

FIUBA - 75.43

Introducción a los Sistemas Distribuidos

Trabajo Práctico N°3: Software-Defined Networks
1er cuatrimestre, 2021

Integrantes:

Nombre	Padrón	Mail
Levi Fernández, Matías	99119	milevif@fi.uba.ar
Pérez Machado, Axel	101127	axelmpm@gmail.com
Franco, Tomás	91013	tomasnfranco@gmail.com

Fecha de entrega: 09/07/2021

Índice

1. Introducción	2
2. README	2
3. Testing	3
4. Primera parte: Topología 1	4
5. Segunda parte: Topología 2	7
5.1. Regla 1	8
5.2. Regla 2	11
5.3. Regla 3	13
6. Preguntas a responder	16

1. Introducción

En este trabajo practico se tiene como objetivo afianzar el entendimiento acerca de las SDN (y en particular el protocolo OpenFlow): se realizara una simulacion de redes interconectadas con distintas topologias y sobre estas se correran tests y experimentos para analizar el funcionamiento de los distintos controladores remotos y componentes de los mismos.

Para simular el controlador remoto de la SDN utilizaremos POX, un controlador de SDN/OpenFlow basado en python y open source

2. README

TOPOLOGIA 1

first, run a console from the pox directory and set up the remote controller:

```
| ./pox.py forwarding.l2_learning openflow.spanning_tree --no-flood --hold-down openflow.discovery host_tracker
```

or

```
| ./pox.py log.level forwarding.l2_learning openflow.spanning_tree --no-flood --hold-down openflow.discovery host_tracker
```

or

```
| ./pox.py log.level --debug forwarding.l2_learning openflow.spanning_tree --no-flood --hold-down openflow.discovery host_tracker
```

then, on a different console set up the configuration and run the test ping all

```
| sudo mn --custom topo1.py --topo topo1,switch_level=3,host_amount=4 --mac --arp --switch ovsk --controller remote
```

TOPOLOGIA 2

first, run a console from the pox directory and set up the remote controller:

```
| ./pox.py misc.firewall forwarding.l2_learning
```

or

```
| ./pox.py log.level misc.firewall forwarding.l2_learning
```

or

```
| ./pox.py log.level --debug misc.firewall forwarding.l2_learning
```

then, on a different console set up the configuration and run the test ping all

```
| sudo mn --custom ./topo2.py --topo topo2,11 --switch ovsk --controller remote
```

<https://github.com/LeviMatias/redes-distribuidos-tps/tree/main/tp3>.

3. Testing

Como ya es costumbre nuestra de otros tps, proveemos en la carpeta el zip 'tests' un archivo de especificacion de los tests corridos para garantizar el correcto funcionamiento del software en casos normales y ante variaciones de los mismos.

4. Primera parte: Topologia 1

El objetivo del desarrollo de la primera topologia fue familiarizarnos con el ambiente y herramientas de trabajo.

Se requirio una estructura en forma de rombo, con hosts en los extremos interconectados por distintos niveles de switches.

Para desplegar la red es necesario en una primera consola levantar el controlador remoto que se encargara de conectar los switches y definir las politicas de comunicacion

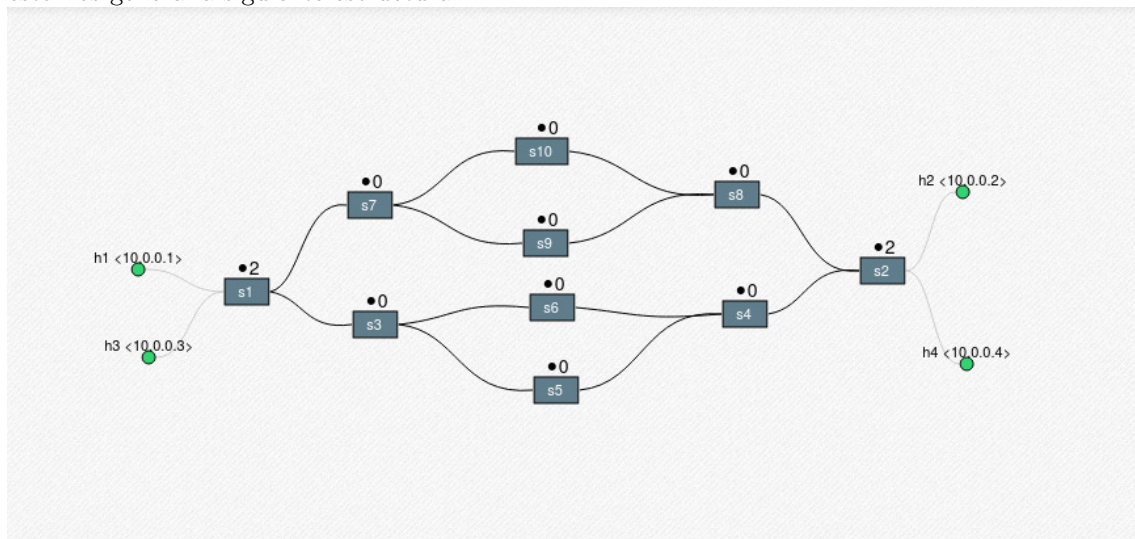
```
./pox.py forwarding.l2_learning openflow.spanning_tree --no-flood --hold-down
openflow.discovery_host_tracker
```

este comando se debe ejecutar dentro de la carpeta de pox.

Luego, se debe inicializar la red con mininet:

```
sudo mn --custom topo1.py --topo topo1,switch_level=3,host_amount=4 --mac --arp --
switch ovsk --controller remote
```

esto nos genera la siguiente estructura:



Esto es una red de nivel 3 con 10 switches OpenFlow y 4 hosts, para analizar la conectividad de la misma se ejecuto el comando 'pingall' encontramos los siguientes resultados con wireshark

The screenshot shows the Wireshark network protocol analyzer interface. The main pane displays a list of captured packets. The selected packet is an HTTP GET request from 10.0.0.1 to 10.0.0.3. The details pane shows the structure of the packet, including Ethernet II, Internet Protocol Version 4, Transmission Control Protocol, and Hypertext Transfer Protocol. The IO Graphs window on the right shows a line graph of network traffic over time, with a sharp peak at approximately 9 seconds.

5

```
Unable to contact the remote controller at 127.0.0.1:6653
Connecting to remote controller at 127.0.0.1:6633
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s1 s2 s3 s4 s5 s6 s7 s8 s9 s10
*** Adding links:
(h1, s1) (h2, s2) (h3, s1) (h4, s2) (s1, s3) (s1, s7) (s2, s4) (s2, s8) (s3, s5)
(s3, s6) (s4, s5) (s4, s6) (s7, s9) (s7, s10) (s8, s9) (s8, s10)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
c0
*** Starting 10 switches
s1 s2 s3 s4 s5 s6 s7 s8 s9 s10 ...
*** Starting CLI:
mininet> pingall
*** 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)
mininet> █
```

Dada la naturaleza de la red propuesta y, tal como dice el enunciado, esta posee numerosos bucles. Estos resultan ser problematicos para ciertas funciones que debe cumplir el controller a la hora de tomar decisiones sobre la red (un ejemplo de esto podria ser establecer un ruteo).

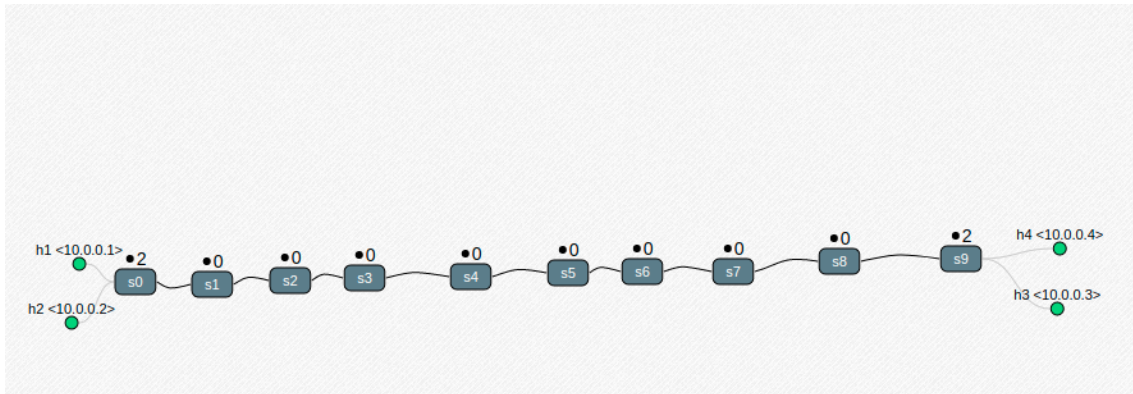
Por esta razon se ejecuta el commando del controller con los flags `openflow.spanning_tree -no-flood -hold-down openflow.discovery host_tracker`.

Esto agrega la funcionalidad de hacer que el controller establezca un arbol de cubrimiento minimo sobre los nodos de la red de forma tal que ahora esta sea conexa (todos se ven con todos) pero de una forma unica (sin bucles)

5. Segunda parte: Topologia 2

En esta parte del tp se pedia una topologia mucho mas sencilla.

Abajo tenemos un ejemplo (generado por nuestro codigo) del tipo de topologia pedida.



La consigna era implementar un controlador openflow con un firewall que fuese capaz de hacer cumplir tres reglas sobre la red.

Como agregado de buen diseño este firewall tiene la capacidad de ser reprogramado y extendido de forma que puede aceptar tantas reglas como se quiera con diferentes parametrizaciones de las mismas.

Esto se logra a traves de la modificacion del archivo `rule_config.json`.

En el zip de la entrega ya vienen implementadas las tres requeridas por el tp y se encuentra en ese archivo un ejemplo de uso bastante comprehensivo.

5.1. Regla 1

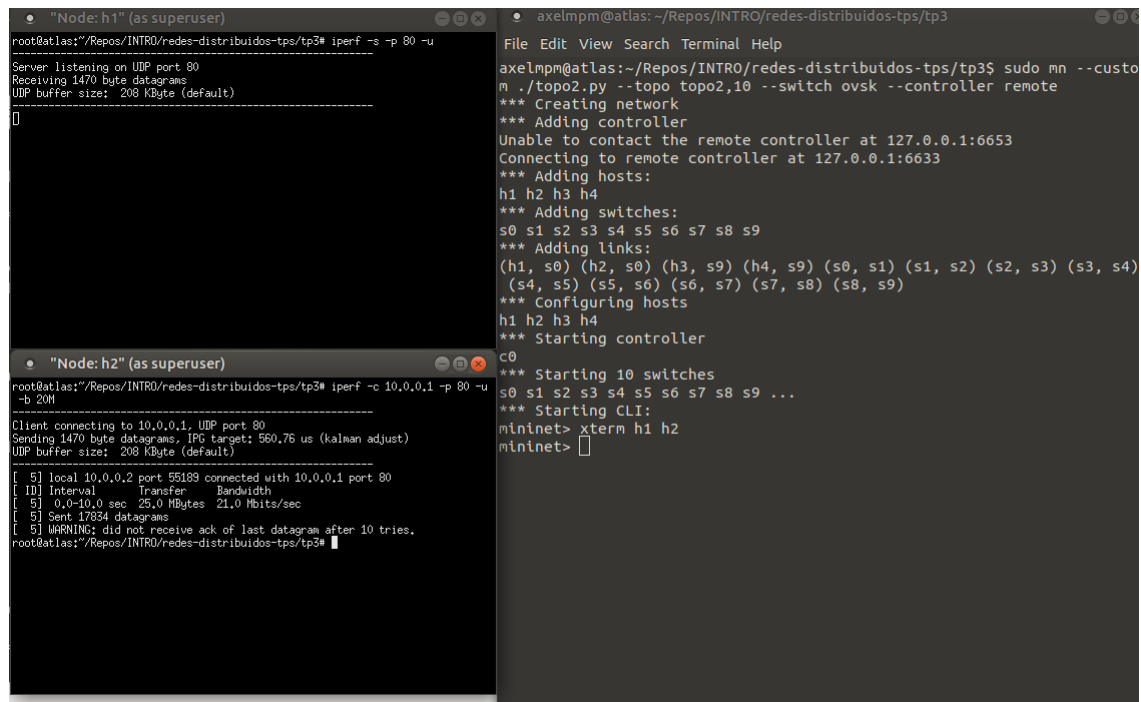
"Se deben descartar todos los mensajes cuyo puerto destino sea 80."

The screenshot shows three terminal windows. The top-left window, titled '"Node: h1" (as superuser)', shows a server listening on TCP port 80. The bottom-left window, titled '"Node: h2" (as superuser)', shows an attempt to connect to h1 on port 80 using the command `iperf -c 10.0.0.1 -p 80`. The right window, titled 'axelmpm@atlas: ~/Repos/INTRO/redes-distribuidos-tps/tp3', shows the execution of `sudo mn --custom ./topo2.py --topo topo2,10 --switch ovsk --controller remote`. The output indicates that the network was created, but the remote controller at 127.0.0.1:6653 could not be contacted. The network configuration includes hosts h1, h2, h3, h4, switches s0 through s9, and a set of links connecting them. The controller is started, and the switches are configured. The mininet prompt shows `mininet> xterm h1 h2`.

Aca vemos lo que pasa al establecer (o intentar establecer) una conexion TCP de h2 a h1 con el puerto destino 80.

Dada la regla la conexion deberia ser imposible y asi lo muestra la captura.

La misma fue heca varios segundos despues de la ejecucion de los comandos, evidenciando asi la incomunicabilidad.



```

• "Node: h1" (as superuser)
root@atlas:~/Repos/INTRO/redes-distribuidos-tps/tp3# iperf -s -p 80 -u
Server listening on UDP port 80
Receiving 1470 byte datagrams
UDP buffer size: 208 KByte (default)
-----
0

• "Node: h2" (as superuser)
root@atlas:~/Repos/INTRO/redes-distribuidos-tps/tp3# iperf -c 10.0.0.1 -p 80 -u -b 20M
Client connecting to 10.0.0.1, UDP port 80
Sending 1470 byte datagrams, IPG target: 560.76 us (kalman adjust)
UDP buffer size: 208 KByte (default)
-----
[ 5] local 10.0.0.2 port 55189 connected with 10.0.0.1 port 80
[ ID] Interval      Transfer    Bandwidth
[ 5] 0.0-10.0 sec   25.0 MBytes  21.0 Mbits/sec
[ 5] Sent 17834 datagrams
[ 5] WARNING: did not receive ack of last datagram after 10 tries.
root@atlas:~/Repos/INTRO/redes-distribuidos-tps/tp3#

axelmpm@atlas:~/Repos/INTRO/redes-distribuidos-tps/tp3$ sudo mn --custo
m ./topo2.py --topo topo2,10 --switch ovsk --controller remote
*** Creating network
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6653
Connecting to remote controller at 127.0.0.1:6633
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s0 s1 s2 s3 s4 s5 s6 s7 s8 s9
*** Adding links:
(h1, s0) (h2, s0) (h3, s9) (h4, s9) (s0, s1) (s1, s2) (s2, s3) (s3, s4)
(s4, s5) (s5, s6) (s6, s7) (s7, s8) (s8, s9)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
c0
*** Starting 10 switches
s0 s1 s2 s3 s4 s5 s6 s7 s8 s9 ...
*** Starting CLI:
mininet> xterm h1 h2
mininet>

```

Vemos que el resultado es el mismo si ahora establecemos un flujo UDP.

Puede verse que al tratarse de UDP se emite un mensaje de incomunicabilidad. UDP manda información sin preocuparse por reintentar o asegurar la entrega. Detecta que no hubo ACKs e informa de que no se pudo comunicar correctamente.

```

• "Node: h1" (as superuser)
root@atlas:~/Repos/INTRO/redes-distribuidos-tps/tp3# iperf -s -p 81
Server listening on TCP port 81
TCP window size: 85.3 KByte (default)
[ 6] local 10.0.0.1 port 81 connected with 10.0.0.2 port 44302
[ ID] Interval      Transfer    Bandwidth
[ 6] 0.0-10.0 sec  55.0 GBytes  47.2 Gbits/sec
[  ]

• "Node: h2" (as superuser)
root@atlas:~/Repos/INTRO/redes-distribuidos-tps/tp3# iperf -c 10.0.0.1 -p 81
Client connecting to 10.0.0.1, TCP port 81
TCP window size: 85.3 KByte (default)
[ 5] local 10.0.0.2 port 44302 connected with 10.0.0.1 port 81
[ ID] Interval      Transfer    Bandwidth
[ 5] 0.0-10.0 sec  55.0 GBytes  47.2 Gbits/sec
root@atlas:~/Repos/INTRO/redes-distribuidos-tps/tp3#

axelmpm@atlas:~/Repos/INTRO/redes-distribuidos-tps/tp3$ sudo mn --custom
m ./topo2.py --topo topo2,10 --switch ovsk --controller remote
*** Creating network
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6653
Connecting to remote controller at 127.0.0.1:6633
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s0 s1 s2 s3 s4 s5 s6 s7 s8 s9
*** Adding links:
(h1, s0) (h2, s0) (h3, s9) (h4, s9) (s0, s1) (s1, s2) (s2, s3) (s3, s4)
(s4, s5) (s5, s6) (s6, s7) (s7, s8) (s8, s9)
*** Configuring hosts
h1 h2 h3 h4
c0
*** Starting controller
c0
*** Starting 10 switches
s0 s1 s2 s3 s4 s5 s6 s7 s8 s9 ...
*** Starting CLI:
mininet> xterm h1 h2
mininet>

```

Por ultimo aca vemos el caso donde se establece un conexion TCP pero apuntando al puerto 81.

Dado que no entramos en el contexto de la regla 1, esta conexion puede realizarse con exito, tal como se ve en la figura de arriba.

5.2. Regla 2

"Se deben descartar todos los mensajes que provengan del host 1, tengan como puerto destino el 5001, y esten utilizando el protocolo UDP".

La cantidad de casos a testear aca son, en rigor, ocho. Dado que no se busca hacer un informe extensivo sobre todo el testing (ver el archivo de especificacion de tests), se provee aca solo dos casos. El caso feliz y un caso de fallo.

The image shows two terminal windows. The left window, titled "Node: h2" (as superuser), shows a server listening on UDP port 5001 and receiving 1470 byte datagrams. The right window, titled "axelmpm@atlas: ~/Repos/INTRO/redes-distribuidos-tps/tp3", shows the execution of a script to create a network topology. The script adds a controller, hosts (h1, h2, h3, h4), switches (s0 to s9), and links. It then starts the controller and switches. The output shows a warning: "WARNING: did not receive ack of last datagram after 10 tries." This indicates a failure in the communication between the host and the server.

```

root@atlas:~/Repos/INTRO/redes-distribuidos-tps/tp3# iperf -s -p 5001 -u
Server listening on UDP port 5001
Receiving 1470 byte datagrams
UDP buffer size: 208 KByte (default)
[ 0]
[ 0]

root@atlas:~/Repos/INTRO/redes-distribuidos-tps/tp3# iperf -c 10.0.0.2 -p 5001 -u -b 20M
Client connecting to 10.0.0.2, UDP port 5001
Sending 1470 byte datagrams, IPG target: 560.76 us (kcalman adjust)
UDP buffer size: 208 KByte (default)
[ 5] local 10.0.0.1 port 45402 connected with 10.0.0.2 port 5001
[ ID] Interval      Transfer    Bandwidth
[ 5] 0.0-10.0 sec  25.0 MBytes  21.0 Mbits/sec
[ 5] Sent 17834 datagrams
[ 5] WARNING: did not receive ack of last datagram after 10 tries.
root@atlas:~/Repos/INTRO/redes-distribuidos-tps/tp3#

axelmpm@atlas:~/Repos/INTRO/redes-distribuidos-tps/tp3$ sudo mn --custo
n ./topo2.py --topo topo2,10 --switch ovsk --controller remote
*** Creating network
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6653
Connecting to remote controller at 127.0.0.1:6653
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s0 s1 s2 s3 s4 s5 s6 s7 s8 s9
*** Adding links:
(h1, s0) (h2, s0) (h3, s9) (h4, s9) (s0, s1) (s1, s2) (s2, s3) (s3, s4)
(s4, s5) (s5, s6) (s6, s7) (s7, s8) (s8, s9)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
c0
*** Starting 10 switches
s0 s1 s2 s3 s4 s5 s6 s7 s8 s9 ...
*** Starting CLI:
mininet> xterm h1 h2
mininet>

```

En esta figura vemos un caso de fallo de la comunicacion tal como se espera del correcto funcionamiento de la regla 2. La conexion es UDP, se apunta al puerto 5001 y esto se realiza desde el host 1.

```

root@atlas:~/Repos/INTRO/redes-distribuidos-tps/tp3# iperf -s -p 5002 -u
Server listening on UDP port 5002
Receiving 1470 byte datagrams
UDP buffer size: 208 KByte (default)
-----
[ 5] local 10.0.0.2 port 5002 connected with 10.0.0.1 port 46410
[ ID] Interval      Transfer    Bandwidth    Jitter    Lost/Totl  Datagrams
[ 5] 0.0-10.0 sec  25.0 MBytes  21.0 Mbits/sec  0.002 ms  0/17834 (0%)
[ 5] 0.00-10.00 sec  7 datagrams received out-of-order
root@atlas:~/Repos/INTRO/redes-distribuidos-tps/tp3#

axelmpm@atlas:~/Repos/INTRO/redes-distribuidos-tps/tp3$ sudo mn --custo
m ./topo2.py --topo topo2,10 --switch ovsk --controller remote
*** Creating network
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6653
Connecting to remote controller at 127.0.0.1:6633
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s0 s1 s2 s3 s4 s5 s6 s7 s8 s9
*** Adding links:
(h1, s0) (h2, s0) (h3, s9) (h4, s9) (s0, s1) (s1, s2) (s2, s3) (s3, s4)
(s4, s5) (s5, s6) (s6, s7) (s7, s8) (s8, s9)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
c0
*** Starting 10 switches
s0 s1 s2 s3 s4 s5 s6 s7 s8 s9 ...
*** Starting CLI:
mininet> xterm h1 h2
mininet>

```

```

root@atlas:~/Repos/INTRO/redes-distribuidos-tps/tp3# iperf -c 10.0.0.2 -p 5002
-u -b 20M
Client connecting to 10.0.0.2, UDP port 5002
Sending 1470 byte datagrams, IPG target: 560.76 us (kalman adjust)
UDP buffer size: 208 KByte (default)
-----
[ 5] local 10.0.0.1 port 46410 connected with 10.0.0.2 port 5002
[ ID] Interval      Transfer    Bandwidth
[ 5] 0.0-10.0 sec  25.0 MBytes  21.0 Mbits/sec
[ 5] Sent 17834 datagrams
[ 5] Server Report:
[ 5] 0.0-10.0 sec  25.0 MBytes  21.0 Mbits/sec  0.000 ms  0/17834 (0%)
[ 5] 0.00-10.00 sec  7 datagrams received out-of-order
root@atlas:~/Repos/INTRO/redes-distribuidos-tps/tp3#

```

Aca puede verse que solo variando una de las condiciones de las reglas ya es capaz de establecerse comunicacion, tal como se esperaria. En este caso se vario el puerto del 5001 al 5002

5.3. Regla 3

"Se debe elegir dos hosts cualquiera, y los mismos no deben poder comunicarse de ninguna forma".

Nuevamente. Para validar la correctitud de implementacion de esta regla habria que tomar todas las combinaciones de dos hosts dado 4, 3, 2 y un host y probar desconectandolos y que realmente no se puedan comunicar de ninguna manera. Esto requeriria probar diferentes protocolos, diferentes cargas de la red, diferentes puertos, etc.

El chequeo de esto es tan complejo que en el archivo de testing se probaron algunos casos mas o menos representativos.

Aca solo se expone un caso visto de dos formas.

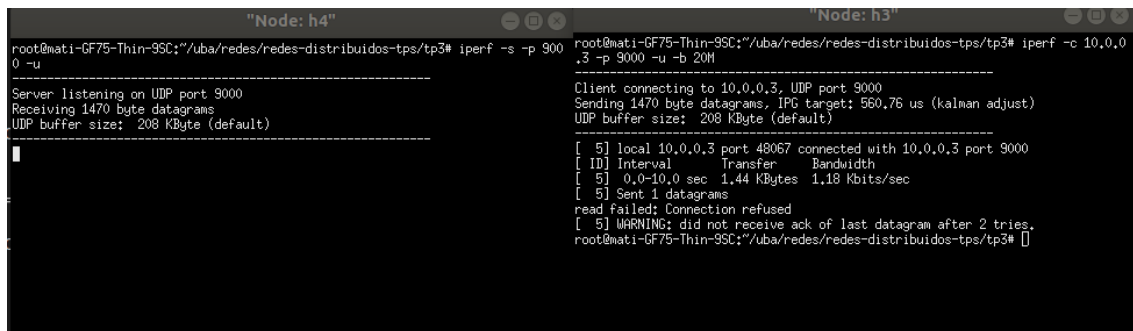
```

axelmpm@atlas:~/Repos/INTRO/redes-distribuidos-tps/tp3/pox$ ./pox.py ml
sc.firewall forwarding.l2_learning
POX 0.7.0 (gar) / Copyright 2011-2020 James McCauley, et al.
WARNING:version:Support for Python 3 is experimental.
INFO:core:POX 0.7.0 (gar) is up.
INFO:openflow.of_01:[00-00-00-00-00-06 2] connected
INFO:misc.firewall:Firewall rules installed on 6
INFO:openflow.of_01:[00-00-00-00-00-01 3] connected
INFO:misc.firewall:Firewall rules installed on 1
INFO:openflow.of_01:[e2-f7-f9-5a-1a-47 4] connected
INFO:misc.firewall:Firewall rules installed on 249554668231239
INFO:openflow.of_01:[00-00-00-00-00-03 5] connected
INFO:misc.firewall:Firewall rules installed on 3
INFO:openflow.of_01:[00-00-00-00-00-02 6] connected
INFO:misc.firewall:Firewall rules installed on 2
INFO:openflow.of_01:[00-00-00-00-00-08 7] connected
INFO:misc.firewall:Firewall rules installed on 8
INFO:openflow.of_01:[00-00-00-00-00-09 8] connected
INFO:misc.firewall:Firewall rules installed on 9
INFO:openflow.of_01:[00-00-00-00-00-07 9] connected
INFO:misc.firewall:Firewall rules installed on 7
INFO:openflow.of_01:[00-00-00-00-00-04 10] connected
INFO:misc.firewall:Firewall rules installed on 4
INFO:openflow.of_01:[00-00-00-00-00-05 11] connected
INFO:misc.firewall:Firewall rules installed on 5

axelmpm@atlas:~/Repos/INTRO/redes-distribuidos-tps/tp3$ sudo mn --custo
n ./topo2.py --topo topo2,10 --switch ovsk --controller remote
*** Creating network
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6653
Connecting to remote controller at 127.0.0.1:6633
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s0 s1 s2 s3 s4 s5 s6 s7 s8 s9
*** Adding links:
(h1, s0) (h2, s0) (h3, s9) (h4, s9) (s0, s1) (s1, s2) (s2, s3) (s3, s4)
(s4, s5) (s5, s6) (s6, s7) (s7, s8) (s8, s9)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
c0
*** Starting 10 switches
s0 s1 s2 s3 s4 s5 s6 s7 s8 s9 ...
*** Starting CLI:
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4
h2 -> h1 h3 h4
h3 -> h1 h2 X
h4 -> h1 h2 X
*** Results: 16% dropped (10/12 received)
mininet>

```

Probamos crear la red, desconectar por config a host 3 y host 4 y al ejecutar pingall vemos que ni h3 puede alcanzar a h4 ni h4 puede alcanzar a h3. Esto apunta a que no hay comunicacion en ningun sentido entre estos hosts, tal como se espera.



```
"Node: h4"
root@mati-GF75-Thin-9SC:~/uba/redes/redes-distribuidos-tps/tp3# iperf -s -p 9000 -u
-----
Server listening on UDP port 9000
Receiving 1470 byte datagrams
UDP buffer size: 208 KByte (default)
-----

"Node: h3"
root@mati-GF75-Thin-9SC:~/uba/redes/redes-distribuidos-tps/tp3# iperf -c 10.0.0.3 -p 9000 -u -b 20M
-----
Client connecting to 10.0.0.3, UDP port 9000
Sending 1470 byte datagrams, IPG target: 560.76 us (kalman adjust)
UDP buffer size: 208 KByte (default)
-----
[ 5] local 10.0.0.3 port 48067 connected with 10.0.0.3 port 9000
[ 10] Interval      Transfer    Bandwidth
[ 5]  0.0-10.0 sec  1.44 KBytes  1.18 Kbits/sec
[ 5] Sent 1 datagrams
read failed: Connection refused
[ 5] WARNING: did not receive ack of last datagram after 2 tries.
root@mati-GF75-Thin-9SC:~/uba/redes/redes-distribuidos-tps/tp3#
```

Vemos aca otra prueba, esta vez estableciendo (entre comillas) una conexion h3 hacia h4 de protocolo UDP con puerto 9000. Nuevamente vemos que se nos informa de la imposibilidad de conexion.

6. Preguntas a responder

1. *¿Cuál es la diferencia entre un Switch y un router? ¿Qué tienen en común?*

La función de los switches es conectar entre sí a los distintos dispositivos dentro de una red, permitiéndoles compartir recursos e información. Los mismos operan a nivel de Data Link Layer.

Por otro lado, de la misma forma en que un switch conecta múltiples dispositivos para formar una red, los routers conectan múltiples switches y sus respectivas redes con el objetivo de formar una red de mayor tamaño.

A su vez, los routers son responsables de direccionar el tráfico de red eligiendo la ruta más eficiente para la información. Los mismos operan a nivel de Network Layer.

2. *¿Cuál es la diferencia entre un Switch convencional y un Switch OpenFlow?*

En los switches convencionales, el forwarding de los paquetes (data plane) y el ruteo (control plane) se realiza en el mismo dispositivo.

En cambio, en los switches OpenFlow, el data plane se desacopla del control plane. El primero se realiza en el dispositivo en cuestión, mientras que el segundo se implementa mediante software con un controlador SDN separado que toma las decisiones concernientes al ruteo.

El switch y el controlador se comunican entre sí por medio del protocolo OpenFlow. La introducción de este último conlleva varias posibles ventajas producto de la posibilidad de realizar una manipulación de los paquetes.

3. *¿Se pueden reemplazar todos los routers de la Internet por Switches OpenFlow? Piense en el escenario interASes para elaborar su respuesta*

La introducción de las SDNs hace posible el reemplazo de routers y switches (con data y control planes) por switches que combinan hardware y un control plane sofisticado implementado en software.

Un Autonomus System (AS) es un grupo de routers que se encuentran bajo el mismo control administrativo. Tradicionalmente las SDNs son aplicables solamente en el contexto de un mismo AS y por lo tanto no se utilizan a la hora de enviar un paquete a través de varios de estos.

Actualmente, la extensión de los conceptos relacionados con las SDNs desde un entorno de intra-AS a uno inter-AS representa un área importante de investigación.