TP numéro 7
16/02/2023
Antoine Lounis

- **Combien de switches et clients sont disponibles dans votre réseau virtuel ?**

```
mininet> nodes
available nodes are:
c0 h1 h2 s1
mininet> dump
<Host h1: h1-eth0:10.0.0.1 pid=795>
<Host h2: h2-eth0:10.0.0.2 pid=798>
<OVSSwitch s1: lo:127.0.0.1,s1-eth1:None,s1-eth2:None pid=803>
<Controller c0: 127.0.0.1:6653 pid=788>
mininet>
```

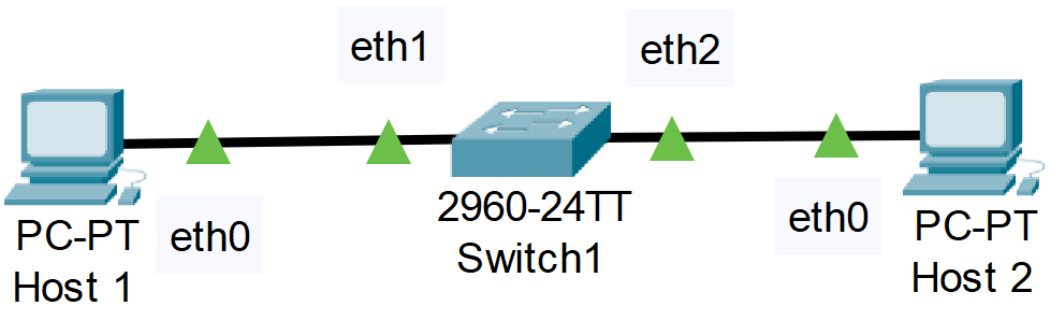
Un seul swtich (s1) est présent dans le réseau virtuel

- **Décrivez ce que vous obtenez comme information des switches et clients du réseau.**

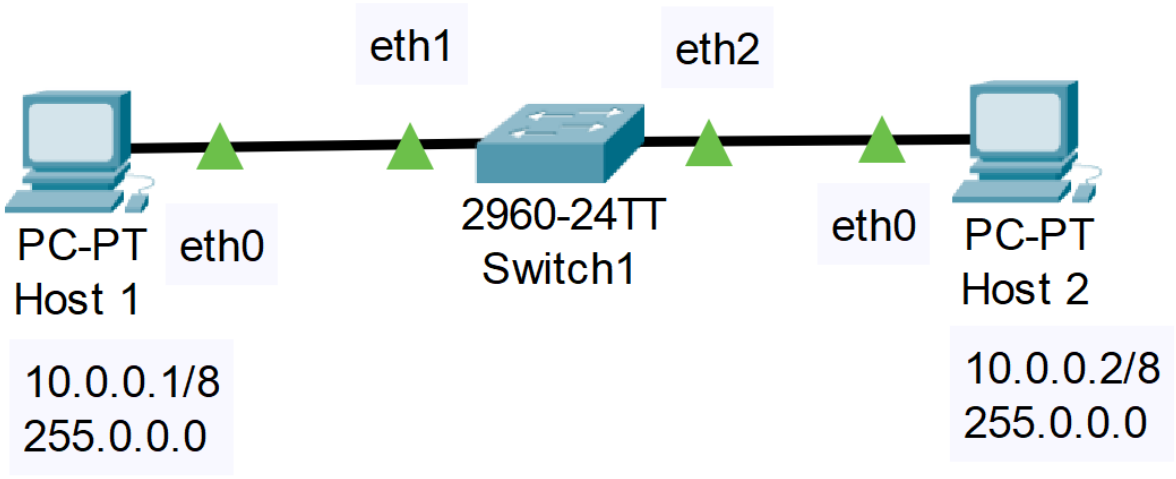
```
mininet> net
h1 h1-eth0:s1-eth1
h2 h2-eth0:s1-eth2
s1 lo:  s1-eth1:h1-eth0 s1-eth2:h2-eth0
c0
mininet>
```

L'host1 est relié depuis sont eth0 au port eth1 du switch et l'host 2 est relié depuis son eth0 au port eth2 du switch

- **Dessinez la topologie créée.**



- **Ajoutez à votre dessin de la topologie réseau les adresses IP associées à chaque client, ainsi que le masque de réseau**



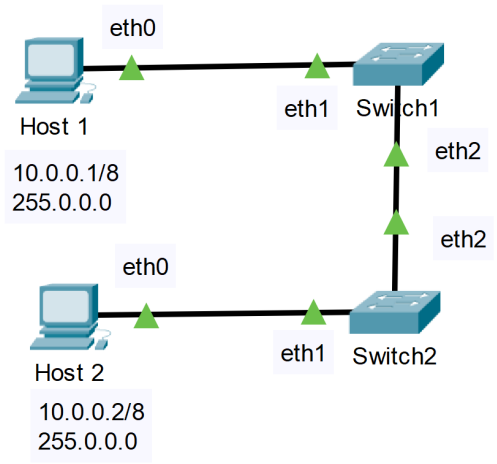
- **Le client h2 peut être atteint par le client h1 ? Vérifiez-le par la commande ping. Donnez l'instruction complète que vous avez utilisée**

Oui h1 et h2 peuvent se ping

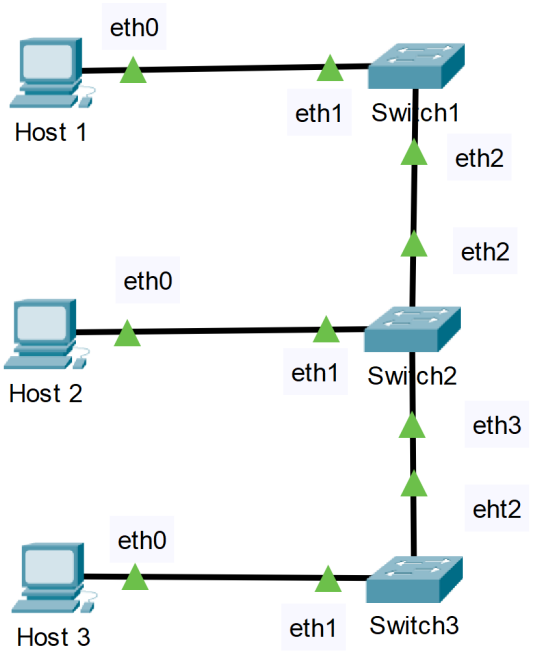
Ping h1 vers h2 > h1 ping h2
Ping h2 vers h1 > h2 ping h1

```
mininet> h1 ping h2 -c 4
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=1.63 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.163 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.042 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.036 ms

--- 10.0.0.2 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3029ms
rtt min/avg/max/mdev = 0.036/0.468/1.632/0.673 ms
mininet> h2 ping h1 -c 4
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=3.15 ms
64 bytes from 10.0.0.1: icmp_seq=2 ttl=64 time=0.039 ms
64 bytes from 10.0.0.1: icmp_seq=3 ttl=64 time=0.035 ms
64 bytes from 10.0.0.1: icmp_seq=4 ttl=64 time=0.037 ms

--- 10.0.0.1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3053ms
rtt min/avg/max/mdev = 0.035/0.815/3.151/1.348 ms
mininet>
```

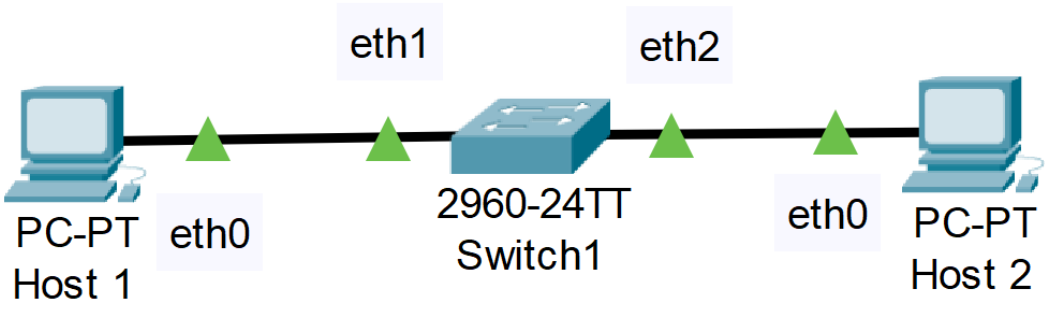
- **Exécutez les commandes suivantes et dessinez la topologie créée :**
  a. **sudo mn --topo linear**



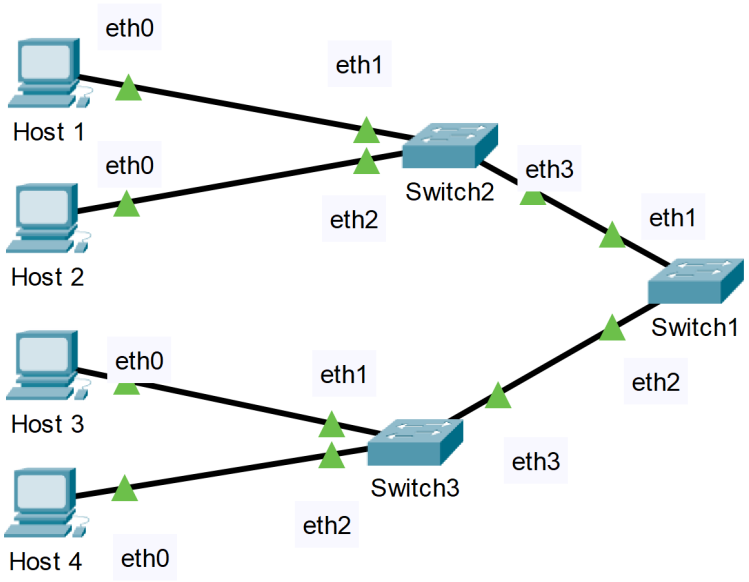
  b. **sudo mn --topo linear,3**



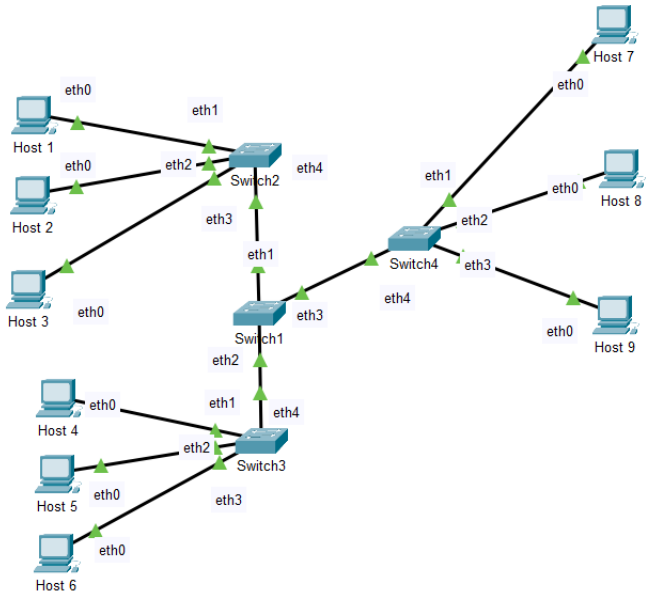
c. sudo mn --topo tree



d. sudo mn --topo tree,2



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



f. **À quoi servent les paramètres 2 et fanout=3 ?**

Le paramètre 2 sert à ajouter un niveau de profondeur les hosts sont reliés à des switchs et ces switchs sont reliés au switch 1.
Fanout 3 ajoute des hosts ainsi que 3 « branches » au tree.

- **Si le réseau est créé correctement ouvrir un terminal pour chaque client et explorez la configuration des cartes réseaux. Combien d'interface disposent h1 et h2 ? Quels sont les adresses assignées ? Configurez par le CLI l'interface réseau du client h1 et l'interface du client h2 qui est connectéà h1. Assignez l'adresse 192.168.0.1 et 192.168.0.2 avec un netmask 255.255.255.0.**
  Comment fait-on pour configurer une carte réseau en ligne de commande ? On utilise la commande ifconfig (voir http://www.tecmint.com/ifconfig-command-examples/) Vérifiez par la commande ping la connectivité entre les machines h1 et h2.

L'host h1 dispose d'une seule carte réseau

- eth0 : 10.0.0.1

L'host h2 dispose de deux cartes réseau

- eth0 :10.0.0.2
- eth1 : Pas d'adresse IP

Changer les adresses IP

py h1.setIP('192.168.0.1/24') ou
h1 ifconfig h1-eth0 192.168.0.1 netmask 255.255.255.0

h2 ifconfig h2-eth1 192.168.0.2 netmask 255.255.255.0

Les 2 réseaux ne communiquent pas entre eux car le protocole utilisé ne fait pas la résolution d'adresse mac. h1 ne peut pas atteindre h2, si on essaye de ping h1 depuis h2, h2 s'auto ping.

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

trois_sw.py :

```python
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())
}
```

```
mininet> h1 ping h2 -c 4
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=2.55 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=3.28 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.170 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.037 ms

--- 10.0.0.2 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3014ms
rtt min/avg/max/mdev = 0.037/1.509/3.281/1.430 ms
mininet> h2 ping h1 -c 4
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=3.65 ms
64 bytes from 10.0.0.1: icmp_seq=2 ttl=64 time=0.042 ms
64 bytes from 10.0.0.1: icmp_seq=3 ttl=64 time=0.038 ms
64 bytes from 10.0.0.1: icmp_seq=4 ttl=64 time=0.037 ms

--- 10.0.0.1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3039ms
rtt min/avg/max/mdev = 0.037/0.940/3.646/1.561 ms
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
```

- **Exécutez le code présenté ci-dessus. Est-ce que le client h1 arrive à atteindre le client h2 ? Si ce n'est pas le cas, en passant par le CLI faites les modifications nécessaires pour que la commande ping réussisse. Enfin, si vous avez trouvé la solution, modifiez le code pour implémenter les commandes que vous avez exécutez par le CLI.**

Ajouter un lien entre h1 et s1

```
py net.addLink(h1, s1)
```

```
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())
}
```

Faite de même avec le code trois_sw.py : Ajoutez des adresses IP de type 192.168.1.X et exécutez un ping entre les 2 clients du réseau en passant par l'API Python.

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')
        h1 = self.addHost('h1', ip='192.168.1.10/24')
        h2 = self.addNode('h2')
        h2 = self.addHost('h2', ip='192.168.1.20/24')
        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())
}
```

```
mininet> h1 ping h2 -c 1
PING 192.168.1.20 (192.168.1.20) 56(84) bytes of data.
64 bytes from 192.168.1.20: icmp_seq=1 ttl=64 time=3.98 ms

--- 192.168.1.20 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 3.976/3.976/3.976/0.000 ms
mininet> h2 ping h1 -c 1
PING 192.168.1.10 (192.168.1.10) 56(84) bytes of data.
64 bytes from 192.168.1.10: icmp_seq=1 ttl=64 time=4.03 ms

--- 192.168.1.10 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 4.031/4.031/4.031/0.000 ms
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
```