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

DOCUMENTACIÓN PRÁCTICA 4

- **Open the script with a text editor and try to understand what it does. Compare it with the first script you used in the Mininet first session. What are the main differences you can see?**

  Después de descargar el archivo en cuestión y compararlo con el *script* que realizamos en la anterior práctica, vemos las siguientes diferencias principales: la existencia de un controlador, cosa que no hemos visto en ninguna otra práctica, que el *switch* es inicializado como *Open Virtual Switch*, es decir, no es el típico *switch* "tonto" y se inicializan los *switches*, cosa que no habíamos hecho antes.

- **Look at the first three loops in the code, can you describe what they do? A fourth loop starts the switch or switches, which was not done in the first topologies we created. Can you imagine why this is needed now?**

  El primer *loop* crea los *sw switches* que le hemos indicado después de los imports y los crea como *OVS Switches*, es decir, *switches* del tipo *Open Virtual Switch*. El segundo bucle se encarga de añadir los *hosts,* según el número de *hosts* por *switch* que hemos establecido a la variable después de los imports, estos se crean. El tercero añade los enlaces entre los *hosts* y sus correspondientes *switches,* permitiendo una comunicación entre ellos. El cuarto *loop* inicializa los *switches*, tal y como dice el enunciado, se hace en este momento porque ahora los *switches* son del tipo OpenVirtual y necesitan inicialización.

- **Modify the script to have one switch and three hosts and execute it. What is the result of the ping tests, why? (VerySimpleNet4.py)**

  Para realizar este ejercicio solo hemos tenido que cambiar el valor de la variable *Nps* por el valor 3 que es el número de hosts por switch y como solo tenemos un switch pues tendremos 3 hosts, tal y como el enunciado nos pide. A partir de observar la siguiente imagen vemos que el *ping* no es exitoso dado que no hay un *controller* conectado que diga al switch si enviar la trama o no y hacia donde. De hecho, se puede observar en la captura como al intentar crear el controlador pone que no se ha podido conectar con el *controller* remoto.

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

```
mininet@mininet:~$ sudo python VerySimpleNet4.py
[sudo] password for mininet:
*** Creating controller, IP: 127.0.0.1
Unable to contact the remote controller at 127.0.0.1:6633
*** Creating switches
        *** Created Switch: s0
*** Creating nodes
        *** Created Node: h0
        *** Created Node: h1
        *** Created Node: h2
*** Creating links between host a Switch
        *** Creating links: s0-h0
        *** Creating links: s0-h1
        *** Creating links: s0-h2
*** Starting network
*** Configuring hosts
h0 h1 h2
*** Testing network
*** Ping: testing ping reachability
h0 -> X X
h1 -> X X
h2 -> X X
*** Results: 100% dropped (0/6 received)
*** Running CLI
*** Starting CLI:
mininet>
```

- **Execute again your modified VerySimpleNet.py script in a separate terminal and check if the ping between the hosts through the OVS is succeeding now. Stop the mininet script.**

  Después de haber instalado y ejecutado los comandos necesarios que se nos presentaban en el enunciado, hemos vuelto a ejecutar el *script*. Como en este caso el mininet se ha conectado al *controller* Ryu los *pings* sí que llegan al destino, es decir, se hace un *pingAlll()* de manera exitosa. Este resultado se puede observar a partir de la siguiente imagen. Además, adjuntamos una captura de la información que recibe el Ryu que son el *data path*, la MAC del *source*, la del destino y el puerto por el que va la trama.

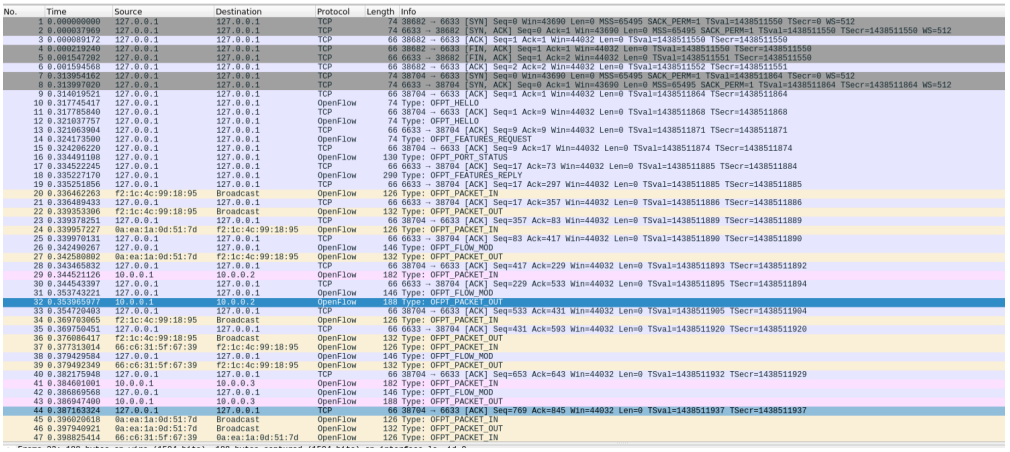
```
mininet@mininet:~/Software/ryu/ryu/app$ cd
mininet@mininet:~$ sudo python VerySimpleNet4.py
[sudo] password for mininet:
*** Creating controller, IP: 127.0.0.1
*** Creating switches
        *** Created Switch: s0
*** Creating nodes
        *** Created Node: h0
        *** Created Node: h1
        *** Created Node: h2
*** Creating links between host a Switch
        *** Creating links: s0-h0
        *** Creating links: s0-h1
        *** Creating links: s0-h2
*** Starting network
*** Configuring hosts
h0 h1 h2
*** Testing network
*** Ping: testing ping reachability
h0 -> h1 h2
h1 -> h0 h2
h2 -> h0 h1
*** Results: 0% dropped (6/6 received)
*** Running CLI
*** Starting CLI:
mininet>
```

```
mininet@mininet:~/Software/ryu/ryu/app$ ryu-manager --ofp-tcp-listen-port 6633 simple_switch.py
loading app simple_switch.py
loading app ryu.controller.ofp_handler
instantiating app simple_switch.py of SimpleSwitch
instantiating app ryu.controller.ofp_handler of OFPHandler
packet in 183031560365124 32:c0:14:9b:77:aa ff:ff:ff:ff:ff:ff 1
packet in 183031560365124 66:f3:f8:d7:c3:13 32:c0:14:9b:77:aa 2
packet in 183031560365124 32:c0:14:9b:77:aa 66:f3:f8:d7:c3:13 1
packet in 183031560365124 32:c0:14:9b:77:aa ff:ff:ff:ff:ff:ff 1
packet in 183031560365124 02:75:21:a0:65:2c 32:c0:14:9b:77:aa 3
```

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

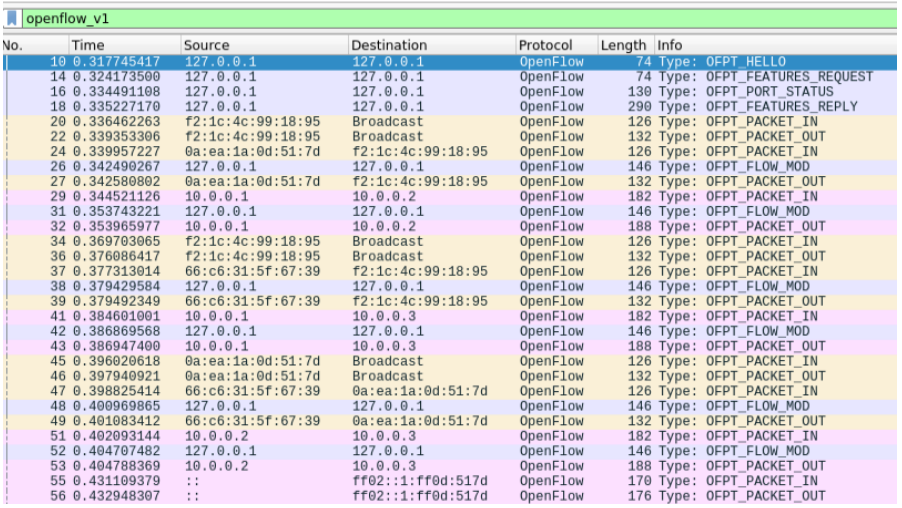
- **From a new terminal window, start Wireshark. Start capturing the traffic over the loopback interface and port 6633, execute again the Mininet script and analyse the packets that were captured. What type of OpenFlow messages are you able to find? Can you describe their mission?**

Hemos vuelto a ejecutar el *script* mientras Wireshark capturaba el tráfico de *loopback* a través del puerto 6633. En dicha aplicación, vemos el siguiente resultado.



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	127.0.0.1	127.0.0.1	TCP	74	38682 → 6633 [SYN] Seq=0 Win=43690 Len=0 MSS=65495 SACK_PERM=1 TSval=1438511550 TSecr=0 WS=512
2	0.000037969	127.0.0.1	127.0.0.1	TCP	74	6633 → 38682 [SYN, ACK] Seq=0 Ack=1 Win=43690 Len=0 MSS=65495 SACK_PERM=1 TSval=1438511550 TSecr=1438511550 WS=512
3	0.000089172	127.0.0.1	127.0.0.1	TCP	66	38682 → 6633 [ACK] Seq=1 Ack=1 Win=44032 Len=0 TSval=1438511550 TSecr=1438511550
4	0.000219240	127.0.0.1	127.0.0.1	TCP	66	38682 → 6633 [FIN, ACK] Seq=1 Ack=1 Win=44032 Len=0 TSval=1438511550 TSecr=1438511550
5	0.001547202	127.0.0.1	127.0.0.1	TCP	66	6633 → 38682 [FIN, ACK] Seq=1 Ack=2 Win=44032 Len=0 TSval=1438511551 TSecr=1438511550
6	0.001594568	127.0.0.1	127.0.0.1	TCP	66	38682 → 6633 [ACK] Seq=2 Ack=2 Win=44032 Len=0 TSval=1438511552 TSecr=1438511551
7	0.313954162	127.0.0.1	127.0.0.1	TCP	74	38704 → 6633 [SYN] Seq=0 Win=43690 Len=0 MSS=65495 SACK_PERM=1 TSval=1438511864 TSecr=0 WS=512
8	0.313997620	127.0.0.1	127.0.0.1	TCP	74	6633 → 38704 [SYN, ACK] Seq=0 Ack=1 Win=43690 Len=0 MSS=65495 SACK_PERM=1 TSval=1438511864 TSecr=1438511864 WS=512
9	0.314019521	127.0.0.1	127.0.0.1	TCP	66	38704 → 6633 [ACK] Seq=1 Ack=1 Win=44032 Len=0 TSval=1438511864 TSecr=1438511864
10	0.317745417	127.0.0.1	127.0.0.1	OpenFlow	74	Type: OFPT_HELLO
11	0.317785840	127.0.0.1	127.0.0.1	TCP	66	38704 → 6633 [ACK] Seq=1 Ack=9 Win=44032 Len=0 TSval=1438511868 TSecr=1438511868
12	0.321037757	127.0.0.1	127.0.0.1	OpenFlow	74	Type: OFPT_HELLO
13	0.321063904	127.0.0.1	127.0.0.1	TCP	66	6633 → 38704 [ACK] Seq=9 Ack=9 Win=44032 Len=0 TSval=1438511871 TSecr=1438511871
14	0.324173500	127.0.0.1	127.0.0.1	OpenFlow	74	Type: OFPT_FEATURES_REQUEST
15	0.324206220	127.0.0.1	127.0.0.1	TCP	66	38704 → 6633 [ACK] Seq=9 Ack=17 Win=44032 Len=0 TSval=1438511874 TSecr=1438511874
16	0.334491108	127.0.0.1	127.0.0.1	OpenFlow	130	Type: OFPT_PORT_STATUS
17	0.334522245	127.0.0.1	127.0.0.1	TCP	66	6633 → 38704 [ACK] Seq=17 Ack=73 Win=44032 Len=0 TSval=1438511885 TSecr=1438511884
18	0.335227170	127.0.0.1	127.0.0.1	OpenFlow	290	Type: OFPT_FEATURES_REPLY
19	0.335251856	127.0.0.1	127.0.0.1	TCP	66	6633 → 38704 [ACK] Seq=17 Ack=297 Win=44032 Len=0 TSval=1438511885 TSecr=1438511885
20	0.336462263	f2:1c:4c:99:18:95	Broadcast	OpenFlow	126	Type: OFPT_PACKET_IN
21	0.336489433	127.0.0.1	127.0.0.1	TCP	66	6633 → 38704 [ACK] Seq=17 Ack=357 Win=44032 Len=0 TSval=1438511886 TSecr=1438511886
22	0.339353306	f2:1c:4c:99:18:95	Broadcast	OpenFlow	132	Type: OFPT_PACKET_OUT
23	0.339378251	127.0.0.1	127.0.0.1	TCP	66	38704 → 6633 [ACK] Seq=357 Ack=83 Win=44032 Len=0 TSval=1438511889 TSecr=1438511889
24	0.339957227	0a:ea:1a:0d:51:7d	f2:1c:4c:99:18:95	OpenFlow	126	Type: OFPT_PACKET_IN
25	0.339970131	127.0.0.1	127.0.0.1	TCP	66	6633 → 38704 [ACK] Seq=83 Ack=417 Win=44032 Len=0 TSval=1438511890 TSecr=1438511890
26	0.342490267	127.0.0.1	127.0.0.1	OpenFlow	146	Type: OFPT_FLOW_MOD
27	0.342580802	0a:ea:1a:0d:51:7d	f2:1c:4c:99:18:95	OpenFlow	132	Type: OFPT_PACKET_OUT
28	0.343465832	127.0.0.1	127.0.0.1	TCP	66	38704 → 6633 [ACK] Seq=417 Ack=229 Win=44032 Len=0 TSval=1438511893 TSecr=1438511892
29	0.344521126	10.0.0.1	10.0.0.2	OpenFlow	182	Type: OFPT_PACKET_IN
30	0.344543397	127.0.0.1	127.0.0.1	TCP	66	6633 → 38704 [ACK] Seq=229 Ack=533 Win=44032 Len=0 TSval=1438511895 TSecr=1438511894
31	0.353743221	127.0.0.1	127.0.0.1	OpenFlow	146	Type: OFPT_FLOW_MOD
32	0.353965977	10.0.0.1	10.0.0.2	OpenFlow	188	Type: OFPT_PACKET_OUT
33	0.354720403	127.0.0.1	127.0.0.1	TCP	66	38704 → 6633 [ACK] Seq=533 Ack=431 Win=44032 Len=0 TSval=1438511905 TSecr=1438511904
34	0.369703065	f2:1c:4c:99:18:95	Broadcast	OpenFlow	126	Type: OFPT_PACKET_IN
35	0.369750451	127.0.0.1	127.0.0.1	TCP	66	6633 → 38704 [ACK] Seq=431 Ack=593 Win=44032 Len=0 TSval=1438511920 TSecr=1438511920
36	0.376086417	f2:1c:4c:99:18:95	Broadcast	OpenFlow	132	Type: OFPT_PACKET_OUT
37	0.377313014	66:c6:31:5f:67:39	f2:1c:4c:99:18:95	OpenFlow	126	Type: OFPT_PACKET_IN
38	0.379429584	127.0.0.1	127.0.0.1	OpenFlow	146	Type: OFPT_FLOW_MOD
39	0.379492349	66:c6:31:5f:67:39	f2:1c:4c:99:18:95	OpenFlow	132	Type: OFPT_PACKET_OUT
40	0.382175948	127.0.0.1	127.0.0.1	TCP	66	38704 → 6633 [ACK] Seq=653 Ack=643 Win=44032 Len=0 TSval=1438511932 TSecr=1438511929
41	0.384601001	10.0.0.1	10.0.0.3	OpenFlow	182	Type: OFPT_PACKET_IN
42	0.386869568	127.0.0.1	127.0.0.1	OpenFlow	146	Type: OFPT_FLOW_MOD
43	0.386947400	10.0.0.1	10.0.0.3	OpenFlow	188	Type: OFPT_PACKET_OUT
44	0.387163324	127.0.0.1	127.0.0.1	TCP	66	38704 → 6633 [ACK] Seq=769 Ack=845 Win=44032 Len=0 TSval=1438511937 TSecr=1438511937
45	0.396020618	0a:ea:1a:0d:51:7d	Broadcast	OpenFlow	126	Type: OFPT_PACKET_IN
46	0.397940921	0a:ea:1a:0d:51:7d	Broadcast	OpenFlow	132	Type: OFPT_PACKET_OUT
47	0.398825414	66:c6:31:5f:67:39	0a:ea:1a:0d:51:7d	OpenFlow	126	Type: OFPT_PACKET_IN

Para centrarnos en los ficheros de tipo OpenFlow hemos establecido el filtro para solo ver estos, en ellos vemos la estructura de comunicación habitual. Primeramente, un *OFTP_HELLO* por parte del *controller* y otro por parte del *vSwitch*. Seguidamente, una *features_request*, la *reply* por parte del *Switch*, el cual también envía una trama de *status*. Luego el *Switch* informa de que ha llegado un paquete y el *controller* informa al *Switch* de hacia dónde ha de ir y se lo devuelve.



openflow_v1

No.	Time	Source	Destination	Protocol	Length	Info
10	0.317745417	127.0.0.1	127.0.0.1	OpenFlow	74	Type: OFPT_HELLO
14	0.324173500	127.0.0.1	127.0.0.1	OpenFlow	74	Type: OFPT_FEATURES_REQUEST
16	0.334491108	127.0.0.1	127.0.0.1	OpenFlow	130	Type: OFPT_PORT_STATUS
18	0.335227170	127.0.0.1	127.0.0.1	OpenFlow	290	Type: OFPT_FEATURES_REPLY
20	0.336462263	f2:1c:4c:99:18:95	Broadcast	OpenFlow	126	Type: OFPT_PACKET_IN
22	0.339353306	f2:1c:4c:99:18:95	Broadcast	OpenFlow	132	Type: OFPT_PACKET_OUT
24	0.339957227	0a:ea:1a:0d:51:7d	f2:1c:4c:99:18:95	OpenFlow	126	Type: OFPT_PACKET_IN
26	0.342490267	127.0.0.1	127.0.0.1	OpenFlow	146	Type: OFPT_FLOW_MOD
27	0.342580802	0a:ea:1a:0d:51:7d	f2:1c:4c:99:18:95	OpenFlow	132	Type: OFPT_PACKET_OUT
29	0.344521126	10.0.0.1	10.0.0.2	OpenFlow	182	Type: OFPT_PACKET_IN
31	0.353743221	127.0.0.1	127.0.0.1	OpenFlow	146	Type: OFPT_FLOW_MOD
32	0.353965977	10.0.0.1	10.0.0.2	OpenFlow	188	Type: OFPT_PACKET_OUT
34	0.369703065	f2:1c:4c:99:18:95	Broadcast	OpenFlow	126	Type: OFPT_PACKET_IN
36	0.376086417	f2:1c:4c:99:18:95	Broadcast	OpenFlow	132	Type: OFPT_PACKET_OUT
37	0.377313014	66:c6:31:5f:67:39	f2:1c:4c:99:18:95	OpenFlow	126	Type: OFPT_PACKET_IN
38	0.379429584	127.0.0.1	127.0.0.1	OpenFlow	146	Type: OFPT_FLOW_MOD
39	0.379492349	66:c6:31:5f:67:39	f2:1c:4c:99:18:95	OpenFlow	132	Type: OFPT_PACKET_OUT
41	0.384601001	10.0.0.1	10.0.0.3	OpenFlow	182	Type: OFPT_PACKET_IN
42	0.386869568	127.0.0.1	127.0.0.1	OpenFlow	146	Type: OFPT_FLOW_MOD
43	0.386947400	10.0.0.1	10.0.0.3	OpenFlow	188	Type: OFPT_PACKET_OUT
45	0.396020618	0a:ea:1a:0d:51:7d	Broadcast	OpenFlow	126	Type: OFPT_PACKET_IN
46	0.397940921	0a:ea:1a:0d:51:7d	Broadcast	OpenFlow	132	Type: OFPT_PACKET_OUT
47	0.398825414	66:c6:31:5f:67:39	0a:ea:1a:0d:51:7d	OpenFlow	126	Type: OFPT_PACKET_IN
48	0.400969865	127.0.0.1	127.0.0.1	OpenFlow	146	Type: OFPT_FLOW_MOD
49	0.401083412	66:c6:31:5f:67:39	0a:ea:1a:0d:51:7d	OpenFlow	132	Type: OFPT_PACKET_OUT
51	0.402093144	10.0.0.2	10.0.0.3	OpenFlow	182	Type: OFPT_PACKET_IN
52	0.404707482	127.0.0.1	127.0.0.1	OpenFlow	146	Type: OFPT_FLOW_MOD
53	0.404788369	10.0.0.2	10.0.0.3	OpenFlow	188	Type: OFPT_PACKET_OUT
55	0.431109379	::	ff02::1:ff0d:517d	OpenFlow	170	Type: OFPT_PACKET_IN
56	0.432948307	::	ff02::1:ff0d:517d	OpenFlow	176	Type: OFPT_PACKET_OUT

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

- **The final challenge: derive a new version of the simple_switch.py controller, that blocks all traffic from/to the IP address of the second node; see how the Ethernet part of the message is handled to figure out how to access the IP source or destination address. Deliver it with the practicum results.**

Para tal de bloquear la dirección del segundo nodo (host 1), hemos tenido que bloquear las tramas IPv4 que envía y recibe dicho nodo. Para ello, hemos tenido que importar la librería de iPv4 de Ryu y sacar la dirección ip del nodo de destino y de origen de la trama. Si se envía desde host 1 o va enviado a este, lo que se hará es desechar el paquete que se envía. Por lo tanto, el resultado esperado es que host 1 no reciba ningún paquete ni que los paquetes que envía sean recibidos por el resto. Por lo cual, el resultado del *ping* debería ser que el host 0 solo pueda establecer conexión con el host 2, el host 1 con ningún host y el host 2 solo con el host 0.

```
mininet@mininet:~$ sudo python VerySimpleNet4.py
*** Creating controller, IP: 127.0.0.1
*** Creating switches
        *** Created Switch: s0
*** Creating nodes
        *** Created Node: h0
        *** Created Node: h1
        *** Created Node: h2
*** Creating links between host a Switch
        *** Creating links: s0-h0
        *** Creating links: s0-h1
        *** Creating links: s0-h2
*** Starting network
*** Configuring hosts
h0 h1 h2
*** Testing network
*** Ping: testing ping reachability
h0 -> X h2
h1 -> X X
h2 -> h0 X
*** Results: 66% dropped (2/6 received)
*** Running CLI
*** Starting CLI:
```

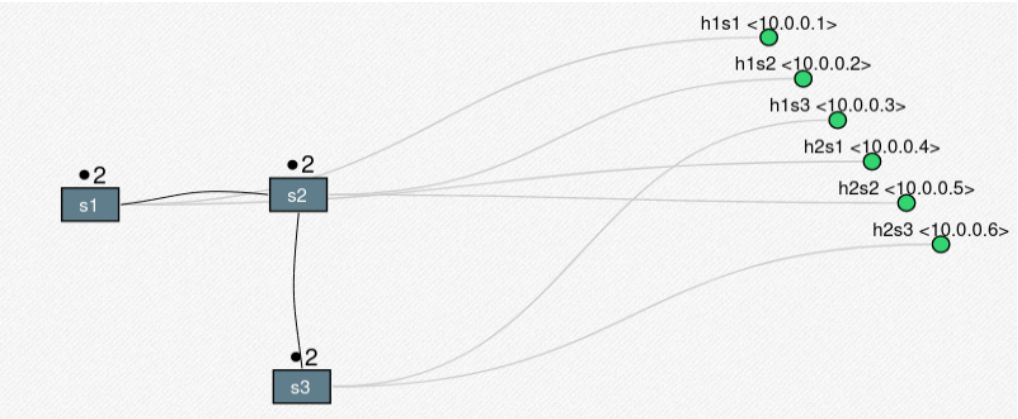
De forma adicional, hemos probado si podíamos realizar lo mismo bloqueando la MAC. Hemos visto que en cada ejecución, la MAC de los hosts es distinta por lo cual solo se podría hacer si se fijase o conociéramos una MAC concreta para cada host.

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

- **Start mininet to execute a network with 3 OVSs connected in a line and 2 hosts per OVS; take a look at the first chapter of the example to understand the meaning and function of each parameter:**
  - **$ sudo mn --topo linear,3,2 --mac --switch ovsk --controller remote**

  Verify the topology you just created, use the dump and links commands of the Mininet CLI; you may use the results in the tool http://demo.spear.narmox.com/app/?apiurl=demo#!/mininet to have a graphical view of the topology.

  Después de haber ejecutado el comando que se nos indicaba en el documento de la práctica, hemos ejecutado los comandos *dump* y *links*. A partir de la información que obtenemos de dichos comandos, hemos ido a la página web que se nos indicaba para ver visualmente la topología y como están distribuidos los distintos *hosts*. El resultado gráfico se muestra en la siguiente imagen:



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

- **Modify the addresses of the hosts to align with the ones proposed at the beginning of the assignment. Use the Mininet CLI to execute the following commands**
  - **mininet> <host> ip addr del <IP>/<máscara> dev <interfaz>**
  - **mininet> <host> ip addr add <IP>/<máscara> dev <interfaz>**

  for each one of the six nodes. First command removes the default address assigned by Mininet and the second one associates the right address and mask. Can you ping the hosts in the same LAN? And between different LANs? Try with the ping command on the mininet CLI.

  Como todavía el controller no está ejecutándose, a la hora de hacer el pingall, como podemos observar en la imagen inferior, no se ha establecido conexión desde ningún host a ningún host, ni de su misma LAN ni a otra LAN.

```
mininet> h1s1 ip addr del 10.0.0.1/8 dev h1s1-eth0
mininet> h1s2 ip addr del 10.0.0.2/8 dev h1s2-eth0
mininet> h1s3 ip addr del 10.0.0.3/8 dev h1s3-eth0
mininet> h2s1 ip addr del 10.0.0.4/8 dev h2s1-eth0
mininet> h2s2 ip addr del 10.0.0.5/8 dev h2s2-eth0
mininet> h2s3 ip addr del 10.0.0.6/8 dev h2s3-eth0
mininet> h1s1 ip addr add 192.168.1.2/24 dev h1s1-eth0
mininet> h2s1 ip addr add 192.168.1.3/24 dev h2s1-eth0
mininet> h1s2 ip addr add 192.168.2.2/24 dev h1s2-eth0
mininet> h2s2 ip addr add 192.168.2.3/24 dev h2s2-eth0
mininet> h1s3 ip addr add 192.168.3.2/24 dev h1s3-eth0
mininet> h2s3 ip addr add 192.168.3.3/24 dev h2s3-eth0
```

```
mininet> pingall
*** Ping: testing ping reachability
h1s1 -> X X X X X
h1s2 -> X X X X X
h1s3 -> X X X X X
h2s1 -> X X X X X
h2s2 -> X X X X X
h2s3 -> X X X X X
*** Results: 100% dropped (0/30 received)
```

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

- **Open a new terminal and start the Ryu controller with the script implementing the router capabilities:**
  - **$ ryu-manager /home/mininet/Software/ryu/ryu/app/rest_router.py**

  , or simpler and equivalent:

  - **$ ryu-manager ryu.app.rest_router**

  You will see in the output that the controller is offering an http server on which it accepts the REST requests that we will use to configure the OVSs: leave it open and running.

La siguiente captura muestra el resultado de haber ejecutado el primer comando que se nos ofrecía en la práctica. Tal y como dice el enunciado, podemos ver, a partir de la información, que se ha iniciado el controlador y se pueden ver en las líneas de por medio tres líneas que dicen “join as a router” la cual cosa informa de que los switches se han inicializado como routers y por lo tanto necesitaran su routing table.

```
loading app /home/mininet/Software/ryu/ryu/app/rest_router.py
loading app ryu.controller.ofp_handler
instantiating app None of DPSet
creating context dpset
creating context wsgi
instantiating app /home/mininet/Software/ryu/ryu/app/rest_router.py of RestRouterAPI
instantiating app ryu.controller.ofp_handler of OFPHandler
(4938) wsgi starting up on http://0.0.0.0:8080
[RT][INFO] switch_id=0000000000000001: Set SW config for TTL error packet in.
[RT][INFO] switch_id=0000000000000001: Set ARP handling (packet in) flow [cookie=0x0]
[RT][INFO] switch_id=0000000000000001: Set L2 switching (normal) flow [cookie=0x0]
[RT][INFO] switch_id=0000000000000001: Set default route (drop) flow [cookie=0x0]
[RT][INFO] switch_id=0000000000000001: Start cyclic routing table update.
[RT][INFO] switch_id=0000000000000001: Join as router.
[RT][INFO] switch_id=0000000000000002: Set SW config for TTL error packet in.
[RT][INFO] switch_id=0000000000000002: Set ARP handling (packet in) flow [cookie=0x0]
[RT][INFO] switch_id=0000000000000002: Set L2 switching (normal) flow [cookie=0x0]
[RT][INFO] switch_id=0000000000000002: Set default route (drop) flow [cookie=0x0]
[RT][INFO] switch_id=0000000000000002: Start cyclic routing table update.
[RT][INFO] switch_id=0000000000000002: Join as router.
[RT][INFO] switch_id=0000000000000003: Set SW config for TTL error packet in.
[RT][INFO] switch_id=0000000000000003: Set ARP handling (packet in) flow [cookie=0x0]
[RT][INFO] switch_id=0000000000000003: Set L2 switching (normal) flow [cookie=0x0]
[RT][INFO] switch_id=0000000000000003: Set default route (drop) flow [cookie=0x0]
[RT][INFO] switch_id=0000000000000003: Start cyclic routing table update.
[RT][INFO] switch_id=0000000000000003: Join as router.
```

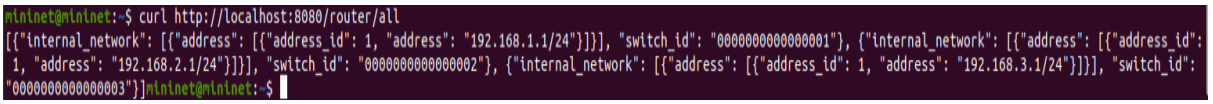
Sara Soriano - 240007
Rubén Vera - 241456
Eneko Treviño - 241679

- **Open a third terminal. You can query at any time the Ryu script for the current configuration of the known OVS through this REST call:**
  - **$ curl http://localhost:8080/router/all**

  Configure the LAN addresses of the three OVSs. Use the REST call seen in setting-the-address:

  - **$ curl -X POST -d '{"address":"<*IP*>/<*mask*>"}' http://localhost:8080/router/<OVS_*ID*>**

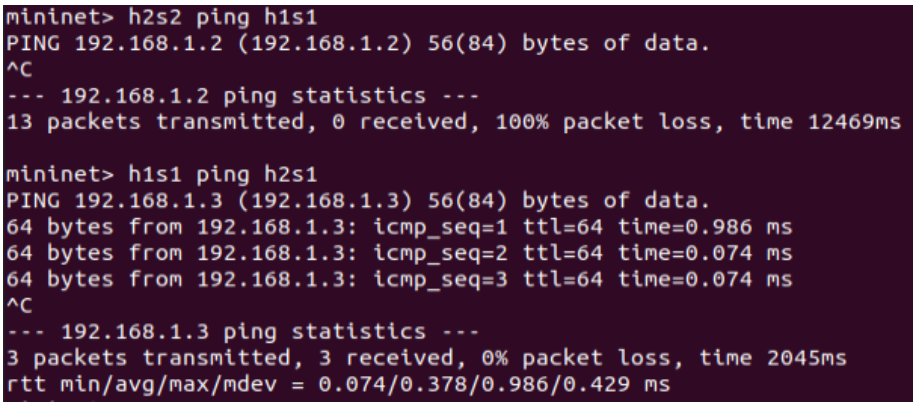
  Una vez configuradas en cada router cómo comunicarse con su propia LAN de los tres OVS, obtenemos el siguiente resultado a partir de imprimir la información que contienen todos los *routers*.



- **Now, define the default gateway for the hosts, this should be the address of the routers you just configured in the previous step:**
  - **mininet> <*host*> ip route add default via <*gateway_ip*>**

  Can you ping now between hosts in the same LAN? And between different LANs?

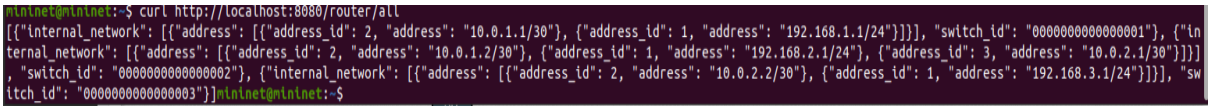
  Primeramente, miramos el *ping* entre *hosts*, utilizando *pingall*. En este caso no conseguimos ver el ping entre los *hosts*. Sin embargo, a la hora de hacer *ping* uno a uno vimos que el *ping* entre hosts de la misma LAN son exitosos. Esto se debe a que ya hay conexión establecida dentro de cada LAN, pero como todavía no se ha hecho la Routing Table no hay conexión entre diversas. Este resultado se puede observar en la siguiente imagen:

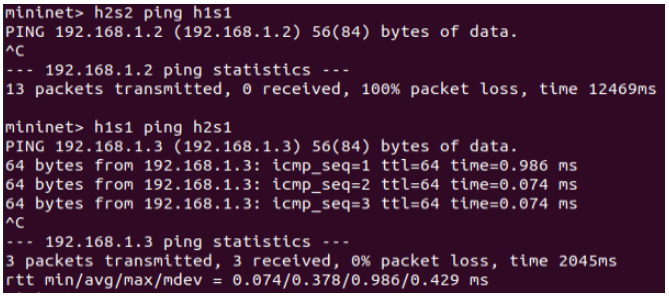


Sara Soriano - 240007
Rubén Vera - 241456
Eneko Treviño - 241679

- **Configure now the addresses of the point-to-point links between the three OVSs. Use the same REST call described above. Again, check the ping between the hosts.**

  Tal y como nos esperábamos los resultados son los mismos que en el apartado anterior, esto se debe a que aunque hayamos configurado las IPs que le corresponde a los enlaces entre cada *router* todavía no hemos definido sus *Routing Tables*. Esto provoca que cuando se quiere enviar un paquete a una LAN diferente de la del *host* destinatario no sabe por dónde debe enviarlo. Con lo cual, hasta que no definamos la información propia de cada uno de los *routers* no se realizará un *pingAll()* con completo éxito.





- **Now configure the routes in the routers. You can make use of other two REST calls from the API of the router Ryu script, either to define the default gateway:**
  - **$ curl -X POST -d '{"gateway": "<*gateway_ip*>"}' http://localhost:8080/router/<*OVS_ID*>**

  , or the route for a given destination network:

  - **$ curl -X POST -d '{"destination": "<*IP*>/<*mask*>", "gateway": "<*gateway_ip*>"}' http://localhost:8080/router/<*OVS_ID*>**

  Para el router 1 y el router 3 hemos usado el primer comando que se nos ofrecía en el enunciado, y, en cambio, para el router 2 hemos utilizado la segunda. Hemos decidido realizarlo de esta manera porque el router 1 si quiere cambiar de LAN deberá siempre enviarlo al 2, lo mismo pasa con el router 3. Pero el router 2 depende de cual sea la dirección final del paquete lo querrá enviar a uno u otro y de esta manera nos parecía más sencillo configurarlo.

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

Como en este apartado hemos establecido toda la información de las *Routing Tables* los *pings* son hechos con éxito. Entre la misma LAN y diferentes, para observar dicho resultado de manera visual hemos realizado el *pingAll()* que se muestra en la segunda imagen que se muestra a continuación: