b73c16dc... x root@ed56cfb1... x
root@a06404897adc: /# ping 10.0.20.3
PING 10.0.20.3 (10.0.20.3) 56(84) bytes of data.
64 bytes from 10.0.20.3: icmp_seq=1 ttl=64 time=16.4 ms
64 bytes from 10.0.20.3: icmp_seq=2 ttl=64 time=1.27 ms
64 bytes from 10.0.20.3: icmp_seq=3 ttl=64 time=0.770 ms
64 bytes from 10.0.20.3: icmp_seq=4 ttl=64 time=3.28 ms

--- 10.0.20.3 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3005ms
rtt min/avg/max/mdev = 0.770/5.429/16.397/6.401 ms
root@a06404897adc: /#
```

On peut donc voir que les tunnels Wireguard fonctionnent parfaitement entre nos trois OVS.

IV. Contrôle SDN avec un Contrôleur (POX)

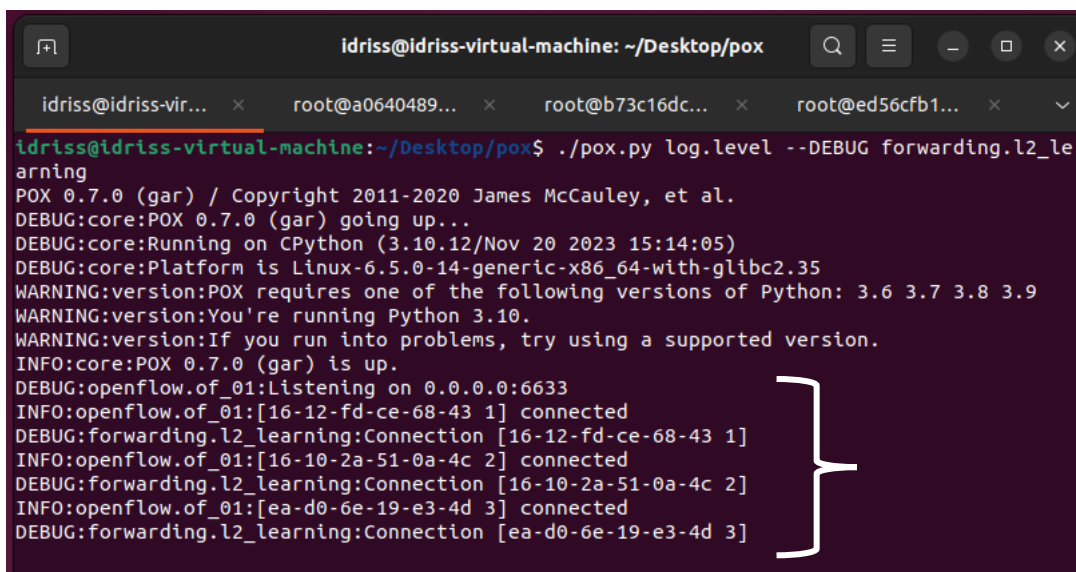
Pour avoir une architecture SDN, nous avons besoin d'un contrôleur qui est au centre du réseau et qui gère le comportement des commutateurs réseau. Les contrôleurs SDN sont responsables de la communication avec les commutateurs, de la collecte d'informations sur le réseau et de la prise de décisions concernant le transfert des données.

J'ai choisi d'utiliser POX car il s'installe beaucoup plus rapidement que le contrôleur ONOS.

Voici les commandes que j'ai réalisé sur ma machine virtuelle (host) :

- Nous avons déjà précisé dans la première étape, dans chaque OVS, l'adresse IP et le Port du contrôleur POX (@ IP de la VM host).
 - Pour OVS1: `docker exec -it ovs1 ovs-vsctl set-controller brO tcp:192.168.30.129:6633`
- Installation au préalable de git
- `git clone https://github.com/noxrepo/pox.git`
- `cd pox`
- `./pox.py log.level --DEBUG forwarding.l2_learning`

On peut voir sur la capture ci-après, que le contrôleur établit une connexion et est connecté aux différents commutateurs du réseau : OVS1, OVS2 et OVS3.



```
idriiss@idriiss-virtual-machine: ~/Desktop/pox
idriiss@idriiss-vir... x root@a0640489... x root@b73c16dc... x root@ed56cfb1... x
idriiss@idriiss-virtual-machine:~/Desktop/pox$ ./pox.py log.level --DEBUG forwarding.l2_learning
POX 0.7.0 (gar) / Copyright 2011-2020 James McCauley, et al.
DEBUG:core:POX 0.7.0 (gar) going up...
DEBUG:core:Running on CPython (3.10.12/Nov 20 2023 15:14:05)
DEBUG:core:Platform is Linux-6.5.0-14-generic-x86_64-with-glibc2.35
WARNING:version:POX requires one of the following versions of Python: 3.6 3.7 3.8 3.9
WARNING:version:You're running Python 3.10.
WARNING:version:If you run into problems, try using a supported version.
INFO:core:POX 0.7.0 (gar) is up.
DEBUG:openflow.of_01:Listening on 0.0.0.0:6633
INFO:openflow.of_01:[16-12-fd-ce-68-43 1] connected
DEBUG:forwarding.l2_learning:Connection [16-12-fd-ce-68-43 1]
INFO:openflow.of_01:[16-10-2a-51-0a-4c 2] connected
DEBUG:forwarding.l2_learning:Connection [16-10-2a-51-0a-4c 2]
INFO:openflow.of_01:[ea-d0-6e-19-e3-4d 3] connected
DEBUG:forwarding.l2_learning:Connection [ea-d0-6e-19-e3-4d 3]
```

V. Démonstration de Connectivité

a. Configuration des hôtes virtuelles

Pour démontrer la connectivité dans notre réseau SD-WAN, nous avons créé deux hôtes virtuels h1 et h2 qui sont respectivement connecté à OVS1 et OVS2.

- Voici la configuration de **h1 et OVS1** :
 - D'abord, voici la configuration dans h1
 - `sudo docker run --cap-add=NET_ADMIN -itd --name h1 ubuntu`
 - J'ai décidé de créer les hôtes virtuels dans des conteneurs Dockers, tout comme pour les instances OVS.
 - `ip addr add 192.168.1.2/24 dev eth0`
 - Ici je configure l'adresse IP de h1 sur l'interface eth0
 - `ip link set dev eth0 up`
 - `apt-get install -y openvswitch-switch`
 - `service openvswitch-switch start`
 - J'installe et je lance le service openvswitch.
 - `ovs-vsctl add-br ovs1`
 - J'ajoute le pont ovs1 pour permettre la connexion avec OVS1.
 - `ip link add eth1 type veth peer name ovs1-h1`
 - `h1 ip link set dev eth1 up`
 - `sudo ovs-vsctl add-port ovs1 ovs1-h1`
 - Je crée ici le port ovs1-h1 qui va permettre la connexion à OVS1.
 - Maintenant passons à la configuration dans OVS1.
 - `ovs-vsctl add-br ovs1`
 - J'ajoute le pont ovs1 pour permettre la connexion avec OVS1.
 - `sudo ovs-vsctl add-port ovs1 ovs1-h1`
 - Je crée ici le port ovs1-h1 qui va permettre la connexion à OVS1.
 - `ip link add ovs1-h1 type veth peer name eth1`
 - `sudo ip addr add 192.168.1.1/24 dev ovs1-h1`
 - Ici je configure l'adresse IP de OVS1 sur l'interface ovs1-h1.
L'adresse de l'interface de OVS1 avec h1 est 192.168.1.1 et celui de h1 avec OVS1 est 192.168.1.2
 - `sudo ip link set dev ovs1-h1 up`
 - Maintenant, notre hôte virtuelle h1, contenue dans un conteneur Docker et connecté à l'instance OVS1 au port ovs1-h1.

- **Voici la configuration de h2 et OVS2 :**
 - **D'abord, voici la configuration dans h2**
 - `sudo docker run --cap-add=NET_ADMIN -itd --name h2 ubuntu`
 - `ip addr add 192.168.2.2/24 dev eth0`
 - `ip link set dev eth0 up`
 - `apt-get install -y openvswitch-switch`
 - `service openvswitch-switch start`
 - `ovs-vsctl add-br ovs2`
 - `ip link add eth1 type veth peer name ovs2-h2`
 - `ip link set dev eth1 up`
 - `sudo ovs-vsctl add-port ovs2 ovs2-h2`
 - **Maintenant passons à la configuration dans OVS2.**
 - `ovs-vsctl add-br ovs2`
 - `sudo ovs-vsctl add-port ovs2 ovs2-h2`
 - `ip link add ovs2-h2 type veth peer name eth1`
 - `sudo ip addr add 192.168.2.1/24 dev ovs2-h2`
 - `sudo ip link set dev ovs2-h2 up`
 - **Maintenant, notre hôte virtuelle h2, contenue dans un conteneur Docker et connecté à l'instance OVS2 au port ovs2-h2.**

b. Démonstration de connectivité entre les 2 hôtes

À présent on peut enfin tester la bonne configuration de notre réseau en testant la connectivité entre nos deux hôtes h1 et h2.

Adresse IP h1 : 172.17.0.5

Adresse IP h2 : 172.17.0.6

- Voici une capture d'un ping depuis h1 vers h2 :

```
root@2d6678b8b1b1:/# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
2: ovs-system: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN group default qlen 1000
    link/ether ca:51:30:8c:49:41 brd ff:ff:ff:ff:ff:ff
3: ovs1: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN group default qlen 1000
    link/ether 3e:ce:fd:b1:84:40 brd ff:ff:ff:ff:ff:ff
12: eth0@if13: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 02:42:ac:11:00:05 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 172.17.0.5/16 brd 172.17.255.255 scope global eth0
        valid_lft forever preferred_lft forever
root@2d6678b8b1b1:/# ping 172.17.0.6
PING 172.17.0.6 (172.17.0.6) 56(84) bytes of data.
64 bytes from 172.17.0.6: icmp_seq=1 ttl=64 time=0.875 ms
64 bytes from 172.17.0.6: icmp_seq=2 ttl=64 time=0.154 ms
64 bytes from 172.17.0.6: icmp_seq=3 ttl=64 time=0.142 ms
64 bytes from 172.17.0.6: icmp_seq=4 ttl=64 time=0.088 ms
^C
--- 172.17.0.6 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3056ms
rtt min/avg/max/mdev = 0.088/0.314/0.875/0.324 ms
root@2d6678b8b1b1:/#
```

- Voici une capture d'un ping depuis h2 vers h1 :

```
root@05f3a45c654c:/# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
2: ovs-system: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN group default qlen 1000
    link/ether 0a:ab:e4:8e:dc:13 brd ff:ff:ff:ff:ff:ff
3: ovs1: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN group default qlen 1000
    link/ether 26:ac:be:42:72:4e brd ff:ff:ff:ff:ff:ff
4: ovs2: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN group default qlen 1000
    link/ether a6:fc:f0:9a:15:43 brd ff:ff:ff:ff:ff:ff
14: eth0@if15: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 02:42:ac:11:00:06 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 172.17.0.6/16 brd 172.17.255.255 scope global eth0
        valid_lft forever preferred_lft forever
root@05f3a45c654c:/# ping 172.17.0.5
PING 172.17.0.5 (172.17.0.5) 56(84) bytes of data.
64 bytes from 172.17.0.5: icmp_seq=1 ttl=64 time=0.990 ms
64 bytes from 172.17.0.5: icmp_seq=2 ttl=64 time=0.152 ms
64 bytes from 172.17.0.5: icmp_seq=3 ttl=64 time=0.109 ms
64 bytes from 172.17.0.5: icmp_seq=4 ttl=64 time=0.099 ms
^C
--- 172.17.0.5 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3045ms
rtt min/avg/max/mdev = 0.099/0.337/0.990/0.377 ms
root@05f3a45c654c:/#
```

VI. Références

- <https://docs.docker.com/engine/install/ubuntu/> : Pour l'installation de Docker.
- <https://www.youtube.com/watch?v=x5INmYtrBK4> : Pour la création des tunnels Wireguard.
- <https://remote-lab.net/sdn-intro-basic-with-ovs-and-pox> : Pour les connexions des OVS avec le contrôleur POX.
- <https://www.brianlinkletter.com/2015/04/using-the-pox-sdn-controller/> : Pour les connexions des OVS avec le contrôleur POX.
- <https://www.openvswitch.org/support/dist-docs/ovs-vsctl.8.html> : Documentation OpenvSwitch.
- <https://github.com/noxrepo/pox> : Pour l'installation de POX.
- [Open AI, ChatGPT](#) : pour régler les problèmes d'importations des différents packages dans les conteneurs Docker.