

## TP N°2: Software-Defined Networks

[75.43] Int. a los Sistemas Distribuidos  
Grupo A7  
Segundo cuatrimestre de 2023

| <b>Padrón</b> | <b>Alumno</b>                     | <b>Correo electrónico</b> |
|---------------|-----------------------------------|---------------------------|
| 104415        | Correa, Valentina Laura           | vcorrea@fi.uba.ar         |
| 101231        | Dimartino, Pablo Salvador         | pdimartino@fi.uba.ar      |
| 104046        | Agustin Ariel Andrade             | aandrade@fi.uba.ar        |
| 104196        | Stephanie Ingrid Izquierdo Osorio | sizquierdo@fi.uba.ar      |
| 100113        | Juan Sebastian Burgos             | jsburgos@fi.uba.ar        |

## Contents

|          |   |          |
|----------|---|----------|
| <b>1</b> | <b>Objetivo</b>   | <b>2</b> |
| <b>2</b> | <b>Implementación</b>   | <b>2</b> |
| 2.1      | Topología . . . . .   | 2        |
| 2.2      | Firewall . . . . .  | 2        |
| <b>3</b> | <b>Ejecución</b>  | <b>3</b> |
| <b>4</b> | <b>Pruebas</b>  | <b>3</b> |
| 4.1      | Bloqueo a cualquier host con puerto destino 80 . . . . .  | 3        |
| 4.2      | Bloqueo de paquetes entre dos hosts elegidos arbitrariamente . . . . .  | 4        |
| 4.3      | Bloqueo de paquetes UDP provenientes del host1 con puerto destino 5001 . . . . .  | 5        |
| <b>5</b> | <b>Preguntas a responder</b>  | <b>5</b> |
| 5.1      | ¿Cuál es la diferencia entre un Switch y un router? ¿Qué tienen en común? . . . .   | 5        |
| 5.2      | ¿Cuál es la diferencia entre un Switch convencional y un Switch OpenFlow? . . . .   | 6        |
| 5.3      | ¿Se pueden reemplazar todos los routers de la Intenet por Switches OpenFlow?<br>Piense en el escenario interASes para elaborar su respuesta . . . . . | 6        |

# 1 Objetivo

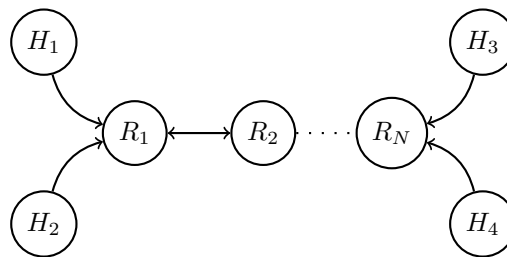
La presente entrega contiene los requerimientos pedidos para el trabajo practico grupal N°2 de la materia Introducción a los Sistemas Distribuidos, curso 1 del prof. Alvarez Hamelin.

Este informe se centra en comprender los desafíos de las SDNs y explorar el protocolo Open-Flow, vital para la administración dinámica del tráfico en redes.

## 2 Implementación

### 2.1 Topología

La topología utilizada es, como muestra la figura, de N routers conectados en serie con un par de hosts en cada extremo:



Es implementada en el archivo `./topology.py` en el cual se define una topología heredando de la clase `Topo` porveida por `mininet`. Para la creación de la topología se utiliza un loop para crear los N routers:

```
for switch in range(0, switches):
    switches_list.append(self.addSwitch(f'switch_{switch}'))
```

Luego se añaden los hosts en cada extremo. Por ejemplo, para el primer hosts se lo crea y luego se añade su link de la forma:

```
h1 = self.addHost("host_1")
self.addLink(switches_list[0], h1)
```

Y finalmente se crean los links entre routers con otro ciclo for:

```
for i in range(1, len(switches_list)):
    self.addLink(switches_list[i - 1], switches_list[i])
```

### 2.2 Firewall

El firewall se implementa en el archivo `./controller.py`. Se ofrece un archivo `./rules.json` para la configuración.

Para definir el firewall creamos una clase `Firewall` que hereda de la clase `EventMixin` proveida por `pox`. Se configura la clase para escuchar los eventos producidos por openflow mediante la línea `self.listenTo(core.openflow)`.

Luego, se utiliza el handler `\_handle\_ConnectionUp` que se invoca por única en el evento de creación de la red para agregar las reglas definidas en el archivo de configuración.

### 3 Ejecución

Para poder ejecutar el programa es necesario situarse dentro de `\TP2` y correr el comando:

```
python3 pox/pox.py log.level --DEBUG openflow.of_01 forwarding.l2_learning misc.controller
```

El cual no solo levanta la herramienta `\pox` sino que además se le indica el nivel de log del programa y también se lo configura para que aprenda automáticamente la topología seteada por `properties`.

A su vez, en una nueva terminal y también desde la ruta `\TP2`, deberemos correr:

```
sudo mn --custom ./topology.py --topo customTopo,switches=2 --mac --arp -x
--switch ovsk --controller remote
```

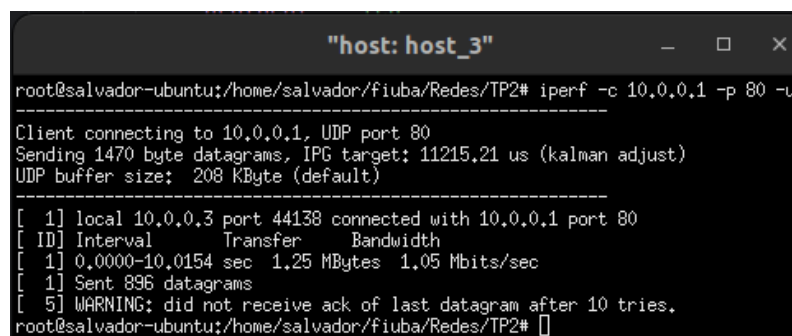
En donde utilizando mininet (`mn`) podremos setear para nuestra topología (`customTopo`) la cantidad de switches deseada (`switches=n`).

## 4 Pruebas

### 4.1 Bloqueo a cualquier host con puerto destino 80

Para la primer regla establecida en nuestro firewall, se decidió bloquear todas las conexiones que tengan como puerto destino al 80, ya sea que provengan tanto del protocolo `udp`, como de `tcp`.

Comenzamos levantando una conexión en el host 1 en el puerto 80 y a través del protocolo `udp`. Una vez levantada esta, procedemos a probar su conexión con un cliente establecido desde el host 3:

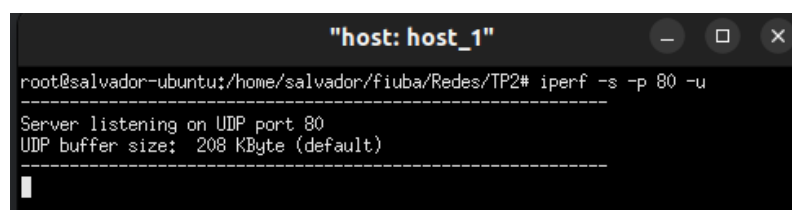


```

"host: host_3"
root@salvador-ubuntu:/home/salvador/fiuba/Redes/TP2# iperf -c 10.0.0.1 -p 80 -u
-----
Client connecting to 10.0.0.1, UDP port 80
Sending 1470 byte datagrams, IPG target: 11215.21 us (kalman adjust)
UDP buffer size: 208 KByte (default)
-----
[ 1] local 10.0.0.3 port 44138 connected with 10.0.0.1 port 80
[ ID] Interval      Transfer    Bandwidth
[ 1] 0.0000-10.0154 sec  1.25 MBytes  1.05 Mbits/sec
[ 1] Sent 896 datagrams
[ 5] WARNING: did not receive ack of last datagram after 10 tries.
root@salvador-ubuntu:/home/salvador/fiuba/Redes/TP2#

```

Como podemos ver en la siguiente imagen, efectivamente no se recibe nada debido a la previa captura del firewall:



```

"host: host_1"
root@salvador-ubuntu:/home/salvador/fiuba/Redes/TP2# iperf -s -p 80 -u
-----
Server listening on UDP port 80
UDP buffer size: 208 KByte (default)
-----

```

Distinto es el caso cuando cambiamos de puerto -por ejemplo el 90- en donde ahora utilizando el protocolo `tcp` vemos que los paquetes si son tanto enviados como recibidos, nuevamente utilizando los hosts 1 y 3:

```

"host: host_1"
root@salvador-ubuntu:/home/salvador/fiuba/Redes/TP2# iperf -s -p 90
-----
Server listening on TCP port 90
TCP window size: 85.3 KByte (default)
-----
[ 1] local 10.0.0.1 port 90 connected with 10.0.0.3 port 48020
[ ID] Interval      Transfer    Bandwidth
[ 1] 0.0000-9.9994 sec 40.0 GBytes 34.4 Gbits/sec
[ ]

```

```

"host: host_3"
root@salvador-ubuntu:/home/salvador/fiuba/Redes/TP2# iperf -c 10.0.0.1 -p 90
-----
Client connecting to 10.0.0.1, TCP port 90
TCP window size: 85.3 KByte (default)
-----
[ 1] local 10.0.0.3 port 48020 connected with 10.0.0.1 port 90
[ ID] Interval      Transfer    Bandwidth
[ 1] 0.0000-10.0154 sec 40.0 GBytes 34.3 Gbits/sec
root@salvador-ubuntu:/home/salvador/fiuba/Redes/TP2#

```

Por otro lado, tambien podemos ver esta comunicaci3n desde wireshark. Aqui podemos filtrar por ejemplo por el puerto utilizado. La imagen de la captura que se muestra a continuaci3n se tomo ante un intento de comunicacion a trav3s del puerto 80 utilizando el protocolo tcp. Como podemos observar, nuevamente utilizamos los hosts anteriormente mencionados y dado que esta conexi3n es intervenida por el firewall notamos en la seccion de info que hay una serie de retransmisi3n de paquetes, la cual atribuimos al intento de realizar dicha conexi3n.

| No. | Time        | Source   | Destination | Protocol | Length | Info   |
|-----|-------------|----------|-------------|----------|--------|--|
| 446 | 2.787547713 | 10.0.0.3 | 10.0.0.1    | TCP      | 70     | 36468 → 80 [SYN] Seq=0 Win=42340 Len=0 MSS=1460 SACK_PERM=1 TSval=489146426 TSecr=0 WS=512                 |
| 447 | 2.788301148 | 10.0.0.3 | 10.0.0.1    | Open     | 160    | Type: OPT_PACKET_IN  |
| 448 | 2.788887980 | 10.0.0.3 | 10.0.0.1    | Open     | 254    | Type: OPT_PACKET_OUT   |
| 451 | 2.789713875 | 10.0.0.3 | 10.0.0.1    | TCP      | 76     | [TCP Retransmission] [TCP Port numbers reused] 36468 → 80 [SYN] Seq=0 Win=42340 Len=0 MSS=1460 SACK_PERM=1 |
| 452 | 2.789717372 | 10.0.0.3 | 10.0.0.1    | TCP      | 76     | [TCP Retransmission] [TCP Port numbers reused] 36468 → 80 [SYN] Seq=0 Win=42340 Len=0 MSS=1460 SACK_PERM=1 |
| 453 | 2.791407964 | 10.0.0.3 | 10.0.0.1    | Open     | 160    | Type: OPT_PACKET_IN  |
| 454 | 2.792987989 | 10.0.0.3 | 10.0.0.1    | Open     | 254    | Type: OPT_PACKET_OUT   |
| 457 | 2.793561920 | 10.0.0.3 | 10.0.0.1    | TCP      | 76     | [TCP Retransmission] [TCP Port numbers reused] 36468 → 80 [SYN] Seq=0 Win=42340 Len=0 MSS=1460 SACK_PERM=1 |
| 458 | 2.793565944 | 10.0.0.3 | 10.0.0.1    | TCP      | 76     | [TCP Retransmission] [TCP Port numbers reused] 36468 → 80 [SYN] Seq=0 Win=42340 Len=0 MSS=1460 SACK_PERM=1 |
| 459 | 2.793565944 | 10.0.0.3 | 10.0.0.1    | TCP      | 76     | [TCP Retransmission] [TCP Port numbers reused] 36468 → 80 [SYN] Seq=0 Win=42340 Len=0 MSS=1460 SACK_PERM=1 |
| 460 | 2.793565944 | 10.0.0.3 | 10.0.0.1    | TCP      | 76     | [TCP Retransmission] [TCP Port numbers reused] 36468 → 80 [SYN] Seq=0 Win=42340 Len=0 MSS=1460 SACK_PERM=1 |
| 461 | 3.886229984 | 10.0.0.3 | 10.0.0.1    | TCP      | 76     | [TCP Retransmission] [TCP Port numbers reused] 36468 → 80 [SYN] Seq=0 Win=42340 Len=0 MSS=1460 SACK_PERM=1 |
| 462 | 3.886235441 | 10.0.0.3 | 10.0.0.1    | TCP      | 76     | [TCP Retransmission] [TCP Port numbers reused] 36468 → 80 [SYN] Seq=0 Win=42340 Len=0 MSS=1460 SACK_PERM=1 |
| 463 | 3.886235441 | 10.0.0.3 | 10.0.0.1    | TCP      | 76     | [TCP Retransmission] [TCP Port numbers reused] 36468 → 80 [SYN] Seq=0 Win=42340 Len=0 MSS=1460 SACK_PERM=1 |
| 464 | 3.886235441 | 10.0.0.3 | 10.0.0.1    | TCP      | 76     | [TCP Retransmission] [TCP Port numbers reused] 36468 → 80 [SYN] Seq=0 Win=42340 Len=0 MSS=1460 SACK_PERM=1 |
| 465 | 3.886235441 | 10.0.0.3 | 10.0.0.1    | TCP      | 76     | [TCP Retransmission] [TCP Port numbers reused] 36468 → 80 [SYN] Seq=0 Win=42340 Len=0 MSS=1460 SACK_PERM=1 |
| 466 | 3.886235441 | 10.0.0.3 | 10.0.0.1    | TCP      | 76     | [TCP Retransmission] [TCP Port numbers reused] 36468 → 80 [SYN] Seq=0 Win=42340 Len=0 MSS=1460 SACK_PERM=1 |
| 467 | 3.886235441 | 10.0.0.3 | 10.0.0.1    | TCP      | 76     | [TCP Retransmission] [TCP Port numbers reused] 36468 → 80 [SYN] Seq=0 Win=42340 Len=0 MSS=1460 SACK_PERM=1 |
| 468 | 3.886235441 | 10.0.0.3 | 10.0.0.1    | TCP      | 76     | [TCP Retransmission] [TCP Port numbers reused] 36468 → 80 [SYN] Seq=0 Win=42340 Len=0 MSS=1460 SACK_PERM=1 |
| 469 | 3.886235441 | 10.0.0.3 | 10.0.0.1    | TCP      | 76     | [TCP Retransmission] [TCP Port numbers reused] 36468 → 80 [SYN] Seq=0 Win=42340 Len=0 MSS=1460 SACK_PERM=1 |
| 470 | 3.886235441 | 10.0.0.3 | 10.0.0.1    | TCP      | 76     | [TCP Retransmission] [TCP Port numbers reused] 36468 → 80 [SYN] Seq=0 Win=42340 Len=0 MSS=1460 SACK_PERM=1 |
| 471 | 3.886235441 | 10.0.0.3 | 10.0.0.1    | TCP      | 76     | [TCP Retransmission] [TCP Port numbers reused] 36468 → 80 [SYN] Seq=0 Win=42340 Len=0 MSS=1460 SACK_PERM=1 |
| 472 | 3.886235441 | 10.0.0.3 | 10.0.0.1    | TCP      | 76     | [TCP Retransmission] [TCP Port numbers reused] 36468 → 80 [SYN] Seq=0 Win=42340 Len=0 MSS=1460 SACK_PERM=1 |

## 4.2 Bloqueo de paquetes entre dos hosts elegidos arbitrariamente

En este trabajo practico decidimos bloquear la conexi3n entre el host 1 y el host 4, aunque al haber hecho esta configuraci3n a traves de un archivo de properties, el cambio de los mismos se puede realizar facilmente.

En la siguiente imagen, vemos la salida del comando pingall, con el que se puede evaluar la llegada de cada uno de los hosts. Aqui notamos como desde host\_1 no se puede acceder a host\_4 y viceversa.

```

mininet> pingall
*** Ping: testing ping reachability
host_1 -> host_2 host_3 X
host_2 -> host_1 host_3 host_4
host_3 -> host_1 host_2 host_4
host_4 -> X host_2 host_3

```

### 4.3 Bloqueo de paquetes UDP provenientes del host1 con puerto destino 5001

Para la tercera prueba creamos server en el host 1 que responda los mensajes udp al puerto 5001.

```

"host: host_3"
root@salvador-ubuntu:/home/salvador/fiuba/Redes/TP2# iperf -s -p 5001 -u
-----
Server listening on UDP port 5001
UDP buffer size: 208 KByte (default)
-----

```

Luego nos intentamos conectar desde el host 3, pasando por el firewall

```

"host: host_1"
root@salvador-ubuntu:/home/salvador/fiuba/Redes/TP2# iperf -c 10.0.0.3 -p 5001 -u
-----
Client connecting to 10.0.0.3, UDP port 5001
Sending 1470 byte datagrams, IPG target: 11215.21 us (kalman adjust)
UDP buffer size: 208 KByte (default)
-----
[ 1] local 10.0.0.1 port 56320 connected with 10.0.0.3 port 5001
[ ID] Interval      Transfer    Bandwidth
[ 1] 0.0000-0.0153 sec 1.25 MBytes 1.05 Mbits/sec
[ 1] Sent 896 datagrams
[ 5] WARNING: did not receive ack of last datagram after 10 tries.
root@salvador-ubuntu:/home/salvador/fiuba/Redes/TP2#

```

En wireshark podemos ver los reintentos de `iperf` de enviar los mensajes udp sin éxito. También se observan dos mensajes del protocolo OpenFlow relacionado al envío de paquetes. `PACKET_IN` es el mensaje que un switch envía al controlador cuando no puede resolver el forwarding. `PACKET_OUT` es la respuesta del controlador luego de realizar las operaciones de routing. En nuestro caso bajo el algoritmo *l2\_learning*.

| No.  | Time         | Source   | Destination | Protocol | Length | Info                  |
|------|--------------|----------|-------------|----------|--------|-----------------------|
| 7205 | 75.152744499 | 10.0.0.1 | 10.0.0.3    | UDP      | 1514   | 56320 → 5001 Len=1470 |
| 7206 | 75.153631181 | 10.0.0.1 | 10.0.0.3    | OpenFlow | 1598   | Type: OFPT_PACKET_IN  |
| 7207 | 75.156295556 | 10.0.0.1 | 10.0.0.3    | OpenFlow | 1692   | Type: OFPT_PACKET_OUT |
| 7210 | 75.156819415 | 10.0.0.1 | 10.0.0.3    | UDP      | 1514   | 56320 → 5001 Len=1470 |
| 7211 | 75.156823973 | 10.0.0.1 | 10.0.0.3    | UDP      | 1514   | 56320 → 5001 Len=1470 |
| 7212 | 75.157342730 | 10.0.0.1 | 10.0.0.3    | OpenFlow | 1598   | Type: OFPT_PACKET_IN  |
| 7215 | 75.160990590 | 10.0.0.1 | 10.0.0.3    | OpenFlow | 1692   | Type: OFPT_PACKET_OUT |
| 7218 | 75.161823498 | 10.0.0.1 | 10.0.0.3    | UDP      | 1514   | 56320 → 5001 Len=1470 |
| 7219 | 75.161827802 | 10.0.0.1 | 10.0.0.3    | UDP      | 1514   | 56320 → 5001 Len=1470 |
| 7220 | 75.164157440 | 10.0.0.1 | 10.0.0.3    | UDP      | 1514   | 56320 → 5001 Len=1470 |
| 7221 | 75.164215561 | 10.0.0.1 | 10.0.0.3    | UDP      | 1514   | 56320 → 5001 Len=1470 |
| 7222 | 75.164368818 | 10.0.0.1 | 10.0.0.3    | UDP      | 1514   | 56320 → 5001 Len=1470 |
| 7223 | 75.164373815 | 10.0.0.1 | 10.0.0.3    | UDP      | 1514   | 56320 → 5001 Len=1470 |
| 7224 | 75.164408098 | 10.0.0.1 | 10.0.0.3    | UDP      | 1514   | 56320 → 5001 Len=1470 |
| 7225 | 75.164409895 | 10.0.0.1 | 10.0.0.3    | UDP      | 1514   | 56320 → 5001 Len=1470 |
| 7226 | 75.164580709 | 10.0.0.1 | 10.0.0.3    | UDP      | 1514   | 56320 → 5001 Len=1470 |
| 7227 | 75.164583867 | 10.0.0.1 | 10.0.0.3    | UDP      | 1514   | 56320 → 5001 Len=1470 |

## 5 Preguntas a responder

### 5.1 ¿Cuál es la diferencia entre un Switch y un router? ¿Qué tienen en común?

La diferencia principal entre ambos es que un switch permite conectar dispositivos dentro de una misma red local mientras que un router permite conectar múltiples switches y sus respectivas redes

para formar una red mas grande, es decir que permite a un dispositivo conectarse con el resto del internet, otras redes. Por ejemplo un conjunto de routers permite conectar una computadora con un servidor.

Otra de las diferencias es la capa en la que funcionan ambos dispositivos. El router funciona en la capa física, la capa de enlace y la capa de red. En el caso de los switches, la capa física quedaría excluida, operando exclusivamente en la capa de enlace de datos y la de red.

Lo que tienen en común los switches y los routers es que permiten conectar distintos dispositivos entre sí.

## 5.2 ¿Cuál es la diferencia entre un Switch convencional y un Switch OpenFlow?

La diferencia entre un switch convencional y un switch que implementa el protocolo de comunicacion OpenFlow es que el primero contiene en un monolito interno ambas capas *control plane* y *data plane*. De esta manera el routing se realiza de una manera distribuida por router.

En cambio un Switch que implementa OpenFlow permite separar estas dos capas. OpenFlow es un protocolo que opera entre los routers que lo implementen y un dispositivo central que los controla. De esta manera, el plano de control es delegado en un controlador el cual contiene información completa de la topología de la red y mediante la utilización del protocolo OpenFlow puede leer y modificar la configuración de los routers de manera centralizada.

La diferenciación y desacoplamiento de estas capas en un elemento central en el concepto de SDN. Es necesario que exista un elemento central que contenga el plano de control para poder definir una red mediante software.

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

Teóricamente, la implementación de switches que utilicen OpenFlow en todos los routers de Internet sería posible, incluso considerando el escenario inter-sistema autónomo. Existen soluciones que integran el uso de BGP con OpenFlow.

Sin embargo, en la práctica, la implementación de OpenFlow en los switches de borde, donde se utiliza BGP, demanda velocidades elevadas para adaptarse a las modificaciones en la política de enrutamiento. Muchas implementaciones recurren a soluciones de hardware, como ASICs, para lograr las velocidades necesarias. En un escenario con el uso de OpenFlow, no solo se enfrenta la dificultad de resolver problemas con múltiples switches de frontera a nivel de software en un controlador central, sino que también se requiere un alto rendimiento en la comunicación entre los switches y dicho controlador.

Otra dificultad que se presenta en la integracion de BGP y OpenFlow es que el primero opera a nivel de red completa, tomando decisiones en base a la informacion de la red como los prefijos y la topologia de los sistemas autonomos conectados. OpenFlow esta basado en el forwarding basado en flujos. Para tomar decisiones, utiliza campos de paquetes especificos y llega a un nivel de granularidad que no esta presente en BGP. En un contexto del uso de OpenFlow en routers a nivel ISP, la cantidad de entradas en una tabla con intencion de forwarding orientado a flujos no sería prácticamente viable.