Sommaire

[1. Mise en place de la plateforme mininet 1](#_Toc127534763)

[2. Test de fonctionnement 3](#_Toc127534764)

[3. Premier contact avec mininet 4](#_Toc127534765)

[4. D’autres topologies réseaux 6](#_Toc127534766)

[5. Pour aller encore plus loin 9](#_Toc127534767)

[6. Et c’est presque la fin 12](#_Toc127534768)

# 1. Mise en place de la plateforme mininet

Lors de ce TP nous allons devoir mettre en place une plateforme mininet

1) Pour se faire je dois commencer par télécharger le fichier .ovf de la VM sur le lien suivant : <https://github.com/mininet/mininet/releases/download/2.3.0/mininet-2.3.0-210211-ubuntu-20.04.1-legacy-server-amd64-ovf.zip>

2) Une fois le fichier téléchargé je le décompresse dans un répertoire

3) Depuis mon hyperviseur VmWare j’importe la VM

4) Je sélectionne mon fichier .ovf

5) Je donne un nom à la VM (ici mininet)

6) L’importation s’effectue correctement et j’ai maintenant une VM de prête :

7)



8) Je laisse la VM dans un réseau privé hôte (NAT)

9) Je démarre la VM

10) Je me connecte avec les identifiants suivants : mininet et mininet

11) Je récupère l’adresse IP de ma VM avec la commande suivante :

ip a

12) la VM se situe bien dans le même réseau que mon ordinateur physique

13) Afin de pouvoir me connecter en SSH à la VM je télécharge et j’installe « putty » sur le lien suivant :

<https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html>

14) Afin de pouvoir lancer les applications graphiques installées sur la VM je télécharge Xming sur le lien suivant :

<https://sourceforge.net/projects/xming/>

15) Une fois les applications installées je lance putty en le configurant de la façon suivante :

Une image contenant texte

Description générée automatiquement

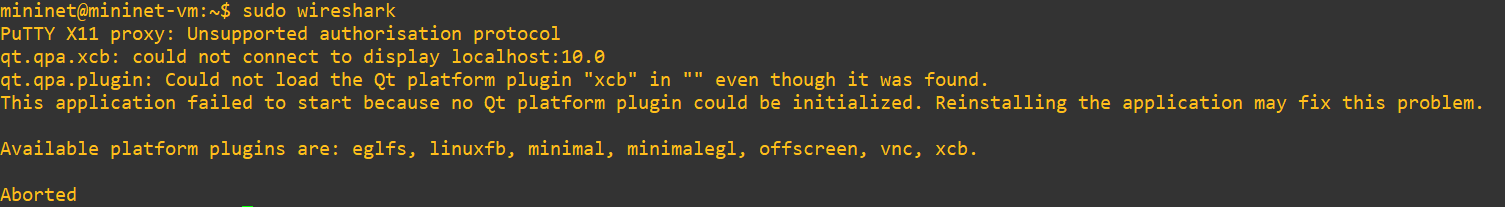
16) Je suis bien connecté en SSH à ma VM :

Une image contenant texte

Description générée automatiquement

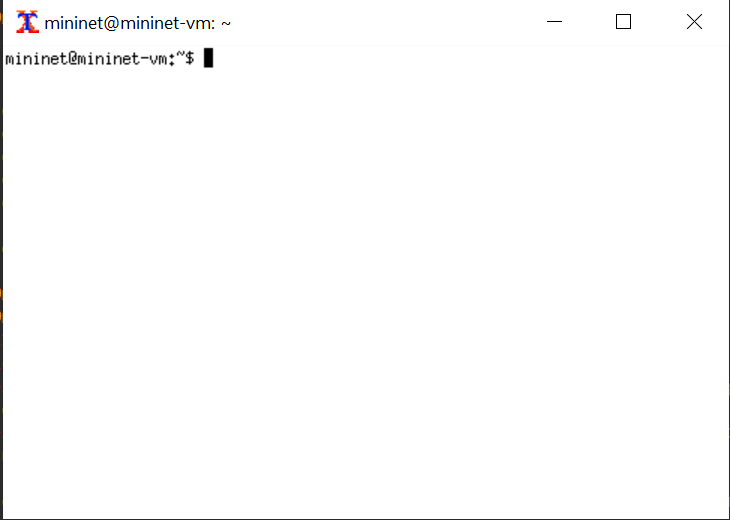
# 2. Test de fonctionnement

Dans l’invite de commande si je lance la commande « sudo wireshark » une fenêtre devrait s’ouvrir sur ma machine physique car la VM ne dispose pas d’un environnement :



Wireshark n’est pas pris en charge par Xming afin d’ouvrir l’application

Nous pouvons tester soit de télécharger un autre moteur soit lancer une autre application comme xterm :



# 3. Premier contact avec mininet

Avant de commencer la pratique sur mininet il est important de savoir qu’il faut toujours l’exécuter en tant qu’administrateur avec la commande sudo

**1) Il y a 1 switch (s1) de disponible dans mon réseau virtuel ainsi que 2 clients (h1, h2) :**

mininet> nodes

available nodes are:

c0 h1 h2 s1

**2) La commande dump permet de données les informations liées à chaque nœud :**

mininet> dump

<Host h1: h1-eth0:10.0.0.1 pid=1606>

<Host h2: h2-eth0:10.0.0.2 pid=1610>

<OVSSwitch s1: lo:127.0.0.1,s1-eth1:None,s1-eth2:None pid=1615>

<Controller c0: 127.0.0.1:6653 pid=1599>

Ces informations sont les **adresses IP** de chaque machine ainsi que le **nom de la carte réseau** qui leur sont associés

**3) Je dessine la topologie réseau actuelle :**

mininet> net

h1 h1-eth0:s1-eth1

h2 h2-eth0:s1-eth2

s1 lo: s1-eth1:h1-eth0 s1-eth2:h2-eth0

c0

Mon client h1 est lié à mon switch sur la carte eth1

Mon client h2 est lié à mon switch sur la carte eth2

Le réseau est donc le suivant :

Contrôleur 0

ETH2

ETH0

Hôte 2

Switch1

Hôte 1

ETH0

ETH1

**4) J’ajoute à mon schéma réseau les IP de chaque client ainsi que leur masque de sous-réseau :**

Le réseau est donc le suivant :

Contrôleur 0

ETH2

ETH0

Hôte 2

Switch1

Hôte 1

IP : 10.0.0.2

Masque : 255.0.0.0

ETH0

ETH1

IP :10.0.0.1

Masque : 255.0.0.0

**5) Le client H2 peut être atteint par le client H1 selon le schéma réseau vu juste au-dessus. Je peux essayer de la prouver en faisant un ping :**

mininet> h1 ping h2

PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.

64 bytes from 10.0.0.2: icmp\_seq=1 ttl=64 time=6.29 ms

64 bytes from 10.0.0.2: icmp\_seq=2 ttl=64 time=0.525 ms

64 bytes from 10.0.0.2: icmp\_seq=3 ttl=64 time=0.120 ms

64 bytes from 10.0.0.2: icmp\_seq=4 ttl=64 time=0.105 ms

^C

--- 10.0.0.2 ping statistics ---

4 packets transmitted, 4 received, 0% packet loss, time 3050ms

rtt min/avg/max/mdev = 0.105/1.759/6.289/2.620 ms

Le client H2 est bien accessible par le client H1 car ces 2 machines se situent dans le même réseau (10.0.0.0/8)

**/ ! \ : Pour sortir de mininet je peux utiliser la commande « exit »**

**Si lors d’un test j’obtiens une erreur et que la topologie ne s’est pas créé correctement il faut nettoyer la topologie avec la commande « sudo mn -c »**

**Information :**

**Pour éviter d’indiquer à chaque fois qu’elle machine doit exécuter quelle commande, et si vous voulez travailler avec une console proche decelle des machines réelles, vous pouvez 1.Exécuter Mininet avec l’option « -x »2.Exécuter la commande xterm et indiquer le client à attacher. mininet> xterm h1La première option ouvrira automatiquement une fenêtre xterm pour chaque client, switch et contrôleur présent dans la topologie.**

# 4. D’autres topologies réseaux

Mininet fournit, en plus de la topologie « minimal », la topologie **« single », « linear » et « tree ».** Pour charger l’une de ces topologies, utilisez l’option « --topo »

Par exemple : $ sudo mn --topo single

**6) J’exécute les commandes suivantes et je dessine les topologies réseaux qui sont créés :**

**sudo mn --topo linear**

mininet> net

h1 h1-eth0:s1-eth1

h2 h2-eth0:s2-eth1

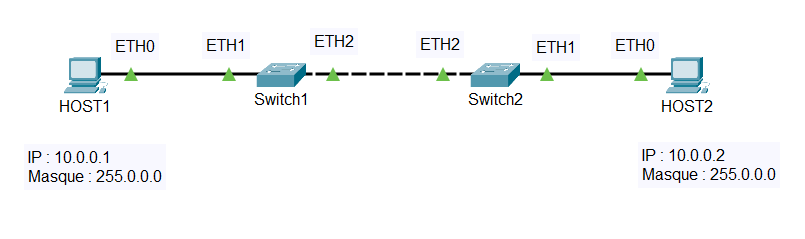
s1 lo: s1-eth1:h1-eth0 s1-eth2:s2-eth2

s2 lo: s2-eth1:h2-eth0 s2-eth2:s1-eth2

c0

Schéma de la topologie réseau :

Contrôleur 0



**sudo mn --topo linear,3**

mininet> net

h1 h1-eth0:s1-eth1

h2 h2-eth0:s2-eth1

h3 h3-eth0:s3-eth1

s1 lo: s1-eth1:h1-eth0 s1-eth2:s2-eth2

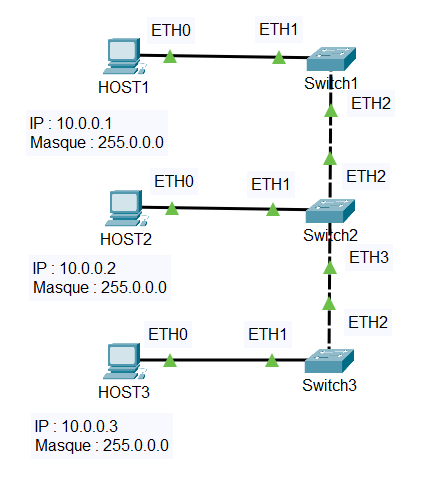
s2 lo: s2-eth1:h2-eth0 s2-eth2:s1-eth2 s2-eth3:s3-eth2

s3 lo: s3-eth1:h3-eth0 s3-eth2:s2-eth3

c0

Schéma de la topologie réseau :

Contrôleur 0



**sudo mn --topo tree**

mininet> net

h1 h1-eth0:s1-eth1

h2 h2-eth0:s1-eth2

s1 lo: s1-eth1:h1-eth0 s1-eth2:h2-eth0

c0

Schéma de la topologie réseau :

Contrôleur 0

ETH2

ETH0

Hôte 2

Switch1

Hôte 1

IP : 10.0.0.2

Masque : 255.0.0.0

ETH0

ETH1

IP :10.0.0.1

Masque : 255.0.0.0

**sudo mn --topo tree,2**

mininet> net

h1 h1-eth0:s2-eth1

h2 h2-eth0:s2-eth2

h3 h3-eth0:s3-eth1

h4 h4-eth0:s3-eth2

s1 lo: s1-eth1:s2-eth3 s1-eth2:s3-eth3

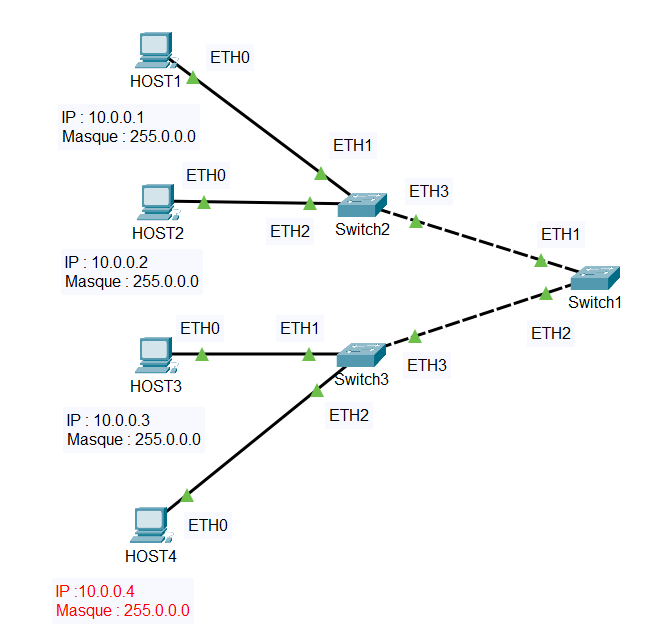
s2 lo: s2-eth1:h1-eth0 s2-eth2:h2-eth0 s2-eth3:s1-eth1

s3 lo: s3-eth1:h3-eth0 s3-eth2:h4-eth0 s3-eth3:s1-eth2

c0

Schéma de la topologie réseau :

Contrôleur 0



**sudo mn --topo tree,2,fanout=3.**

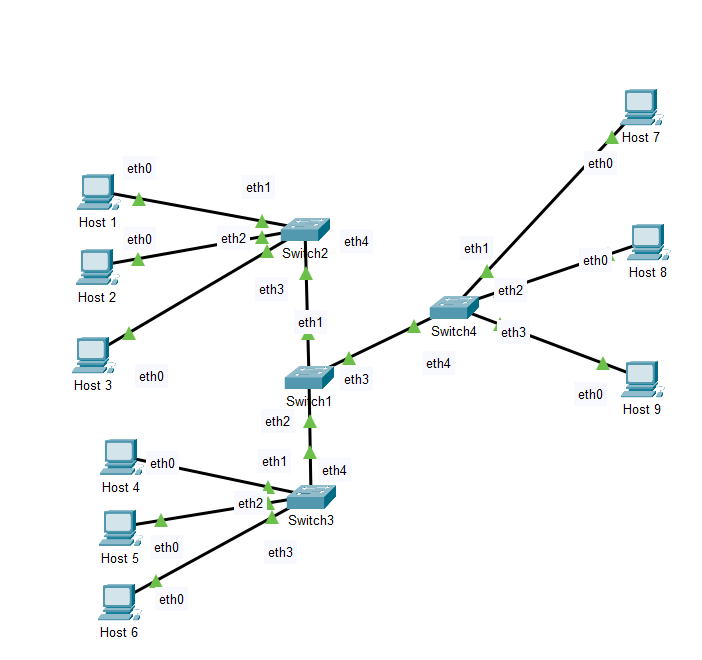
À quoi servent les paramètres 2 et fanout=3 ?

La paramètre Fanout 3 créer 3 hôtes par switch

Le paramètre 2 permet d’ajouter de la profondeur à la topologie (des switchs supplémentaires)

2 nouveaux switchs en plus de celui crée à la base

Contrôleur 0



# 5. Pour aller encore plus loin

Je commence par écrire un petit script (toptest.py) en python qui est le suivant :

from mininet.topo import Topo

class Test\_Topo(Topo):

  def \_\_init\_\_(self):

    "Create P2P topology"

    # Initialiser la topologie

    Topo.\_\_init\_\_(self)

    # Ajouter les hôtes et les switchs

    h1 = self.addNode('h1')

    h2 = self.addNode('h2')

    s1 = self.addSwitch('s1')

    # Ajouter les liaisons

    self.addLink(h2, s1)

    self.addLink(h1, h2)

topos = {

          'toptest' : (lambda: Test\_Topo())

}

Je le sauvegarde et via l’outil WinSCP je copie le script dans la VM mininet dans le répertoire /home/mininet/mytopos/

mininet@mininet-vm:~$ mkdir /home/mininet/mytopos

mininet@mininet-vm:~$ ls /home/mininet/mytopos/

toptest.py

**7) Je peux maintenant exécuter le script afin de créer une topologie réseau custom :**

mininet@mininet-vm:~$ sudo mn --custom /home/mininet/mytopos/toptest.py --topo toptest

\*\*\* Creating network

\*\*\* Adding controller

... ... ... ...

Je regarde la configuration réseau de chaque hôte :

**Hôte 1 :**

mininet> h1 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: h1-eth0@if2: <BROADCAST,MULTICAST,UP,LOWER\_UP> mtu 1500 qdisc noqueue state UP group default qlen 1000

link/ether 9e:41:c7:c3:c6:25 brd ff:ff:ff:ff:ff:ff link-netnsid 0

inet 10.0.0.1/8 brd 10.255.255.255 scope global h1-eth0

valid\_lft forever preferred\_lft forever

h1 n’a que 2 cartes réseaux (la loopback et la eth0 lié à l’hôte h2)

**Hôte 2 :**

mininet> h2 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: h2-eth1@if2: <BROADCAST,MULTICAST,UP,LOWER\_UP> mtu 1500 qdisc noqueue state UP group default qlen 1000

link/ether ca:4b:bd:44:c3:f0 brd ff:ff:ff:ff:ff:ff link-netnsid 0

3: h2-eth0@if67: <BROADCAST,MULTICAST,UP,LOWER\_UP> mtu 1500 qdisc noqueue state UP group default qlen 1000

link/ether 6a:fd:79:cd:1c:9e brd ff:ff:ff:ff:ff:ff link-netnsid 1

inet 10.0.0.2/8 brd 10.255.255.255 scope global h2-eth0

valid\_lft forever preferred\_lft forever

h2 a 3 cartes réseaux (1 loopback, 1 liée à h1 et 1 liée au switch)

J’attribue par le CLI des adresses IP aux 2 clients (192.168.0.1 et .2 en /24) :

**H1 :**

mininet> h1 ip a f dev h1-eth0

mininet> h1 ifconfig h1-eth0 192.168.0.1 netmask 255.255.255.0

2: h1-eth0@if2: <BROADCAST,MULTICAST,UP,LOWER\_UP> mtu 1500 qdisc noqueue state UP group default qlen 1000

link/ether 9e:41:c7:c3:c6:25 brd ff:ff:ff:ff:ff:ff link-netnsid 0

inet 192.168.0.1/24 scope global h1-eth0

valid\_lft forever preferred\_lft forever

**H2 :**

mininet> h2 ip a f dev h2-eth0

mininet> h2 ifconfig h2-eth0 192.168.0.2 netmask 255.255.255.0

La connectivité entre les 2 clients ne se fait pas car des machines dans un réseau SDN doivent obligatoirement passer par un switch qui utilise le protocole OpenFlow. Le lien direct entre 2 machines ne peut pas fonctionner dans ce type de réseau.

Je dois donc configurer une IP sur chaque interface entre le client et le switch

h1 --------s1 --------s2 --------s3--------h2

Contenu du script trois\_sw.py :

from mininet.topo import Topo

class Test\_Topo(Topo):

  def \_\_init\_\_(self):

    "Create P2P topology"

    # Initialiser la topologie

    Topo.\_\_init\_\_(self)

    # Ajouter les hôtes et les switchs

    h1 = self.addNode('h1')

    h2 = self.addNode('h2')

    s1 = self.addSwitch('s1')

    s2 = self.addSwitch('s2')

    s3 = self.addSwitch('s3')

    # Ajouter les liaisons

    self.addLink(h1, s1)

    self.addLink(s1, s2)

    self.addLink(s2, s3)

    self.addLink(s3, h2)

topos = {

          'toptest' : (lambda: Test\_Topo())

}

Lancement de la topologie mininet :

mininet@mininet-vm:~/trois$ sudo mn --custom /home/mininet/trois/troissw.py --topo toptest

Vérification de la connectivité avec un ping :

mininet> net

h1 h1-eth0:s1-eth1

h2 h2-eth0:s3-eth2

s1 lo: s1-eth1:h1-eth0 s1-eth2:s2-eth1

s2 lo: s2-eth1:s1-eth2 s2-eth2:s3-eth1

s3 lo: s3-eth1:s2-eth2 s3-eth2:h2-eth0

c0

mininet> h1 ping h2

PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.

64 bytes from 10.0.0.2: icmp\_seq=1 ttl=64 time=4.49 ms

mininet> h2 ping h1

PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data.

64 bytes from 10.0.0.1: icmp\_seq=1 ttl=64 time=5.32 ms

# 6. Et c’est presque la fin

J’améliore maintenant le script (toptest.py) pour éviter de devoir attribuer des IP à la main :

from mininet.net import Mininet

from mininet.topo import Topo

from mininet.node import Node

from mininet.cli import CLI

from mininet.link import TCLink

from mininet.log import setLogLevel

class Test\_Topo(Topo):

  def \_\_init\_\_(self):

    "Create P2P topology"

    # Initialiser la topologie

    Topo.\_\_init\_\_(self)

    # Ajouter les hôtes et les switchs

    h1 = self.addNode('h1')

    h2 = self.addNode('h2')

    s1 = self.addSwitch('s1')

    # Ajouter les liaisons

    self.addLink(h2, s1)

    self.addLink(h1, h2)

topos = {

          'toptest' : (lambda: Test\_Topo())

}

def topTest():

  topo = Test\_Topo()

  net = Mininet(topo=topo, link=TCLink)

  net.start()

  h1 = net.get('h1')

  h2 = net.get('h2')

  h1int0 = h1.intf('h1-eth0')

  h2int1 = h2.intf('h2-eth1')

  h1.setIP('11.0.0.1', 8, h1int0)

  h2.setIP('11.0.0.3', 8, h2int1)

  CLI(net) # Sans cette ligne, on ne verrait jamais le CLI

  net.stop() # Ne pas oublier de detruire le reseau

if \_\_name\_\_ == '\_\_main\_\_':

    setLogLevel('info')

    topTest()

**8) Partie sur le script toptest.py :**

**Je test un ping entre h1 et h2. Il ne fonctionne pas pour la même raison que précédemment.**

Il faut que mon client h1 ai une liaison avec le switch pour que les 2 clients puissent communiquer

**Résolution du problème en CLI :**

Je supprime la liaison entre mes 2 clients (qui ne sert pas) :

mininet> py net.delLinkBetween(h1,h2)

[<mininet.link.TCLink object at 0x7fd0e10a5c70>]

Je crée une nouvelle liaison entre mon client h1 et mon switch s1 :

mininet> py net.addLink(h1,s1)

<mininet.link.TCLink object at 0x7f5fa1126f40>

J’attribue une IP et un masque et la nouvelle interface liée au switch :

h1 ifconfig h1-eth0 10.0.0.1 netmask 255.0.0.0

Le ping devrait être maintenant fonctionnel mais le mininet ne prend pas en compte l’ajout de l’adresse IP

**Modification à apporter au script pour que ce soit fonctionnel dès le début :**

from mininet.net import Mininet

from mininet.topo import Topo

from mininet.node import Node

from mininet.cli import CLI

from mininet.link import TCLink

from mininet.log import setLogLevel

class Test\_Topo(Topo):

  def \_\_init\_\_(self):

    "Create P2P topology"

    # Initialiser la topologie

    Topo.\_\_init\_\_(self)

    # Ajouter les hôtes et les switchs

    h1 = self.addNode('h1')

    h2 = self.addNode('h2')

    s1 = self.addSwitch('s1')

    # Ajouter les liaisons

    self.addLink(h2, s1)

    self.addLink(h1, s1)

topos = {

          'toptest' : (lambda: Test\_Topo())

}

def topTest():

  topo = Test\_Topo()

  net = Mininet(topo=topo, link=TCLink)

  net.start()

  h1 = net.get('h1')

  h2 = net.get('h2')

  h1int0 = h1.intf('h1-eth0')

  h2int0 = h2.intf('h2-eth0')

  h1.setIP('11.0.0.1', 8, h1int0)

  h2.setIP('11.0.0.3', 8, h2int0)

  CLI(net) # Sans cette ligne, on ne verrait jamais le CLI

  net.stop() # Ne pas oublier de detruire le reseau

if \_\_name\_\_ == '\_\_main\_\_':

    setLogLevel('info')

    topTest()

Maintenant je peux relancer le script et vérifier que le ping fonctionne bien entre h1 et h2 :

mininet> h1 ping h2

PING 11.0.0.3 (11.0.0.3) 56(84) bytes of data.

64 bytes from 11.0.0.3: icmp\_seq=1 ttl=64 time=1.78 ms

...

mininet> h2 ping h1

PING 11.0.0.1 (11.0.0.1) 56(84) bytes of data.

64 bytes from 11.0.0.1: icmp\_seq=1 ttl=64 time=1.01 ms

...

**Partie sur le script trois\_sw.py :**

Modification du script pour donner des IP automatiquement au lancement de la topologie :

from mininet.net import Mininet

from mininet.topo import Topo

from mininet.node import Node

from mininet.cli import CLI

from mininet.link import TCLink

from mininet.log import setLogLevel

class Test\_Topo(Topo):

  def \_\_init\_\_(self):

    "Create P2P topology"

    # Initialiser la topologie

    Topo.\_\_init\_\_(self)

    # Ajouter les hôtes et les switchs

    h1 = self.addNode('h1')

    h2 = self.addNode('h2')

    s1 = self.addSwitch('s1')

    s2 = self.addSwitch('s2')

    s3 = self.addSwitch('s3')

    # Ajouter les liaisons

    self.addLink(h1, s1)

    self.addLink(s1, s2)

    self.addLink(s2, s3)

    self.addLink(s3, h2)

topos = {

          'toptest' : (lambda: Test\_Topo())

}

def topTest():

  topo = Test\_Topo()

  net = Mininet(topo=topo, link=TCLink)

  net.start()

  h1 = net.get('h1')

  h2 = net.get('h2')

  h1int0 = h1.intf('h1-eth0')

  h2int0 = h2.intf('h2-eth0')

  h1.setIP('192.168.1.1', 24, h1int0)

  h2.setIP('192.168.1.2', 24, h2int0)

  net.pingAll() # Test d’un ping a la creation du reseau

  CLI(net) # Sans cette ligne, on ne verrait jamais le CLI

  net.stop() # Ne pas oublier de detruire le reseau

if \_\_name\_\_ == '\_\_main\_\_':

    setLogLevel('info')

    topTest()

Création du réseau :

mininet@mininet-vm:~/trois$ sudo python /home/mininet/trois/trois\_sw.py

Vérification de la connectivité des clients via le ping dans l’API python :

\*\*\* Ping: testing ping reachability

h1 -> h2

h2 -> h1

\*\*\* Results: 0% dropped (2/2 received)

\*\*\* Starting CLI:

Les clients arrivent bien à se ping