



# TÉLÉCOMMUNICATION ET RÉSEAUX

## LICENCE 3

ANNÉE-SCOLAIRE 2019 – 2020

### **RAPPORT: Applications des SDN**

Étudiants:

Innonce Disu – Hafid  
Adama Thiam  
Junior Abotsi

Encadrant:

Prof. Samuel Ouya

# **Objectifs**

## **I – Présentation de VXLAN**

### **II – Principe de fonctionnement**

II.1 – Terminologie

II.2 – Format de trame

II.3 – Flux de trafic

    II.4 – Rappel sur l'unicast, le multicast et le broadcast

### **III – Mise en place de VXLAN en unicast**

    III.1 – Cas 1 : VXLAN en unicast sous Linux

    III.2 – Cas 2 : VXLAN en unicast avec l'émulateur Mininet

### **IV – Mise en place de VXLAN en multicast**

### **V – Mise en place de VXLAN avec SDN (Software Defined Networking)**

    V.1 – Cas 1 : Centralisation du contrôle des serveurs OVS interconnectés avec des VM

    V.2 – Cas 2 : Centralisation du contrôle des serveurs OVS interconnectés avec des VM et  
                des équipements physiques

    V.3 – Cas 3 : Déploiement d'un système de communication au sein d'un réseau de  
                superposition centralisé

## I – Présentation de VXLAN

Traditionnellement, tous les centres de données utilisent des VLAN pour appliquer l'isolation Layer2 et au fur et à mesure que les centres de données se développent et que des besoins se font sentir pour étendre les réseaux de couche 2 à travers le centre de données ou peuvent dépasser un centre de données, les lacunes des VLAN sont évidentes. Ces lacunes sont:

Dans un centre de données, des milliers de VLAN sont nécessaires pour partitionner le trafic dans un environnement multi-locataire partageant la même infrastructure L2 / L3 pour un fournisseur de services cloud. La limite actuelle de 4096 VLAN (certains sont réservés) n'est pas suffisante.

En raison de la virtualisation du serveur, chaque machine virtuelle (VM) nécessite une adresse MAC unique et une adresse IP. Il existe donc des milliers d'entrées de table MAC sur les commutateurs en amont. Cela impose une demande beaucoup plus importante sur la capacité de table des commutateurs.

Les VLAN sont trop restrictifs en termes de distance et de déploiement. Le VTP peut être utilisé pour déployer des VLAN sur les commutateurs L2, mais la plupart des gens préfèrent désactiver le VTP en raison de sa nature destructrice.

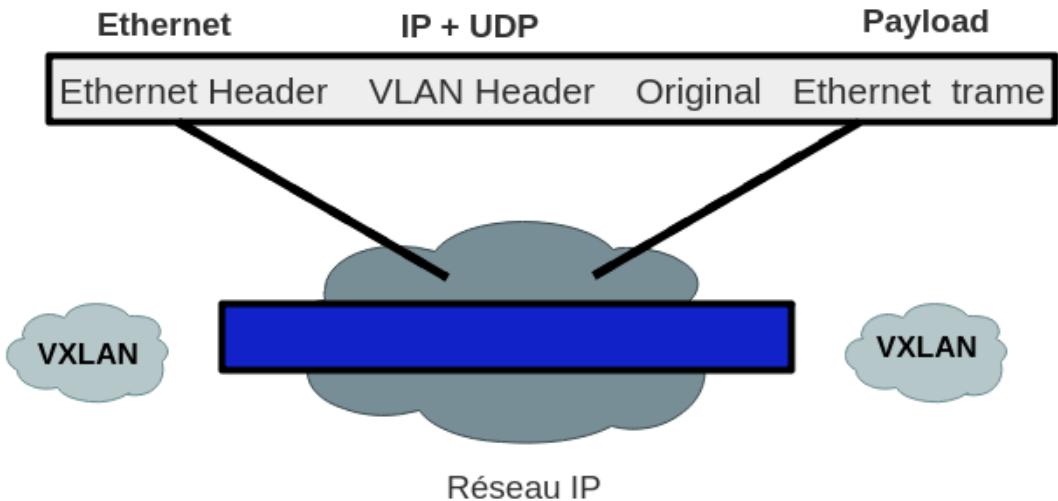
L'utilisation de STP pour fournir une topologie sans boucle L2 désactive la plupart des liaisons redondantes. Par conséquent, ECMP (Equal-Cost Multi-Path) est difficile à réaliser dans un réseau IP.

Très présent dans les technologies réseau liées au Cloud et aux conteneurs, VXLAN est un protocole de tunnelisation qui permet d'étendre un réseau de couche 2 au dessus de réseaux routés. On identifie un réseau VXLAN par sa VNI (VXLAN Network Identifier). Celle-ci est codée sur 24 bits, ce qui donne 16777216 possibilités, on est alors bien loin de la limitation de 4096 induite par les VLANs.

Grâce à ce format, il est possible de faire transiter des trames de niveau 2 dans UDP.

## II – Principe de fonctionnement

Le fonctionnement de VXLAN est simple, il s'agit d'encapsuler une trame Ethernet (couche 2 du modèle OSI), dans un datagramme UDP (couche 4). On veut pouvoir déployer un réseau Ethernet sur un ensemble de sous-réseaux IP. Cette communication doit pouvoir se faire au travers de n'importe quel réseau. Pour cela, on a besoin d'utiliser la couche transport. Par conséquent, la trame Ethernet est : Encapsulée dans un segment UDP, lui-même encapsulé dans un paquet IP, lui-même encapsulé dans une trame Niveau 2 (Ethernet par exemple).



**Figure:** Protocole VXLAN

## II.1 – Terminologie

- Virtual Tunnel End-point : VTEP
- Virtual Tunnel Identifier : VTI
- Virtual Network Identifier : VNI
- VXLAN Header

Les éléments chargés de cette encapsulation (et de la dés-encapsulation) sont appelés VTEP, pour VXLAN Tunnel End Point. Il suffit pour monter un réseau d'overlay VXLAN, que deux VTEP soient en mesure de communiquer en IP et en UDP, sur un numéro de port dédié (le port désigné par l'IANA est 4789, mais selon les implémentations on peut retrouver 8472 comme port par défaut, il est également possible d'en changer).

Un VTEP peut être dit matériel, lorsqu'il s'agit d'un équipement réseau physique disposant de cette fonctionnalité ou logiciel lorsqu'il s'agit d'une implémentation déployée sur un serveur (classiquement un hyperviseur ou un serveur hébergeant des conteneurs).

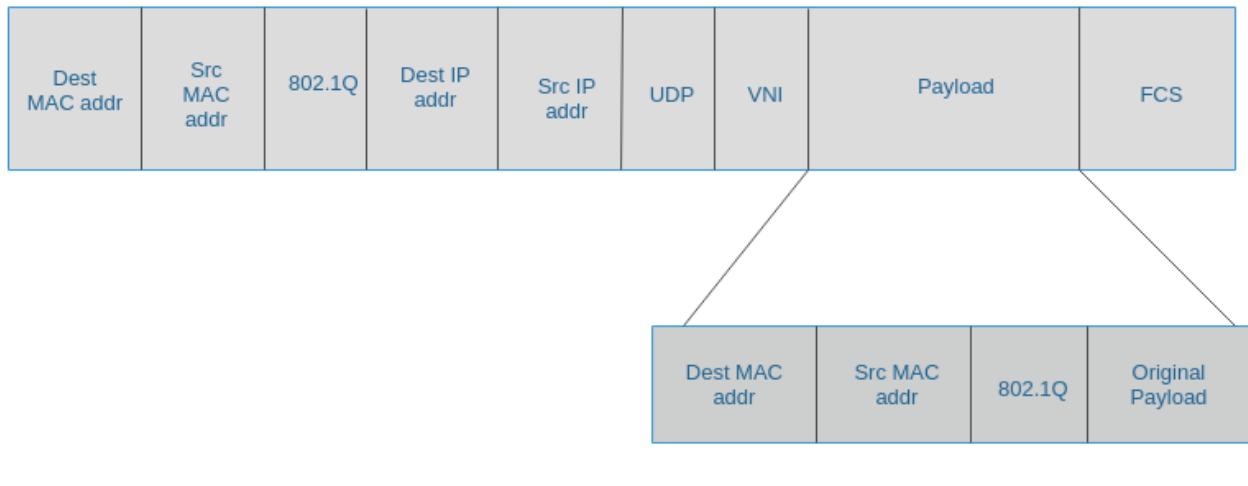
De ce fonctionnement, la problématique qui se pose est de savoir comment un VTEP apprend l'existence des autres VTEP.

Pour rappel il s'agit de faire fonctionner un réseau de couche 2 au dessus d'un réseau de couche 3. Derrière chaque VTEP sont présents plusieurs équipements, soit d'un point de vue réseau, plusieurs adresses MAC. Comment fait le VTEP A pour savoir si l'adresse MAC 00:00:5E:00:53:01, se trouve derrière le VTEP B ou bien derrière le VTEP C, avant de transmettre une trame à son attention ?

La réponse est simple, il va falloir qu'un mécanisme soit mis en place pour que chaque VTEP apprenne les adresses MACs déployées derrière ses homologues. Pour ce faire, le VTEP dispose d'une table de correspondance (adresse MAC → adresse IP d'un VTEP) que l'on appelle la FDB (pour Forwarding DataBase). La question est maintenant de savoir comment la FDB de chaque VTEP va être remplie.

## II.2 – Format de trame

- Header Ethernet utilise la local VTEP MAC et la MAC du routeur : 14 bytes + 4 optional 802.1q
- Encapsulation VXLAN Ip source/destination sont ceux du local/remote VTI
- UDP Header avec
- 24-bits VNI pour atteindre 16 millions de VLAN



**Figure:** trame VXLAN

## II.3 – Flux de trafic

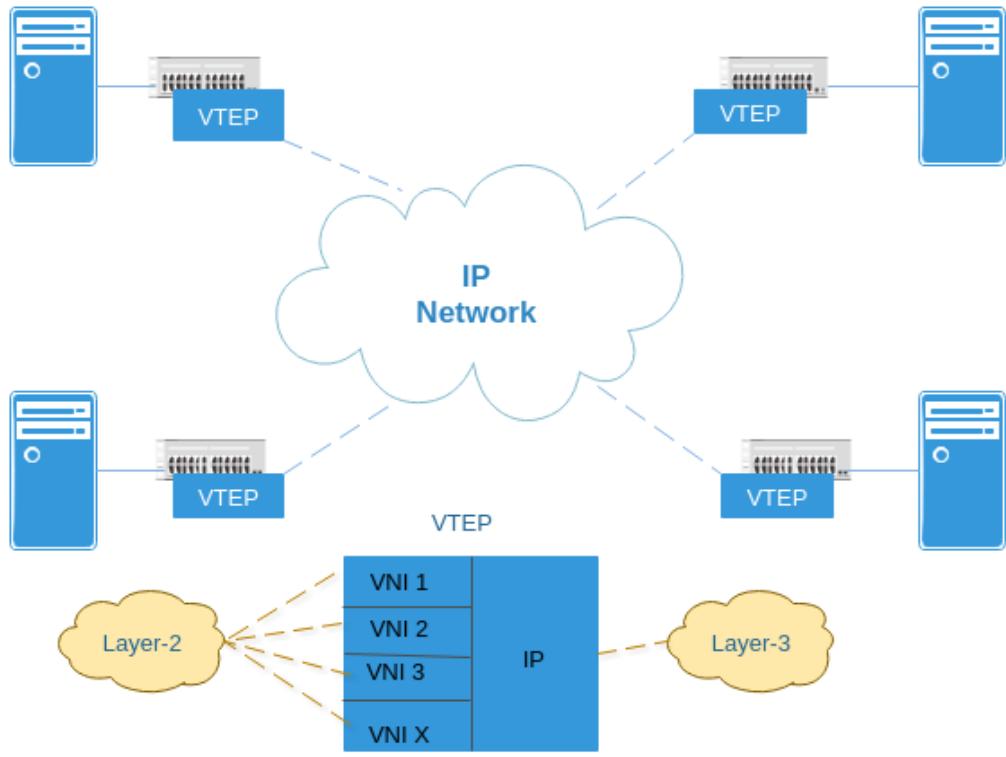
### Encapsulation et décapsulation

Comme nous l'avons vu, le trafic VxLAN est encapsulé avant d'être envoyé sur le réseau. Cela crée des tunnels sans état sur le réseau, du commutateur source au commutateur de destination.

L'encapsulation et la décapsulation sont gérées par un composant appelé **VTEP** (VxLAN Tunnel End Point). Dans la plate-forme Cisco Nexus, cela s'appelle également une interface **NVE**.

Comme le montre le diagramme ci-dessous, un VTEP a une adresse IP dans le réseau sous-jacent. Il est également associé à un ou plusieurs VNI. Lorsque les trames de l'un de ces VNI arrivent au *VTEP d'entrée*, le VTEP l'encapsule avec des en-têtes UDP et IP.

Le paquet encapsulé est envoyé sur le réseau IP vers le *VTEP de sortie*. Lorsqu'il arrive, le VTEP supprime les en-têtes IP et UDP et délivre la trame normalement.



**Figure:** encapsulation et décapsulation

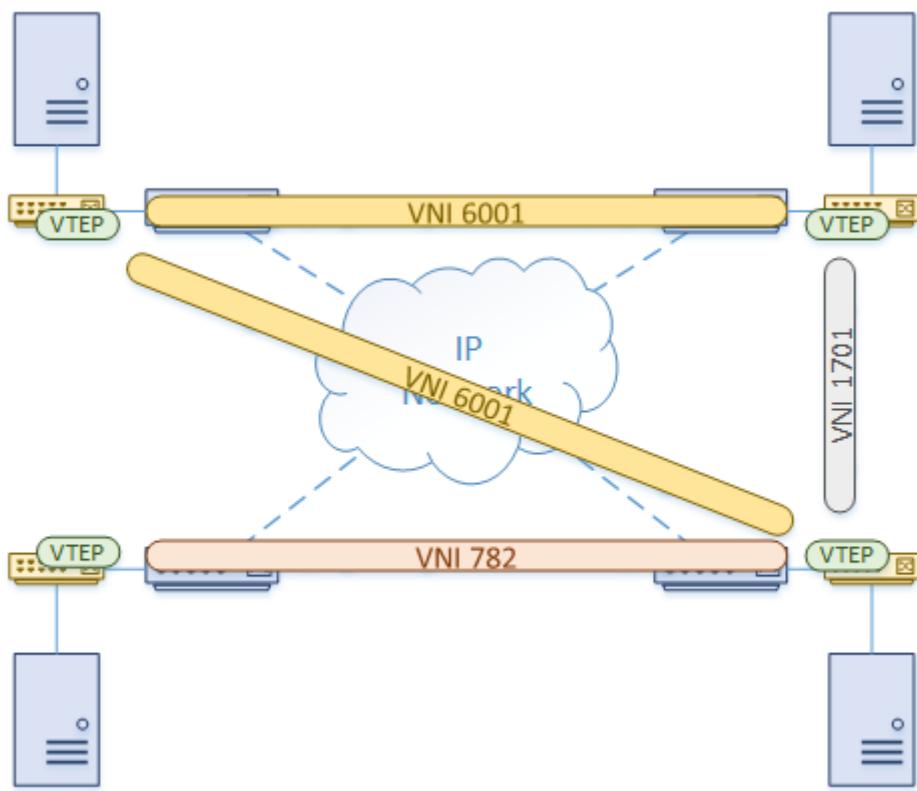
**Comment un VTEP sait-il où transférer le trafic? Comment un VTEP trouve-t-il un autre VTEP?**

#### Hôtes et passerelles

Le VxLAN est indépendant du fournisseur, il existe donc différentes manières de le déployer. Les deux méthodes principales sont **basées sur l'hôte** et **passerelle**.

Les hôtes, tels qu'un hyperviseur, peuvent avoir VxLAN configuré sur leurs commutateurs virtuels. D'autres périphériques, tels que les pare-feu et les équilibriseurs de charge, peuvent également parler VxLAN de manière native.

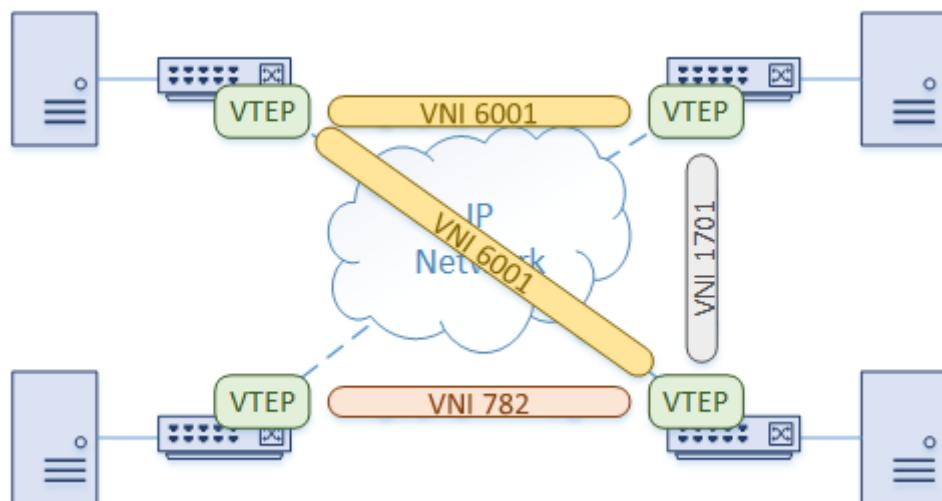
Dans des cas comme celui-ci, il n'est pas nécessaire de traduire les VLAN en VNI. De plus, les VTEP sont sur les hôtes eux-mêmes. L'infrastructure de réseau physique voit le trafic IP, pas VxLAN.



Si les hôtes ne prennent pas en charge l'exécution de VxLAN en mode natif, ou s'ils ne l'utilisent tout simplement pas, les commutateurs peuvent agir comme une **passerelle VxLAN**. Les hôtes appartiennent à un VLAN. Les commutateurs mappent le VLAN à un VNI et fournissent les fonctions VTEP.

Dans ce modèle, les hôtes ne sont pas conscients du VxLAN, car les commutateurs font tout le travail.

L'un des avantages de cette méthode est que VxLAN peut (selon la plate-forme) être implémenté dans le matériel, offrant des avantages en termes de performances.



Bien entendu, une combinaison de ces deux méthodes peut être utilisée. Les commutateurs peuvent fournir des services de passerelle à certains hôtes, tandis que d'autres hôtes parlent VxLAN de manière native.

Prenons un moment pour voir comment le trafic passe par un simple réseau VxLAN.



1. Une trame arrive sur un port de commutateur à partir d'un hôte. Ce port est un port (d'accès) non balisé régulier, qui attribue un VLAN au trafic
2. Le commutateur détermine que la trame doit être transférée vers un autre emplacement. Le commutateur distant est connecté par un réseau IP. Il peut être proche ou à plusieurs sauts
3. Le VLAN est associé à un VNI, donc un en-tête VxLAN est appliqué. Le VTEP encapsule le trafic dans les en-têtes UDP et IP. Le port UDP 4789 est utilisé comme port de destination. Le trafic est envoyé sur le réseau IP
4. Le commutateur distant reçoit le paquet et le décapsule. Une trame de couche 2 régulière avec un ID VLAN est laissée
5. Le commutateur sélectionne un port de sortie pour envoyer la trame. Ceci est basé sur les recherches MAC normales. Le reste du processus est normal.

Plusieurs modes de déploiement de VXLAN adressent cette problématique :

- **multicast** : la découverte automatique des VTEP est permise par multicast, les adresses MAC sources et leur localisation sont retenues par le VTEP.
- **Unicast** en automatisant la découverte des VTEP et en s'appuyant sur la capacité d'apprentissage des VTEP pour retenir la localisation des adresses MAC sources.
- Unicast en automatisant par un moyen externe la découverte des VTEP et la localisation des adresses MAC (les solution SDN ou BGP EVPN en sont un exemple).

Pour remarque, les limites des VTEP (VXLAN Tunnel End Point) et de la bonne configuration du réseau peuvent limiter les performances du protocole.

## II.4 – Rappel sur l'unicast, le multicast et le broadcast

Quand vous tentez de contacter un ou plusieurs hôtes (interfaces) dans un réseau, vous pouvez utiliser trois types d'adresses :

**Unicast** : Ce type d'adresse se réfère à un hôte unique (interface) dans un sous-réseau. Un exemple d'adresse unicast est 192.168.100.9. Un exemple d'adresse MAC unicast est, par exemple, 80:C0:F6:A0:4A:B1.

**Broadcast** : Cette adresse vous permet d'appeler tous les hôtes (interfaces) à l'intérieur d'un sous-réseau . Une adresse IP broadcast est 192.168.100.255 et un broadcast MAC est FF:FF:FF:FF:FF:FF.

**Multicast** : Ce type d'adresse permet d'appeler un groupe spécifique d'hôtes (interfaces) dans un sous-réseau.

### II.4.1 – Unicast

L'adressage unicast est le plus utilisé et le plus simple. Les ordinateurs possédant chacun une adresse IP, on peut envoyer les trames en spécifiant l'adresse IP de l'ordinateur à qui on veut envoyer les informations. Les éléments actifs et passifs du réseau (répéteurs, commutateurs, routeurs... ) dirigent l'information dans la bonne direction pour que les trames arrivent au bon endroit. Seule la machine ayant l'adresse contenue dans la trame regarde et traite l'information.

En IPv4 il existe 3 classes d'adresses unicast :

La classe A : Adresses comprises entre 1.0.0.x et 127.255.255.x

La classe B : Adresses comprises entre 128.0.0.x et 191.255.255.x

La classe C : Adresses comprises entre 192.0.0.x et 223.255.255.x

#### **II.4.2 – Broadcast**

Le broadcast (diffusion) consiste à envoyer l'information à tous les ordinateurs du réseau ou sous-réseau où on se situe (domaine de diffusion). Au lieu d'envoyer les informations en unicast vers l'adresse IP de la chaque machine (ex. 193.169.1.37 avec un masque 255.255.255.0), on envoie la trame à tous les ordinateurs du sous-réseau en utilisant l'adresse de broadcast (ici, 193.169.1.255). Cette adresse est réservée à cet usage. Chacun des ordinateurs du sous-réseau regarde et traite la trame comme si elle leur était personnellement adressée.

Les trames de broadcast ont une caractéristique particulière : c'est de ne pas pouvoir passer les routeurs puisqu'il s'adresse uniquement à tous les ordinateurs d'un même sous-réseau (pensez « niveau OSI »).

#### **II.4.3 – Multicast**

Plutôt que d'envoyer les fichiers du serveur vers chacune des machines clientes (unicast) on peut n'envoyer l'information qu'une seule fois et chaque ordinateur client la récupère. En effet, dans un réseau Ethernet par exemple, toutes les trames qui circulent passent par tous les ordinateurs. C'est le principe du multicast : on envoie l'information à une adresse et tous les clients écoutent cette adresse.

En Ipv4, en plus des classes d'adresses A, B et C, il existe une quatrième classe (D) réservée aux adresses multicast. Les adresses IPv4 entre 224.0.0.0 et 239.255.255.255 appartiennent à cette classe. Les 4 bits les plus significatifs de l'adresse IP autorisent des valeurs comprises entre 224 et 239. Les autres 28 bits, moins significatifs, sont réservés à l'identificateur du groupe multicast.

Chaque client multicast s'enregistre donc avec une adresse IP multicast de classe D (entre 224.0.0.0 et 239.255.255.255 sauf 224.0.0.0 non utilisée et 224.0.0.1 qui correspond au "broadcast du multicast"). C'est sur cette adresse que les informations vont être envoyées.

Les clients écoutent ce qui arrive sur cette adresse et suivent la procédure décrite par le protocole multicast implémenté.

Au niveau du réseau, les adresses multicast IPv4 doivent être « mappées » sur des adresses physiques du réseau. Dans le cas d'un adressage unicast, les adresses physiques sont obtenues en utilisant le protocole ARP. Dans le cas des adresses multicast, ARP ne peut pas être utilisé et les adresses physiques doivent être obtenues d'une autre manière (RFC 1112).

Dans les réseaux Ethernet les plus étendus, le "mappage" est effectué en fixant les 24 bits les plus significatifs de l'adresse Ethernet à 01:00:5E. Le bit suivant est fixé à 0 et les 23 bits les moins significatifs utilisent les 23 bits les moins significatifs de l'adresse multicast IPv4.

Par exemple, l'adresse multicast IPv4 224.0.0.5 correspondra à l'adresse physique ethernet 01:00:5E:00:00:05.

Il existe quelques adresses multicast IPv4 particulières :

- L'adresse 224.0.0.1 identifie tous les hôtes d'un sous-réseau. Tous les hôtes ayant des capacités multicast dans un sous-réseau doivent faire partie de ce groupe.
- L'adresse 224.0.0.2 identifie tout routeur multicast dans un réseau.

- Le champ d'adresse 224.0.0.0 - 224.0.0.255 est alloué pour les protocoles bas niveau. Les datagrammes envoyés dans cette plage d'adresse ne seront pas routés par des routeurs multicast.
- La plage d'adresse 239.0.0.0 - 239.255.255.255 est allouée à des fins administratives. Les adresses sont allouées localement pour chaque organisation mais elles ne peuvent exister à l'extérieur de celles-ci. Les routeurs de l'organisation ne doivent pas pouvoir router ces adresses à l'extérieur du réseau de l'entreprise.

Il existe un nombre important d'adresses multicast allouées, autres que celles mentionnées cidessus (voir RFC).

Le tableau ci-dessous montre le champ complet des adresses multicast avec les noms usuels pour chaque plage d'adresse et leur TTL associé. Avec le multicast IPv4, le TTL a une double signification. Comme le sait probablement le lecteur, il contrôle la durée de vie d'un datagramme dans le réseau pour éviter toute boucle infinie dans le cas de tables de routage mal configurées. En travaillant avec le multicast, la valeur TTL définit aussi le champ d'activité (scope) du datagramme, par exemple, jusqu'où circulera-t-il au sein du réseau. Ceci permet une définition de champ d'activité basée sur la catégorie du datagramme.

Champ d'activité	TTL	Plage d'adresse	Description
Nœud	0		Le datagramme est limité à l'hôte local. Il n'atteindra aucune autre interface du réseau.
Lien	1	224.0.0.0 - 224.0.0.255	Le datagramme sera limité au sous-réseau de l'hôte émetteur et ne passera pas les routeurs.
Département	< 32	239.255.0.0 - 239.255.255.255	Limité à un département de l'organisation.
Organisation	< 64	239.192.0.0 - 239.195.255.255	Limité à une organisation spécifique.
Global	< 254	224.0.1.0 - 238.255.255.255	Pas de restriction, application globale.

### Principe de fonctionnement du multicast

Dans un LAN, l'interface réseau d'un hôte enverra aux couches supérieures tous les paquets qui ont l'hôte comme destination. Ces paquets seront ceux qui auront pour adresse de destination les adresses de l'interface physique ou une adresse broadcast.

Si l'hôte fait partie d'un groupe multicast, l'interface réseau reconnaîtra aussi les paquets destinés à ce groupe : tous ceux avec une adresse de destination correspondant au groupe multicast dont l'hôte est membre.

C'est pourquoi, si une interface d'hôte possède l'adresse physique 80:C0:F6:A0:4A:B1 et fait partie du groupe multicast 224.0.1.10, les paquets qui seront reconnus comme appartenant à l'hôte seront ceux ayant l'une des adresses suivantes :

- L'adresse d'interface : 80:C0:F6:A0:4A:B1
- L'adresse broadcast : FF:FF:FF:FF:FF:FF
- L'adresse associée au groupe multicast : 01:00:5E:00:01:0A

Pour travailler avec multicast sur un WAN, les routeurs doivent supporter le routage multicast. Lorsqu'un processus actif sur un hôte adhère à un groupe multicast, l'hôte envoie un message IGMP à tous les routeurs multicast du sous-réseau, pour les informer que les messages multicast envoyés au groupe multicast doivent être transmis au sous-réseau local afin de permettre la réception par le processus nouvellement inscrit. Les routeurs informeront tous les autres routeurs multicast, en leur indiquant quels sont les messages multicast qui doivent être routés vers le sous-réseau.

Les routeurs envoient aussi des messages IGMP au groupe 224.0.0.1 demandant à chaque hôte des informations sur les groupes auxquels il adhère. Après avoir reçu ce message, un hôte définit un compteur sur une valeur aléatoire et répondra lorsqu'il sera arrivé à zéro. Ceci permet d'éviter que tous les hôtes répondant en même temps, provoquent une surcharge réseau. Lorsque l'hôte répond, il envoie le message à l'adresse multicast du groupe et tous les autres hôtes membres de ce groupe peuvent voir la réponse sans avoir à répondre eux-mêmes, puisqu'un hôte membre est suffisant au routeur du sous-réseau pour gérer les messages multicast de ce groupe.

Si tous les hôtes inscrits à un groupe se sont retirés, aucun ne répondra et le routeur décidera qu'aucun hôte n'est intéressé par ce groupe et cessera le routage des messages correspondants pour ce sous réseau. Une autre option existe dans IGMPv2 : l'hôte informe de son retrait en envoyant un message à l'adresse 224.0.0.2.

### III – Mise en place de VXLAN en unicast

#### CAS 1 : VXLAN en unicast sous Linux

Dans ce cas de figure, nous disposons de trois machines dans le même réseau qui va permettre d'en montrer le fonctionnement

Sous Linux, VXLAN est utilisable à partir des noyaux supérieurs à 3.x et grâce au module dont nous pouvons voir la description ci-dessous:

```
root@innonce-Lenovo:~# modinfo /lib/modules/$(uname -r)/kernel/drivers/net/vxlan.ko
filename:      /lib/modules/4.15.0-112-generic/kernel/drivers/net/vxlan.ko
alias:         rtnl-link-vxlan
description:   Driver for VXLAN encapsulated traffic
author:        Stephen Hemminger <stephen@networkplumber.org>
version:       0.1
license:       GPL
srcversion:    926C5A2AD6D629723DBFC1A
depends:       udp_tunnel,ip6_udp_tunnel
retpoline:     Y
intree:        Y
name:          vxlan
vermagic:     4.15.0-112-generic SMP mod_unload
signat:        PKCS#7
signer:
sig_key:
sig_hashalgo: md4
parm:          udp_port:Destination UDP port (ushort)
parm:          log_ecn_error:Log packets received with corrupted ECN (bool)
root@innonce-Lenovo:~#
```

Lecture du contenu de la FDB

```
root@innonce-Lenovo:~# bridge fdb show
01:00:5e:00:00:01 dev enp1s0 self permanent
01:00:5e:00:00:01 dev wlp2s0b1 self permanent
33:33:00:00:00:01 dev wlp2s0b1 self permanent
33:33:ff:8c:1d:45 dev wlp2s0b1 self permanent
33:33:00:00:00:fb dev wlp2s0b1 self permanent
01:00:5e:00:00:fb dev wlp2s0b1 self permanent
01:00:5e:7f:66:12 dev wlp2s0b1 self permanent
33:33:00:00:00:01 dev ovs-system self permanent
33:33:00:00:00:01 dev switch1 self permanent
01:00:5e:00:00:01 dev virbr0 self permanent
01:00:5e:00:00:fb dev virbr0 self permanent
52:54:00:3d:fa:ac dev virbr0-nic master virbr0 permanent
52:54:00:3d:fa:ac dev virbr0-nic vlan 1 master virbr0 permanent
33:33:00:00:00:01 dev docker0 self permanent
01:00:5e:00:00:01 dev docker0 self permanent
01:00:5e:00:00:fb dev docker0 self permanent
02:42:26:f8:03:7b dev docker0 master docker0 permanent
02:42:26:f8:03:7b dev docker0 vlan 1 master docker0 permanent
root@innonce-Lenovo:~#
```

La création d'une interface de type VXLAN se fait simplement via la commande ip en mottant un tunnel avec le VNI 10 entre la première et la seconde machine

```
root@innonce-Lenovo:~# ip link add vxlan10 type vxlan id 10 local 192.168.4.50 remote 192.168.5.212
vxlan: destination port not specified
Will use Linux kernel default (non-standard value)
Use 'dstport 4789' to get the IANA assigned value
Use 'dstport 0' to get default and quiet this message
root@innonce-Lenovo:~# ip link set vxlan10 up
root@innonce-Lenovo:~#
```

On peut à présent voir apparaître dans la FDB une entrée indiquant comment joindre le VTEP de la seconde machine

```
root@innonce-Lenovo:~# bridge fdb show
01:00:5e:00:00:01 dev enp1s0 self permanent
01:00:5e:00:00:01 dev wlp2s0b1 self permanent
33:33:00:00:00:01 dev wlp2s0b1 self permanent
33:33:ff:10:88:8d dev wlp2s0b1 self permanent
33:33:00:00:00:fb dev wlp2s0b1 self permanent
01:00:5e:00:00:fb dev wlp2s0b1 self permanent
01:00:5e:7f:66:12 dev wlp2s0b1 self permanent
33:33:00:00:00:01 dev ovs-system self permanent
33:33:00:00:00:01 dev switch1 self permanent
01:00:5e:00:00:01 dev virbr0 self permanent
01:00:5e:00:00:fb dev virbr0 self permanent
01:00:5e:7f:66:12 dev virbr0 self permanent
52:54:00:3d:fa:ac dev virbr0-nic master virbr0 permanent
52:54:00:3d:fa:ac dev virbr0-nic vlan 1 master virbr0 permanent
33:33:00:00:00:01 dev docker0 self permanent
01:00:5e:00:00:01 dev docker0 self permanent
01:00:5e:00:00:fb dev docker0 self permanent
02:42:bb:ca:a7:56 dev docker0 vlan 1 master docker0 permanent
02:42:bb:ca:a7:56 dev docker0 master docker0 permanent
00:00:00:00:00:00 dev vxlan10 dst 192.168.5.212 self permanent
root@innonce-Lenovo:~#
```

On peut voir que la machine est en écoute sur le port udp/8472

```
root@innonce-Lenovo:~# netstat -pan | grep -w 8472
udp      0      0 0.0.0.0:8472          0.0.0.0:*
root@innonce-Lenovo:~#
```

On effectue les mêmes actions sur la seconde machine pour que les deux VTEP se connaissent :

```
root@innonce-VirtualBox:~# ip link add vxlan10 type vxlan id 10 local 192.168.5.212 remote 192.168.4.50
vxlan: destination port not specified
Will use Linux kernel default (non-standard value)
Use 'dstport 4789' to get the IANA assigned value
Use 'dstport 0' to get default and quiet this message
root@innonce-VirtualBox:~# ip link set vxlan10 up
root@innonce-VirtualBox:~#
```

```
root@innonce-VirtualBox:~# bridge fdb show | grep vxlan10
00:00:00:00:00:00 dev vxlan10 dst 192.168.4.50 self permanent
root@innonce-VirtualBox:~#
```

À ce stade, il est possible de communiquer à travers le tunnel en ajoutant une adresse IP à chaque interface VXLAN

```
root@innonce-Lenovo:~# ip addr add 10.10.10.1/24 dev vxlan10
root@innonce-Lenovo:~#
```

```
root@innonce-VirtualBox:~# ip addr add 10.10.10.2/24 dev vxlan10
root@innonce-VirtualBox:~#
```

Un test de connectivité lancer au niveau des deux machines afin d'affirmer que les deux VTEP arrivent à communiquer via le tunnel établi.

```
root@innonce-Lenovo:~# ping -c2 10.10.10.2
PING 10.10.10.2 (10.10.10.2) 56(84) bytes of data.
64 bytes from 10.10.10.2: icmp_seq=1 ttl=64 time=0.429 ms
64 bytes from 10.10.10.2: icmp_seq=2 ttl=64 time=0.418 ms

--- 10.10.10.2 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1000ms
rtt min/avg/max/mdev = 0.418/0.423/0.429/0.021 ms
root@innonce-Lenovo:~#
```

```
root@innonce-VirtualBox:~# ping -c2 10.10.10.1
PING 10.10.10.1 (10.10.10.1) 56(84) bytes of data.
64 bytes from 10.10.10.1: icmp_seq=1 ttl=64 time=0.741 ms
64 bytes from 10.10.10.1: icmp_seq=2 ttl=64 time=0.281 ms

--- 10.10.10.1 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1001ms
rtt min/avg/max/mdev = 0.281/0.511/0.741/0.230 ms
root@innonce-VirtualBox:~#
```

On doit avoir dans la FDB une entrée indiquant que l'adresse MAC de vxlan10 sur la deuxième machine accessible derrière l'adresse IP du VTEP distant :

```
root@innonce-VirtualBox:~# bridge fdb show | grep vxlan10
00:00:00:00:00:00 dev vxlan10 dst 192.168.4.50 self permanent
12:9f:c7:f9:00 dev vxlan10 dst 192.168.4.50 self
root@innonce-VirtualBox:~#
```

On peut vérifier que c'est bien la bonne adresse MAC

```

root@innonce-Lenovo:~# ifconfig vxlan10
vxlan10: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
      inet 10.10.10.1 netmask 255.255.255.0 broadcast 0.0.0.0
      inet6 fe80::10f:79ff:fe7:f900 prefixlen 64 scopeid 0x20<link>
        ether 12:9f:79:c7:f9:00 txqueuelen 1000 (Ethernet)
          RX packets 61 bytes 6029 (6.0 KB)
          RX errors 0 dropped 0 overruns 0 frame 0
          TX packets 101 bytes 15513 (15.5 KB)
          TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

root@innonce-Lenovo:~# 
```

C'est le cas. On constate ici que le VTEP source apprend de lui même la localisation des adresses MAC de destination. Si l'on regarde la FDB du VTEP de destination, une entrée est également apparue :

```

root@innonce-VirtualBox:~# bridge fdb show | grep vxlan10
00:00:00:00:00:00 dev vxlan10 dst 192.168.4.50 self permanent
12:9f:79:c7:f9:00 dev vxlan10 dst 192.168.4.50 self
root@innonce-VirtualBox:~# 
```

Le VTEP de destination retient donc la localisation de l'adresse MAC source.

Ce mécanisme d'apprentissage peut être désactivé, si au moment de la création de l'interface VXLAN, on passe l'option nolearning. La déclaration des entrées dans la FDB sera alors à notre charge (en plus de la découverte des VTEP). Les solutions SDN gèrent généralement ces deux aspects d'apprentissage en maintenant une FDB centralisée. Une autre approche consiste à utiliser BGP pour communiquer la localisation des adresses MAC (voir BGP EVPN).

On peut visualiser ce qui se passe sous le capot avec la commande **tcpdump**

```

root@innonce-Lenovo:~# tcpdump -i wlp2s0b1 -n udp | head
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on wlp2s0b1, link-type EN10MB (Ethernet), capture size 262144 bytes
16:51:26.942837 IP6 fe80::45a2:f0a9:8599:7a67.546 > ff02::1:2.547: dhcp6 solicit
16:51:27.864555 IP6 fe80::4910:99e8:e041:ce79.47447 > ff08::2:7ffe.9875: UDP, length 219
16:51:28.564724 IP 192.168.5.212.57047 > 192.168.4.50.8472: OTV, flags [I] (0x08), overlay 0, instance 10
IP 10.10.10.1 > 10.10.10.1: ICMP echo request, id 11456, seq 1, length 64
16:51:28.564784 IP 192.168.4.50.33577 > 192.168.5.212.8472: OTV, flags [I] (0x08), overlay 0, instance 10
IP 10.10.10.1 > 10.10.10.2: ICMP echo reply, id 11456, seq 1, length 64
16:51:29.502166 IP 192.168.2.226.43072 > 224.2.127.254.9875: UDP, length 267
16:51:29.594052 IP 192.168.5.212.57047 > 192.168.4.50.8472: OTV, flags [I] (0x08), overlay 0, instance 10
IP 10.10.10.2 > 10.10.10.1: ICMP echo request, id 11456, seq 2, length 64
16:51:29.594118 IP 192.168.4.50.33577 > 192.168.5.212.8472: OTV, flags [I] (0x08), overlay 0, instance 10
tcpdump: Unable to write output: Broken pipe
root@innonce-Lenovo:~# 
```

On voit les paquets ICMP encapsulés.

On ajoute une troisième machine en montant une interface VXLAN et en ajoutant le VTEP de l'autre machine à la FDB :

```

root@mm:~# ip link add vxlan10 type vxlan id 10 local 192.168.2.32 remote 192.168.4.50
vxlan: destination port not specified
Will use Linux kernel default (non-standard value)
Use 'dstport 4789' to get the IANA assigned value
Use 'dstport 0' to get default and quiet this message
root@mm:~# ip link set vxlan10 up
root@mm:~# bridge fdb append 00:00:00:00:00:00 dev vxlan10 dst 192.168.4.50
root@mm:~# ip addr add 10.10.10.3/24 dev vxlan10
root@mm:~#

```

On a donné une adresse IP à la troisième machine dans le réseau d'overlay.

Test de connectivité

```

root@mm:~# ping -c2 10.10.10.1
PING 10.10.10.1 (10.10.10.1) 56(84) bytes of data.
64 bytes from 10.10.10.1: icmp_seq=1 ttl=64 time=3221 ms
64 bytes from 10.10.10.1: icmp_seq=2 ttl=64 time=2213 ms

--- 10.10.10.1 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1007ms
rtt min/avg/max/mdev = 2213.823/2717.734/3221.646/503.914 ms, pipe 2
root@mm:~# ping -c2 10.10.10.1

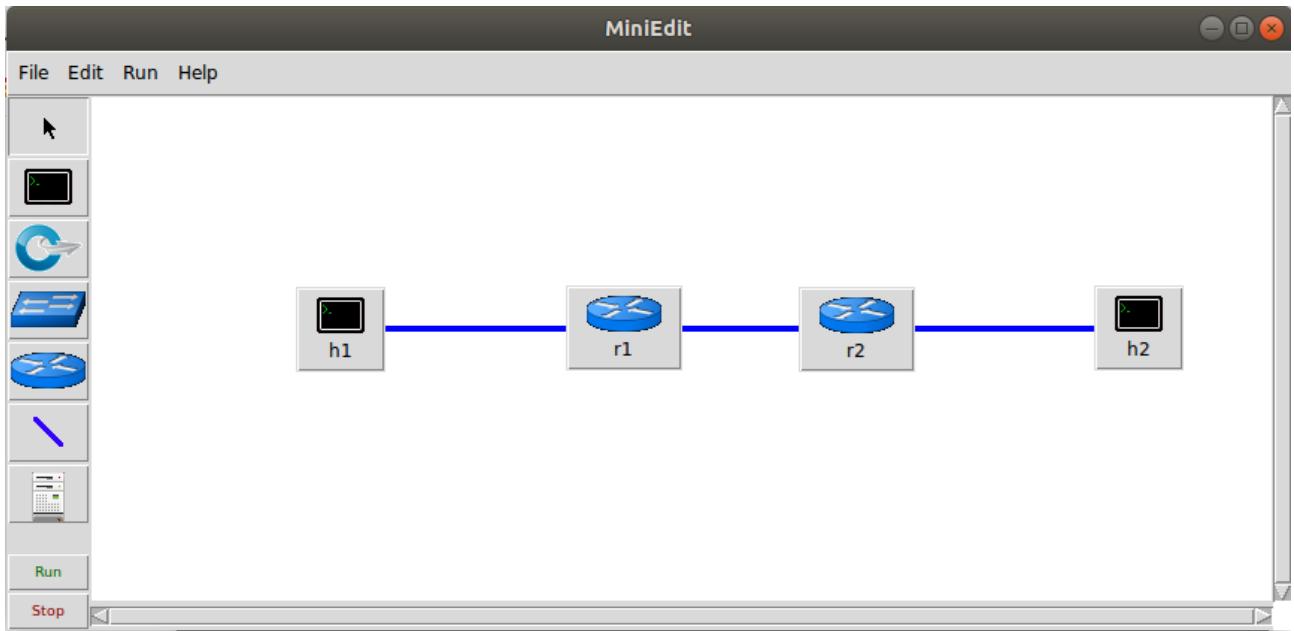
```

## CAS 2 : VXLAN en unicast avec l'émulateur Mininet

### Architecture



## Montage de l'architecture avec l'outil mininet



Affectation des adresses IP au niveau des équipements

Routeur r1

```
"Host: r1" (sur mininet-vm)
root@mininet-vm:~# ifconfig r1-eth0 192.168.12.1
root@mininet-vm:~# ifconfig r1-eth1 192.168.1.1
root@mininet-vm:~#"
```

Routeur r2

```
"Host: r2" (sur mininet-vm)
root@mininet-vm:~# ifconfig r2-eth0 192.168.13.1
root@mininet-vm:~# ifconfig r2-eth1 192.168.1.2
root@mininet-vm:~#"
```

Machine1

"Host: h1" (sur mininet-vm)

```
root@mininet-vm:~# ifconfig h1-eth0 192.168.12.2
root@mininet-vm:~#
```

Machine2

"Host: h2" (sur mininet-vm)

```
root@mininet-vm:~# ifconfig h2-eth0 192.168.13.2
root@mininet-vm:~#
```

Passerelle aux machines (server1 & server2)

"Host: h1" (sur mininet-vm)

```
root@mininet-vm:~# route add default gw 192.168.12.1
root@mininet-vm:~#
```

"Host: h2" (sur mininet-vm)

```
root@mininet-vm:~# route add default gw 192.168.13.1
root@mininet-vm:~#
```

Vérification de la table de routage

"Host: h1" (sur mininet-vm)

"Host: h2" (sur mininet-vm)

```
root@mininet-vm:~# route -n
Kernel IP routing table
Destination     Gateway         Genmask        Flags Metric Ref    Use Iface
0.0.0.0         192.168.13.1   0.0.0.0       UG     0      0        0 h2-eth0
192.168.13.0   0.0.0.0        255.255.255.0  U      0      0        0 h2-eth0
root@mininet-vm:~#
```

Routage (r1 & r2)

```
root@mininet-vm:~# route add -net 192.168.13.0 netmask 255.255.255.0 gw 192.168.1.2
root@mininet-vm:~#
```

```
"Host: r2" (sur mininet-vm)
root@mininet-vm:~# route add -net 192.168.12.0 netmask 255.255.255.0 gw 192.168.1.1
root@mininet-vm:~# [ ]
```

### Communication entre les machines au niveau de la couche 3

```
"Host: h1" (sur mininet-vm)
root@mininet-vm:~# ping -c2 192.168.12.1; ping -c2 192.168.1.2; ping -c2 192.168.13.2
PING 192.168.12.1 (192.168.12.1) 56(84) bytes of data.
64 bytes from 192.168.12.1: icmp_seq=1 ttl=64 time=0.044 ms
64 bytes from 192.168.12.1: icmp_seq=2 ttl=64 time=0.054 ms

--- 192.168.12.1 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 999ms
rtt min/avg/max/mdev = 0.044/0.049/0.054/0.005 ms
PING 192.168.1.2 (192.168.1.2) 56(84) bytes of data.
64 bytes from 192.168.1.2: icmp_seq=1 ttl=63 time=0.056 ms
64 bytes from 192.168.1.2: icmp_seq=2 ttl=63 time=0.066 ms

--- 192.168.1.2 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 999ms
rtt min/avg/max/mdev = 0.056/0.061/0.066/0.005 ms
PING 192.168.13.2 (192.168.13.2) 56(84) bytes of data.
64 bytes from 192.168.13.2: icmp_seq=1 ttl=62 time=0.072 ms
64 bytes from 192.168.13.2: icmp_seq=2 ttl=62 time=0.080 ms

--- 192.168.13.2 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1000ms
rtt min/avg/max/mdev = 0.072/0.076/0.080/0.004 ms
root@mininet-vm:~# [ ]
```

```
"Host: h2" (sur mininet-vm)
root@mininet-vm:~# ping -c2 192.168.13.1; ping -c2 192.168.1.1; ping -c2 192.168.12.2
PING 192.168.13.1 (192.168.13.1) 56(84) bytes of data.
64 bytes from 192.168.13.1: icmp_seq=1 ttl=64 time=0.039 ms
64 bytes from 192.168.13.1: icmp_seq=2 ttl=64 time=0.056 ms

--- 192.168.13.1 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 999ms
rtt min/avg/max/mdev = 0.039/0.047/0.056/0.010 ms
PING 192.168.1.1 (192.168.1.1) 56(84) bytes of data.
64 bytes from 192.168.1.1: icmp_seq=1 ttl=63 time=0.059 ms
64 bytes from 192.168.1.1: icmp_seq=2 ttl=63 time=0.066 ms

--- 192.168.1.1 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 999ms
rtt min/avg/max/mdev = 0.059/0.062/0.066/0.008 ms
PING 192.168.12.2 (192.168.12.2) 56(84) bytes of data.
64 bytes from 192.168.12.2: icmp_seq=1 ttl=62 time=0.075 ms
64 bytes from 192.168.12.2: icmp_seq=2 ttl=62 time=0.081 ms

--- 192.168.12.2 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1000ms
rtt min/avg/max/mdev = 0.075/0.078/0.081/0.003 ms
root@mininet-vm:~# [ ]
```

### Création et activation des interfaces de type VXLAN au niveau des machines

"Host: h1" (sur mininet-vm)

```
root@mininet-vm:~# ip link add vxlan100 type vxlan id 100 local 192.168.12.2 remote 192.168.13.2
root@mininet-vm:~# ip link set vxlan100 up
root@mininet-vm:~#
```

"Host: h2" (sur mininet-vm)

```
root@mininet-vm:~# ip link add vxlan100 type vxlan id 100 local 192.168.13.2 remote 192.168.12.2
root@mininet-vm:~# ip link set vxlan100 up
root@mininet-vm:~#
```

Table FDB

"Host: h1" (sur mininet-vm)

```
root@mininet-vm:~# bridge fdb show
01:00:5e:00:00:01 dev h1-eth0 self permanent
00:00:00:00:00:00 dev vxlan100 dst 192.168.13.2 self permanent
root@mininet-vm:~#
```

"Host: h2" (sur mininet-vm)

```
root@mininet-vm:~# bridge fdb show
01:00:5e:00:00:01 dev h2-eth0 self permanent
00:00:00:00:00:00 dev vxlan100 dst 192.168.12.2 self permanent
root@mininet-vm:~#
```

Affectation des adresses IP au niveau des interfaces VXLAN (machine1 & machine2)

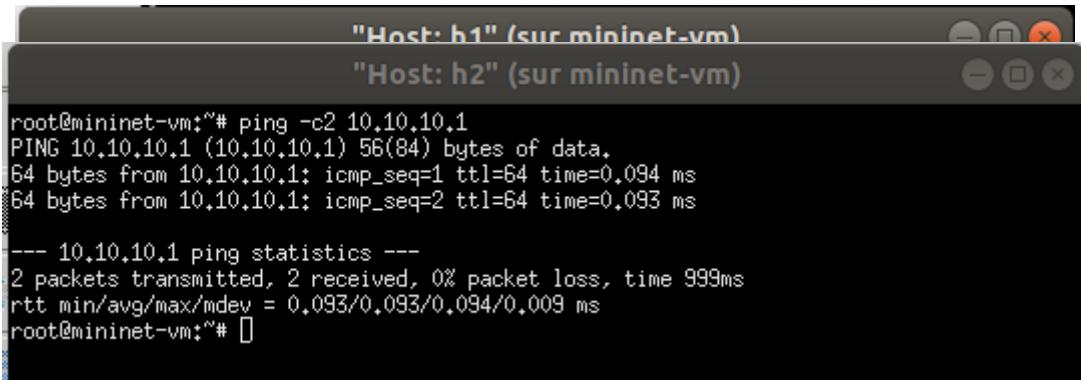
"Host: h1" (sur mininet-vm)

```
root@mininet-vm:~# ip addr add 10.10.10.2/24 dev vxlan100
root@mininet-vm:~#
```

"Host: h2" (sur mininet-vm)

```
root@mininet-vm:~# ip addr add 10.10.10.1/24 dev vxlan100
root@mininet-vm:~#
```

## Test de communication



The screenshot shows two terminal windows side-by-side. The left window is titled "Host: h1" (sur mininet-vm) and the right window is titled "Host: h2" (sur mininet-vm). Both windows are running on a Linux desktop environment.

In the "Host: h1" window, the command "root@mininet-vm:~# ping -c2 10.10.10.1" is run, followed by the output:

```
PING 10.10.10.1 (10.10.10.1) 56(84) bytes of data.  
64 bytes from 10.10.10.1: icmp_seq=1 ttl=64 time=0.094 ms  
64 bytes from 10.10.10.1: icmp_seq=2 ttl=64 time=0.093 ms
```

In the "Host: h2" window, the command "root@mininet-vm:~# ping 10.10.10.1" is run, followed by the output:

```
--- 10.10.10.1 ping statistics ---  
2 packets transmitted, 2 received, 0% packet loss, time 999ms  
rtt min/avg/max/mdev = 0.093/0.093/0.094/0.009 ms
```

A blue box highlights the command "IV -" at the top left of the windows.

## Mise en place de VXLAN en multicast

### Prérequis

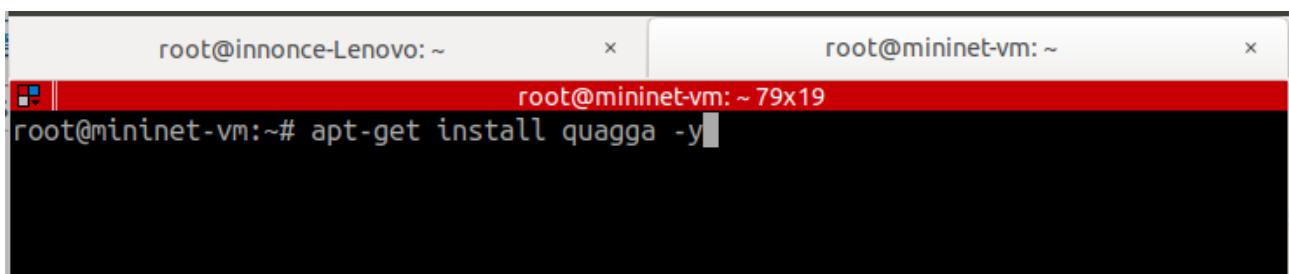
#### Installation et configuration de Quagga

Quagga est l'un des principaux projets open source utilisés pour fournir des services de routage sur la plate-forme Linux. Il se compose de différents composants pour différents protocoles dynamiques tels que Open Shortest Path First (OSPF), Routing Information Protocol (RIP), Border Gateway Protocol (BGP), Intermediate System to Intermediate System (IS-IS) et Multiprotocol Label Switching (MPLS) . En partie, il fournit le même terminal virtuel ou CLI (vty / vtysh) que CISCO / JUNIPER pour la configuration des protocoles.

Quagga peut être installé à partir du code source cependant, nous allons installer le package deb/binary.

#### Installation

La commande suivante permet d'installer le logiciel de routage Quagga



The screenshot shows a single terminal window with two tabs. The left tab is titled "root@innonce-Lenovo: ~" and the right tab is titled "root@mininet-vm: ~".

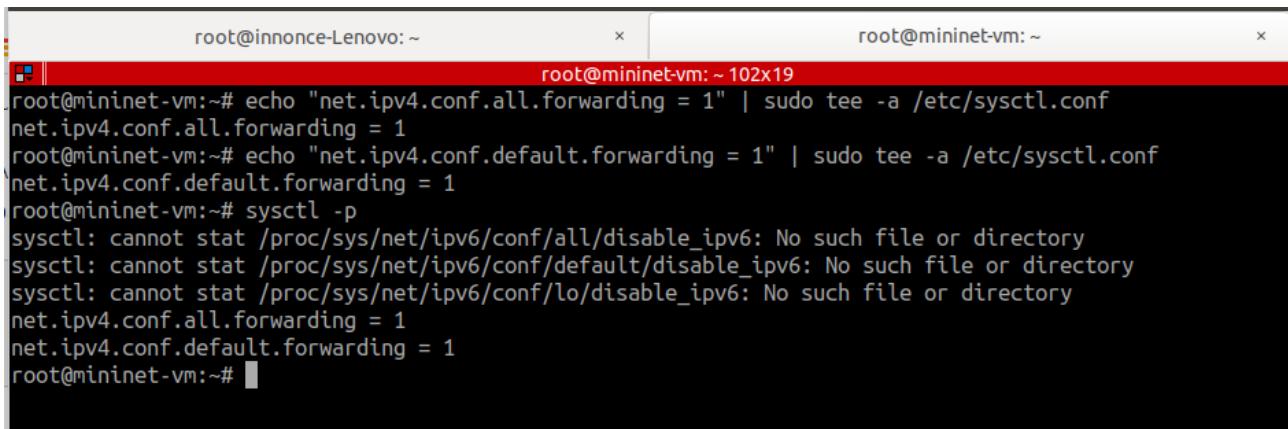
The command "root@mininet-vm:~# apt-get install quagga -y" is being typed into the terminal. The command is completed and the output shows the installation process.

```
root@mininet-vm:~# apt-cache policy quagga
quagga:
  Installed: 0.99.22.4-3ubuntu1.5
  Candidate: 0.99.22.4-3ubuntu1.5
  Version table:
*** 0.99.22.4-3ubuntu1.5 0
    500 http://us.archive.ubuntu.com/ubuntu/ trusty-updates/main amd64 Packages
      500 http://security.ubuntu.com/ubuntu/ trusty-security/main amd64 Packages
    100 /var/lib/dpkg/status
  0.99.22.4-3ubuntu1.0
    500 http://us.archive.ubuntu.com/ubuntu/ trusty/main amd64 Packages
root@mininet-vm:~#
```

## Configuration

La configuration par défaut des périphériques basés sur Linux ne prend pas en charge le transfert de paquets jusqu'à ce que quelques paramètres du noyau soient activés.

Activons le transfert de paquets pour IPv4 à l'aide des commandes suivantes, illustrées dans la figure. Le paramètre sera enregistré en permanence dans le fichier **/etc/sysctl.conf**.



The screenshot shows two terminal windows side-by-side. The left window is titled 'root@innonce-Lenovo:' and the right window is titled 'root@mininet-vm: ~'. Both windows have a red header bar. The left window contains a single command: 'root@mininet-vm:~# echo "net.ipv4.conf.all.forwarding = 1" | sudo tee -a /etc/sysctl.conf'. The right window contains several commands: 'root@mininet-vm:~# echo "net.ipv4.conf.all.forwarding = 1" | sudo tee -a /etc/sysctl.conf', 'net.ipv4.conf.all.forwarding = 1', 'root@mininet-vm:~# echo "net.ipv4.conf.default.forwarding = 1" | sudo tee -a /etc/sysctl.conf', 'net.ipv4.conf.default.forwarding = 1', 'root@mininet-vm:~# sysctl -p', which outputs: 'sysctl: cannot stat /proc/sys/net/ipv6/conf/all/disable\_ipv6: No such file or directory', 'sysctl: cannot stat /proc/sys/net/ipv6/conf/default/disable\_ipv6: No such file or directory', 'sysctl: cannot stat /proc/sys/net/ipv6/conf/lo/disable\_ipv6: No such file or directory', 'net.ipv4.conf.all.forwarding = 1', 'net.ipv4.conf.default.forwarding = 1'. Both windows end with a black prompt area.

```
root@innonce-Lenovo: ~
root@mininet-vm: ~ 102x19
root@mininet-vm:~# echo "net.ipv4.conf.all.forwarding = 1" | sudo tee -a /etc/sysctl.conf
net.ipv4.conf.all.forwarding = 1
root@mininet-vm:~# echo "net.ipv4.conf.default.forwarding = 1" | sudo tee -a /etc/sysctl.conf
net.ipv4.conf.default.forwarding = 1
root@mininet-vm:~# sysctl -p
sysctl: cannot stat /proc/sys/net/ipv6/conf/all/disable_ipv6: No such file or directory
sysctl: cannot stat /proc/sys/net/ipv6/conf/default/disable_ipv6: No such file or directory
sysctl: cannot stat /proc/sys/net/ipv6/conf/lo/disable_ipv6: No such file or directory
net.ipv4.conf.all.forwarding = 1
net.ipv4.conf.default.forwarding = 1
root@mininet-vm:~#
```

Après avoir activé le transfert de paquets, nous allons maintenant configurer le logiciel de routage Quagga pour qu'il s'exécute sous Linux. Configuration suivante requise pour exécuter le démon Quagga sur Ubuntu.

L'utilisateur peut créer ces fichiers de configuration ou copier ces exemples de fichiers depuis **/usr/share/doc/quagga/examples/** vers le chemin **/etc/quagga/\*.conf**.

```
root@innonce-Lenovo: ~          root@mininet-vm: ~
root@mininet-vm:~# ls -l /usr/share/doc/quagga/examples/
total 40
-rw-r--r-- 1 root root 655 Feb  8 2018 babeld.conf.sample
-rw-r--r-- 1 root root 582 Feb  8 2018 bgpd.conf.sample
-rw-r--r-- 1 root root 2801 Feb  8 2018 bgpd.conf.sample2
-rw-r--r-- 1 root root 805 Feb  8 2018 isisd.conf.sample
-rw-r--r-- 1 root root 1110 Feb  8 2018 ospf6d.conf.sample
-rw-r--r-- 1 root root 182 Feb  8 2018 ospfd.conf.sample
-rw-r--r-- 1 root root 422 Feb  8 2018 ripd.conf.sample
-rw-r--r-- 1 root root 390 Feb  8 2018 ripngd.conf.sample
-rw-r--r-- 1 root root 126 Feb  8 2018 vtysh.conf.sample
-rw-r--r-- 1 root root 385 Feb  8 2018 zebra.conf.sample
root@mininet-vm:~#
```

La copie de fichiers d'exemple sous **/etc/quagga/** est illustrée dans la figure suivante

```
root@innonce-Lenovo: ~          root@mininet-vm: ~
root@mininet-vm:~# cp /usr/share/doc/quagga/examples/*.sample /etc/quagga/
root@mininet-vm:~#
root@mininet-vm:~# ls -l /etc/quagga/
total 44
-rw-r--r-- 1 root root 655 Sep  9 22:12 babeld.conf.sample
-rw-r--r-- 1 root root 582 Sep  9 22:12 bgpd.conf.sample
-rw-r----- 1 quagga quagga 990 Feb  8 2018 daemons
-rw-r----- 1 quagga quagga 945 Feb  8 2018 debian.conf
-rw-r--r-- 1 root root 805 Sep  9 22:12 isisd.conf.sample
-rw-r--r-- 1 root root 1110 Sep  9 22:12 ospf6d.conf.sample
-rw-r--r-- 1 root root 182 Sep  9 22:12 ospfd.conf.sample
-rw-r--r-- 1 root root 422 Sep  9 22:12 ripd.conf.sample
-rw-r--r-- 1 root root 390 Sep  9 22:12 ripngd.conf.sample
-rw-r--r-- 1 root root 126 Sep  9 22:12 vtysh.conf.sample
-rw-r--r-- 1 root root 385 Sep  9 22:12 zebra.conf.sample
root@mininet-vm:~#
```

Renommons les fichiers d'exemple après les avoir copiés dans le répertoire **/etc/quagga** .

```
root@innonce-Lenovo: ~          root@mininet-vm: /etc/quagga
root@mininet-vm:~# cd /etc/quagga/
root@mininet-vm:/etc/quagga# mv babeld.conf.sample babeld.conf
root@mininet-vm:/etc/quagga# mv isisd.conf.sample isisd.conf
root@mininet-vm:/etc/quagga# mv ospfd.conf.sample ospf6d.conf
root@mininet-vm:/etc/quagga# mv ripngd.conf.sample ripngd.conf
root@mininet-vm:/etc/quagga# mv zebra.conf.sample zebra.conf
root@mininet-vm:/etc/quagga# mv bgpd.conf.sample bgpd.conf
root@mininet-vm:/etc/quagga# mv ospf6d.conf.sample ospf6d.conf
root@mininet-vm:/etc/quagga# mv ripd.conf.sample ripd.conf
root@mininet-vm:/etc/quagga# mv vtysh.conf.sample vtysh.conf
root@mininet-vm:/etc/quagga#
root@mininet-vm:/etc/quagga# ls
babeld.conf      daemons      isisd.conf      ospfd.conf      ripngd.conf      zebra.conf
bgpd.conf        debian.conf  ospf6d.conf    ripd.conf       vtysh.conf
root@mininet-vm:/etc/quagga#
```

Exécutons les commandes suivantes sous le dossier **/etc/quagga** pour changer le propriétaire et les autorisations.

```
root@innonce-Lenovo: ~          x      root@mininet-vm: /etc/quagga
root@mininet-vm:/etc/quagga# ls -l
total 44
-rw-r--r-- 1 root  root   655 Sep  9 22:12 babeld.conf
-rw-r--r-- 1 root  root   582 Sep  9 22:12 bgpd.conf
-rw-r----- 1 quagga quagga 990 Feb  8 2018 daemons
-rw-r----- 1 quagga quagga 945 Feb  8 2018 debian.conf
-rw-r--r-- 1 root  root   805 Sep  9 22:12 isisd.conf
-rw-r--r-- 1 root  root  1110 Sep  9 22:12 ospf6d.conf
-rw-r--r-- 1 root  root   182 Sep  9 22:12 ospfd.conf
-rw-r--r-- 1 root  root   422 Sep  9 22:12 ripd.conf
-rw-r--r-- 1 root  root   390 Sep  9 22:12 ripngd.conf
-rw-r--r-- 1 root  root   126 Sep  9 22:12 vtysh.conf
-rw-r--r-- 1 root  root   385 Sep  9 22:12 zebra.conf
root@mininet-vm:/etc/quagga#
```

```
root@innonce-Lenovo: ~          x      root@mininet-vm: /etc/quagga
root@mininet-vm:/etc/quagga# chown quagga:quagga *
root@mininet-vm:/etc/quagga#
root@mininet-vm:/etc/quagga# ls -l
total 44
-rw-r--r-- 1 quagga quagga 655 Sep  9 22:12 babeld.conf
-rw-r--r-- 1 quagga quagga 582 Sep  9 22:12 bgpd.conf
-rw-r----- 1 quagga quagga 990 Feb  8 2018 daemons
-rw-r----- 1 quagga quagga 945 Feb  8 2018 debian.conf
-rw-r--r-- 1 quagga quagga 805 Sep  9 22:12 isisd.conf
-rw-r--r-- 1 quagga quagga 1110 Sep  9 22:12 ospf6d.conf
-rw-r--r-- 1 quagga quagga 182 Sep  9 22:12 ospfd.conf
-rw-r--r-- 1 quagga quagga 422 Sep  9 22:12 ripd.conf
-rw-r--r-- 1 quagga quagga 390 Sep  9 22:12 ripngd.conf
-rw-r--r-- 1 quagga quagga 126 Sep  9 22:12 vtysh.conf
-rw-r--r-- 1 quagga quagga 385 Sep  9 22:12 zebra.conf
root@mininet-vm:/etc/quagga#
```

L'autorisation « 640 » est déjà définie sur les fichiers.

En fin de compte, nous devons activer ou désactiver différents démons de Quagga. Le démon Zebra est la partie centrale de la suite de routage, il doit donc être activé dans le fichier **/etc/quagga/daemons**. Dans notre cas, seuls les démons de protocole dynamique OSPF et RIP seront activés dans le fichier.

La configuration par défaut du fichier des démons est indiquée dans l'instantané suivant. Tous les démons de routage sont désactivés dans le fichier de configuration par défaut.

```
root@innonce-Lenovo: ~          x      root@mininet-vm: ~          x
root@mininet-vm:~# tail /etc/quagga/daemons
# that can be changed via /etc/quagga/debian.conf.
#
zebra=no
bgpd=no
ospfd=no
ospf6d=no
ripd=no
ripngd=no
isisd=no
babeld=no
root@mininet-vm:~#
```

Le fichier de configuration avec OSPF et RIP activés est illustré ci-dessous

```
root@innonce-Lenovo: ~          x      root@mininet-vm: ~          x
root@mininet-vm:~# cat /etc/quagga/daemons
# be u=rw,g=r,o=.
# When using "vtysh" such a config file is also needed. It should be owned by
# group "quaggavty" and set to ug=rw,o= though. Check /etc/pam.d/quagga, too.
#
# The watchquagga daemon is always started. Per default in monitoring-only but
# that can be changed via /etc/quagga/debian.conf.
#
zebra=yes
bgpd=no
ospfd=yes
ospf6d=no
ripd=yes
ripngd=no
isisd=no
babeld=no
"/etc/quagga/daemons" 31L, 993C written
```

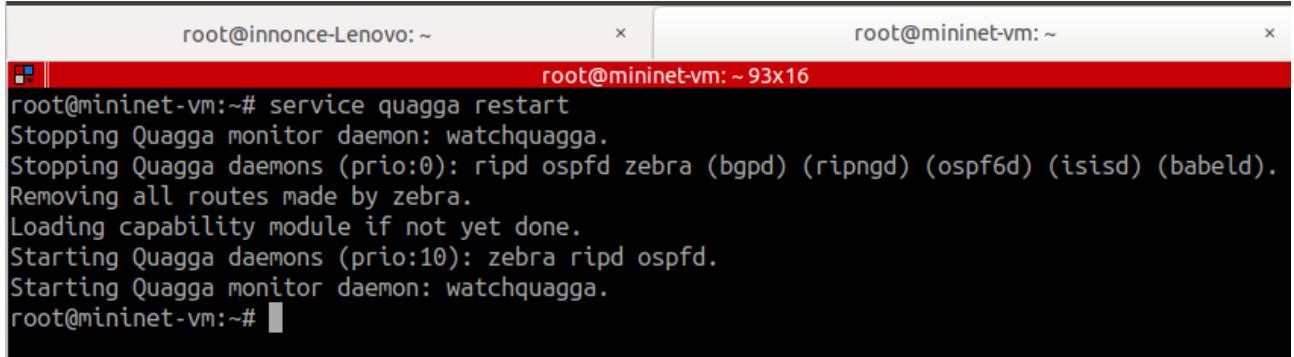
23,1

Bot

Différents démons de la suite Quagga fonctionneront sur le protocole TCP et les ports d'écoute seront de 2600 à 2800.

```
root@innonce-Lenovo: ~          x      root@mininet-vm: ~          x
root@mininet-vm:~# cat /etc/services | grep zebra
zebrasrv      2600/tcp          # zebra service
zebra         2601/tcp          # zebra vty
ripd          2602/tcp          # ripd vty (zebra)
ripngd        2603/tcp          # ripngd vty (zebra)
ospfd          2604/tcp          # ospfd vty (zebra)
bgpd          2605/tcp          # bgpd vty (zebra)
ospf6d        2606/tcp          # ospf6d vty (zebra)
isisd          2608/tcp          # ISISd vty (zebra)
root@mininet-vm:~#
```

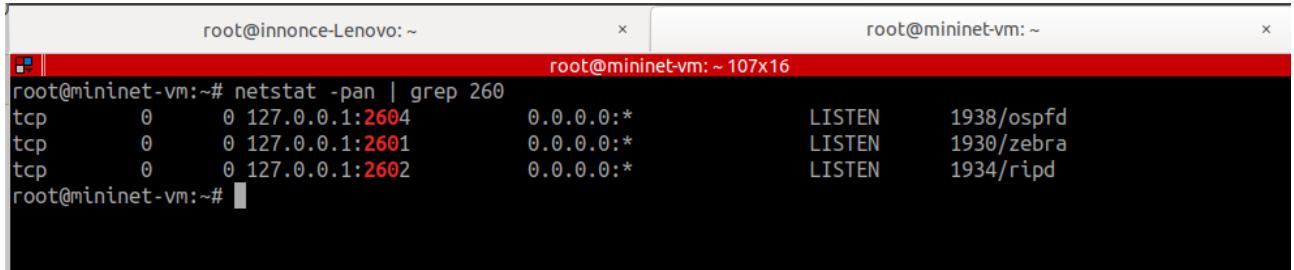
Redémarrons le service



The screenshot shows two terminal windows side-by-side. The left window is titled 'root@innonce-Lenovo: ~' and the right window is titled 'root@mininet-vm: ~'. Both windows have a red header bar. The left window contains the command 'service quagga restart' followed by its output: 'Stopping Quagga monitor daemon: watchquagga.', 'Stopping Quagga daemons (prio:0): ripd ospfd zebra (bgpd) (ripngd) (ospf6d) (isisd) (babeld).', 'Removing all routes made by zebra.', 'Loading capability module if not yet done.', 'Starting Quagga daemons (prio:10): zebra ripd ospfd.', and 'Starting Quagga monitor daemon: watchquagga.'. The right window has a similar structure but no visible output.

```
root@innonce-Lenovo: ~
root@mininet-vm: ~ 93x16
root@mininet-vm:~# service quagga restart
Stopping Quagga monitor daemon: watchquagga.
Stopping Quagga daemons (prio:0): ripd ospfd zebra (bgpd) (ripngd) (ospf6d) (isisd) (babeld).
Removing all routes made by zebra.
Loading capability module if not yet done.
Starting Quagga daemons (prio:10): zebra ripd ospfd.
Starting Quagga monitor daemon: watchquagga.
root@mininet-vm:~#
```

En utilisant la commande **netstat**, nous pouvons confirmer l'exécution réussie des démons



The screenshot shows two terminal windows side-by-side. The left window is titled 'root@innonce-Lenovo: ~' and the right window is titled 'root@mininet-vm: ~'. Both windows have a red header bar. The left window contains the command 'netstat -pan | grep 260' followed by its output: 'tcp 0 0 127.0.0.1:2604 0.0.0.0:\* LISTEN 1938/ospfd', 'tcp 0 0 127.0.0.1:2601 0.0.0.0:\* LISTEN 1930/zebra', and 'tcp 0 0 127.0.0.1:2602 0.0.0.0:\* LISTEN 1934/ripd'. The right window has a similar structure but no visible output.

```
root@innonce-Lenovo: ~
root@mininet-vm: ~ 107x16
root@mininet-vm:~# netstat -pan | grep 260
tcp      0      0 127.0.0.1:2604          0.0.0.0:*
tcp      0      0 127.0.0.1:2601          0.0.0.0:*
tcp      0      0 127.0.0.1:2602          0.0.0.0:*
LISTEN    1938/ospfd
LISTEN    1930/zebra
LISTEN    1934/ripd
root@mininet-vm:~#
```

Le routage Quagga peut-être configuré à l'aide des méthodes **telnet** ou **vtysh**.

vtysh fournit un emplacement unique pour la configuration de tous les démons. Exécutons la commande suivante dans le terminal pour démarrer le shell virtuel (vtysh) pour la configuration de Quagga

The screenshot shows two terminal windows side-by-side. The left window is titled 'root@innonce-Lenovo: ~' and the right window is titled 'root@mininet-vm: /etc/quagga'. Both windows have a red header bar.

The left window contains the command:

```
root@mininet-vm:~# cd /etc/quagga/
root@mininet-vm:/etc/quagga#
root@mininet-vm:/etc/quagga# vtysh
```

The right window displays the output of the 'vtysh' command, which is the Quagga Virtual Terminal Service Help. It lists various commands and their descriptions:

```
Hello, this is Quagga (version 0.99.22.4).
Copyright 1996-2005 Kunihiro Ishiguro, et al.

mininet-vm#
  clear      Reset functions
  configure  Configuration from vty interface
  copy       Copy from one file to another
  debug      Enable debug messages for specific or all part.
  disable    Turn off privileged mode command
  end        End current mode and change to enable mode
  exit       Exit current mode and down to previous mode
  list       Print command list
  no         Negate a command or set its defaults
  ping       Send echo messages
  quit      Exit current mode and down to previous mode
  show       Show running system information
  ssh        Open an ssh connection
  start-shell Start UNIX shell
  telnet     Open a telnet connection
  terminal   Set terminal line parameters
  traceroute Trace route to destination
  undebug   Disable debugging functions (see also 'debug')
  write     Write running configuration to memory, network, or terminal
```

Zebra, ospfd et ripd peuvent être configurés à l'aide de vtysh.

The screenshot shows a single terminal window with a red header bar. The title is 'root@innonce-Lenovo: ~' and the command being run is 'root@mininet-vm: /etc/quagga'.

The command entered is:

```
mininet-vm# show daemons
zebra ripd ospfd
```

La suite de routage Quagga est principalement utilisée sur la plate-forme Linux pour effectuer un routage dynamique. Il prend en charge plusieurs façons de configurer les protocoles de routage tels que OSPF et RIP. Les dispositifs de routage basés sur Quagga peuvent être utilisés pour les petites et moyennes entreprises (PME).

## V – Mise en place de VXLAN avec SDN (Software Defined Networking)

### V.1 – Cas 1 : Centralisation du contrôle des serveurs OVS interconnectés avec des VM

Cette mise en place consiste à déployer un contrôleur SDN OpenFlow pour la gestion des VTEP dans VXLAN.

Le projet peut-être divisé en principaux composants fonctionnels suivants :

#### 1. Topologie réseau

Cette étape va consister à créer les nœuds des serveurs (serveur 1, 2 et 3) qui hébergeront des machines virtuelles. OVS installé sur les serveurs est compatible avec VXLAN.

Les serveurs sont directement connectés au réseau L3.

#### 2. Réseau de superposition

OVS-switch1, OVS-switch2 et OVS-switch3 sont connectés au contrôleur SDN en occurrence Floodlight.

Les tunnels VXLAN sont créés entre :

OVS-switch1 ↔ OVS-switch2

OVS-switch1 ↔ OVS-switch3

OVS-switch2 ↔ OVS-switch3

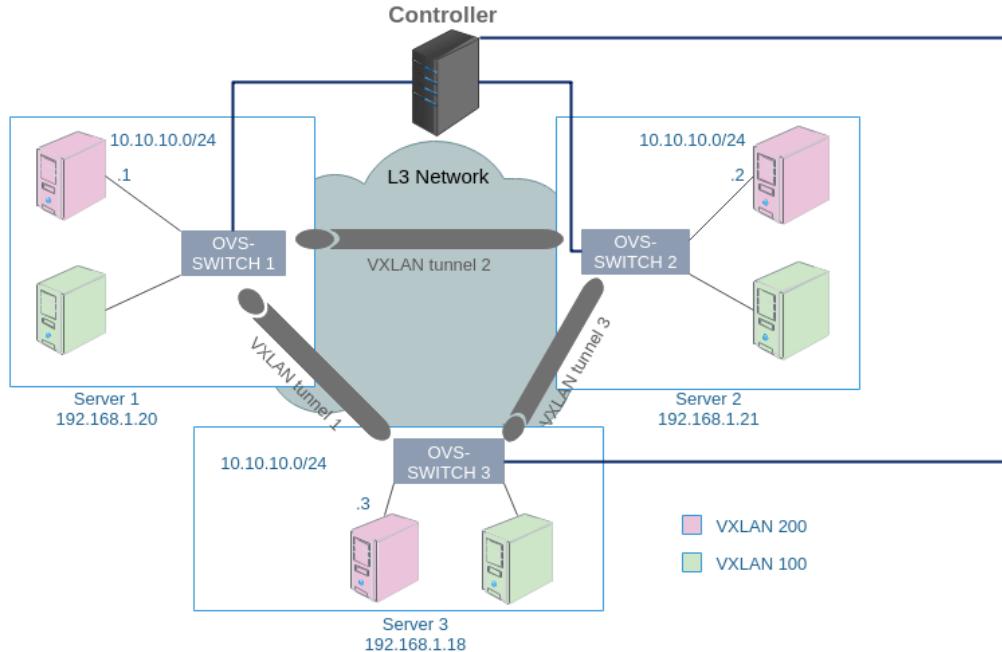
Le contrôleur pourra gérer les entrées de flux.

#### 3. Générer du trafic et effectuer la vérification et la validation du système qui comprend l'unité test

Le test inclut des vérifications ci-après :

- La résolution ARP se produit avec succès
- Les entrées des flux sont correctement mises à jour dans le tableau
- L'isolation du trafic se produit avec succès.

Architecture



### Création de ponts OVS

Une fois que nous avons créé nos ponts, nous allons utiliser le protocole OpenFlow14 et connecter chaque OVS au contrôleur.

```
root@innonce-VirtualBox:~# ovs-vsctl add-br switch1
root@innonce-VirtualBox:~#
root@innonce-VirtualBox:~# ovs-vsctl set-fail-mode switch1 secure
root@innonce-VirtualBox:~# ovs-vsctl set bridge switch1 protocols=OpenFlow14
root@innonce-VirtualBox:~# ovs-vsctl set-controller switch1 tcp:192.168.1.18:6653
root@innonce-VirtualBox:~#
```

```
root@innonce-Lenovo:~# ovs-vsctl add-br switch2
root@innonce-Lenovo:~#
root@innonce-Lenovo:~# ovs-vsctl set-fail-mode switch2 secure
root@innonce-Lenovo:~# ovs-vsctl set bridge switch2 protocols=OpenFlow14
root@innonce-Lenovo:~# ovs-vsctl set-controller switch2 tcp:192.168.1.18:6653
root@innonce-Lenovo:~#
```

```
root@dama-MacBookPro:~# ovs-vsctl add-br switch3
root@dama-MacBookPro:~# ovs-vsctl set-fail-mode switch3 secure
root@dama-MacBookPro:~# ovs-vsctl set bridge switch3 protocols=OpenFlow14
root@dama-MacBookPro:~# ovs-vsctl set-controller switch3 tcp:192.168.1.18:6653
root@dama-MacBookPro:~#
```

The screenshot shows a web-based interface for managing network switches. On the left, a sidebar lists navigation options: Applications, Gmail, YouTube, Installation, Maps, W3Schools, Welcome on..., PHP Sources, and Présentation... Below these are links for Commutateurs, Hôtes, Liens, Topologie, Pare-feu, Listes de contrôle d'accès, Statistiques, and Modifier les contrôleurs.

The main content area is titled "Commutateurs" and contains two sections:

- Commutateurs connectés**: A table listing three connected switches with their MAC addresses, IPv4 addresses, and connection times. All three switches are listed as "MAÎTRE" (Master).
- Échanger les rôles**: A table showing the current master status for three switches. All three switches are listed as "MAÎTRE" (Master).

ID de commutateur	Adresse IPv4	Connecté depuis
00: 00: 92: 5b: 05: f8: 4c: 4c	/192.168.1.20:58608	Mer 23 septembre 2020 13:00:39 GMT + 0000 (heure moyenne de Greenwich)
00: 00: da: 7f: 11: 56: 85: 49	/192.168.1.21:47770	Mer 23 septembre 2020 13:59:54 GMT + 0000 (heure moyenne de Greenwich)
00: 00: f6: 83: 71: a6: ce: 4b	/192.168.1.18:53744	Mer 23 septembre 2020 13:01:25 GMT + 0000 (heure moyenne de Greenwich)

Changer de MAC	Rôle
00: 00: f6: 83: 71: a6: ce: 4b	MAÎTRE
00: 00: 92: 5b: 05: f8: 4c: 4c	MAÎTRE
00: 00: da: 7f: 11: 56: 85: 49	MAÎTRE

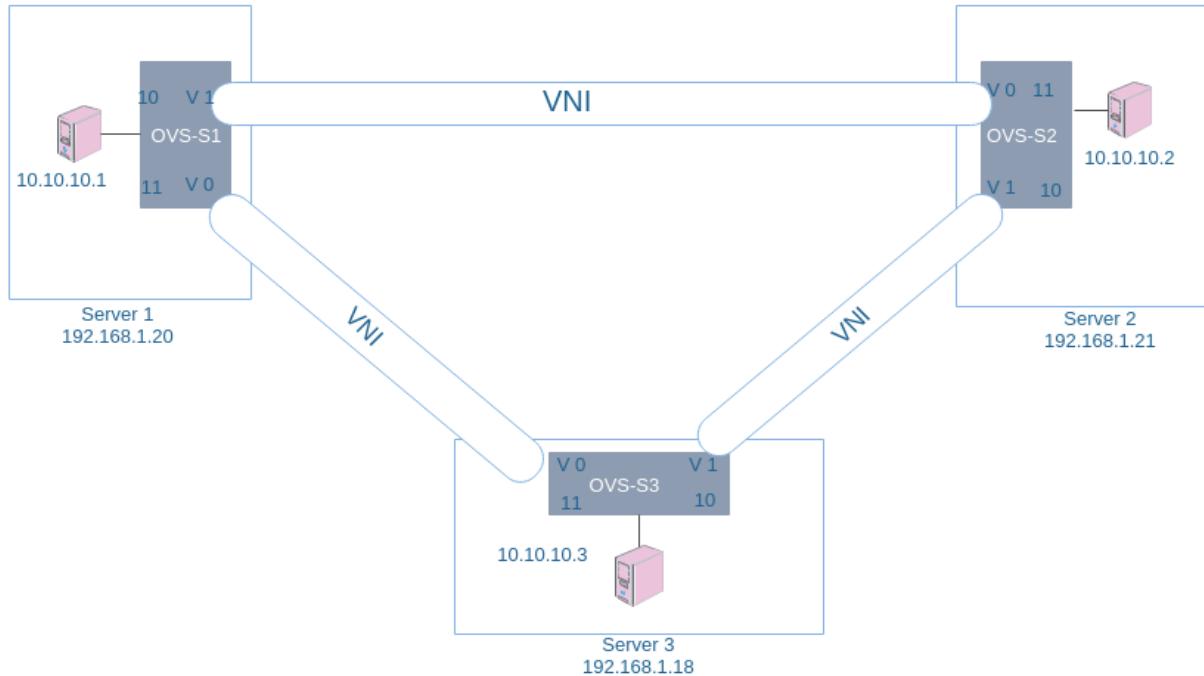
## Création des tunnels VXLAN sur les serveurs

```
root@innonce-VirtualBox:~# ovs-vsctl add-port switch1 v0 -- set interface v0 type=vxlan option:remote_ip=192.168.1.18 ofport_request=11
root@innonce-VirtualBox:~#
root@innonce-VirtualBox:~# ovs-vsctl add-port switch1 v1 -- set interface v1 type=vxlan option:remote_ip=192.168.1.21 ofport_request=10
root@innonce-VirtualBox:~# 
```

```
root@innonce-Lenovo:~# ovs-vsctl add-port switch2 v0 -- set interface v0 type=vxlan option:remote_ip=192.168.1.20 ofport_request=11
root@innonce-Lenovo:~# ovs-vsctl add-port switch2 v1 -- set interface v1 type=vxlan option:remote_ip=192.168.1.21 ofport_request=10
root@innonce-Lenovo:~# 
```

```
root@dama-MacBookPro:~# ovs-vsctl add-port switch3 v0 -- set interface v0 type=vxlan option:remote_ip=192.168.1.20 ofport_request=11
root@dama-MacBookPro:~# ovs-vsctl add-port switch3 v1 -- set interface v1 type=vxlan option:remote_ip=192.168.1.18 ofport_request=10
root@dama-MacBookPro:~# 
```

Conceptuellement, le réseau de superposition est représenté par ce montage :



Nous pouvons visualiser cela au niveau du contrôleur comme le montre la capture ci-dessous :

The screenshot shows a web-based interface for managing network links. The left sidebar includes links for Hôtes, Liens (selected), Topologie, Pare-feu, Listes de contrôle d'accès, Statistiques, and Modifier les contrôleurs.

**Liens internes**

Direction	Port source	Commutateur de source	Le port de destination	Commutateur de destination	Type
bidirectionnel	dix	00: 00: 92: 5b: 05: f8: 4c: 4c	11	00: 00: da: 7f: 11: 56: 85: 49	interne
bidirectionnel	11	00: 00: 92: 5b: 05: f8: 4c: 4c	11	00: 00: f6: 83: 71: a6: ce: 4b	interne
bidirectionnel	dix	00: 00: da: 7f: 11: 56: 85: 49	dix	00: 00: f6: 83: 71: a6: ce: 4b	interne

Affiche 1 à 3 sur 3 entrées

**Liens externes**

Direction	Port source	Commutateur de source	Le port de destination	Commutateur de destination	Type

aucune donnée disponible

Affichage de 0 à 0 sur 0 entrées

Test de connectivité entre les VM

```
root@innonce-VirtualBox:~# ping 10.10.10.2
PING 10.10.10.2 (10.10.10.2) 56(84) bytes of data.
64 bytes from 10.10.10.2: icmp_seq=1 ttl=64 time=300 ms
64 bytes from 10.10.10.2: icmp_seq=2 ttl=64 time=0.672 ms
64 bytes from 10.10.10.2: icmp_seq=3 ttl=64 time=0.274 ms
^C
--- 10.10.10.2 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2001ms
rtt min/avg/max/mdev = 0.274/100.505/300.570/141.467 ms
root@innonce-VirtualBox:~#
```

```
root@innonce-Lenovo:~# ping 10.10.10.1
PING 10.10.10.1 (10.10.10.1) 56(84) bytes of data.
64 bytes from 10.10.10.1: icmp_seq=1 ttl=64 time=14.4 ms
64 bytes from 10.10.10.1: icmp_seq=2 ttl=64 time=0.435 ms
^C
--- 10.10.10.1 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1001ms
rtt min/avg/max/mdev = 0.435/7.463/14.492/7.029 ms
root@innonce-Lenovo:~#
```

```
root@innonce-VirtualBox:~# ping -c3 10.10.10.3
PING 10.10.10.3 (10.10.10.3) 56(84) bytes of data.
64 bytes from 10.10.10.3: icmp_seq=1 ttl=64 time=41.2 ms
64 bytes from 10.10.10.3: icmp_seq=2 ttl=64 time=22.3 ms
64 bytes from 10.10.10.3: icmp_seq=3 ttl=64 time=2.30 ms

--- 10.10.10.3 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 2.309/21.962/41.250/15.900 ms
root@innonce-VirtualBox:~#
```

```
root@innonce-Lenovo:~# ping -c3 10.10.10.3
PING 10.10.10.3 (10.10.10.3) 56(84) bytes of data.
64 bytes from 10.10.10.3: icmp_seq=1 ttl=64 time=107 ms
64 bytes from 10.10.10.3: icmp_seq=2 ttl=64 time=244 ms
64 bytes from 10.10.10.3: icmp_seq=3 ttl=64 time=53.1 ms

--- 10.10.10.3 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2001ms
rtt min/avg/max/mdev = 53.121/134.973/244.682/80.647 ms
root@innonce-Lenovo:~#
```

## Application des règles

La règle établie va interdire la communication entre la VM de OVS-switch1 et celle de OVS-switch2

Créer une règle

Protocole

TCP  UDP  ICMP

IPv4 source  
10.10.10.1/24

Dest IPv4  
10.10.10.2/24

Dest TP  
Dest Tp

action

AUTORISER  NIER

Annuler Créer

Une fois que nous avons fini de remplir, on clique sur «créer» pour que la règle soit prise en compte au niveau des VTEP.

Listes de contrôle d'accès

ID	La source	Dest	IP source	Masque	IP de destination	Masque	Prot.	Dest TP	Acte.	Supprir
1	10.10.10.1/24	10.10.10.2/24	168430081	24	168430082	24	1	0	AUTORISER	<span style="background-color: red; color: white; border-radius: 5px; padding: 2px 10px;">Supprir</span>

Affiche 1 à 1 de 1 entrées

On réessaie un test de communication pour constater que les VM dans OVS-switch1 et OVS-switch2 ne peuvent plus communiquer.

```

root@innonce-VirtualBox:~# ping -c3 10.10.10.2
PING 10.10.10.2 (10.10.10.2) 56(84) bytes of data.

--- 10.10.10.2 ping statistics ---
3 packets transmitted, 0 received, 100% packet loss, time 2039ms

root@innonce-VirtualBox:~# 
```

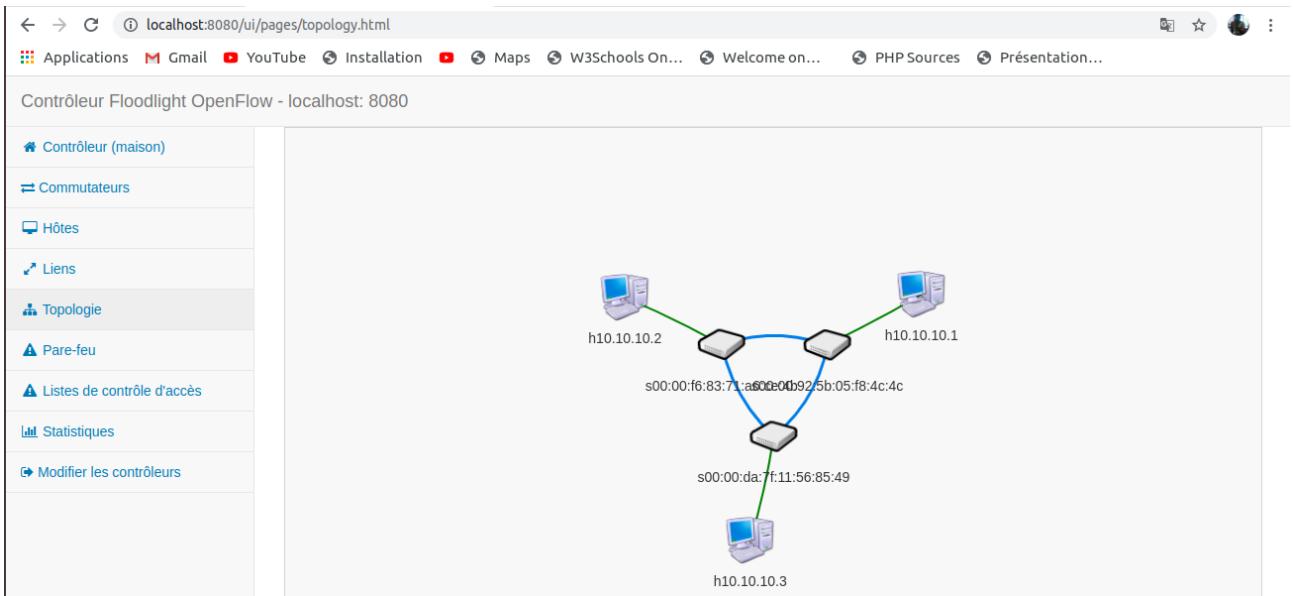
```

root@innonce-Lenovo:~# ping -c3 10.10.10.1
PING 10.10.10.1 (10.10.10.1) 56(84) bytes of data.

--- 10.10.10.1 ping statistics ---
3 packets transmitted, 0 received, 100% packet loss, time 2048ms

root@innonce-Lenovo:~# 
```

## Vue de la topologie au niveau du contrôleur

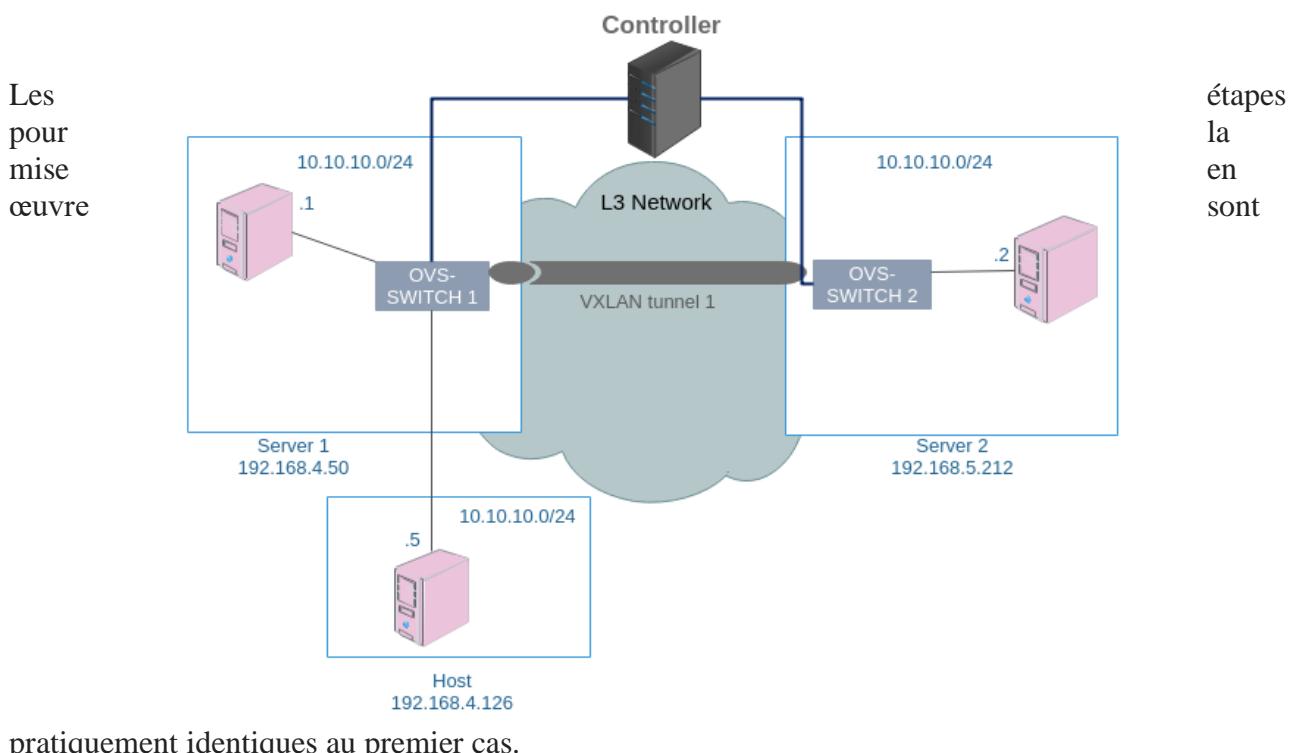


## V.2 – Cas 2 : Centralisation du contrôle des serveurs OVS interconnectés avec des VM et des équipements physiques

Ce second cas de figure va consister à interconnecter deux serveurs OVS dans lequel un pont sera créé au niveau du premier OVS afin d'ajouter une machine physique qui pourra dialoguer avec des machines virtuelles dans les serveurs.

Il faut noter que c'est bien cette méthode qui peut-être déployé en production grâce à cette notion de pont qui permet de connecter des équipements physiques au niveau des OVS.

### Architecture



pratiquement identiques au premier cas.

### Création des OVS

```
root@innonce-Lenovo:~# ovs-vsctl add-br ovs-switch1
root@innonce-Lenovo:~#
root@innonce-Lenovo:~# ovs-vsctl set-fail-mode ovs-switch1 secure
root@innonce-Lenovo:~# ovs-vsctl set bridge ovs-switch1 protocols=OpenFlow14
root@innonce-Lenovo:~#
root@innonce-Lenovo:~# ovs-vsctl set-controller ovs-switch1 tcp:192.168.4.50:6653
root@innonce-Lenovo:~#
```

```
root@innonce-VirtualBox:~# ovs-vsctl add-br ovs-switch2
root@innonce-VirtualBox:~# ovs-vsctl set-fail-mode ovs-switch2 secure
root@innonce-VirtualBox:~# ovs-vsctl set bridge ovs-switch2 protocols=OpenFlow14
root@innonce-VirtualBox:~# ovs-vsctl set-controller ovs-switch2 tcp:192.168.4.50:6653
root@innonce-VirtualBox:~#
```

The screenshot shows a web-based management interface for an Open vSwitch setup. On the left, a sidebar menu includes options like Contrôleur (maison), Commutateurs, Hôtes, Liens, Topologie, Pare-feu, Listes de contrôle d'accès, Statistiques, and Modifier les contrôleurs. The main content area is titled "Commutateurs" and displays two connected switches:

ID de commutateur	Adresse IPv4	Connecté depuis
00: 00: 00: e0: 4c: 36: 04: 55	/192.168.4.50:60730	Ven 02 Oct 2020 16:31:17 GMT + 0000 (heure moyenne de Greenwich)
00: 00: 6a: 71: 62: 14: 46: 4b	/192.168.5.212:41304	Ven 02 Oct 2020 16:31:49 GMT + 0000 (heure moyenne de Greenwich)

Below this, a section titled "Échanger les rôles" shows the current roles assigned to each switch:

Changer de MAC	Rôle
00: 00: 6a: 71: 62: 14: 46: 4b	MAÎTRE
00: 00: 00: e0: 4c: 36: 04: 55	MAÎTRE

## Création des tunnels VXLAN

```
root@innonce-Lenovo:~# ovs-vsctl add-port ovs-switch1 v0 -- set interface v0 type=vxlan option:remote_ip=192.168.5.212
ofport_request=2
root@innonce-Lenovo:~#
```

```
root@innonce-VirtualBox:~# ovs-vsctl add-port ovs-switch2 v0 -- set interface v0 type=vxlan option:remote_ip=192.168.4.50
ofport_request=2
root@innonce-VirtualBox:~#
```

Chaque tunnel est bidirectionnel comme on peut le voir au niveau du contrôleur

The screenshot shows the Floodlight OpenFlow controller interface at [localhost:8080/ui/pages/links.html](http://localhost:8080/ui/pages/links.html). The left sidebar has a tree view with nodes: Contrôleur (maison), Commutateurs, Hôtes, Liens (selected), Topologie, Pare-feu, Listes de contrôle d'accès, Statistiques, and Modifier les contrôleurs. The main area is titled "Liens" and contains two tables. The first table, "Liens internes", has columns: Direction, Port source, Commutateur de source, Le port de destination, Commutateur de destination, and Type. It shows one entry: bidirectionnel, port 2, source switch 00:00:00:e0:4c:36:04:55, destination port 2, destination switch 00:00:6a:71:62:14:46:4b, and type interne. The second table, "Liens externes", has the same columns but shows "aucune donnée disponible". A message at the bottom says "Affichage de 0 à 0 sur 0 entrées".

Ajout de l'interface où est connecté la machine physique au sein de l'OVS

```
root@innonce-Lenovo:~# ovs-vsctl add-port ovs-switch1 enx00e04c360455
root@innonce-Lenovo:~#
```

On liste les ports de l'OVS

```
root@innonce-Lenovo:~# ovs-vsctl list-ports ovs-switch1
enx00e04c360455
root@innonce-Lenovo:~#
```

Affectation des adresses IP

```
root@innonce-Lenovo:~# #ip addr add 10.10.10.1 dev ovs-switch1
root@innonce-Lenovo:~# ifconfig ovs-switch1 up
root@innonce-Lenovo:~#
```

```
root@innonce-VirtualBox:~# ip addr add 10.10.10.2 dev ovs-switch2
root@innonce-VirtualBox:~# ifconfig ovs-switch2 up
root@innonce-VirtualBox:~#
```

```

root@dama-MacBookPro:~# #ip addr add 10.10.10.5/24 dev enp2s0f0
root@dama-MacBookPro:~# #ifconfig enp2s0f0 up
root@dama-MacBookPro:~# ifconfig enp2s0f0
enp2s0f0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
      inet 10.10.10.5  netmask 255.255.255.0  broadcast 0.0.0.0
        ether 10:9a:dd:70:f0:74  txqueuelen 1000  (Ethernet)
          RX packets 852  bytes 181060 (181.0 KB)
          RX errors 0  dropped 229  overruns 0  frame 0
          TX packets 212  bytes 33943 (33.9 KB)
          TX errors 0  dropped 0  overruns 0  carrier 0  collisions 0
          device interrupt 16

root@dama-MacBookPro:~# 

```

Il faut s'assurer que la connexion entre les OVS et le contrôleur soit fiable afin que les messages échangés entre les deux entités parviennent de part et d'autre. Le cas défaillant le contrôleur ne verra pas les adresses IP attribuées aux hôtes.

The screenshot shows the Floodlight OpenFlow web interface at [localhost:8080/ui/pages/hosts.html](http://localhost:8080/ui/pages/hosts.html). The left sidebar has a tree view with nodes: Contrôleur (maison), Commutateurs, Hôtes (selected), Liens, Topologie, Pare-feu, Listes de contrôle d'accès, Statistiques, and Modifier les contrôleurs. The main content area is titled "Hôtes" and contains a sub-section "Hôtes connectés". A table lists three connected hosts:

MAC	Adresse IPv4	Adresse IPv6	Commutateur	Port	Dernière vue
00:e0:4c:36:04:55	10.10.10.1	fe80 :: 2e0: 4cff: fe36: 455	00:00:00:e0:4c:36:04:55	local	1601656417926
10:9a:jj:70:f0:74	10.10.10.5		00:00:00:e0:4c:36:04:55	1	1601656410070
6a:71:62:14:46:4b	10.10.10.2	fe80 :: 6871: 62ff: fe14: 464b	00:00:6a:71:62:14:46:4b	local	1601656439462

Below the table, it says "Affiche 1 à 3 sur 3 entrées".

### Test de connectivité entre les hôtes

```

root@innonce-Lenovo:~# ping -c2 10.10.10.2; ping -c2 10.10.10.5
PING 10.10.10.2 (10.10.10.2) 56(84) bytes of data.
64 bytes from 10.10.10.2: icmp_seq=1 ttl=64 time=8.11 ms
64 bytes from 10.10.10.2: icmp_seq=2 ttl=64 time=0.800 ms

--- 10.10.10.2 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1001ms
rtt min/avg/max/mdev = 0.800/4.457/8.115/3.658 ms
PING 10.10.10.5 (10.10.10.5) 56(84) bytes of data.
64 bytes from 10.10.10.5: icmp_seq=1 ttl=64 time=4.70 ms
64 bytes from 10.10.10.5: icmp_seq=2 ttl=64 time=1.42 ms

--- 10.10.10.5 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1001ms
rtt min/avg/max/mdev = 1.421/3.064/4.708/1.644 ms
root@innonce-Lenovo:~# 

```

```

root@innonce-VirtualBox:~# ping -c2 10.10.10.1; ping -c2 10.10.10.5
PING 10.10.10.1 (10.10.10.1) 56(84) bytes of data.
64 bytes from 10.10.10.1: icmp_seq=1 ttl=64 time=8.16 ms
64 bytes from 10.10.10.1: icmp_seq=2 ttl=64 time=0.825 ms

--- 10.10.10.1 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1000ms
rtt min/avg/max/mdev = 0.825/4.494/8.163/3.669 ms
PING 10.10.10.5 (10.10.10.5) 56(84) bytes of data.
64 bytes from 10.10.10.5: icmp_seq=1 ttl=64 time=7.75 ms
64 bytes from 10.10.10.5: icmp_seq=2 ttl=64 time=1.79 ms

--- 10.10.10.5 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1001ms
rtt min/avg/max/mdev = 1.796/4.777/7.758/2.981 ms
root@innonce-VirtualBox:~#

```

```

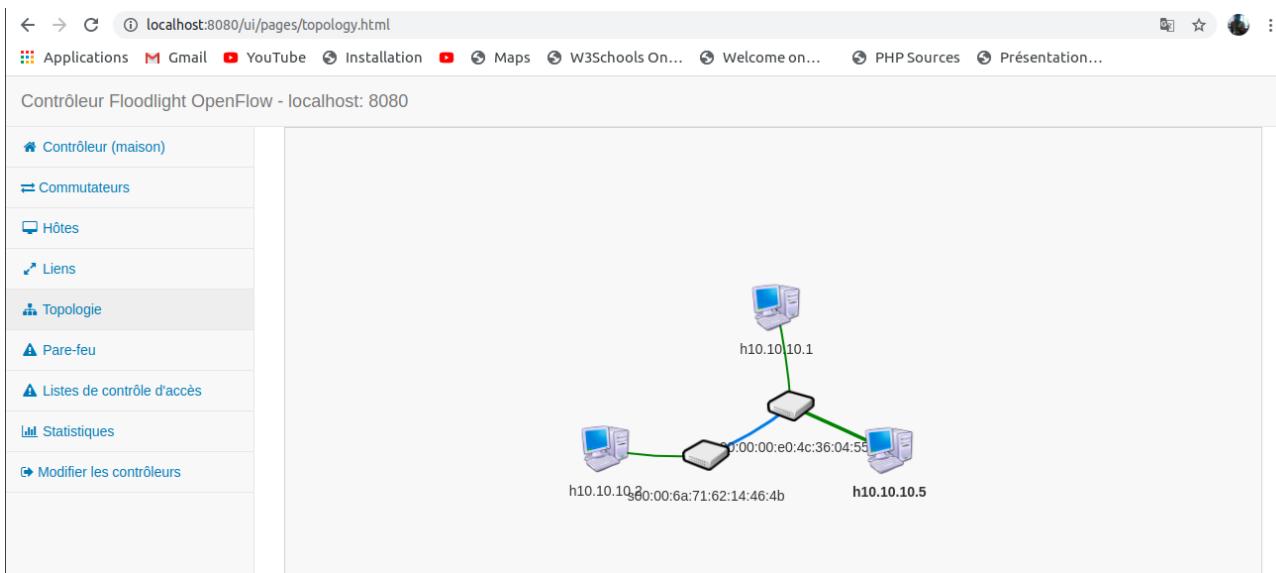
root@dama-MacBookPro:~# ping -c2 10.10.10.1; ping -c2 10.10.10.2
PING 10.10.10.1 (10.10.10.1) 56(84) bytes of data.
64 bytes from 10.10.10.1: icmp_seq=1 ttl=64 time=4.76 ms
64 bytes from 10.10.10.1: icmp_seq=2 ttl=64 time=1.25 ms

--- 10.10.10.1 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1001ms
rtt min/avg/max/mdev = 1.250/3.008/4.766/1.758 ms
PING 10.10.10.2 (10.10.10.2) 56(84) bytes of data.
64 bytes from 10.10.10.2: icmp_seq=1 ttl=64 time=8.22 ms
64 bytes from 10.10.10.2: icmp_seq=2 ttl=64 time=1.88 ms

--- 10.10.10.2 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1001ms
rtt min/avg/max/mdev = 1.888/5.056/8.224/3.168 ms
root@dama-MacBookPro:~#

```

## Vue de la topologie au niveau du contrôleur

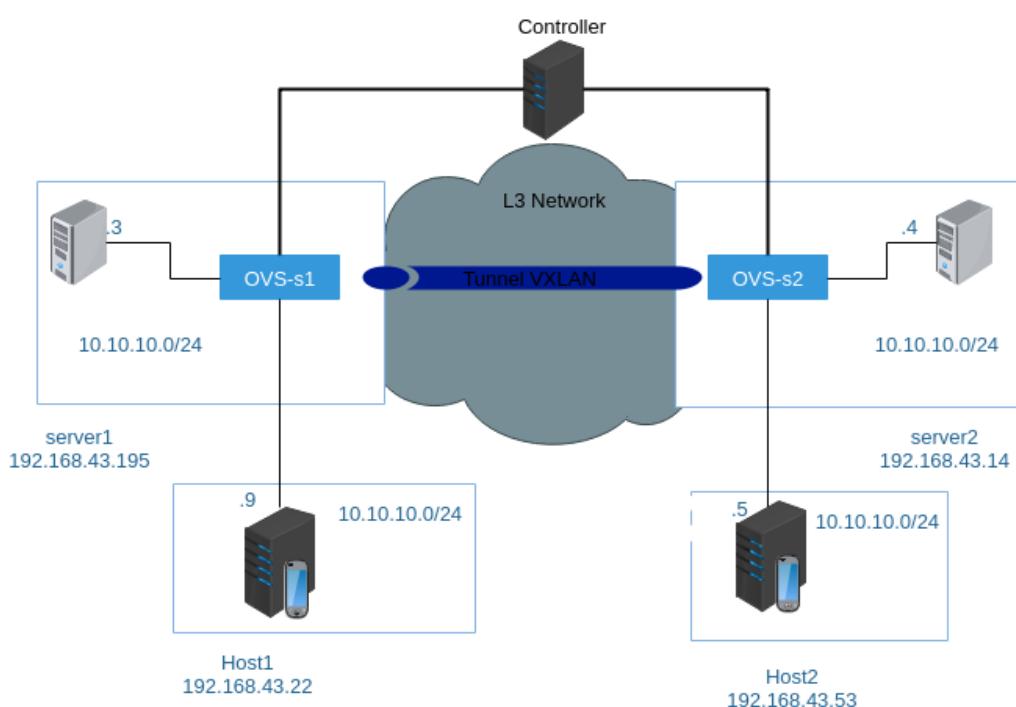


### **V.3 – Cas 3 :** Déploiement d'un système de communication au sein d'un réseau de superposition centralisé

Cette troisième partie de notre travail va consister à mettre en place un service de communication avec l'outil BLINK au sein d'un réseau de superposition.

Notre architecture se compose de deux serveurs interconnectés par un tunnel VXLAN. Sur chaque serveur on trouve un OVS auquel est connecté des machines aussi bien physiques que virtuelles. L'idée de ce type de montage s'inscrit dans l'optique où une entreprise ayant deux sites ou même plusieurs puisse établir un système de communication permettant aux employés entre les différents sites d'échanger.

#### **Architecture**



#### **Mise en œuvre**

- La première va consister à créer les OVS au niveau de chaque serveur (**ovs-s1 & ovs-s2**), puis choisir le protocole en occurrence **OpenFlow14** dans notre cas et enfin connecter l'OVS au contrôleur.
- Dans la deuxième étape, nous allons créer un tunnel VxlAN entre les deux serveurs.
- La troisième étape servira à créer des ponts afin d'ajouter les interfaces auxquelles seront rattachés les équipements.
- A l'étape quatre, nous allons attribuer les adresses IP aux équipements et tester la communication.
- Système de communication avec BLINK

```
root@innonce-Lenovo:~# ovs-vsctl add-br ovs-s1
root@innonce-Lenovo:~#
root@innonce-Lenovo:~# ovs-vsctl set-fail-mode ovs-s1 secure
root@innonce-Lenovo:~# ovs-vsctl set bridge ovs-s1 protocols=OpenFlow14
root@innonce-Lenovo:~# ovs-vsctl set-controller ovs-s1 tcp:192.168.43.195:6653
root@innonce-Lenovo:~#
root@innonce-Lenovo:~# ovs-vsctl add-port ovs-s1 v0 -- set interface v0 type=vxlan option:remote_ip=192.168.43.14 ofport_request=2
root@innonce-Lenovo:~#
root@innonce-Lenovo:~# ovs-vsctl add-port ovs-s1 enx8026891cc394
root@innonce-Lenovo:~#
root@innonce-Lenovo:~# ovs-vsctl list-ports ovs-s1
enx8026891cc394
v0
```

```
root@latyr-Lenovo:~# ovs-vsctl add-br ovs-s2
root@latyr-Lenovo:~# ovs-vsctl set-fail-mode ovs-s2
ovs-vsctl: 'set-fail-mode' command requires at least 2 arguments
root@latyr-Lenovo:~# ovs-vsctl set-fail-mode ovs-s2 secure
root@latyr-Lenovo:~# ovs-vsctl set bridge ovs-s2
ovs-vsctl: 'set' command requires at least 3 arguments
root@latyr-Lenovo:~# ovs-vsctl set bridge ovs-s2 protocols=OpenFlow14
root@latyr-Lenovo:~# ovs-vsctl set-controller ovs-s2 tcp:192.168.43.195
root@latyr-Lenovo:~# ovs-vsctl set-controller ovs-s2 tcp:192.168.43.195:6653
root@latyr-Lenovo:~#
```

```
root@latyr-Lenovo:~# ovs-vsctl add-port ovs-s2 v0 -- set interface v0 type=vxlan option:remote_ip=192.168.43.195 ofport_request=2
root@latyr-Lenovo:~# ifconfig
enp3s0: flags=4099<UP,BROADCAST,MULTICAST> mtu 1500
      ether 54:e1:ad:24:7d:ee txqueuelen 1000  (Ethernet)
      RX packets 254 bytes 65746 (65.7 KB)
      RX errors 0 dropped 0 overruns 0 frame 0
      TX packets 358 bytes 59648 (59.6 KB)
      TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

enx00e04c360455: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
      inet6 fe80::238a:4bfc:b463:ec51 prefixlen 64 scopeid 0x20<link>
      ether 00:e0:4c:36:04:55 txqueuelen 1000  (Ethernet)
      RX packets 3 bytes 582 (582.0 B)
      RX errors 0 dropped 0 overruns 0 frame 0
      TX packets 272 bytes 45003 (45.0 KB)
      TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

## OVS au niveau du contrôleur

Switch ID	IPv4 Address	Connected Since
00:00:00:e0:4c:36:04:55	/192.168.43.14:46872	Sat Oct 03 2020 17:52:30 GMT+0000 (heure moyenne de Greenwich)
00:00:80:26:89:1c:c3:94	/192.168.43.195:38110	Sat Oct 03 2020 17:17:22 GMT+0000 (heure moyenne de Greenwich)

## Tunnel OVS

Direction	Source Port	Source Switch	Destination Port	Destination Switch	Type
bidirectional	2	00:00:00:e0:4c:36:04:55	2	00:00:80:26:89:1c:c3:94	internal

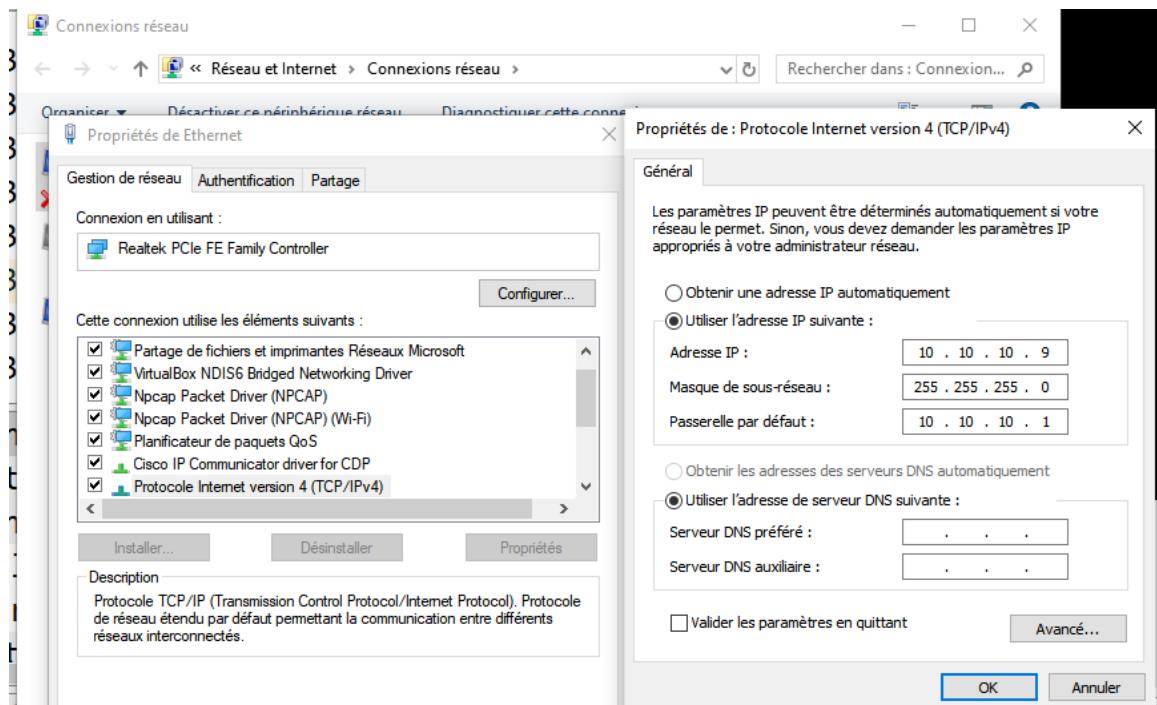
No data available in table

## Attribution des adresses IP

Il faut noter que les adresses IP au sein du réseau de superposition ont été affecté manuellement.

```
root@dama-MacBookPro:~# ip addr add 10.10.10.5/24 dev enp2s0f0
root@dama-MacBookPro:~# ifconfig enp2s0f0 up
root@dama-MacBookPro:~# ifconfig enp2s0f0
enp2s0f0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
        inet 10.10.10.5 netmask 255.255.255.0 broadcast 0.0.0.0
          ether 10:9a:dd:70:f0:74 txqueuelen 1000 (Ethernet)
            RX packets 852 bytes 181060 (181.0 KB)
            RX errors 0 dropped 229 overruns 0 frame 0
            TX packets 212 bytes 33943 (33.9 KB)
            TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
            device interrupt 16

root@dama-MacBookPro:~#
```



Ci-dessous une capture qui nous permet de visualiser au niveau du contrôleur.

MAC	IPv4 Address	IPv6 Address	Switch	Port	Last
00:1e:68:fa:09:c0	10.10.10.5	fe80::c53d:6600:2fa4:6be8	00:00:00:e0:4c:36:04:55000:00:80:26:89:1c:c3:94	302	16017
00:e0:4c:36:04:55	10.10.10.4	fe80::2e0:4cff:fe36:455	00:00:00:e0:4c:36:04:55000:00:80:26:89:1c:c3:94	local02	16017
2c:27:d7:82:68:fa	10.10.10.7,169.254.74.79	fe80::35b2:6a5f:649e:4af			16017
80:26:89:1c:c3:94	10.10.10.3	fe80::8226:89ff:fe1c:c394	00:00:00:e0:4c:36:04:55000:00:80:26:89:1c:c3:94	20local	16017
94:57:a5:db:b3:5d	10.10.10.9,169.254.140.201	fe80::e95c:5e06:caa8:8cc9	00:00:00:e0:4c:36:04:55000:00:80:26:89:1c:c3:94	201	16017

## Test de connectivité

```

root@innonce-Lenovo:~# ping -c2 10.10.10.4; ping -c2 10.10.10.9; ping -c2 10.10.10.5
PING 10.10.10.4 (10.10.10.4) 56(84) bytes of data.
64 bytes from 10.10.10.4: icmp_seq=1 ttl=64 time=72.8 ms
64 bytes from 10.10.10.4: icmp_seq=2 ttl=64 time=13.8 ms

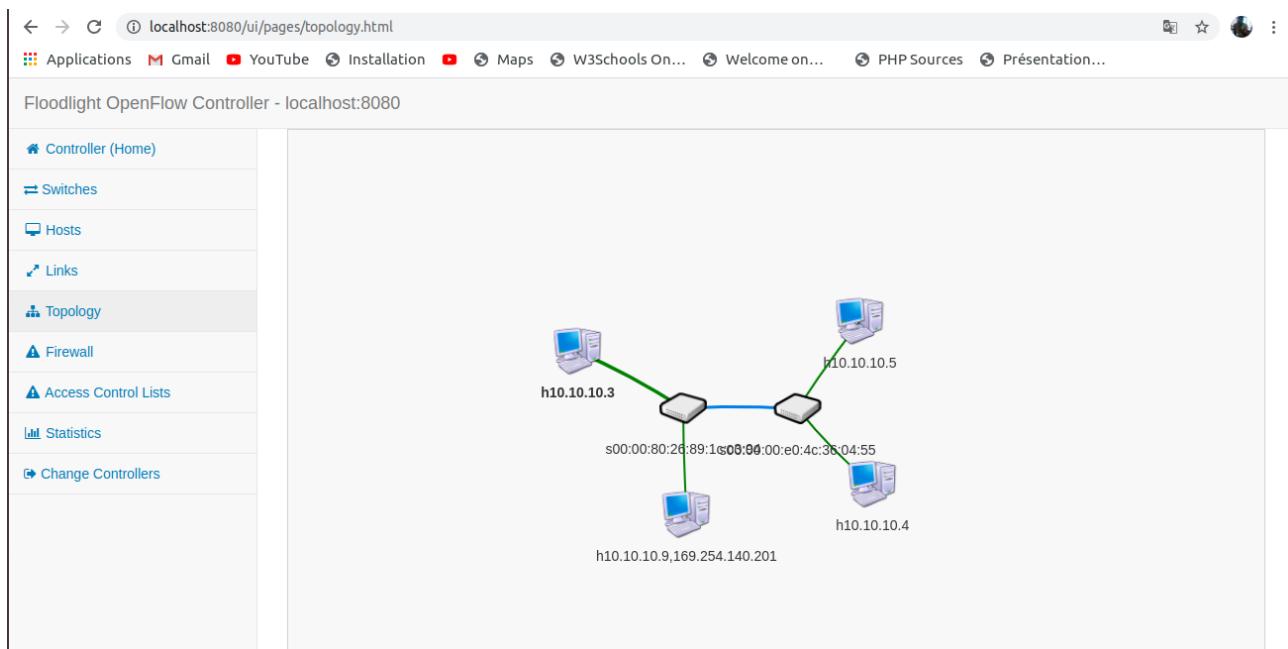
--- 10.10.10.4 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1001ms
rtt min/avg/max/mdev = 13.817/43.309/72.801/29.492 ms
PING 10.10.10.9 (10.10.10.9) 56(84) bytes of data.
64 bytes from 10.10.10.9: icmp_seq=1 ttl=128 time=4.07 ms
64 bytes from 10.10.10.9: icmp_seq=2 ttl=128 time=1.02 ms

--- 10.10.10.9 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1001ms
rtt min/avg/max/mdev = 1.029/2.553/4.077/1.524 ms
PING 10.10.10.5 (10.10.10.5) 56(84) bytes of data.
64 bytes from 10.10.10.5: icmp_seq=1 ttl=128 time=191 ms
64 bytes from 10.10.10.5: icmp_seq=2 ttl=128 time=228 ms

--- 10.10.10.5 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 999ms
rtt min/avg/max/mdev = 191.268/209.735/228.203/18.473 ms
root@innonce-Lenovo:~# 

```

## Topologie



## Système de communication avec l'outil BLINK

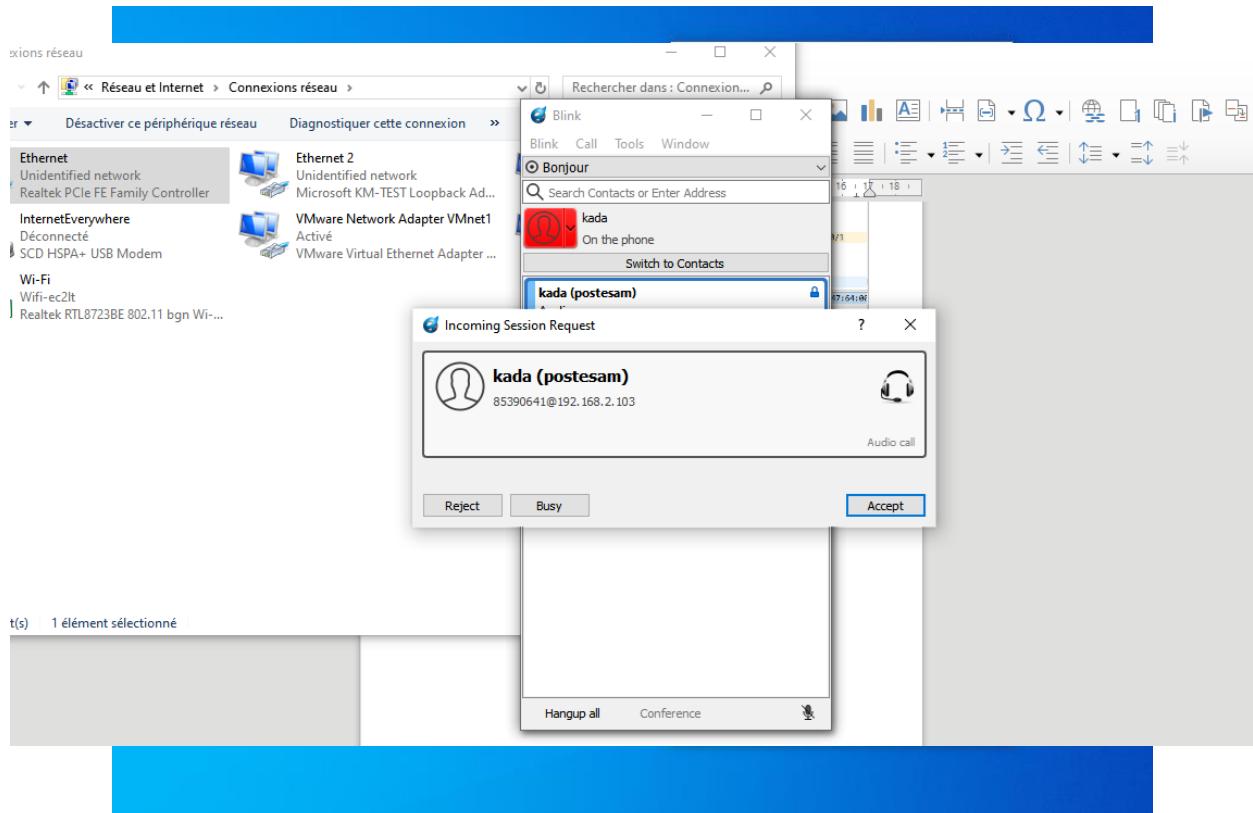
Blink est un client SIP à la pointe de la technologie et facile à utiliser. Blink propose des communications multimédias riches basées sur le protocole SIP en utilisant Future Proof Addressing sous la forme d'une adresse e-mail. Intégration facile avec les fournisseurs de services SIP existants. Open Source, on obtient le code source et on peut contribuer.

Blink est un client SIP basé sur l'IETF qui implémente des sessions audio et vidéo à l'aide du protocole RTP, une messagerie instantanée basée sur une session, un transfert de fichiers et des sessions de discussion multi-parties utilisant le protocole MSRP, le partage d'écran à l'aide du protocole VNC, la publication / abonnement pour les informations de présence et la gestion de la liste d'amis en utilisant les protocoles RLS et XCAP.

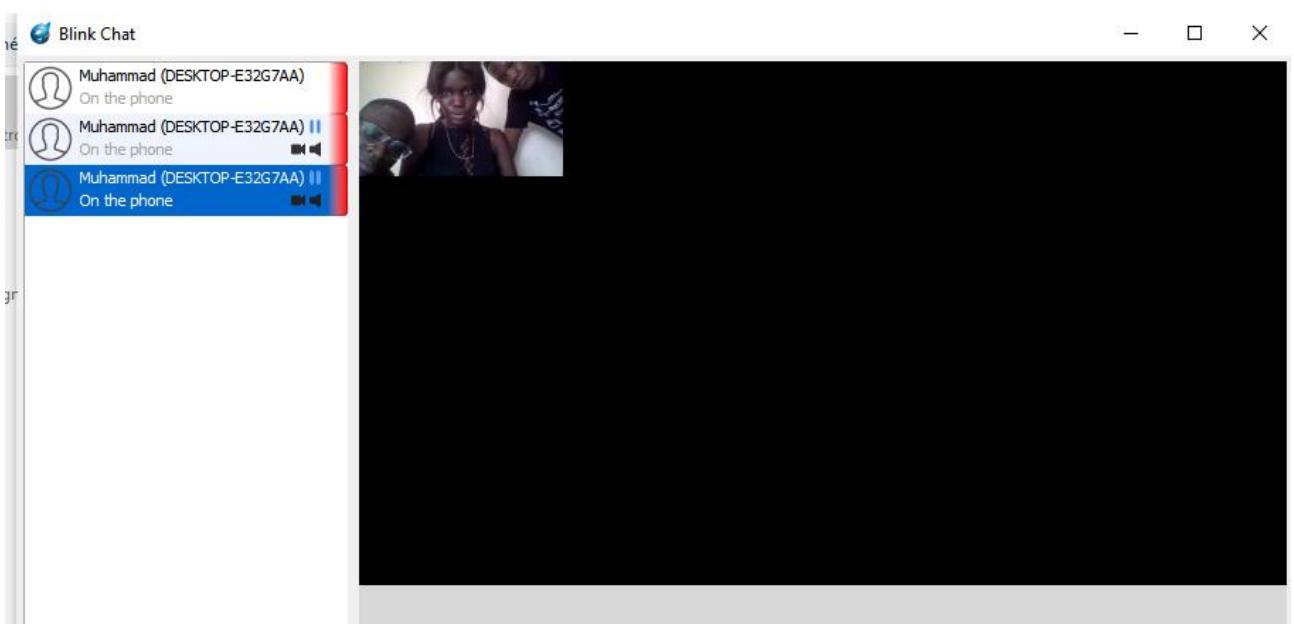
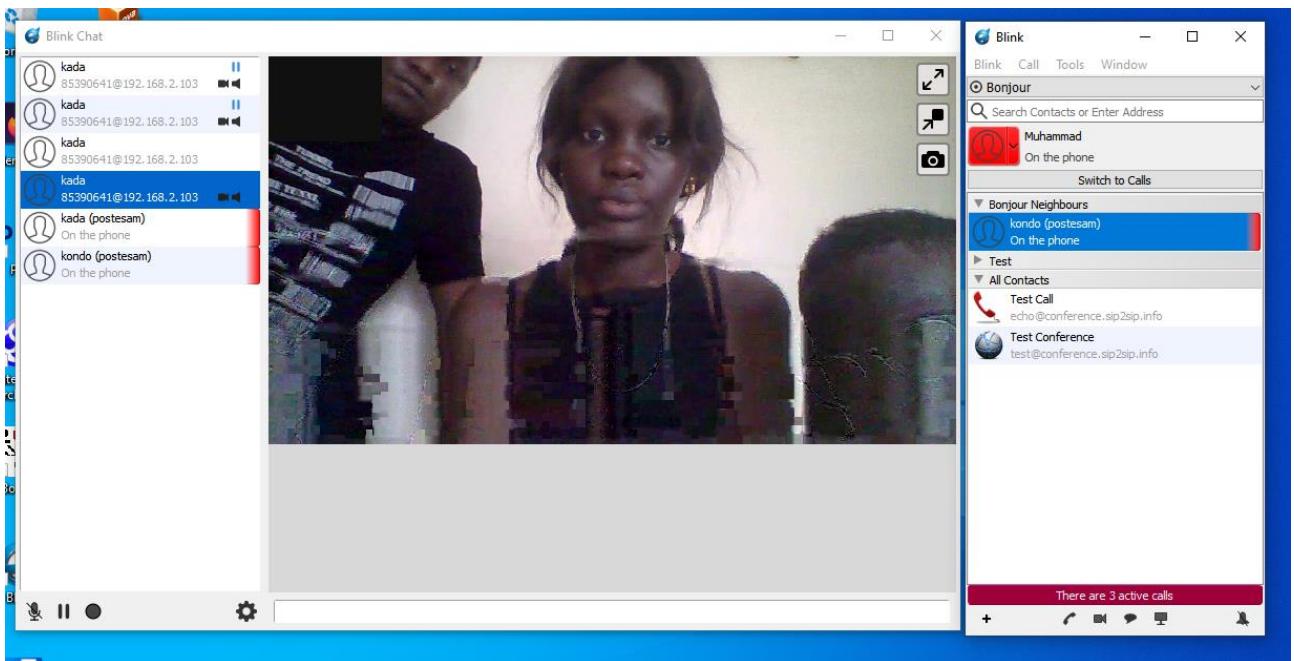
- Audio large bande (codec Opus)
- Vidéo HD (codec H.264)
- Chat et transfert de fichiers (protocole MSRP)
- Présence (SIP SIMPLE)
- Partage d'écran (protocole VNC)
- Chiffrement de bout en bout (protocoles ZRTP et ORT)
- Conférence multipartite

Le lien ci-dessous permettra de télécharger le logiciel  
<http://icanblink.com/>

- **Test d'appel**



- **Appel vidéo**



- **Transfert de fichiers**

