75.43 Introducción a los sistemas distribuidos

TP 3 Software-Defined Networks

Nombre	**Padrón**	**Email**
Israel, Pablo	95849	pabloisrael94@gmail.com
Lazzari, Santiago	96735	santilazzari1994@gmail.com
Mintrone, Luciano Guido	95463	lucianomintrone@gmail.com
Ortiz, Luciano	100323	lnortiz@fi.uba.ar
Rossini, Federico	97161	frossini@fi.uba.ar

Link al repositorio: https://github.com/lucianoMintrone/tp-openflow

Introducción

El objetivo de este trabajo práctico es familiarizarse con los desafíos por los cuales surgen las SDNs. Se pide construir una topología parametrizable en la que se utilizará OpenFlow para implementar un Firewall a nivel de capa de enlace. Para poder plantear este escenario se emulará el comportamiento de la topología a través de mininet.

Herramientas utilizadas

Mininet

Mininet es un emulador de software para hacer prototipos de topologías de redes en una sola computadora. Permite crear rápidamente una red virtual y operar con switches OpenFlow.

Pox

Pox es un sistema open source que usa python para definir un SDN utilizando el protocolo Openflow. Nos permite crear fácilmente controladores donde podemos definir distintas reglas.

VirtualBox

Es un sistema de virtualización compatible con sistema operativo host MacOs y sistema operativo guest Ubuntu.

Implementación

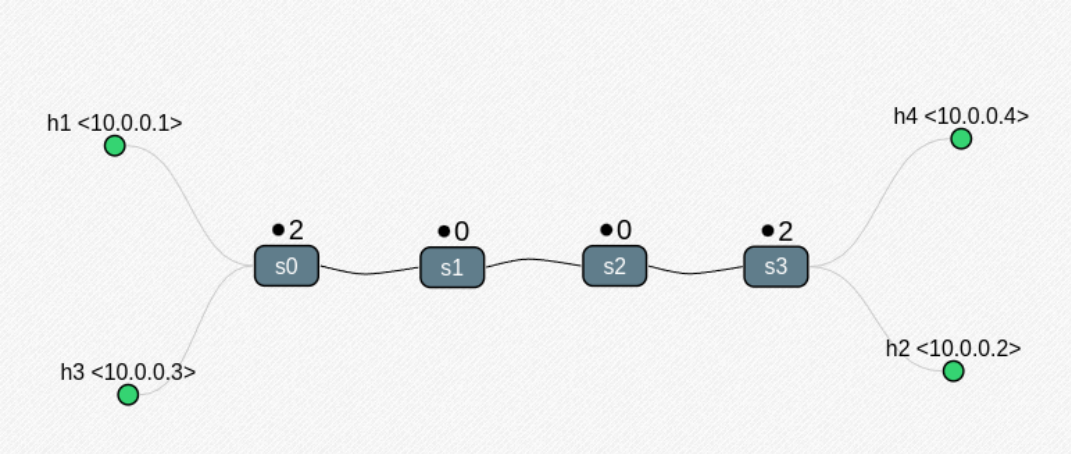
Para crear la topología propuesta por el trabajo práctico, utilizamos la clase “Topo” que provee Mininet, de la cual extendemos para crear una propia que cumpla las condiciones. Las direcciones IP y MAC de cada host siguen el patrón “10.0.0.x” y “00:00:00:00:00:0x” respectivamente, en donde x es el número del host (de 1 a 4). Para los switches, solo se les setea manualmente su nombre, comenzando por s0.

En cuanto al controlador, en primer lugar se utilizó el complemento que ya trae POX “forwarding.l2_learning” para hacer que los switches actualicen sus tablas dinámicamente.

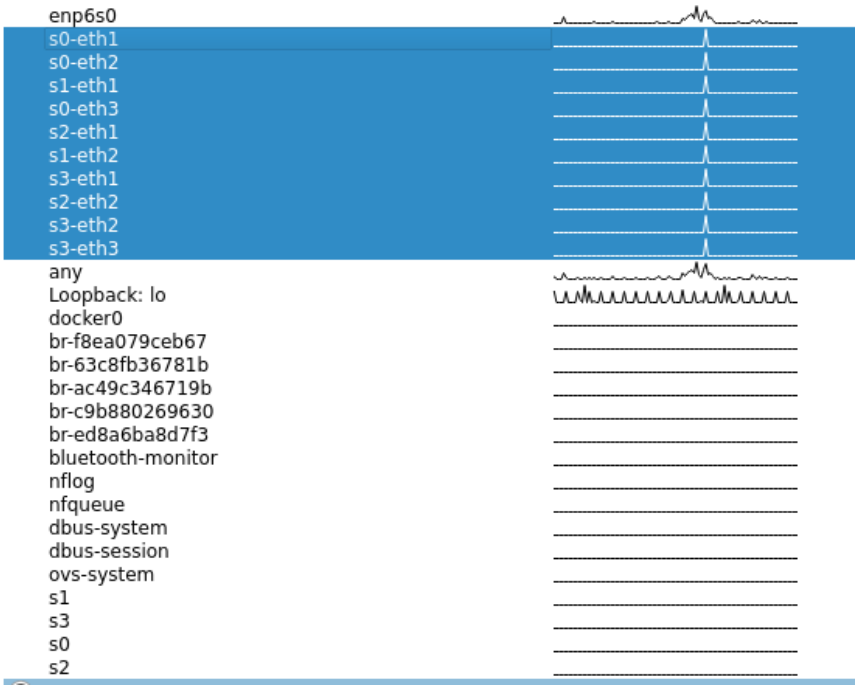
Para la implementación del firewall nos basamos en el archivo sugerido por el enunciado y en el mismo se implementaron las tres reglas. Para saber en qué switch se deben instalar las reglas y para saber qué dos hosts no se deben poder comunicar entre sí (tercera regla), se lee un archivo de configuración JSON.

Resultados de simulaciones

Para los siguientes ejemplos se probó con una topología de 4 switches en cadena, con 2 hosts en cada extremo, cuyos dump y links producen la siguiente visualización:



Se configuró al switch s0 para que actúe con las reglas del firewall.
Dicha topología nos permite capturar 10 interfaces distintas para analizar tráfico usando Wireshark (remarcadas en azul).



Cada una de las interfaces representa un enlace entre el switch y un host, o entre el switch y otro switch, por lo tanto dependiendo el trafico que estemos queriendo capturar podemos elegir un conjunto de interfaces diferente sobre el cual capturar.

Al correr el comando para iniciar el controlador vemos lo siguiente:

```
mininet@mininet-vm:~/tp-openflow$ python3 ../pox/pox.py forwarding.l2_learning firewall
POX 0.7.0 (gar) / Copyright 2011-2020 James McCauley, et al.
INFO:firewall:Enabling Firewall Module
WARNING:version:Support for Python 3 is experimental.
INFO:core:POX 0.7.0 (gar) is up.
INFO:openflow.of_01:[00-00-00-00-00-01 2] connected
INFO:firewall:Switch s1 connecting
INFO:openflow.of_01:[00-00-00-00-00-03 3] connected
INFO:firewall:Switch s3 connecting
INFO:openflow.of_01:[5e-29-9e-f7-2c-4d 4] connected
INFO:firewall:Switch s0 connecting
INFO:firewall:Firewall rules installed on switch s0
INFO:openflow.of_01:[00-00-00-00-00-02 5] connected
INFO:firewall:Switch s2 connecting
```

Mientras que mininet nos informa lo siguiente:

```
mininet@mininet-vm:~/tp-openflow$ sudo mn --custom tp3-topo.py --topo TP3topo --controller=remote
Numero de switches: 4
*** 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
*** Adding links:
(h1, s0) (h3, s0) (s0, s1) (s1, s2) (s2, s3) (s3, h2) (s3, h4)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
c0
*** Starting 4 switches
s0 s1 s2 s3 ...
*** Starting CLI:
mininet>
```

A continuación mostramos los resultados de las diferentes pruebas:

1) La primer prueba es utilizar el comando pingall y analizar los resultados y su captura en wireshark.

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> X h3 h4
h2 -> X h3 h4
h3 -> h1 h2 h4
h4 -> h1 h2 h3
*** Results: 16% dropped (10/12 received)
```

Como era de esperarse, debido a la regla que bloquea cualquier tipo de comunicación entre h1 y h2, estos dos no reciben la respuesta de ping entre ellos y por lo tanto no figura alcance entre los mismos.

icmp

No.	Time	Source	Destination	Protocol	Lengt	Info
1	0.000000000	10.0.0.1	10.0.0.2	ICMP	98	Echo (ping) request id=0x4224, seq=1/256, ttl=64 (no response found!)
5	10.013408642	10.0.0.1	10.0.0.4	ICMP	98	Echo (ping) request id=0x68df, seq=1/256, ttl=64 (reply in 7)
6	10.006520529	10.0.0.1	10.0.0.3	ICMP	98	Echo (ping) request id=0x423c, seq=1/256, ttl=64 (reply in 9)
7	10.022796562	10.0.0.4	10.0.0.1	ICMP	98	Echo (ping) reply id=0x68df, seq=1/256, ttl=64 (request in 5)
9	10.006545212	10.0.0.3	10.0.0.1	ICMP	98	Echo (ping) reply id=0x423c, seq=1/256, ttl=64 (request in 6)
11	10.004463890	10.0.0.1	10.0.0.3	ICMP	98	Echo (ping) request id=0x423c, seq=1/256, ttl=64 (reply in 12)
12	10.008136588	10.0.0.3	10.0.0.1	ICMP	98	Echo (ping) reply id=0x423c, seq=1/256, ttl=64 (request in 11)
13	10.011616177	10.0.0.1	10.0.0.4	ICMP	98	Echo (ping) request id=0x68df, seq=1/256, ttl=64 (reply in 14)
14	10.024108523	10.0.0.4	10.0.0.1	ICMP	98	Echo (ping) reply id=0x68df, seq=1/256, ttl=64 (request in 13)
25	13.115474877	10.0.0.2	10.0.0.3	ICMP	98	Echo (ping) request id=0xeb87, seq=1/256, ttl=64 (no response found!)
26	13.113612294	10.0.0.2	10.0.0.3	ICMP	98	Echo (ping) request id=0xeb87, seq=1/256, ttl=64 (reply in 27)
27	13.115515923	10.0.0.3	10.0.0.2	ICMP	98	Echo (ping) reply id=0xeb87, seq=1/256, ttl=64 (request in 26)
28	13.134651783	10.0.0.3	10.0.0.1	ICMP	98	Echo (ping) request id=0x02bc, seq=1/256, ttl=64 (reply in 31)
29	13.117692768	10.0.0.3	10.0.0.2	ICMP	98	Echo (ping) reply id=0xeb87, seq=1/256, ttl=64
30	13.144030436	10.0.0.3	10.0.0.2	ICMP	98	Echo (ping) request id=0xa655, seq=1/256, ttl=64 (reply in 32)
31	13.138642756	10.0.0.1	10.0.0.3	ICMP	98	Echo (ping) reply id=0x02bc, seq=1/256, ttl=64 (request in 28)
32	13.154469108	10.0.0.2	10.0.0.3	ICMP	98	Echo (ping) reply id=0xa655, seq=1/256, ttl=64 (request in 30)
33	13.161915589	10.0.0.3	10.0.0.4	ICMP	98	Echo (ping) request id=0xfe16, seq=1/256, ttl=64 (reply in 34)
34	13.173456463	10.0.0.4	10.0.0.3	ICMP	98	Echo (ping) reply id=0xfe16, seq=1/256, ttl=64 (request in 33)
35	13.142258645	10.0.0.3	10.0.0.2	ICMP	98	Echo (ping) request id=0xa655, seq=1/256, ttl=64 (reply in 37)
36	13.182580452	10.0.0.4	10.0.0.1	ICMP	98	Echo (ping) request id=0xe278, seq=1/256, ttl=64 (reply in 39)
37	13.156198456	10.0.0.2	10.0.0.3	ICMP	98	Echo (ping) reply id=0xa655, seq=1/256, ttl=64 (request in 35)
38	13.159840367	10.0.0.3	10.0.0.4	ICMP	98	Echo (ping) request id=0xfe16, seq=1/256, ttl=64 (reply in 40)
39	13.185540960	10.0.0.1	10.0.0.4	ICMP	98	Echo (ping) reply id=0xe278, seq=1/256, ttl=64 (request in 36)
40	13.175136225	10.0.0.4	10.0.0.3	ICMP	98	Echo (ping) reply id=0xfe16, seq=1/256, ttl=64 (request in 38)
41	13.199605173	10.0.0.4	10.0.0.3	ICMP	98	Echo (ping) request id=0x0b2d, seq=1/256, ttl=64 (no response found!)
42	13.200506349	10.0.0.4	10.0.0.3	ICMP	98	Echo (ping) request id=0x0b2d, seq=1/256, ttl=64 (reply in 43)
43	13.201356423	10.0.0.3	10.0.0.4	ICMP	98	Echo (ping) reply id=0x0b2d, seq=1/256, ttl=64 (request in 42)
44	13.200526286	10.0.0.3	10.0.0.4	ICMP	98	Echo (ping) reply id=0x0b2d, seq=1/256, ttl=64
45	13.136726206	10.0.0.3	10.0.0.1	ICMP	98	Echo (ping) request id=0x02bc, seq=1/256, ttl=64 (reply in 46)
46	13.136758706	10.0.0.1	10.0.0.3	ICMP	98	Echo (ping) reply id=0x02bc, seq=1/256, ttl=64 (request in 45)
47	13.184100976	10.0.0.4	10.0.0.1	ICMP	98	Echo (ping) request id=0xe278, seq=1/256, ttl=64 (reply in 48)
48	13.184134396	10.0.0.1	10.0.0.4	ICMP	98	Echo (ping) reply id=0xe278, seq=1/256, ttl=64 (request in 47)

Viendo la captura de wireshark (capturando los enlaces del switch 0 y filtrando los paquetes por ICMP), podemos ver los distintos request y replies de ping que se generaron para el pingall.
Al principio vemos que 10.0.0.1 hace un ping a 10.0.0.2 pero no obtiene respuesta en ningún momento. En cambio podemos ver que en algunas ocasiones otros hosts no obtuvieron respuesta en el primer ping pero si lo obtuvieron en un segundo intento.
Además el intento de ping entre h2 (10.0.0.2) y h1 (10.0.0.1) ni siquiera figura capturado. Esto puede deberse a que se haya bloqueado la solicitud de h2 mediante el protocolo ARP de la dirección IP de h1, y por lo tanto este ni siquiera haya podido hacer ping a h1.

2) En este ejemplo se ve que levantamos un server udp en el host 4 escuchando en el puerto 81 y un cliente en el host 3 haciendole pedidos. Como no hay ninguna restricción para este escenario los 895 paquetes enviados llegaron correctamente.

```
mininet> h4 iperf -s -p 81 -D -u
Running Iperf Server as a daemon
mininet> h3 iperf -c 10.0.0.4 -p 81 -u
------------------------------------------------------------
Client connecting to 10.0.0.4, UDP port 81
Sending 1470 byte datagrams, IPG target: 11215.21 us (kalman adjust)
UDP buffer size:   208 KByte (default)
------------------------------------------------------------
[   1] local 10.0.0.3 port 55639 connected with 10.0.0.4 port 81
[ ID] Interval        Transfer      Bandwidth
[   1] 0.0000-10.0155 sec   1.25 MBytes   1.05 Mbits/sec
[   1] Sent 896 datagrams
[   1] Server Report:
[ ID] Interval        Transfer      Bandwidth         Jitter    Lost/Total Datagram
s
[   1] 0.0000-9.9982 sec   1.25 MBytes   1.05 Mbits/sec    0.017 ms 0/895 (0%)
mininet>
```

Se capturó el tráfico de todos los enlaces del switch 0 y se capturó el doble de paquetes (895 * 2), debido a que el switch 0 recibió los paquetes del host 3 y luego los envió por otro enlace para que le lleguen al host 4. Además al final se pueden ver paquetes utilizados por protocolo ARP, de los cuales se capturaron 8 en total para este ejemplo (otros 4 aparecieron entre medio de los paquetes UDP). Por otra parte se puede ver que el server (10.0.0.4) le responde al cliente (10.0.0.3), confirmando que los paquetes llegan sin problema.

No.	Time	Source	Destination	Protocol	Length	Info
1761	9.779978466	10.0.0.3	10.0.0.4	UDP	1512	36075 → 81 Len=1470
1762	9.791072705	10.0.0.3	10.0.0.4	UDP	1512	36075 → 81 Len=1470
1763	9.802318805	10.0.0.3	10.0.0.4	UDP	1512	36075 → 81 Len=1470
1764	9.813443150	10.0.0.3	10.0.0.4	UDP	1512	36075 → 81 Len=1470
1765	9.824733432	10.0.0.3	10.0.0.4	UDP	1512	36075 → 81 Len=1470
1766	9.835949907	10.0.0.3	10.0.0.4	UDP	1512	36075 → 81 Len=1470
1767	9.847097988	10.0.0.3	10.0.0.4	UDP	1512	36075 → 81 Len=1470
1768	9.858374100	10.0.0.3	10.0.0.4	UDP	1512	36075 → 81 Len=1470
1769	9.869664695	10.0.0.3	10.0.0.4	UDP	1512	36075 → 81 Len=1470
1770	9.880782010	10.0.0.3	10.0.0.4	UDP	1512	36075 → 81 Len=1470
1771	9.892151124	10.0.0.3	10.0.0.4	UDP	1512	36075 → 81 Len=1470
1772	9.903321677	10.0.0.3	10.0.0.4	UDP	1512	36075 → 81 Len=1470
1773	9.914577271	10.0.0.3	10.0.0.4	UDP	1512	36075 → 81 Len=1470
1774	9.925814096	10.0.0.3	10.0.0.4	UDP	1512	36075 → 81 Len=1470
1775	9.936937273	10.0.0.3	10.0.0.4	UDP	1512	36075 → 81 Len=1470
1776	9.948127978	10.0.0.3	10.0.0.4	UDP	1512	36075 → 81 Len=1470
1777	9.959272878	10.0.0.3	10.0.0.4	UDP	1512	36075 → 81 Len=1470
1778	9.970615400	10.0.0.3	10.0.0.4	UDP	1512	36075 → 81 Len=1470
1779	9.892161010	10.0.0.3	10.0.0.4	UDP	1512	36075 → 81 Len=1470
1780	9.903329873	10.0.0.3	10.0.0.4	UDP	1512	36075 → 81 Len=1470
1781	9.914587224	10.0.0.3	10.0.0.4	UDP	1512	36075 → 81 Len=1470
1782	9.925824713	10.0.0.3	10.0.0.4	UDP	1512	36075 → 81 Len=1470
1783	9.936943792	10.0.0.3	10.0.0.4	UDP	1512	36075 → 81 Len=1470
1784	9.948134603	10.0.0.3	10.0.0.4	UDP	1512	36075 → 81 Len=1470
1785	9.959282676	10.0.0.3	10.0.0.4	UDP	1512	36075 → 81 Len=1470
1786	9.970634842	10.0.0.3	10.0.0.4	UDP	1512	36075 → 81 Len=1470
1787	9.981747045	10.0.0.3	10.0.0.4	UDP	1512	36075 → 81 Len=1470
1788	9.992930832	10.0.0.3	10.0.0.4	UDP	1512	36075 → 81 Len=1470
1789	10.004144566	10.0.0.3	10.0.0.4	UDP	1512	36075 → 81 Len=1470
1790	10.015360940	10.0.0.3	10.0.0.4	UDP	1512	36075 → 81 Len=1470
1791	10.028730197	10.0.0.4	10.0.0.3	UDP	170	81 → 36075 Len=128
1792	9.981738747	10.0.0.3	10.0.0.4	UDP	1512	36075 → 81 Len=1470
1793	9.992922527	10.0.0.3	10.0.0.4	UDP	1512	36075 → 81 Len=1470
1794	10.004136598	10.0.0.3	10.0.0.4	UDP	1512	36075 → 81 Len=1470
1795	10.015353189	10.0.0.3	10.0.0.4	UDP	1512	36075 → 81 Len=1470
1796	10.030292211	10.0.0.4	10.0.0.3	UDP	170	81 → 36075 Len=128
1797	15.230489166	00:00:00_00:00:04	00:00:00_00:00:03	ARP	42	Who has 10.0.0.3? Tell 10.0.0.4
1798	15.232992573	00:00:00_00:00:03	00:00:00_00:00:04	ARP	42	10.0.0.3 is at 00:00:00:00:00:03
1799	15.231636585	00:00:00_00:00:04	00:00:00_00:00:03	ARP	42	Who has 10.0.0.3? Tell 10.0.0.4
1800	15.231646911	00:00:00 00:00:03	00:00:00 00:00:04	ARP	42	10.0.0.3 is at 00:00:00:00:00:03

3) Ahora levantamos el mismo escenario anterior pero en el puerto 80 y vemos que los mensajes no llegan a destino ya que la primera regla lo prohíbe. Tanto para udp como tcp.

```
mininet> h4 iperf -s -p 80 -D -u
Running Iperf Server as a daemon
mininet> h3 iperf -c 10.0.0.4 -p 80 -u
------------------------------------------------------------
Client connecting to 10.0.0.4, 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 60159 connected with 10.0.0.4 port 80
[ ID] Interval       Transfer     Bandwidth
[   1] 0.0000-10.0568 sec   1.25 MBytes   1.05 Mbits/sec
[   1] Sent 896 datagrams
[   3] WARNING: did not receive ack of last datagram after 10 tries.
mininet>
```

Haciendo captura de paquetes con wireshark nuevamente en el switch 0, podemos ver que en contraste con el escenario anterior, esta vez no encontramos el doble de paquetes debido a que una vez que los paquetes llegan al switch estos se descartan y no son enviados por otro enlace. Debido a esto esta vez encontramos unos 919 paquetes, entre los cuales algunos son de protocolo ARP.

Por último, no se vieron paquetes desde el server al cliente, confirmando que el server no respondió nada y por lo tanto no recibió nada.

No.	Time	Source	Destination	Protocol	Length	Info
884	9.847250807	10.0.0.3	10.0.0.4	UDP	1512	57885 → 80 Len=1470
885	9.858457008	10.0.0.3	10.0.0.4	UDP	1512	57885 → 80 Len=1470
886	9.869700365	10.0.0.3	10.0.0.4	UDP	1512	57885 → 80 Len=1470
887	9.880890922	10.0.0.3	10.0.0.4	UDP	1512	57885 → 80 Len=1470
888	9.892111301	10.0.0.3	10.0.0.4	UDP	1512	57885 → 80 Len=1470
889	9.903309192	10.0.0.3	10.0.0.4	UDP	1512	57885 → 80 Len=1470
890	9.914452816	10.0.0.3	10.0.0.4	UDP	1512	57885 → 80 Len=1470
891	9.925717195	10.0.0.3	10.0.0.4	UDP	1512	57885 → 80 Len=1470
892	9.936943440	10.0.0.3	10.0.0.4	UDP	1512	57885 → 80 Len=1470
893	9.948186308	10.0.0.3	10.0.0.4	UDP	1512	57885 → 80 Len=1470
894	9.959314777	10.0.0.3	10.0.0.4	UDP	1512	57885 → 80 Len=1470
895	9.970590400	10.0.0.3	10.0.0.4	UDP	1512	57885 → 80 Len=1470
896	9.981792690	10.0.0.3	10.0.0.4	UDP	1512	57885 → 80 Len=1470
897	9.993016492	10.0.0.3	10.0.0.4	UDP	1512	57885 → 80 Len=1470
898	10.004250560	10.0.0.3	10.0.0.4	UDP	1512	57885 → 80 Len=1470
899	10.015444539	10.0.0.3	10.0.0.4	UDP	1512	57885 → 80 Len=1470
900	10.129354047	10.0.0.3	10.0.0.4	UDP	1512	57885 → 80 Len=1470
901	10.229498533	10.0.0.3	10.0.0.4	UDP	1512	57885 → 80 Len=1470
902	10.329714886	10.0.0.3	10.0.0.4	UDP	1512	57885 → 80 Len=1470
903	10.429919995	10.0.0.3	10.0.0.4	UDP	1512	57885 → 80 Len=1470
904	10.530128525	10.0.0.3	10.0.0.4	UDP	1512	57885 → 80 Len=1470
905	10.630356611	10.0.0.3	10.0.0.4	UDP	1512	57885 → 80 Len=1470
906	10.730637008	10.0.0.3	10.0.0.4	UDP	1512	57885 → 80 Len=1470
907	10.830882205	10.0.0.3	10.0.0.4	UDP	1512	57885 → 80 Len=1470
908	10.931151357	10.0.0.3	10.0.0.4	UDP	1512	57885 → 80 Len=1470
909	11.031341799	10.0.0.3	10.0.0.4	UDP	1512	57885 → 80 Len=1470
910	11.131499975	10.0.0.3	10.0.0.4	UDP	1512	57885 → 80 Len=1470
911	11.231752016	10.0.0.3	10.0.0.4	UDP	1512	57885 → 80 Len=1470
912	11.332075434	10.0.0.3	10.0.0.4	UDP	1512	57885 → 80 Len=1470
913	11.432279076	10.0.0.3	10.0.0.4	UDP	1512	57885 → 80 Len=1470
914	11.532504718	10.0.0.3	10.0.0.4	UDP	1512	57885 → 80 Len=1470
915	11.632675604	10.0.0.3	10.0.0.4	UDP	1512	57885 → 80 Len=1470
916	11.732956979	10.0.0.3	10.0.0.4	UDP	1512	57885 → 80 Len=1470
917	11.833226131	10.0.0.3	10.0.0.4	UDP	1512	57885 → 80 Len=1470
918	11.933514839	10.0.0.3	10.0.0.4	UDP	1512	57885 → 80 Len=1470
919	12.033783013	10.0.0.3	10.0.0.4	UDP	1512	57885 → 80 Len=1470

```
------------------------------------------------------------
mininet> h4 iperf -s -p 80 -D
Running Iperf Server as a daemon
mininet> h3 iperf -c 10.0.0.4 -p 80
tcp connect failed: Connection timed out
------------------------------------------------------------
Client connecting to 10.0.0.4, TCP port 80
TCP window size: -1.00 Byte (default)
------------------------------------------------------------
```

No.	Time	Source	Destination	Protocol	Lengtl	Info
1	0.000000000	00:00:00_00:00:03	Broadcast	ARP	42	Who has 10.0.0.4? Tell 10.0.0.3
2	0.009783972	00:00:00_00:00:04	00:00:00_00:00:03	ARP	42	10.0.0.4 is at 00:00:00:00:00:04
3	-0.000990962	00:00:00_00:00:03	Broadcast	ARP	42	Who has 10.0.0.4? Tell 10.0.0.3
4	-0.000005867	00:00:00_00:00:03	Broadcast	ARP	42	Who has 10.0.0.4? Tell 10.0.0.3
5	0.011823094	00:00:00_00:00:04	00:00:00_00:00:03	ARP	42	10.0.0.4 is at 00:00:00:00:00:04
6	0.011848027	10.0.0.3	10.0.0.4	TCP	74	47624 → 80 [SYN] Seq=0 Win=42340 Len=0
7	1.006552847	10.0.0.3	10.0.0.4	TCP	74	[TCP Retransmission] 47624 → 80 [SYN]
8	3.026566896	10.0.0.3	10.0.0.4	TCP	74	[TCP Retransmission] 47624 → 80 [SYN]
9	7.182518327	10.0.0.3	10.0.0.4	TCP	74	[TCP Retransmission] 47624 → 80 [SYN]
10	15.374644545	10.0.0.3	10.0.0.4	TCP	74	[TCP Retransmission] 47624 → 80 [SYN]
11	31.502551999	10.0.0.3	10.0.0.4	TCP	74	[TCP Retransmission] 47624 → 80 [SYN]
12	36.624631168	00:00:00_00:00:03	00:00:00_00:00:04	ARP	42	Who has 10.0.0.4? Tell 10.0.0.3
13	36.635708716	00:00:00_00:00:04	00:00:00_00:00:03	ARP	42	10.0.0.4 is at 00:00:00:00:00:04
14	36.622493292	00:00:00_00:00:03	00:00:00_00:00:04	ARP	42	Who has 10.0.0.4? Tell 10.0.0.3
15	36.638378494	00:00:00_00:00:04	00:00:00_00:00:03	ARP	42	10.0.0.4 is at 00:00:00:00:00:04
16	65.038541470	10.0.0.3	10.0.0.4	TCP	74	[TCP Retransmission] 47624 → 80 [SYN]
17	70.158492995	00:00:00_00:00:03	00:00:00_00:00:04	ARP	42	Who has 10.0.0.4? Tell 10.0.0.3
18	70.168959441	00:00:00_00:00:04	00:00:00_00:00:03	ARP	42	10.0.0.4 is at 00:00:00:00:00:04
19	70.159920038	00:00:00_00:00:03	00:00:00_00:00:04	ARP	42	Who has 10.0.0.4? Tell 10.0.0.3
20	70.167898570	00:00:00_00:00:04	00:00:00_00:00:03	ARP	42	10.0.0.4 is at 00:00:00:00:00:04

En este caso se ve como h3 se intenta conectar con h4, pero jamás llega el SYN+ACK para establecer el handshake de TCP, y por eso se producen las retransmisiones, confirmando que los paquetes se descartan por tener como destino el puerto 80.

4) Para probar la segunda regla vamos a levantar un cliente en el host 1 y un server udp en el host 4 en el puerto 5001. También podemos observar como los mensajes son descartados por el switch.

```
mininet> h4 iperf -s -p 5001 -D -u
Running Iperf Server as a daemon
mininet> h1 iperf -c 10.0.0.4 -p 5001 -u
------------------------------------------------------------
Client connecting to 10.0.0.4, 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 49335 connected with 10.0.0.4 port 5001
[ ID] Interval       Transfer     Bandwidth
[  1] 0.0000-10.0043 sec  1.25 MBytes  1.05 Mbits/sec
[  1] Sent 895 datagrams
[  3] WARNING: did not receive ack of last datagram after 10 tries.
mininet>
```

La captura de wireshark es muy similar a la captura del ejemplo anterior, pero en este caso el origen es el host 1 (10.0.0.1).
Los paquetes no se forwardean, y el server no responde.

No.	Time	Source	Destination	Protocol	Length	Info
884	9.847331472	10.0.0.1	10.0.0.4	UDP	1512	50292 → 5001 Len=1470
885	9.858521541	10.0.0.1	10.0.0.4	UDP	1512	50292 → 5001 Len=1470
886	9.869740942	10.0.0.1	10.0.0.4	UDP	1512	50292 → 5001 Len=1470
887	9.880892388	10.0.0.1	10.0.0.4	UDP	1512	50292 → 5001 Len=1470
888	9.892169967	10.0.0.1	10.0.0.4	UDP	1512	50292 → 5001 Len=1470
889	9.903394746	10.0.0.1	10.0.0.4	UDP	1512	50292 → 5001 Len=1470
890	9.914678680	10.0.0.1	10.0.0.4	UDP	1512	50292 → 5001 Len=1470
891	9.925790527	10.0.0.1	10.0.0.4	UDP	1512	50292 → 5001 Len=1470
892	9.937055395	10.0.0.1	10.0.0.4	UDP	1512	50292 → 5001 Len=1470
893	9.948465461	10.0.0.1	10.0.0.4	UDP	1512	50292 → 5001 Len=1470
894	9.959475620	10.0.0.1	10.0.0.4	UDP	1512	50292 → 5001 Len=1470
895	9.970732665	10.0.0.1	10.0.0.4	UDP	1512	50292 → 5001 Len=1470
896	9.981838156	10.0.0.1	10.0.0.4	UDP	1512	50292 → 5001 Len=1470
897	9.993119157	10.0.0.1	10.0.0.4	UDP	1512	50292 → 5001 Len=1470
898	10.004260826	10.0.0.1	10.0.0.4	UDP	1512	50292 → 5001 Len=1470
899	10.015479249	10.0.0.1	10.0.0.4	UDP	1512	50292 → 5001 Len=1470
900	10.119652141	10.0.0.1	10.0.0.4	UDP	1512	50292 → 5001 Len=1470
901	10.219831828	10.0.0.1	10.0.0.4	UDP	1512	50292 → 5001 Len=1470
902	10.320113691	10.0.0.1	10.0.0.4	UDP	1512	50292 → 5001 Len=1470
903	10.420396043	10.0.0.1	10.0.0.4	UDP	1512	50292 → 5001 Len=1470
904	10.520628040	10.0.0.1	10.0.0.4	UDP	1512	50292 → 5001 Len=1470
905	10.620911859	10.0.0.1	10.0.0.4	UDP	1512	50292 → 5001 Len=1470
906	10.721194211	10.0.0.1	10.0.0.4	UDP	1512	50292 → 5001 Len=1470
907	10.821413497	10.0.0.1	10.0.0.4	UDP	1512	50292 → 5001 Len=1470
908	10.921575095	10.0.0.1	10.0.0.4	UDP	1512	50292 → 5001 Len=1470
909	11.021917091	10.0.0.1	10.0.0.4	UDP	1512	50292 → 5001 Len=1470
910	11.122202865	10.0.0.1	10.0.0.4	UDP	1512	50292 → 5001 Len=1470
911	11.222515528	10.0.0.1	10.0.0.4	UDP	1512	50292 → 5001 Len=1470
912	11.322655126	10.0.0.1	10.0.0.4	UDP	1512	50292 → 5001 Len=1470
913	11.422933078	10.0.0.1	10.0.0.4	UDP	1512	50292 → 5001 Len=1470
914	11.523265786	10.0.0.1	10.0.0.4	UDP	1512	50292 → 5001 Len=1470
915	11.623600448	10.0.0.1	10.0.0.4	UDP	1512	50292 → 5001 Len=1470
916	11.723897467	10.0.0.1	10.0.0.4	UDP	1512	50292 → 5001 Len=1470
917	11.824222841	10.0.0.1	10.0.0.4	UDP	1512	50292 → 5001 Len=1470
918	11.924554570	10.0.0.1	10.0.0.4	UDP	1512	50292 → 5001 Len=1470
919	12.024865278	10.0.0.1	10.0.0.4	UDP	1512	50292 → 5001 Len=1470

5) Para la última regla levantamos un server en el host 2 en el puerto 8080 y un cliente en el host 1 y podemos ver como no se pueden comunicar entre ellos de ninguna manera.

```
mininet> h2 iperf -s -p 8080 -D -u
Running Iperf Server as a daemon
mininet> h1 iperf -c 10.0.0.2 -p 8080 -u
------------------------------------------------------------
Client connecting to 10.0.0.2, UDP port 8080
Sending 1470 byte datagrams, IPG target: 11215.21 us (kalman adjust)
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[  1] local 10.0.0.1 port 56714 connected with 10.0.0.2 port 8080
[ ID] Interval       Transfer     Bandwidth
[  1] 0.0000-10.0204 sec  1.25 MBytes  1.05 Mbits/sec
[  1] Sent 896 datagrams
[  3] WARNING: did not receive ack of last datagram after 10 tries.
mininet>
```

En este caso el host 1 intenta primero obtener la dirección IP del host 2 utilizando el protocolo ARP, por lo que en principio manda una solicitud en broadcast de quien tiene la direccion MAC 10.0.0.2.

Luego el host 2 escucha esa solicitud y manda una respuesta, pero el switch 0 debido a la regla de firewall descarta esa respuesta y por lo tanto no le deja saber al host 1 la dirección. De esta manera queda inhabilitada la comunicación entre h1 y h2.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	00:00:00_00:00:01	Broadcast	ARP	42	Who has 10.0.0.2? Tell 10.0.0.1
2	-0.000927886	00:00:00_00:00:01	Broadcast	ARP	42	Who has 10.0.0.2? Tell 10.0.0.1
3	0.000004889	00:00:00_00:00:01	Broadcast	ARP	42	Who has 10.0.0.2? Tell 10.0.0.1
4	0.004508902	00:00:00_00:00:02	00:00:00_00:00:01	ARP	42	10.0.0.2 is at 00:00:00:00:00:02
5	1.006093415	00:00:00_00:00:01	Broadcast	ARP	42	Who has 10.0.0.2? Tell 10.0.0.1
6	1.004814027	00:00:00_00:00:01	Broadcast	ARP	42	Who has 10.0.0.2? Tell 10.0.0.1
7	1.006098793	00:00:00_00:00:01	Broadcast	ARP	42	Who has 10.0.0.2? Tell 10.0.0.1
8	1.008509929	00:00:00_00:00:02	00:00:00_00:00:01	ARP	42	10.0.0.2 is at 00:00:00:00:00:02
9	2.028839446	00:00:00_00:00:01	Broadcast	ARP	42	Who has 10.0.0.2? Tell 10.0.0.1
10	2.029902751	00:00:00_00:00:01	Broadcast	ARP	42	Who has 10.0.0.2? Tell 10.0.0.1
11	2.029907151	00:00:00_00:00:01	Broadcast	ARP	42	Who has 10.0.0.2? Tell 10.0.0.1
12	2.031811811	00:00:00_00:00:02	00:00:00_00:00:01	ARP	42	10.0.0.2 is at 00:00:00:00:00:02
13	3.062074833	00:00:00_00:00:01	Broadcast	ARP	42	Who has 10.0.0.2? Tell 10.0.0.1
14	3.064395038	00:00:00_00:00:02	00:00:00_00:00:01	ARP	42	10.0.0.2 is at 00:00:00:00:00:02
15	3.061038416	00:00:00_00:00:01	Broadcast	ARP	42	Who has 10.0.0.2? Tell 10.0.0.1
16	3.062067500	00:00:00_00:00:01	Broadcast	ARP	42	Who has 10.0.0.2? Tell 10.0.0.1
17	4.082125760	00:00:00_00:00:01	Broadcast	ARP	42	Who has 10.0.0.2? Tell 10.0.0.1
18	4.080872282	00:00:00_00:00:01	Broadcast	ARP	42	Who has 10.0.0.2? Tell 10.0.0.1
19	4.082132115	00:00:00_00:00:01	Broadcast	ARP	42	Who has 10.0.0.2? Tell 10.0.0.1
20	4.084375078	00:00:00_00:00:02	00:00:00_00:00:01	ARP	42	10.0.0.2 is at 00:00:00:00:00:02
21	5.100828401	00:00:00_00:00:01	Broadcast	ARP	42	Who has 10.0.0.2? Tell 10.0.0.1
22	5.101751888	00:00:00_00:00:01	Broadcast	ARP	42	Who has 10.0.0.2? Tell 10.0.0.1
23	5.101757754	00:00:00_00:00:01	Broadcast	ARP	42	Who has 10.0.0.2? Tell 10.0.0.1
24	5.103991917	00:00:00_00:00:02	00:00:00_00:00:01	ARP	42	10.0.0.2 is at 00:00:00:00:00:02
25	6.139113492	00:00:00_00:00:01	Broadcast	ARP	42	Who has 10.0.0.2? Tell 10.0.0.1
26	6.142688154	00:00:00_00:00:02	00:00:00_00:00:01	ARP	42	10.0.0.2 is at 00:00:00:00:00:02
27	6.134226199	00:00:00_00:00:01	Broadcast	ARP	42	Who has 10.0.0.2? Tell 10.0.0.1
28	6.139096382	00:00:00_00:00:01	Broadcast	ARP	42	Who has 10.0.0.2? Tell 10.0.0.1
29	7.153897860	00:00:00_00:00:01	Broadcast	ARP	42	Who has 10.0.0.2? Tell 10.0.0.1
30	7.152822333	00:00:00_00:00:01	Broadcast	ARP	42	Who has 10.0.0.2? Tell 10.0.0.1
31	7.153905682	00:00:00_00:00:01	Broadcast	ARP	42	Who has 10.0.0.2? Tell 10.0.0.1
32	7.156005404	00:00:00_00:00:02	00:00:00_00:00:01	ARP	42	10.0.0.2 is at 00:00:00:00:00:02
33	8.173977765	00:00:00_00:00:01	Broadcast	ARP	42	Who has 10.0.0.2? Tell 10.0.0.1
34	8.172819129	00:00:00_00:00:01	Broadcast	ARP	42	Who has 10.0.0.2? Tell 10.0.0.1
35	8.173986565	00:00:00_00:00:01	Broadcast	ARP	42	Who has 10.0.0.2? Tell 10.0.0.1
36	8.177220969	00:00:00_00:00:02	00:00:00_00:00:01	ARP	42	10.0.0.2 is at 00:00:00:00:00:02
37	9.208048847	00:00:00_00:00:01	Broadcast	ARP	42	Who has 10.0.0.2? Tell 10.0.0.1
38	9.206999231	00:00:00_00:00:01	Broadcast	ARP	42	Who has 10.0.0.2? Tell 10.0.0.1
39	9.208055692	00:00:00_00:00:01	Broadcast	ARP	42	Who has 10.0.0.2? Tell 10.0.0.1
40	9.209647961	00:00:00_00:00:02	00:00:00_00:00:01	ARP	42	10.0.0.2 is at 00:00:00:00:00:02
41	10.222169420	00:00:00_00:00:01	Broadcast	ARP	42	Who has 10.0.0.2? Tell 10.0.0.1
42	10.223798355	00:00:00_00:00:02	00:00:00_00:00:01	ARP	42	10.0.0.2 is at 00:00:00:00:00:02
43	10.220815722	00:00:00_00:00:01	Broadcast	ARP	42	Who has 10.0.0.2? Tell 10.0.0.1
44	10.222164042	00:00:00_00:00:01	Broadcast	ARP	42	Who has 10.0.0.2? Tell 10.0.0.1
45	11.244818468	00:00:00_00:00:01	Broadcast	ARP	42	Who has 10.0.0.2? Tell 10.0.0.1
46	11.245690623	00:00:00_00:00:01	Broadcast	ARP	42	Who has 10.0.0.2? Tell 10.0.0.1
47	11.245696000	00:00:00_00:00:01	Broadcast	ARP	42	Who has 10.0.0.2? Tell 10.0.0.1
48	11.247120585	00:00:00 00:00:02	00:00:00 00:00:01	ARP	42	10.0.0.2 is at 00:00:00:00:00:02

También se probó utilizando TCP:

```
mininet> h1 iperf -s -p 8080 -D
Running Iperf Server as a daemon
mininet> h2 iperf -c 10.0.0.1 -p 8080
tcp connect failed: No route to host
------------------------------------------------------------
Client connecting to 10.0.0.1, TCP port 8080
TCP window size: -1.00 Byte (default)
------------------------------------------------------------
```

En la captura de wireshark se vé algo parecido a lo que sucede en UDP, en el que el host 2 nunca se entera de a dónde tiene que enviar los paquetes, y por lo tanto no se envían.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	00:00:00_00:00:02	Broadcast	ARP	42	Who has 10.0.0.1? Tell 10.0.0.2
2	0.000020533	00:00:00_00:00:01	00:00:00_00:00:02	ARP	42	10.0.0.1 is at 00:00:00:00:00:01
3	0.000007333	00:00:00_00:00:02	Broadcast	ARP	42	Who has 10.0.0.1? Tell 10.0.0.2
4	-0.001175766	00:00:00_00:00:02	Broadcast	ARP	42	Who has 10.0.0.1? Tell 10.0.0.2
5	1.001007107	00:00:00_00:00:02	Broadcast	ARP	42	Who has 10.0.0.1? Tell 10.0.0.2
6	1.001904209	00:00:00_00:00:02	Broadcast	ARP	42	Who has 10.0.0.1? Tell 10.0.0.2
7	1.001922297	00:00:00_00:00:01	00:00:00_00:00:02	ARP	42	10.0.0.1 is at 00:00:00:00:00:01
8	1.001911053	00:00:00_00:00:02	Broadcast	ARP	42	Who has 10.0.0.1? Tell 10.0.0.2
9	2.024761463	00:00:00_00:00:02	Broadcast	ARP	42	Who has 10.0.0.1? Tell 10.0.0.2
10	2.026041361	00:00:00_00:00:02	Broadcast	ARP	42	Who has 10.0.0.1? Tell 10.0.0.2
11	2.026030606	00:00:00_00:00:02	Broadcast	ARP	42	Who has 10.0.0.1? Tell 10.0.0.2
12	2.026062872	00:00:00_00:00:01	00:00:00_00:00:02	ARP	42	10.0.0.1 is at 00:00:00:00:00:01
13	3.052599433	00:00:00_00:00:02	Broadcast	ARP	42	Who has 10.0.0.1? Tell 10.0.0.2
14	3.052594056	00:00:00_00:00:02	Broadcast	ARP	42	Who has 10.0.0.1? Tell 10.0.0.2
15	3.052609211	00:00:00_00:00:01	00:00:00_00:00:02	ARP	42	10.0.0.1 is at 00:00:00:00:00:01
16	3.052150149	00:00:00_00:00:02	Broadcast	ARP	42	Who has 10.0.0.1? Tell 10.0.0.2
17	4.075601819	00:00:00_00:00:02	Broadcast	ARP	42	Who has 10.0.0.1? Tell 10.0.0.2
18	4.075947459	00:00:00_00:00:02	Broadcast	ARP	42	Who has 10.0.0.1? Tell 10.0.0.2
19	4.075942571	00:00:00_00:00:02	Broadcast	ARP	42	Who has 10.0.0.1? Tell 10.0.0.2
20	4.075955770	00:00:00_00:00:01	00:00:00_00:00:02	ARP	42	10.0.0.1 is at 00:00:00:00:00:01
21	5.103103858	00:00:00_00:00:02	Broadcast	ARP	42	Who has 10.0.0.1? Tell 10.0.0.2
22	5.104163758	00:00:00_00:00:02	Broadcast	ARP	42	Who has 10.0.0.1? Tell 10.0.0.2
23	5.104148603	00:00:00_00:00:02	Broadcast	ARP	42	Who has 10.0.0.1? Tell 10.0.0.2
24	5.104195047	00:00:00_00:00:01	00:00:00_00:00:02	ARP	42	10.0.0.1 is at 00:00:00:00:00:01

Readme

En el repositorio del código podemos encontrar el [ReadMe del proyecto].

Preguntas a responder

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

Las diferencias principales entre el Router y el Switch es que el router opera en la capa de red mientras que el switch opera en la capa de enlace.

La idea del router es que reciba un datagrama de la capa de red y pueda decidir a donde forwardearlo. Ya sea de forma centralizada o no centralizada. En general el router tiene una noción más grande de la red que los switches.

El switch tiene como función recibir un fragmento de la capa de enlace e insertarlo en otro enlace.

Las similitudes son que ambos hacen la operación de switching del paquete, es decir, de recibir de un enlace y depositarlo en otro enlace.

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

El switch Openflow tiene la capacidad de forwardear basándose en información de las capas de enlace, red y transporte. Es decir que puede comprender en los destinos MAC (enlace), ip destino y src (Red) y protocolo de transporte, puerto y tipo de protocolo (Transporte).

Dado el nivel de comprensión del paquete que llega, se pueden aplicar varias técnicas y distintos tratamientos a los paquetes como el caso del firewall.

Además, los switches Openflow se conectan a un controlador que orquestra las configuraciones de la red para tener un control centralizado de la misma.

Por otro lado los switches convencionales solo tienen visibilidad de la capa de enlace y tienen como tarea principal el agarrar un fragmento de un enlace y depositarlo en otro enlace.

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

Con las respuestas de la pregunta 1 y la pregunta 2, podemos concluir que los routers y los switches Openflow tienen un comportamiento equivalente, ya que no solo hacen switching, si no también toman operaciones de ruteo y además los switches Openflow tienen conocimiento de la capa de transporte (que los routers no tienen).

Si tenemos solo eso en cuenta parecería que serían equivalentes (si no mejores) los switches Openflow.

Ahora la diferencia principal radica en la naturaleza de la centralidad de los switches Openflow que dependen de un controlador central, mientras que los routers tienen la información implementada en el mismo router.

Esta centralidad hace que la implementación a gran escala de los switches Openflow sea como mínimo insegura, ya que la centralidad de las decisiones de la red puede ser vulnerada y eso impacta en todos los ruteos.

Dificultades

Instalación de la Virtual Machine

La mayoría de los integrantes del grupo tenemos computadoras mac por lo que no podemos instalar mininet de forma nativa y tuvimos la dificultad extra de tener que instalar una virtual machine con sistema operativo Ubuntu.

Wireshark

Al usar la virtual machine no pudimos encontrar una forma de levantar wireshark para que registre el tráfico de mininet, por lo que sí o sí dependemos de correr mininet en una máquina ubuntu.