Sara Soriano - 240007
Rubén Vera - 241456
Eneko Treviño - 241679

DOCUMENTACIÓN PRÁCTICA 3

- **Execute this topology with command "sudo python SingleSwitchTopo.py". What did this first script do? Match the output of the script with the source contents to understand it. Deliver the script with the practicum results.**

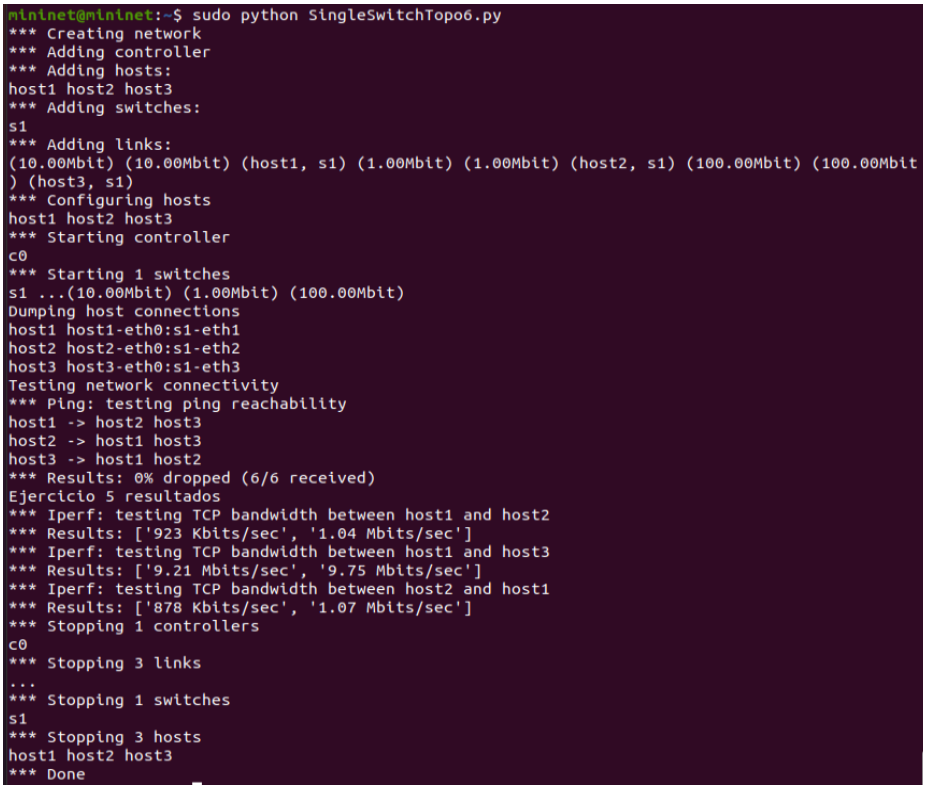
  Primeramente crea la network y añade los *hosts* h1, h2, h3 y h4 y añade el *switch* s1. Posteriormente, genera los *links* (hi, s1) y configura los *hosts*. Seguidamente, startea el *controller* y el *switch*. Ahora se realiza un *dump* de las conexiones, es decir, imprime información acerca de los nodos, *switches* y *controllers* de la *network* simulada. En este caso, muestra que puerto de cada host está conectado con que puerto del *switch* y se hace un "*ping"* desde cada *host* a todos los demás para comprobar la conectividad entre ellos. Una vez realizado todo, *Stopea* el *controller*, los *links* y los *hosts*. Todo esto lo podemos observar en la siguiente captura realizada:

```
mininet@mininet:~$ sudo python SingleSwitchTopo.py
[sudo] password for mininet:
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1) (h3, s1) (h4, s1)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
c0
*** Starting 1 switches
s1 ...
Dumping host connections
h1 h1-eth0:s1-eth1
h2 h2-eth0:s1-eth2
h3 h3-eth0:s1-eth3
h4 h4-eth0:s1-eth4
Testing network connectivity
*** Ping: testing ping reachability
h1 -> h2 h3 h4
h2 -> h1 h3 h4
h3 -> h1 h2 h4
h4 -> h1 h2 h3
*** Results: 0% dropped (12/12 received)
*** Stopping 1 controllers
c0
*** Stopping 4 links
....
*** Stopping 1 switches
s1
*** Stopping 4 hosts
h1 h2 h3 h4
*** Done
```

Sara Soriano - 240007
Rubén Vera - 241456
Eneko Treviño - 241679

- **Modify the sample file to have a fixed topology with three hosts, with the IP addresses shown in the first image, and linked to the switch with the indicated bandwidths. Make sure you did setup the links' bandwidth correctly by using the iPerf method of the Mininet object. You just need to pass two hosts to it to test the bandwidth of the link between them.**

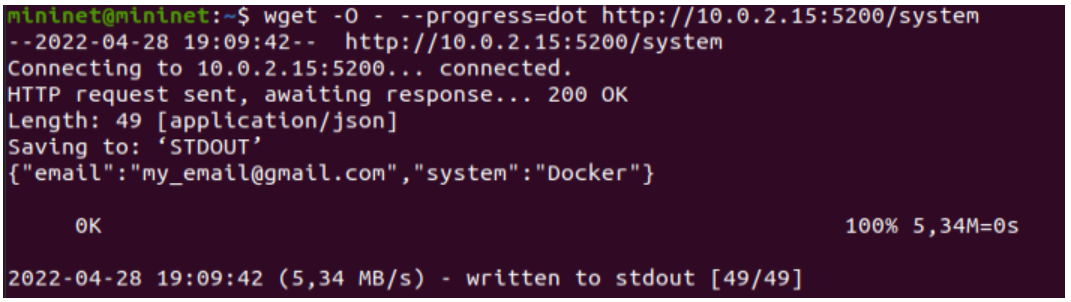
  Como se puede ver a través del archivo SingleSwitchTopo6.py, hemos creado los 3 hosts utilizando *self.addHost()* con los parámetros necesarios, entre ellos las direcciones IP con sus respectivas máscaras de red, y hemos creado los enlaces con *self.addLink()*, con los parámetros necesarios especificando el *bandwidth* de cada enlace. Además como se nos pedía en la función *simpleTest()* hemos analizado el ancho de banda de cada host y los resultados obtenidos confirman que hemos configurado bien, en la primera prueba usamos el host 2 que tiene un ancho de banda de 1Mbps, en el segundo caso usamos el host 1 y el 3, por lo tanto, el de mínimo ancho de banda es el primero con 10 Mbps. La última prueba es entre 2 y 1 otra vez para acabar de confirmar que la configuración era la correcta. Vemos que en los 3 casos el ancho de banda resultante es coherente con lo que debería según nuestra configuración. La configuración del ancho de banda del tercer *host* no la podíamos comprobar del todo ya que ningún otro *host* soportaba 100Mbps, pero se ha configurado de la misma manera así que también es correcto.



Sara Soriano - 240007
Rubén Vera - 241456
Eneko Treviño - 241679

- **Try it: execute it on a terminal with “sudo python Flask1.py”, and invoke it from another one with “wget -O - --progress=dot http://10.0.2.15:5200/system” (if that is the IP address of your VM). Stop the Flask server with Ctrl+C.**

  Una vez descargado el archivo Flask1.py en la dirección correspondiente de nuestra VM, hemos ejecutado el comando que se nos pedía *“sudo python Flask1.py”* y en otro terminal el comando que vemos en la captura que se muestra a continuación. En ella vemos como el cliente se ha conectado con el servidor y por lo tanto el proceso ha salido correcto.



Sara Soriano - 240007
Rubén Vera - 241456
Eneko Treviño - 241679

- **Now look for the way to run programs inside the mininet hosts and use it to execute the server in host h1 and the wget client in host h2; remove the bandwidth checking code before. You probably need to run the server in background mode (append '&' at the end of the command) to avoid the script to stop at that point. To give some time for the server to start accepting connections, add a line with "sleep(1)" to you script, after starting the server in h1; you need to import that function from the time python module (*from time import sleep*).**

```
mininet@mininet:~$ sudo python SingleSwitchTopo11.py
*** Creating network
*** Adding controller
*** Adding hosts:
host1 host2 host3
*** Adding switches:
s1
*** Adding links:
(10.00Mbit) (10.00Mbit) (host1, s1) (1.00Mbit) (1.00Mbit) (host2, s1) (100.00Mbit) (100.00Mbi
) (host3, s1)
*** Configuring hosts
host1 host2 host3
*** Starting controller
c0
*** Starting 1 switches
s1 ...(10.00Mbit) (1.00Mbit) (100.00Mbit)
Dumping host connections
host1 host1-eth0:s1-eth1
host2 host2-eth0:s1-eth2
host3 host3-eth0:s1-eth3
Testing network connectivity
*** Ping: testing ping reachability
host1 -> host2 host3
host2 -> host1 host3
host3 -> host1 host2
*** Results: 0% dropped (6/6 received)
--2022-04-28 19:27:56--  http://192.168.1.1:5200/system
Connecting to 192.168.1.1:5200... connected.
HTTP request sent, awaiting response... 200 OK
Length: 49 [application/json]
Saving to: 'STDOUT'
{"email":"my_email@gmail.com","system":"Docker"}

     0K                                                       100% 2,35M=0s

2022-04-28 19:27:56 (2,35 MB/s) - written to stdout [49/49]


*** Stopping 1 controllers
c0
*** Stopping 3 links
...
*** Stopping 1 switches
s1
*** Stopping 3 hosts
host1 host2 host3
*** Done
```

- **Explain in detail your conclusions about these first features tested from the mininet framework.**

  Las conclusiones son que con Mininet de momento hemos podido comprobar que se puede realizar una emulación de una red perfectamente y podemos usar con diferentes funciones que nos proporcionan las librerías, las cuales usaríamos en una red real, por así llamarla. Por ejemplo, hemos podido ver como se puede realizar un "*ping*", o una conexión servidor-cliente que también afirma la correcta conexión entre los *hosts*. Además, hemos visto que hay muchas opciones para configurar como queramos el *host* o el tipo de enlace, como por ejemplo poner una IP concreta a los *hosts* o especificar el ancho de banda o el tipo de enlace.

Sara Soriano - 240007
Rubén Vera - 241456
Eneko Treviño - 241679

- **Starting from the last script you created in the first session, extend it so that it opens a CLI session.**

  Como se puede observar en la siguiente imagen hemos cogido el *script* de la anterior sesión y lo hemos extendido para poder ejecutar una CLI. Para ello hemos tenido que importar la librería de CLI y añadir la línea de código *CLI(net)*.

```
def simpleTest():
    "Create and test a simple network"
    topo = SingleSwitchTopo(n=4)
    net = Mininet(topo, link = TCLink)
    net.start()
    print( "Dumping host connections" )
    dumpNodeConnections(net.hosts)
    #print( "Testing network connectivity" )
    #net.pingAll()
    #h1 = net.get('host1')
    #h3 = net.get('host3')
    #h1.cmd('sudo python Flask1.py &')
    #sleep(1)
    #print(h3.cmd('wget -O - --progress=dot http://192.168.1.1:5200/system'))
    CLI(net)
    net.stop()
```

- **Execute your script, it should stop and prompt "mininet>". Type "help" for a quick view on the available commands. You can execute commands on any of the running hosts by typing the host name (h1, h2…) and the command to run.**

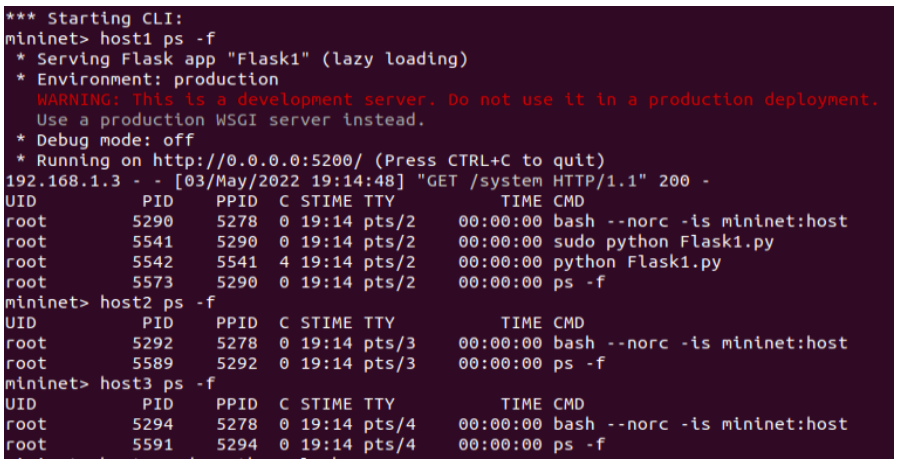
  En las siguientes imágenes se muestra como después de ejecutar nuestro *script* hemos mirado los distintos comandos disponibles y hemos probado a ejecutar algunos de ellos (en nuestro caso, "*ifconfig*" y "*ping*").

```
mininet> help

Documented commands (type help <topic>):
========================================
EOF    gterm  iperfudp  nodes        pingpair      py      switch
dpctl  help   link      noecho       pingpairfull  quit    time
dump   intfs  links     pingall      ports         sh      x
exit   iperf  net       pingallfull  px            source  xterm

You may also send a command to a node using:
  <node> command {args}
For example:
  mininet> h1 ifconfig

The interpreter automatically substitutes IP addresses
for node names when a node is the first arg, so commands
like
  mininet> h2 ping h3
should work.

Some character-oriented interactive commands require
noecho:
  mininet> noecho h2 vi foo.py
However, starting up an xterm/gterm is generally better:
  mininet> xterm h2
```

```
mininet> host1 ifconfig
host1-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.1.1  netmask 255.255.255.0  broadcast 192.168.1.255
        inet6 fe80::1468:7aff:fe00:9961  prefixlen 64  scopeid 0x20<link>
        ether 16:68:7a:00:99:61  txqueuelen 1000  (Ethernet)
        RX packets 52  bytes 5094 (5.0 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 16  bytes 1244 (1.2 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

mininet> host1 ping host2
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=64 time=20.5 ms
64 bytes from 192.168.1.2: icmp_seq=2 ttl=64 time=0.570 ms
^C
--- 192.168.1.2 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1002ms
```

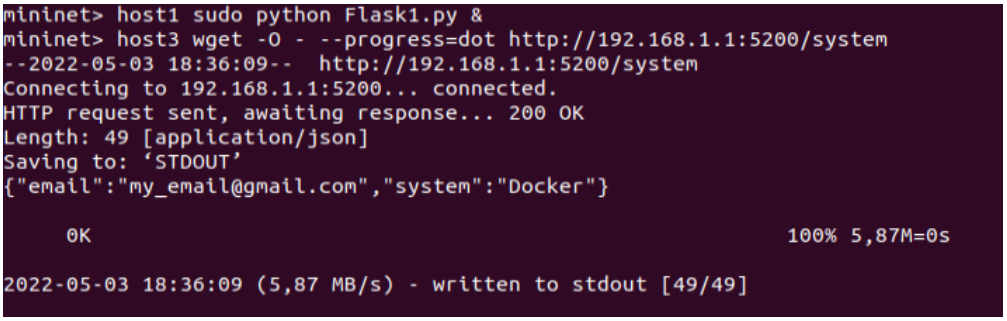
Sara Soriano - 240007
Rubén Vera - 241456
Eneko Treviño - 241679

- **Verify the running processes (“ps -f”) in all three hosts.**



De la misma manera que en el apartado anterior hemos realizado el “*ping*” y “*ifconfig*”, en este apartado hemos ejecutado el comando “*ps -f*” para ver los procesos activos de los tres hosts. Los resultados de dichos comandos son los que se muestran en la imagen anterior. En este caso podemos observar que como el host 1 está haciendo de servidor, en su lista de procesos se puede observar como hay 2 procesos adicionales a los otros 2 *hosts* que corresponden a los procesos de servidor.

- **Try to run now the wget test in the CLI from h2, accessing server that runs on h1, just as you did from the mininet script.**



Se puede observar cómo ejecutando el servidor en el host 1 y el cliente en el host 3, recibimos el *output* del “*wget*” confirmando que el cliente se ha conectado al servidor, la cual cosa confirma que nuestra CLI funciona correctamente. Lo hemos hecho con el host 3, pero con el host 2 el resultado sería el mismo.

Sara Soriano - 240007
Rubén Vera - 241456
Eneko Treviño - 241679

- **Now start the Flask http server also on h3, just as it is done from the mininet script. You should have the same server on h1 and h3. Verify with "wget" commands from h2, now using the IP addresses that were assigned to h1 and h3.**

```
*** Starting CLI:
mininet> host1 sudo python Flask1.py &
mininet> host3 sudo python Flask1.py &
mininet> host2 wget -O - --progress=dot http://192.168.1.1:5200/system
--2022-05-03 18:49:44--  http://192.168.1.1:5200/system
Connecting to 192.168.1.1:5200... connected.
HTTP request sent, awaiting response... 200 OK
Length: 49 [application/json]
Saving to: 'STDOUT'
{"email":"my_email@gmail.com","system":"Docker"}

     0K                                                       100% 3,07M=0s

2022-05-03 18:49:44 (3,07 MB/s) - written to stdout [49/49]

mininet> host2 wget -O - --progress=dot http://192.168.1.3:5200/system
--2022-05-03 18:49:53--  http://192.168.1.3:5200/system
Connecting to 192.168.1.3:5200... connected.
HTTP request sent, awaiting response... 200 OK
Length: 49 [application/json]
Saving to: 'STDOUT'
{"email":"my_email@gmail.com","system":"Docker"}

     0K                                                       100% 6,13M=0s

2022-05-03 18:49:53 (6,13 MB/s) - written to stdout [49/49]
```

En este caso hacemos que el host 1 y el host 3 hagan de servidores y el host 2 se conecta a ambos poniendo sus correspondientes *IPs* en el comando "*wget*". Además, vemos cómo en ambos casos el output confirma que la conexión se ha establecido correctamente.

- **Check the processes running in h3, again with "ps". You will probably see now one or two more processes, compared with the ones seen in step 3. And you probably see some unexpected output not related to the ps command. Can you give an explanation to both things?**

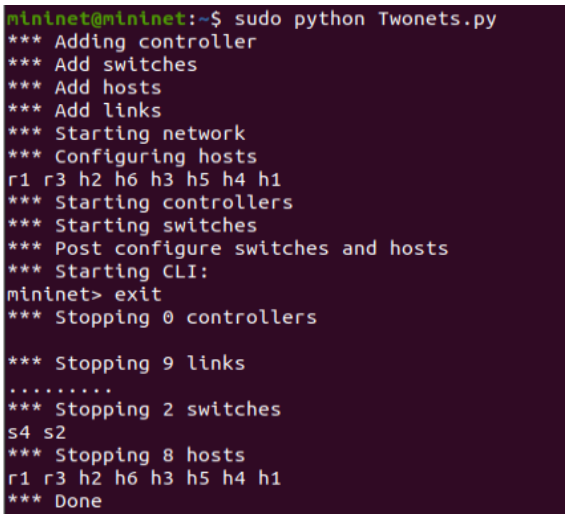
```
mininet> host3 ps -f
 * Serving Flask app "Flask1" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://0.0.0.0:5200/ (Press CTRL+C to quit)
192.168.1.2 - - [03/May/2022 18:49:53] "GET /system HTTP/1.1" 200 -
UID          PID    PPID  C STIME TTY          TIME CMD
root        3248    3232  0 18:48 pts/4    00:00:00 bash --norc -is mininet:host
root        3417    3248  0 18:49 pts/4    00:00:00 sudo python Flask1.py
root        3419    3417  0 18:49 pts/4    00:00:00 python Flask1.py
root        3472    3248  0 18:51 pts/4    00:00:00 ps -f
```

Comparando el resultado del apartado 3 con los resultados que hemos obtenido en este (los que se muestran en la imagen anterior) vemos que la única diferencia són los procesos que se generan por el hecho de estar

ejecutando el servidor. En función de cómo se haya escrito el código, encontraremos esos procesos adicionales que se mencionan en el enunciado, pero en nuestro caso no los hemos obtenido.

- **Save the topology as a level-2 script, and you will get a .py script you should be familiar with. Execute it as a regular mininet script, again you will see nothing very much interesting.**

  Después de haber guardado el *script* y de ejecutarlo nos encontramos con el siguiente resultado, donde se nos muestran los distintos componentes que forman la topología, tal y como se nos indicaba en el enunciado de la práctica.



- **Modify the script to add, before calling the CLI, the ping test you already saw in the BLA. Execute it. Does the test succeed? Execute from the CLI the "ifconfig" command on the router nodes. Does the router have an address assigned to the two network interfaces it has? Execute also the "netstat -rn" command to check the configured routes in the routers.**

  Una vez ejecutado el *script* para realizar un "*ping_all*" desde todos los hosts a todos los nodos de la topología, observamos que no todos los hosts reciben respuesta de todos los hosts. Esto es porque no están configuradas las distintas interfaces de red ni las rutas en los routers y, por lo tanto, cualquier paquete que llegue al router de un *ping* se perderá ahí porque no sabrá dónde enviarlo para que llegue a destino. Por lo cual, solo llegarán los pings a aquellos hosts que estén conectados a través de un mismo *switch,* que lo que hace es enviar todos los paquetes que le llegan hacia todo lo que tenga conectado. Por lo tanto, siguiendo la topología, lo esperado y lo que vemos en la captura que sucede, es que el host 1 solo podrá recibir respuesta del ping del host 2 y el host 3 y viceversa. En cambio, el host 4 sólo podrá

establecer comunicación con los hosts 5 y 6 que son los que están conectados al mismo switch que él.

```
mininet@mininet:~$ sudo python Twonets2.py
*** Adding controller
*** Add switches
*** Add hosts
*** Add links
*** Starting network
*** Configuring hosts
r1 r3 h2 h6 h3 h5 h4 h1
*** Starting controllers
*** Starting switches
*** Post configure switches and hosts
*** Ping: testing ping reachability
r1 -> r3 X X X X X X
r3 -> r1 X X X X X X
h2 -> r1 r3 X h3 X X h1
h6 -> r1 r3 X X h5 h4 X
h3 -> r1 r3 h2 X X X h1
h5 -> r1 r3 X h6 X h4 X
h4 -> r1 r3 X h6 X h5 X
h1 -> r1 r3 h2 X h3 X X
*** Results: 53% dropped (26/56 received)
*** Starting CLI:
```

A partir de ejecutar los comandos "*ifconfig*" y "*netstat -rn*" vemos que en el caso del "*ifconfig*", ninguno de los puertos de los routers están configurados y no tienen dirección IP. Esto lo sabemos ya que si tuviesen, la segunda línea debería ser la dirección IP, la máscara de red…. En el caso del estado de la red (*netstat*) se puede observar como la *Routing Table* está vacía y como se ha dicho antes, si le llega un paquete para una dirección IP en concreto, no sabe dónde enviarlo para hacerlo llegar a esa red.

```
mininet> r1 ifconfig
lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 2  bytes 168 (168.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 2  bytes 168 (168.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

r1-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet6 fe80::3044:46ff:fe40:263d  prefixlen 64  scopeid 0x20<link>
        ether 32:44:46:40:26:3d  txqueuelen 1000  (Ethernet)
        RX packets 81  bytes 6295 (6.2 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 9  bytes 726 (726.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

r1-eth1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet6 fe80::5c35:caff:fe21:4c18  prefixlen 64  scopeid 0x20<link>
        ether 5e:35:ca:21:4c:18  txqueuelen 1000  (Ethernet)
        RX packets 10  bytes 796 (796.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 10  bytes 796 (796.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Sara Soriano - 240007
Rubén Vera - 241456
Eneko Treviño - 241679

```
mininet> r3 ifconfig
lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 2  bytes 168 (168.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 2  bytes 168 (168.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

r3-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet6 fe80::9831:34ff:fe11:fad9  prefixlen 64  scopeid 0x20<link>
        ether 9a:31:34:11:fa:d9  txqueuelen 1000  (Ethernet)
        RX packets 10  bytes 796 (796.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 10  bytes 796 (796.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

r3-eth1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet6 fe80::8c23:e5ff:fe22:263f  prefixlen 64  scopeid 0x20<link>
        ether 8e:23:e5:22:26:3f  txqueuelen 1000  (Ethernet)
        RX packets 81  bytes 6275 (6.2 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 10  bytes 796 (796.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

```
mininet> r1 netstat -rn
Kernel IP routing table
Destination     Gateway         Genmask         Flags   MSS Window  irtt Iface
mininet> r3 netstat -rn
Kernel IP routing table
Destination     Gateway         Genmask         Flags   MSS Window  irtt Iface
```

- **Find out how to configure a network interface (you can browse the Internet for that…) and add in the python script, before the ping test call, the necessary commands to the router nodes; notice you will already find a first command "sysctl …" for the router, that was added by miniedit to enable IP routing to the router nodes. You can discriminate the interfaces (LAN vs. point-to-point) through the "links" mininet command. Now execute the script, does the ping test succeed? Run again from the CLI the "ifconfig" and "netstat -rn" commands on the router node. What do you get now?**

  Los resultados esperados de la prueba, del "*ping*", son los mismos que en el apartado anterior dado que todavía no se ha realizado la *Routing Table* y, por lo tanto, al llegarle un paquete al router para una dirección IP determinada, no sabe dónde enviarlo para que llegue a destino. Pero a diferencia del caso anterior, los puertos *Ethernet* ya están configurados con sus correspondientes IPs. En el caso del "*ifconfig*" vemos que en la segunda línea se puede observar ya la información del puerto, tanto la IP como la máscara… Y en el "*netstat -rn*" vemos que para llegar a las redes (en este caso 2, como su número de puertos) pertenecientes a la IP de sus puertos, sabe como llegar ya que el propio *router* es la *gateway* de los *hosts* que están conectados a él y si le llega una trama para su propia red sabe hacia dónde enviarlo.

Sara Soriano - 240007
Rubén Vera - 241456
Eneko Treviño - 241679

```
mininet@mininet:~$ sudo python Twonets3.py
*** Adding controller
*** Add switches
*** Add hosts
*** Add links
*** Starting network
*** Configuring hosts
r1 r3 h2 h6 h3 h5 h4 h1
*** Starting controllers
*** Starting switches
*** Post configure switches and hosts
*** Ping: testing ping reachability
r1 -> r3 h2 X h3 X X h1
r3 -> r1 X h6 X h5 h4 X
h2 -> r1 r3 X h3 X X h1
h6 -> r1 r3 X X h5 h4 X
h3 -> r1 r3 h2 X X X h1
h5 -> r1 r3 X h6 X h4 X
h4 -> r1 r3 X h6 X h5 X
h1 -> r1 r3 h2 X h3 X X
*** Results: 42% dropped (32/56 received)
*** Starting CLI:
```

```
mininet> r1 ifconfig
lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 2  bytes 168 (168.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 2  bytes 168 (168.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

r1-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.1.1  netmask 255.255.255.0  broadcast 192.168.1.255
        inet6 fe80::44e4:29ff:feed:30b2  prefixlen 64  scopeid 0x20<link>
        ether 46:e4:29:ed:30:b2  txqueuelen 1000  (Ethernet)
        RX packets 71  bytes 6407 (6.4 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 27  bytes 2182 (2.1 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

r1-eth1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.10.10.1  netmask 255.255.255.252  broadcast 10.10.10.3
        inet6 fe80::fc90:b6ff:fe2d:e2db  prefixlen 64  scopeid 0x20<link>
        ether fe:90:b6:2d:e2:db  txqueuelen 1000  (Ethernet)
        RX packets 9  bytes 726 (726.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 9  bytes 726 (726.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

```
mininet> r3 ifconfig
lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 2  bytes 168 (168.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 2  bytes 168 (168.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

r3-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.10.10.2  netmask 255.255.255.252  broadcast 10.10.10.3
        inet6 fe80::44a6:c6ff:fedb:d1f7  prefixlen 64  scopeid 0x20<link>
        ether 46:a6:c6:db:d1:f7  txqueuelen 1000  (Ethernet)
        RX packets 10  bytes 796 (796.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 9  bytes 726 (726.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

r3-eth1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.2.1  netmask 255.255.255.0  broadcast 192.168.2.255
        inet6 fe80::7491:feff:fe71:567b  prefixlen 64  scopeid 0x20<link>
        ether 76:91:fe:71:56:7b  txqueuelen 1000  (Ethernet)
        RX packets 72  bytes 6457 (6.4 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 28  bytes 2224 (2.2 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

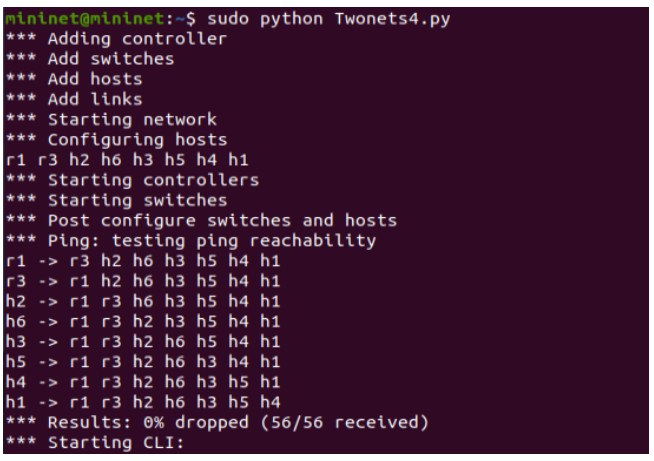
```
mininet> r1 netstat -rn
Kernel IP routing table
Destination     Gateway         Genmask         Flags   MSS Window  irtt Iface
10.10.10.0      0.0.0.0         255.255.255.252 U         0 0          0 r1-eth1
192.168.1.0     0.0.0.0         255.255.255.0   U         0 0          0 r1-eth0
mininet> r3 netstat -rn
Kernel IP routing table
Destination     Gateway         Genmask         Flags   MSS Window  irtt Iface
10.10.10.0      0.0.0.0         255.255.255.252 U         0 0          0 r3-eth0
192.168.2.0     0.0.0.0         255.255.255.0   U         0 0          0 r3-eth1
```

Sara Soriano - 240007
Rubén Vera - 241456
Eneko Treviño - 241679

- **Find out how to configure a route in an Ubuntu node and add in the python script, after the interface configuration code you added in previous point, the commands needed to create two routes that send traffic from each router to the opposite LAN via the other route. Now execute the script, does the ping test succeed? Run again from the CLI the "ifconfig" and "netstat -rn" commands on the router node. What do you get now, what are your conclusions?**

  Como ahora ya hemos configurado también los routers y cuál es el *next hop* de cada uno de ellos, la prueba que hemos estado realizando en estos tres apartados es exitosa, tal y como se muestra en la primera imagen. Además, mirando la información de la *Routing Table* vemos que también aparece la nueva información, es decir, ahora los routers saben dónde enviar los paquetes para que lleguen a la LAN opuesta. Esto se debe a que hemos usado el comando *ip route add <red/máscara> via <next hop> dev <puerto>*, donde le decimos que para llegar a la red en cuestión, que en este caso será la red de la LAN opuesta, ha de enviar el paquete al next hop, en este caso la IP del puerto *Ethernet* que conecta al router opuesto con el router que estemos configurando. Luego, el router opuesto tiene configurado como llegar a la red, ya que él es la gateway de esa red como se ha mencionado ya en el apartado anterior.

  Por la cual cosa, el viaje de un paquete ICMP en este caso será del host en cuestión al *switch*, que enviará el paquete a todo lo que tenga conectado, y uno de ellos será el router el cual, consultará si tiene información en la *Routing Table* de cómo llegar a la red de la dirección IP de destino del paquete. Como es así, envía el paquete al *next hop*, es decir, al router opuesto. Una vez llega el paquete al router opuesto, este envía el paquete al switch y este lo envía a todos, donde uno de esos irá a la IP que estaba destinado el paquete. Una vez lo recibe, envía un paquete de respuesta el cual hará el mismo camino pero inverso, y llegará ya que el router también tiene configurado como llegar a la red de donde venía el paquete.



Sara Soriano - 240007
Rubén Vera - 241456
Eneko Treviño - 241679

```
mininet> r1 ifconfig
lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 2  bytes 168 (168.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 2  bytes 168 (168.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

r1-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.1.1  netmask 255.255.255.0  broadcast 192.168.1.255
        inet6 fe80::a0d8:8fff:fe99:35ac  prefixlen 64  scopeid 0x20<link>
        ether a2:d8:8f:99:35:ac  txqueuelen 1000  (Ethernet)
        RX packets 56  bytes 5522 (5.5 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 33  bytes 2994 (2.9 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

r1-eth1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.10.10.1  netmask 255.255.255.252  broadcast 10.10.10.3
        inet6 fe80::b08b:b7ff:feb1:884b  prefixlen 64  scopeid 0x20<link>
        ether b2:8b:b7:b1:88:4b  txqueuelen 1000  (Ethernet)
        RX packets 31  bytes 2910 (2.9 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 31  bytes 2910 (2.9 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

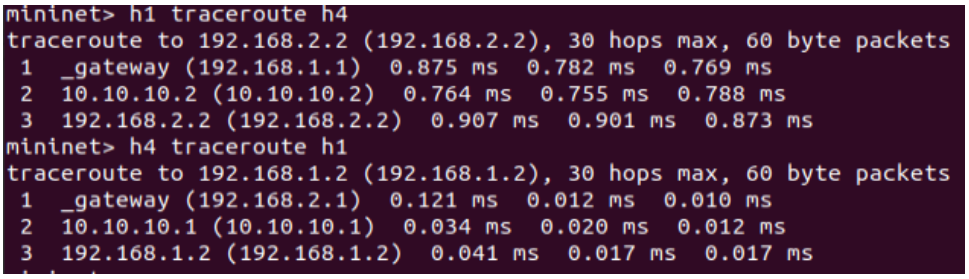
```
mininet> r3 ifconfig
lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 2  bytes 168 (168.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 2  bytes 168 (168.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

r3-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.10.10.2  netmask 255.255.255.252  broadcast 10.10.10.3
        inet6 fe80::8cce:1cff:fe5c:a000  prefixlen 64  scopeid 0x20<link>
        ether 8e:ce:1c:5c:a0:00  txqueuelen 1000  (Ethernet)
        RX packets 33  bytes 3022 (3.0 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 33  bytes 3022 (3.0 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

r3-eth1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.2.1  netmask 255.255.255.0  broadcast 192.168.2.255
        inet6 fe80::c8a2:d1ff:fe6a:d816  prefixlen 64  scopeid 0x20<link>
        ether ca:a2:d1:6a:d8:16  txqueuelen 1000  (Ethernet)
        RX packets 66  bytes 6301 (6.3 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 37  bytes 3190 (3.1 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

```
mininet> r1 netstat -rn
Kernel IP routing table
Destination     Gateway         Genmask         Flags   MSS Window  irtt Iface
10.10.10.0      0.0.0.0         255.255.255.252 U         0 0          0 r1-eth1
192.168.1.0     0.0.0.0         255.255.255.0   U         0 0          0 r1-eth0
192.168.2.0     10.10.10.2      255.255.255.0   UG        0 0          0 r1-eth1
mininet> r3 netstat -rn
Kernel IP routing table
Destination     Gateway         Genmask         Flags   MSS Window  irtt Iface
10.10.10.0      0.0.0.0         255.255.255.252 U         0 0          0 r3-eth0
192.168.1.0     10.10.10.1      255.255.255.0   UG        0 0          0 r3-eth0
192.168.2.0     0.0.0.0         255.255.255.0   U         0 0          0 r3-eth1
```

- **Run traceroute on one node in each network, to see the path the IP packets follow to get to a node on the other network, or to the router itself.**

  Una vez comprobado que traceroute está correctamente instalado, lo hemos ejecutado entre dos hosts de las diferentes LANs para comprobar que la conexión es correcta.

Sara Soriano - 240007
Rubén Vera - 241456
Eneko Treviño - 241679



A partir de la imagen anterior podemos ver el camino que siguen los paquetes enviados. En el primer caso vemos que el paquete llega al router correspondiente (r1). A continuación, se envía a la otra LAN, es decir, llega a r3. Este router se encarga de enviar el paquete al host receptor, en este caso a h4.

En el segundo caso, sigue el proceso contrario ya que se envía del h4 al h1. Por lo tanto, primero pasa por r3 y después por r1, donde este envía el paquete al destino final que es h1.

Esta conexión entre hosts de distintas LANs ha sido posible a través de configurar las interfaces de red y los routers, ya que si esto no se hubiese relacionado no se podrían comunicar. Los paquetes se perderían, como pasaba en un principio.