ARQUITECTURAS DE REDES AVANZADAS
PRÁCTICA 2
Balanceo de Carga con SDN
ENERO 2015

José Luis Cánovas Sánchez
joseluis.canovas2@um.es
48636907A

Resumen

Índice

1. Introducción

2. Topología mininet

La topología implementada en mininet corresponde, como se puede ver en la Figura 1 a dos switch conectados al controlador, 6 máquinas cliente y 4 servidores.

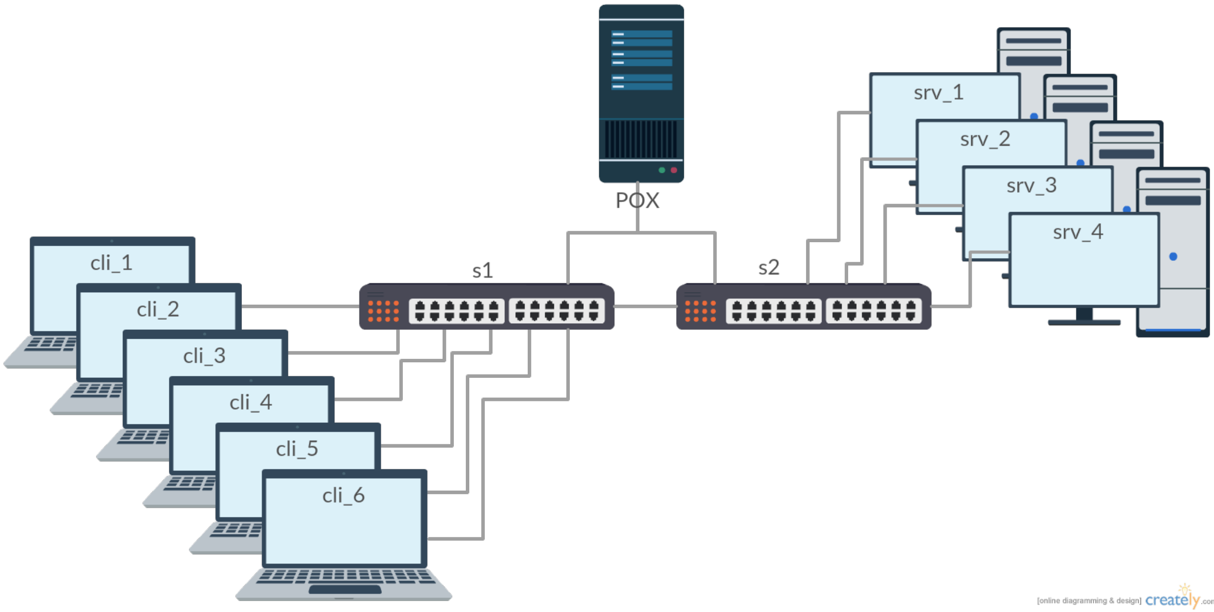

Figura 1: Topología

El código en python que define a la topología es el siguiente. Para facilitar las pruebas, a los servidores se les da una dirección MAC prefijada en el método *addHost*, y a las conexiones con el switch 's2' siempre se indican a qué puerto conectar, con el parámetro *port2* del método *addLink*.

```python
# -*- coding: utf-8 -*-

from mininet.topo import Topo

class MyTopo (Topo):

    def __init__ (self):
        Topo.__init__( self )

        # Add switches
        sw_clients = self.addSwitch('s1')
        sw_servers = self.addSwitch('s2')

        # Add clients
        c1 = self.addHost('cli_1')
        c2 = self.addHost('cli_2')
        c3 = self.addHost('cli_3')
        c4 = self.addHost('cli_4')
        c5 = self.addHost('cli_5')
        c6 = self.addHost('cli_6')

        # Add servers
        s1 = self.addHost('srv_1', ip='10.0.0.101', mac='00:00:00:00:01:01')
        s2 = self.addHost('srv_2', ip='10.0.0.102', mac='00:00:00:00:01:02')
        s3 = self.addHost('srv_3', ip='10.0.0.103', mac='00:00:00:00:01:03')
```

```python
        s4 = self.addHost('srv_4', ip='10.0.0.104', mac='00:00:00:00:01:04')

        # Add links
        self.addLink(sw_clients, sw_servers, port2=1)

        self.addLink(c1, sw_clients)
        self.addLink(c2, sw_clients)
        self.addLink(c3, sw_clients)
        self.addLink(c4, sw_clients)
        self.addLink(c5, sw_clients)
        self.addLink(c6, sw_clients)

        self.addLink(s1, sw_servers, port2=2)
        self.addLink(s2, sw_servers, port2=3)
        self.addLink(s3, sw_servers, port2=4)
        self.addLink(s4, sw_servers, port2=5)


topos = {'mytopo': lambda: MyTopo()}
```

3. Controlador POX con l2_learning

Realizamos un test básico de la topología anterior usando el fichero *l2_learning* que proporciona POX. La salida por pantalla de mininet durante la prueba es la siguiente:

```
$sudo mn --custom topo.py --topo mytopo --controller remote --test pingall
        *** Creating network
        *** Adding controller
        *** Adding hosts:
        cli_1 cli_2 cli_3 cli_4 cli_5 cli_6 srv_1 srv_2 srv_3 srv_4
        *** Adding switches:
        s1 s2
        *** Adding links:
        (cli_1, s1) (cli_2, s1) (cli_3, s1) (cli_4, s1) (cli_5, s1)
         (cli_6, s1) (s1, s2) (srv_1, s2) (srv_2, s2) (srv_3, s2) (srv_4, s2)
        *** Configuring hosts
        cli_1 cli_2 cli_3 cli_4 cli_5 cli_6 srv_1 srv_2 srv_3 srv_4
        *** Starting controller
        c0
        *** Starting 2 switches
        s1 s2 ...
        *** Waiting for switches to connect
        s1 s2
```

```
*** Ping: testing ping reachability
cli_1 -> cli_2 cli_3 cli_4 cli_5 cli_6 srv_1 srv_2 srv_3 srv_4
cli_2 -> cli_1 cli_3 cli_4 cli_5 cli_6 srv_1 srv_2 srv_3 srv_4
cli_3 -> cli_1 cli_2 cli_4 cli_5 cli_6 srv_1 srv_2 srv_3 srv_4
cli_4 -> cli_1 cli_2 cli_3 cli_5 cli_6 srv_1 srv_2 srv_3 srv_4
cli_5 -> cli_1 cli_2 cli_3 cli_4 cli_6 srv_1 srv_2 srv_3 srv_4
cli_6 -> cli_1 cli_2 cli_3 cli_4 cli_5 srv_1 srv_2 srv_3 srv_4
srv_1 -> cli_1 cli_2 cli_3 cli_4 cli_5 cli_6 srv_2 srv_3 srv_4
srv_2 -> cli_1 cli_2 cli_3 cli_4 cli_5 cli_6 srv_1 srv_3 srv_4
srv_3 -> cli_1 cli_2 cli_3 cli_4 cli_5 cli_6 srv_1 srv_2 srv_4
srv_4 -> cli_1 cli_2 cli_3 cli_4 cli_5 cli_6 srv_1 srv_2 srv_3
*** Results: 0\% dropped (90/90 received)
*** Stopping 1 controllers
c0
*** Stopping 11 links
...........
*** Stopping 2 switches
s1 s2
*** Stopping 10 hosts
cli_1 cli_2 cli_3 cli_4 cli_5 cli_6 srv_1 srv_2 srv_3 srv_4
*** Done
completed in 6.529 seconds
```

Todos los hosts (clientes y servidores) tienen conectividad entre ellos y ningún paquete ICMP se ha perdido. La topología de mininet es correcta y se conecta con el controlador POX.

4. Construcción del balanceador de carga

4.1. Módulo de balanceo en POX

Partimos del fichero *l2_learning.py* y añadimos las siguientes líneas de código en la clase LearningSwitch.

Las primeras líneas sirven para definir el método que por round-robin devuelve el siguiente puerto del switch s2 por el que balancear una nueva conexión a los servidores.

```
class LearningSwitch (object):
    [...]
     def __init__ (self, connection, transparent):
        [...]
        self.round_robin = 0
        self.max_srvs = 4
        self.frst_prt = 2

    def roundRobin(self):
        rr = (self.round_robin%self.max_srvs) + self.frst_prt
```

```
        self.round_robin+=1
        return rr
```

Estas líneas se añaden en _handle_PacketIn casi al inicio del método, después de aprender el puerto para la MAC del origen (añadiendo una entrada en *macToPort*), y comprobamos que sea un mensaje del switch 2, que es el que debe balancear, que es un ARP REQUEST y que pregunta por la máquina con IP la de nuestros servidores balanceados.

En caso de ser un paquete que cumple todo lo anterior, creamos un mensaje para el switch s2 que añadirá a su tabla un nuevo flujo para todo paquete con MAC origen la del cliente que se envíe por el puerto del switch que indique el método del round-robin.

```
def _handle_PacketIn (self, event):
    [...]
    # Round-Robin
    if ( dpid_to_str(event.dpid) == "00-00-00-00-00-02" and
                        packet.type == packet.ARP_TYPE and
                packet.payload.opcode == arp.REQUEST and
                    packet.next.protodst == "10.0.0.101" ):
        msg = of.ofp_flow_mod()
        msg.match.dl_src = packet.src
        msg.actions.append(of.ofp_action_output(port = self.roundRobin()))
        #msg.idle_timeout = 10
        #msg.hard_timeout = 30
        msg.data = event.ofp
        self.connection.send(msg)
        return

    if packet.dst.is_multicast:
        flood() # 3a
    [...]
```

4.2. Modificación topología

A la topología de mininet el único cambio que hay que aplicarle es modificar las líneas 22 a 26, modificando la IP en *addHost* por la 10.0.0.101, y que así sea única para todos los servidores.

4.3. Ejecución y prueba de PING

En el anexo al final de esta memoria se encuentra la salida por pantalla completa de las pruebas realizadas, y a continuación se muestra parte de las pruebas con ping aplicadas.

Para cada cliente se ejecuta la orden *cli_i ping -c 4 10.0.0.101*, 4 pings a la dirección de los servidores, que como vemos el primero tiene un retardo bastante considerable frente a los 3 siguientes, debido a que con el primero de todos el switch debe comunicarse con el controlador que se asigna el flujo para la MAC del cliente. Como el flujo se guarda en el switch, se aplica instantáneamente con los siguientes mensajes.

Además se realiza un *pingall* que muestra que los 6 clientes alcanzan, como en l2_learning a todas las máquinas. Sin embargo, los servidores, a pesar de poder hacer ping al resto de servidores, sólo reciben respuesta de uno o dos clientes, pues las respuestas tienen MAC origen la del cliente, y se reenvían al puerto asignado por round-robin en los pings anteriores. Por ejemplo, a los clientes 1 y 5 se les asignó, de los 4 servidores, el *srv_1*.

```
mininet> cli_6 ping -c 4 10.0.0.101
PING 10.0.0.101 (10.0.0.101) 56(84) bytes of data.
64 bytes from 10.0.0.101: icmp_seq=1 ttl=64 time=38.9 ms
64 bytes from 10.0.0.101: icmp_seq=2 ttl=64 time=0.448 ms
64 bytes from 10.0.0.101: icmp_seq=3 ttl=64 time=0.597 ms
64 bytes from 10.0.0.101: icmp_seq=4 ttl=64 time=0.352 ms

--- 10.0.0.101 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3006ms
rtt min/avg/max/mdev = 0.352/10.082/38.933/16.657 ms
mininet> pingall
*** Ping: testing ping reachability
cli_1 -> cli_2 cli_3 cli_4 cli_5 cli_6 srv_1 srv_2 srv_3 srv_4
cli_2 -> cli_1 cli_3 cli_4 cli_5 cli_6 srv_1 srv_2 srv_3 srv_4
cli_3 -> cli_1 cli_2 cli_4 cli_5 cli_6 srv_1 srv_2 srv_3 srv_4
cli_4 -> cli_1 cli_2 cli_3 cli_5 cli_6 srv_1 srv_2 srv_3 srv_4
cli_5 -> cli_1 cli_2 cli_3 cli_4 cli_6 srv_1 srv_2 srv_3 srv_4
cli_6 -> cli_1 cli_2 cli_3 cli_4 cli_5 srv_1 srv_2 srv_3 srv_4
srv_1 -> cli_1 X X X cli_5 X srv_2 srv_3 srv_4
srv_2 -> X cli_2 X X X cli_6 srv_1 srv_3 srv_4
srv_3 -> X X cli_3 X X X srv_1 srv_2 srv_4
srv_4 -> X X X cli_4 X X srv_1 srv_2 srv_3
*** Results: 20% dropped (72/90 received)
```

4.4. Tráfico y flujos

Tráfico analizado con wireshark:

En los ficheros *srv1.pcapng* a *srv4.pcapng* se encuentran las capturas de tráfico de las pruebas en cada uno de los servidores. A continuación se muestran capturas de cada fichero con las trazas más interesantes:

1	0.000000	26:e6:fa:5d:60:15	Broadcast	ARP	42	Who has 10.0.0.101? Tell 10.0.0.1
2	0.000016	00:00:00_00:01:01	26:e6:fa:5d:60:15	ARP	42	10.0.0.101 is at 00:00:00:00:01:01
3	0.004522	10.0.0.1	10.0.0.101	ICMP	98	Echo (ping) request id=0x0c28, seq=1/256, ttl=64 (reply in 4)
4	0.004541	10.0.0.101	10.0.0.1	ICMP	98	Echo (ping) reply id=0x0c28, seq=1/256, ttl=64 (request in 3)
5	0.987393	10.0.0.1	10.0.0.101	ICMP	98	Echo (ping) request id=0x0c28, seq=2/512, ttl=64 (reply in 6)
6	0.987417	10.0.0.101	10.0.0.1	ICMP	98	Echo (ping) reply id=0x0c28, seq=2/512, ttl=64 (request in 5)
7	1.988990	10.0.0.1	10.0.0.101	ICMP	98	Echo (ping) request id=0x0c28, seq=3/768, ttl=64 (reply in 8)
8	1.989014	10.0.0.101	10.0.0.1	ICMP	98	Echo (ping) reply id=0x0c28, seq=3/768, ttl=64 (request in 7)
9	2.991998	10.0.0.1	10.0.0.101	ICMP	98	Echo (ping) request id=0x0c28, seq=4/1024, ttl=64 (reply in 10)
10	2.992016	10.0.0.101	10.0.0.1	ICMP	98	Echo (ping) reply id=0x0c28, seq=4/1024, ttl=64 (request in 9)
11	5.009885	00:00:00_00:01:01	26:e6:fa:5d:60:15	ARP	42	Who has 10.0.0.1? Tell 10.0.0.101
12	5.023755	26:e6:fa:5d:60:15	00:00:00_00:01:01	ARP	42	10.0.0.1 is at 26:e6:fa:5d:60:15
13	27.625645	0a:54:b3:e6:fe:77	Broadcast	ARP	42	Who has 10.0.0.101? Tell 10.0.0.5
14	27.625665	00:00:00_00:01:01	0a:54:b3:e6:fe:77	ARP	42	10.0.0.101 is at 00:00:00:00:01:01
15	27.630373	10.0.0.5	10.0.0.101	ICMP	98	Echo (ping) request id=0x0c39, seq=1/256, ttl=64 (reply in 16)
16	27.630395	10.0.0.101	10.0.0.5	ICMP	98	Echo (ping) reply id=0x0c39, seq=1/256, ttl=64 (request in 15)
17	28.587150	10.0.0.5	10.0.0.101	ICMP	98	Echo (ping) request id=0x0c39, seq=2/512, ttl=64 (reply in 18)
18	28.587168	10.0.0.101	10.0.0.5	ICMP	98	Echo (ping) reply id=0x0c39, seq=2/512, ttl=64 (request in 17)
19	29.588473	10.0.0.5	10.0.0.101	ICMP	98	Echo (ping) request id=0x0c39, seq=3/768, ttl=64 (reply in 20)
20	29.588491	10.0.0.101	10.0.0.5	ICMP	98	Echo (ping) reply id=0x0c39, seq=3/768, ttl=64 (request in 19)
21	30.590814	10.0.0.5	10.0.0.101	ICMP	98	Echo (ping) request id=0x0c39, seq=4/1024, ttl=64 (reply in 22)
22	30.590845	10.0.0.101	10.0.0.5	ICMP	98	Echo (ping) reply id=0x0c39, seq=4/1024, ttl=64 (request in 21)
23	32.642023	00:00:00_00:01:01	0a:54:b3:e6:fe:77	ARP	42	Who has 10.0.0.5? Tell 10.0.0.101
24	32.684851	0a:54:b3:e6:fe:77	00:00:00_00:01:01	ARP	42	10.0.0.5 is at 0a:54:b3:e6:fe:77
25	43.536815	26:e6:fa:5d:60:15	Broadcast	ARP	42	Who has 10.0.0.2? Tell 10.0.0.1
26	43.547963	26:e6:fa:5d:60:15	Broadcast	ARP	42	Who has 10.0.0.3? Tell 10.0.0.1
27	43.560453	26:e6:fa:5d:60:15	Broadcast	ARP	42	Who has 10.0.0.4? Tell 10.0.0.1
28	43.569829	26:e6:fa:5d:60:15	Broadcast	ARP	42	Who has 10.0.0.5? Tell 10.0.0.1
29	43.579948	26:e6:fa:5d:60:15	Broadcast	ARP	42	Who has 10.0.0.6? Tell 10.0.0.1
30	43.589007	10.0.0.1	10.0.0.101	ICMP	98	Echo (ping) request id=0x0c46, seq=1/256, ttl=64 (reply in 31)
31	43.589031	10.0.0.101	10.0.0.1	ICMP	98	Echo (ping) reply id=0x0c46, seq=1/256, ttl=64 (request in 30)
32	43.594815	10.0.0.1	10.0.0.101	ICMP	98	Echo (ping) request id=0x0c47, seq=1/256, ttl=64 (reply in 33)
33	43.594836	10.0.0.101	10.0.0.1	ICMP	98	Echo (ping) reply id=0x0c47, seq=1/256, ttl=64 (request in 32)

En la captura de srv_1 vemos 2 ARP seguidos de 8 PING entre 10.0.0.1 y la IP de los servidores. Corresponde al *cli_1 ping -c 4 10.0.0.101*, con el ARP request que analizará el controlador, la respuesta con MAC la del primer servidor, y 4 ping reply y 4 request con números de secuencia del 1 al 4.

A continuación, la misma situación pero con *cli_5*, pues por round-robin le correspondía el servidor 1 de nuevo, y por ello no aparecen aquí el resto de pings de los clientes.

Finalmente, una sucesión de ARP request desde el servidor al resto de clientes, pero sin respuesta, seguidos de varios mensajes PING desde y hacia los clientes 1 y 5 todos con número de secuencia 1.

Corresponde a la prueba de *pingall*, donde mininet manda la orden al servidor 1 de hacer un ping a cada IP cliente, pero por los flujos en el switch, los paquetes de respuesta se reenvían a otro servidor, y como esto ocurre con cada uno de los 4 servidores, con la misma IP, se explica por qué hay 4 pares de mensajes ICMP Ping con número de secuencia siempre 1 desde cada cliente 1 y 5: por pingall, deben hacer un ping a la IP de las máquinas servidor, que tienen el mismo valor 10.0.0.101, y el switch los reenvía a la misma máquina, en este caso srv_1.

En resumen, en pingall el servidor recibe los PING destinados a los otros servidores.

1	0.000000	d2:82:8a:eb:bd:6c	Broadcast	ARP	42	Who has 10.0.0.101? Tell 10.0.0.2
2	0.000131	00:00:00_00:01:02	d2:82:8a:eb:bd:6c	ARP	42	10.0.0.101 is at 00:00:00:00:01:02
3	0.003835	10.0.0.2	10.0.0.101	ICMP	98	Echo (ping) request id=0x0c2d, seq=1/256, ttl=64 (reply in 4)
4	0.003853	10.0.0.101	10.0.0.2	ICMP	98	Echo (ping) reply id=0x0c2d, seq=1/256, ttl=64 (request in 3)
5	0.984629	10.0.0.2	10.0.0.101	ICMP	98	Echo (ping) request id=0x0c2d, seq=2/512, ttl=64 (reply in 6)
6	0.984667	10.0.0.101	10.0.0.2	ICMP	98	Echo (ping) reply id=0x0c2d, seq=2/512, ttl=64 (request in 5)
7	1.984439	10.0.0.2	10.0.0.101	ICMP	98	Echo (ping) request id=0x0c2d, seq=3/768, ttl=64 (reply in 8)
8	1.984465	10.0.0.101	10.0.0.2	ICMP	98	Echo (ping) reply id=0x0c2d, seq=3/768, ttl=64 (request in 7)
9	2.983427	10.0.0.2	10.0.0.101	ICMP	98	Echo (ping) request id=0x0c2d, seq=4/1024, ttl=64 (reply in 10)
10	2.983458	10.0.0.101	10.0.0.2	ICMP	98	Echo (ping) reply id=0x0c2d, seq=4/1024, ttl=64 (request in 9)
11	5.006234	00:00:00_00:01:02	d2:82:8a:eb:bd:6c	ARP	42	Who has 10.0.0.2? Tell 10.0.0.101
12	5.052046	d2:82:8a:eb:bd:6c	00:00:00_00:01:02	ARP	42	10.0.0.2 is at d2:82:8a:eb:bd:6c
13	28.176554	72:77:b6:49:6d:84	Broadcast	ARP	42	Who has 10.0.0.101? Tell 10.0.0.6
14	28.176571	00:00:00_00:01:02	72:77:b6:49:6d:84	ARP	42	10.0.0.101 is at 00:00:00:00:01:02
15	28.179456	10.0.0.6	10.0.0.101	ICMP	98	Echo (ping) request id=0x0c3d, seq=1/256, ttl=64 (reply in 16)
16	28.179472	10.0.0.101	10.0.0.6	ICMP	98	Echo (ping) reply id=0x0c3d, seq=1/256, ttl=64 (request in 15)
17	29.150051	10.0.0.6	10.0.0.101	ICMP	98	Echo (ping) request id=0x0c3d, seq=2/512, ttl=64 (reply in 18)
18	29.150068	10.0.0.101	10.0.0.6	ICMP	98	Echo (ping) reply id=0x0c3d, seq=2/512, ttl=64 (request in 17)
19	30.151509	10.0.0.6	10.0.0.101	ICMP	98	Echo (ping) request id=0x0c3d, seq=3/768, ttl=64 (reply in 20)
20	30.151532	10.0.0.101	10.0.0.6	ICMP	98	Echo (ping) reply id=0x0c3d, seq=3/768, ttl=64 (request in 19)
21	31.153945	10.0.0.6	10.0.0.101	ICMP	98	Echo (ping) request id=0x0c3d, seq=4/1024, ttl=64 (reply in 22)
22	31.153961	10.0.0.101	10.0.0.6	ICMP	98	Echo (ping) reply id=0x0c3d, seq=4/1024, ttl=64 (request in 21)
23	33.182207	00:00:00_00:01:02	72:77:b6:49:6d:84	ARP	42	Who has 10.0.0.6? Tell 10.0.0.101
24	33.229658	72:77:b6:49:6d:84	00:00:00_00:01:02	ARP	42	10.0.0.6 is at 72:77:b6:49:6d:84
25	36.587934	d2:82:8a:eb:bd:6c	Broadcast	ARP	42	Who has 10.0.0.3? Tell 10.0.0.2
26	36.594920	d2:82:8a:eb:bd:6c	Broadcast	ARP	42	Who has 10.0.0.4? Tell 10.0.0.2
27	36.618397	d2:82:8a:eb:bd:6c	Broadcast	ARP	42	Who has 10.0.0.5? Tell 10.0.0.2
28	36.631629	d2:82:8a:eb:bd:6c	Broadcast	ARP	42	Who has 10.0.0.6? Tell 10.0.0.2
29	36.647599	10.0.0.2	10.0.0.101	ICMP	98	Echo (ping) request id=0x0c4f, seq=1/256, ttl=64 (reply in 30)
30	36.647622	10.0.0.101	10.0.0.2	ICMP	98	Echo (ping) reply id=0x0c4f, seq=1/256, ttl=64 (request in 29)
31	36.656254	10.0.0.2	10.0.0.101	ICMP	98	Echo (ping) request id=0x0c50, seq=1/256, ttl=64 (reply in 32)
32	36.656269	10.0.0.101	10.0.0.2	ICMP	98	Echo (ping) reply id=0x0c50, seq=1/256, ttl=64 (request in 31)
33	36.660054	10.0.0.2	10.0.0.101	ICMP	98	Echo (ping) request id=0x0c51, seq=1/256, ttl=64 (reply in 34)

En el servidor 2 ocurre lo mismo que en el 1 pero con los clientes 2 y 6 que le corresponden por round-robin.

1	0.000000	be:11:f5:32:df:a2	Broadcast	ARP	42	Who has 10.0.0.101? Tell 10.0.0.3
2	0.000068	00:00:00_00:01:03	be:11:f5:32:df:a2	ARP	42	10.0.0.101 is at 00:00:00:00:01:03
3	0.005645	10.0.0.3	10.0.0.101	ICMP	98	Echo (ping) request id=0x0c32, seq=1/256, ttl=64 (reply in 4)
4	0.005669	10.0.0.101	10.0.0.3	ICMP	98	Echo (ping) reply id=0x0c32, seq=1/256, ttl=64 (request in 3)
5	0.984738	10.0.0.3	10.0.0.101	ICMP	98	Echo (ping) request id=0x0c32, seq=2/512, ttl=64 (reply in 6)
6	0.984757	10.0.0.101	10.0.0.3	ICMP	98	Echo (ping) reply id=0x0c32, seq=2/512, ttl=64 (request in 5)
7	1.985034	10.0.0.3	10.0.0.101	ICMP	98	Echo (ping) request id=0x0c32, seq=3/768, ttl=64 (reply in 8)
8	1.985051	10.0.0.101	10.0.0.3	ICMP	98	Echo (ping) reply id=0x0c32, seq=3/768, ttl=64 (request in 7)
9	2.986500	10.0.0.3	10.0.0.101	ICMP	98	Echo (ping) request id=0x0c32, seq=4/1024, ttl=64 (reply in 10)
10	2.986528	10.0.0.101	10.0.0.3	ICMP	98	Echo (ping) reply id=0x0c32, seq=4/1024, ttl=64 (request in 9)
11	5.020968	00:00:00_00:01:03	be:11:f5:32:df:a2	ARP	42	Who has 10.0.0.3? Tell 10.0.0.101
12	5.051501	be:11:f5:32:df:a2	00:00:00_00:01:03	ARP	42	10.0.0.3 is at be:11:f5:32:df:a2
13	28.605111	be:11:f5:32:df:a2	Broadcast	ARP	42	Who has 10.0.0.4? Tell 10.0.0.3
14	28.613915	be:11:f5:32:df:a2	Broadcast	ARP	42	Who has 10.0.0.5? Tell 10.0.0.3
15	28.627538	be:11:f5:32:df:a2	Broadcast	ARP	42	Who has 10.0.0.6? Tell 10.0.0.3
16	28.638277	10.0.0.3	10.0.0.101	ICMP	98	Echo (ping) request id=0x0c58, seq=1/256, ttl=64 (reply in 17)
17	28.638305	10.0.0.101	10.0.0.3	ICMP	98	Echo (ping) reply id=0x0c58, seq=1/256, ttl=64 (request in 16)
18	28.644154	10.0.0.3	10.0.0.101	ICMP	98	Echo (ping) request id=0x0c59, seq=1/256, ttl=64 (reply in 19)
19	28.644169	10.0.0.101	10.0.0.3	ICMP	98	Echo (ping) reply id=0x0c59, seq=1/256, ttl=64 (request in 18)
20	28.646515	10.0.0.3	10.0.0.101	ICMP	98	Echo (ping) request id=0x0c5a, seq=1/256, ttl=64 (reply in 21)
21	28.646528	10.0.0.101	10.0.0.3	ICMP	98	Echo (ping) reply id=0x0c5a, seq=1/256, ttl=64 (request in 20)
22	28.649032	10.0.0.3	10.0.0.101	ICMP	98	Echo (ping) request id=0x0c5b, seq=1/256, ttl=64 (reply in 23)
23	28.649045	10.0.0.101	10.0.0.3	ICMP	98	Echo (ping) reply id=0x0c5b, seq=1/256, ttl=64 (request in 22)
24	28.972967	00:00:00_00:01:01	Broadcast	ARP	42	Who has 10.0.0.2? Tell 10.0.0.101 (duplicate use of 10.0.0.101 detected!
25	30.000869	00:00:00_00:01:01	Broadcast	ARP	42	Who has 10.0.0.2? Tell 10.0.0.101 (duplicate use of 10.0.0.101 detected!
26	30.976135	00:00:00_00:01:01	Broadcast	ARP	42	Who has 10.0.0.2? Tell 10.0.0.101 (duplicate use of 10.0.0.101 detected!
27	32.004809	00:00:00_00:01:01	Broadcast	ARP	42	Who has 10.0.0.3? Tell 10.0.0.101 (duplicate use of 10.0.0.101 detected!
28	32.007830	be:11:f5:32:df:a2	00:00:00_00:01:01	ARP	42	10.0.0.3 is at be:11:f5:32:df:a2 (duplicate use of 10.0.0.101 detected!)
29	32.986817	00:00:00_00:01:01	Broadcast	ARP	42	Who has 10.0.0.3? Tell 10.0.0.101 (duplicate use of 10.0.0.101 detected!

En el servidor 3 ocurren *tres cuartos de lo mismo* que en el resto pero con sólo el cliente 3. En esta captura aprovecho para comentar lo que ocurre en los 4 servidores, pero que se ve en la imagen: wireshark detecta una IP duplicada en la red. Lo descubre porque en un ARP reply previo la IP y MAC destino eran la del servidor 3, pero el ARP request donde se detecta la duplicidad proviene de otro servidor, en la traza 24 corresponde a srv_1, en la 39 al srv_2, etc.

Como estos mensajes ARP request no preguntan por la IP de los servidores, se reenvían como cualquier otro paquete y no se ven afectados por los flujos que hemos programado para el balanceo.

1	0.000000	a2:44:fc:bd:86:61	Broadcast	ARP	42	Who has 10.0.0.101? Tell 10.0.0.4
2	0.000022	00:00:00_00:01:04	a2:44:fc:bd:86:61	ARP	42	10.0.0.101 is at 00:00:00:00:01:04
3	0.006280	10.0.0.4	10.0.0.101	ICMP	98	Echo (ping) request id=0x0c37, seq=1/256, ttl=64 (reply in 4)
4	0.006301	10.0.0.101	10.0.0.4	ICMP	98	Echo (ping) reply id=0x0c37, seq=1/256, ttl=64 (request in 3)
5	0.961345	10.0.0.4	10.0.0.101	ICMP	98	Echo (ping) request id=0x0c37, seq=2/512, ttl=64 (reply in 6)
6	0.961368	10.0.0.101	10.0.0.4	ICMP	98	Echo (ping) reply id=0x0c37, seq=2/512, ttl=64 (request in 5)
7	1.962604	10.0.0.4	10.0.0.101	ICMP	98	Echo (ping) request id=0x0c37, seq=3/768, ttl=64 (reply in 8)
8	1.962623	10.0.0.101	10.0.0.4	ICMP	98	Echo (ping) reply id=0x0c37, seq=3/768, ttl=64 (request in 7)
9	2.963892	10.0.0.4	10.0.0.101	ICMP	98	Echo (ping) request id=0x0c37, seq=4/1024, ttl=64 (reply in 10)
10	2.963919	10.0.0.101	10.0.0.4	ICMP	98	Echo (ping) reply id=0x0c37, seq=4/1024, ttl=64 (request in 9)
11	5.014662	00:00:00_00:01:04	a2:44:fc:bd:86:61	ARP	42	Who has 10.0.0.4? Tell 10.0.0.101
12	5.054401	a2:44:fc:bd:86:61	00:00:00_00:01:04	ARP	42	10.0.0.4 is at a2:44:fc:bd:86:61
13	22.151813	a2:44:fc:bd:86:61	Broadcast	ARP	42	Who has 10.0.0.5? Tell 10.0.0.4
14	22.162728	a2:44:fc:bd:86:61	Broadcast	ARP	42	Who has 10.0.0.6? Tell 10.0.0.4
15	22.174303	10.0.0.4	10.0.0.101	ICMP	98	Echo (ping) request id=0x0c61, seq=1/256, ttl=64 (reply in 16)
16	22.174326	10.0.0.101	10.0.0.4	ICMP	98	Echo (ping) reply id=0x0c61, seq=1/256, ttl=64 (request in 15)
17	22.180767	10.0.0.4	10.0.0.101	ICMP	98	Echo (ping) request id=0x0c62, seq=1/256, ttl=64 (reply in 18)
18	22.180782	10.0.0.101	10.0.0.4	ICMP	98	Echo (ping) reply id=0x0c62, seq=1/256, ttl=64 (request in 17)
19	22.183782	10.0.0.4	10.0.0.101	ICMP	98	Echo (ping) request id=0x0c63, seq=1/256, ttl=64 (reply in 20)
20	22.183795	10.0.0.101	10.0.0.4	ICMP	98	Echo (ping) reply id=0x0c63, seq=1/256, ttl=64 (request in 19)
21	22.187903	10.0.0.4	10.0.0.101	ICMP	98	Echo (ping) request id=0x0c64, seq=1/256, ttl=64 (reply in 22)
22	22.187924	10.0.0.101	10.0.0.4	ICMP	98	Echo (ping) reply id=0x0c64, seq=1/256, ttl=64 (request in 21)
23	22.438602	00:00:00_00:01:01	Broadcast	ARP	42	Who has 10.0.0.2? Tell 10.0.0.101 (duplicate use of 10.0.0.101 detec
24	23.466504	00:00:00_00:01:01	Broadcast	ARP	42	Who has 10.0.0.2? Tell 10.0.0.101 (duplicate use of 10.0.0.101 detec
25	24.441770	00:00:00_00:01:01	Broadcast	ARP	42	Who has 10.0.0.2? Tell 10.0.0.101 (duplicate use of 10.0.0.101 detec
26	25.470444	00:00:00_00:01:01	Broadcast	ARP	42	Who has 10.0.0.3? Tell 10.0.0.101 (duplicate use of 10.0.0.101 detec
27	26.452452	00:00:00_00:01:01	Broadcast	ARP	42	Who has 10.0.0.3? Tell 10.0.0.101 (duplicate use of 10.0.0.101 detec
28	27.468814	00:00:00_00:01:01	Broadcast	ARP	42	Who has 10.0.0.3? Tell 10.0.0.101 (duplicate use of 10.0.0.101 detec
29	28.462066	00:00:00_00:01:01	Broadcast	ARP	42	Who has 10.0.0.4? Tell 10.0.0.101 (duplicate use of 10.0.0.101 detec
30	28.465653	a2:44:fc:bd:86:61	00:00:00_00:01:01	ARP	42	10.0.0.4 is at a2:44:fc:bd:86:61 (duplicate use of 10.0.0.101 detecte

Y en el servidor 4 ocurre con el cliente 4 lo mismo que en el servidor 3 con el cliente 3.

Flujos de los switches:

En el código del controlador de la sección anterior había comentadas dos líneas que indicaban al switch que la regla del nuevo flujo debería caducar. Para facilitar las pruebas, se comentan, y pasados unos segundos en los que las reglas de aprendizaje de macToPort sí caducan, con la orden en mininet *dpctl dump-flows* se muestran las tablas con los flujos asignados por round-robin.

```
mininet> dpctl dump-flows
*** s1 ------------------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
```

```
*** s2 ------------------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
cookie=0x0, duration=390.693s, table=0, n_packets=24, n_bytes=1512, idle_age=309,
dl_src=fe:1c:e8:bb:47:2b actions=output:4
cookie=0x0, duration=397.979s, table=0, n_packets=25, n_bytes=1554, idle_age=312,
dl_src=5a:fb:3a:24:c3:d0 actions=output:3
cookie=0x0, duration=368.291s, table=0, n_packets=21, n_bytes=1386, idle_age=303,
dl_src=26:1a:ac:72:e4:51 actions=output:3
cookie=0x0, duration=383.206s, table=0, n_packets=23, n_bytes=1470, idle_age=303,
dl_src=ba:d3:b1:08:9d:20 actions=output:5
cookie=0x0, duration=406.713s, table=0, n_packets=26, n_bytes=1596, idle_age=315,
dl_src=86:bd:bc:16:52:e3 actions=output:2
cookie=0x0, duration=376.218s, table=0, n_packets=21, n_bytes=1386, idle_age=306,
dl_src=ca:6d:f2:8c:c1:a9 actions=output:2
```

Se observa que todas pertenecen a la tabla 0, y que en *dl_src* se indica la MAC de un cliente y en *actions=output* el puerto por donde reenviar el paquete. Usando la información de los *ifconfig* que se encuentra en el anexo, se puede hacer la correspondencia de cada cliente con su MAC y por tanto su flujo.

A los clientes 1 y 5 les corresponde el puerto 2, es decir, el servidor 1. A los clientes 2 y 6 el puerto 3, servidor 2. Al cliente 3 el puerto 4, servidor 3. Y al cliente 4 el puerto 5, servidor 4.

Las tablas de flujos de los switchs sin que hayan caducado las entradas de macToPort son las siguientes:

```
mininet> dpctl dump-flows
*** s1 ------------------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
cookie=0x0, duration=10.672s, table=0, n_packets=3, n_bytes=126, idle_timeout=10,
 hard_timeout=30, idle_age=8, priority=65535,arp,in_port=7,vlan_tci=0x0000,
  dl_src=02:3f:e7:62:af:f1,dl_dst=00:00:00:00:01:04,arp_spa=10.0.0.6,
  arp_tpa=10.0.0.101,arp_op=2 actions=output:1
cookie=0x0, duration=8.627s, table=0, n_packets=1, n_bytes=42, idle_timeout=10,
 hard_timeout=30, idle_age=8, priority=65535,arp,in_port=5,vlan_tci=0x0000,
  dl_src=0a:27:a9:ce:34:da,dl_dst=00:00:00:00:01:04,arp_spa=10.0.0.4,
  arp_tpa=10.0.0.101,arp_op=2 actions=output:1
cookie=0x0, duration=8.666s, table=0, n_packets=1, n_bytes=42, idle_timeout=10,
 hard_timeout=30, idle_age=8, priority=65535,arp,in_port=1,vlan_tci=0x0000,
  dl_src=00:00:00:00:01:04,dl_dst=0a:27:a9:ce:34:da,arp_spa=10.0.0.101,
  arp_tpa=10.0.0.4,arp_op=1 actions=output:5
*** s2 ------------------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
cookie=0x0, duration=8.673s, table=0, n_packets=1, n_bytes=42, idle_timeout=10,
 hard_timeout=30, idle_age=8, priority=65535,arp,in_port=5,vlan_tci=0x0000,dl_src=00:00:00:00:01:04,dl_
```

```
cookie=0x0, duration=117.856s, table=0, n_packets=26, n_bytes=1596, idle_age=20,
 dl_src=ba:b9:7f:25:67:db actions=output:2
cookie=0x0, duration=110.578s, table=0, n_packets=25, n_bytes=1554, idle_age=17,
 dl_src=d6:a7:d3:3c:41:ed actions=output:3
cookie=0x0, duration=78.586s, table=0, n_packets=21, n_bytes=1386, idle_age=8,
 dl_src=02:3f:e7:62:af:f1 actions=output:3
cookie=0x0, duration=86.319s, table=0, n_packets=21, n_bytes=1386, idle_age=11,
 dl_src=e6:aa:ea:b9:70:6a actions=output:2
cookie=0x0, duration=103.169s, table=0, n_packets=24, n_bytes=1512, idle_age=14,
 dl_src=f6:46:72:d8:ae:cc actions=output:4
cookie=0x0, duration=93.593s, table=0, n_packets=23, n_bytes=1470, idle_age=8,
 dl_src=0a:27:a9:ce:34:da actions=output:5
```

A. Test ping

```
mininet@mininet-vm:~/GitHub/mininet/scripts$ sudo ./testWcont topo.py
*** Creating network
*** Adding controller
*** Adding hosts:
cli_1 cli_2 cli_3 cli_4 cli_5 cli_6 srv_1 srv_2 srv_3 srv_4
*** Adding switches:
s1 s2
*** Adding links:
(cli_1, s1) (cli_2, s1) (cli_3, s1) (cli_4, s1) (cli_5, s1) (cli_6, s1)
 (s1, s2) (srv_1, s2) (srv_2, s2) (srv_3, s2) (srv_4, s2)
*** Configuring hosts
cli_1 cli_2 cli_3 cli_4 cli_5 cli_6 srv_1 srv_2 srv_3 srv_4
*** Starting controller
c0
*** Starting 2 switches
s1 s2 ...
*** Starting CLI:
mininet> cli_1 ping -c 4 10.0.0.101
PING 10.0.0.101 (10.0.0.101) 56(84) bytes of data.
64 bytes from 10.0.0.101: icmp_seq=1 ttl=64 time=12.2 ms
64 bytes from 10.0.0.101: icmp_seq=2 ttl=64 time=1.21 ms
64 bytes from 10.0.0.101: icmp_seq=3 ttl=64 time=0.200 ms
64 bytes from 10.0.0.101: icmp_seq=4 ttl=64 time=0.063 ms

--- 10.0.0.101 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3006ms
```

```
rtt min/avg/max/mdev = 0.063/3.441/12.288/5.127 ms
mininet> cli_2 ping -c 4 10.0.0.101
PING 10.0.0.101 (10.0.0.101) 56(84) bytes of data.
64 bytes from 10.0.0.101: icmp_seq=1 ttl=64 time=31.9 ms
64 bytes from 10.0.0.101: icmp_seq=2 ttl=64 time=0.810 ms
64 bytes from 10.0.0.101: icmp_seq=3 ttl=64 time=0.071 ms
64 bytes from 10.0.0.101: icmp_seq=4 ttl=64 time=0.776 ms

--- 10.0.0.101 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3000ms
rtt min/avg/max/mdev = 0.071/8.392/31.914/13.583 ms
mininet> cli_3 ping -c 4 10.0.0.101
PING 10.0.0.101 (10.0.0.101) 56(84) bytes of data.
64 bytes from 10.0.0.101: icmp_seq=1 ttl=64 time=40.4 ms
64 bytes from 10.0.0.101: icmp_seq=2 ttl=64 time=1.28 ms
64 bytes from 10.0.0.101: icmp_seq=3 ttl=64 time=0.072 ms
64 bytes from 10.0.0.101: icmp_seq=4 ttl=64 time=0.671 ms

--- 10.0.0.101 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3003ms
rtt min/avg/max/mdev = 0.072/10.614/40.426/17.217 ms
mininet> cli_4 ping -c 4 10.0.0.101
PING 10.0.0.101 (10.0.0.101) 56(84) bytes of data.
64 bytes from 10.0.0.101: icmp_seq=1 ttl=64 time=28.5 ms
64 bytes from 10.0.0.101: icmp_seq=2 ttl=64 time=0.407 ms
64 bytes from 10.0.0.101: icmp_seq=3 ttl=64 time=0.073 ms
64 bytes from 10.0.0.101: icmp_seq=4 ttl=64 time=0.083 ms

--- 10.0.0.101 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3004ms
rtt min/avg/max/mdev = 0.073/7.284/28.574/12.292 ms
mininet> cli_5 ping -c 4 10.0.0.101
PING 10.0.0.101 (10.0.0.101) 56(84) bytes of data.
64 bytes from 10.0.0.101: icmp_seq=1 ttl=64 time=19.3 ms
64 bytes from 10.0.0.101: icmp_seq=2 ttl=64 time=0.447 ms
64 bytes from 10.0.0.101: icmp_seq=3 ttl=64 time=0.846 ms
64 bytes from 10.0.0.101: icmp_seq=4 ttl=64 time=0.087 ms

--- 10.0.0.101 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3004ms
rtt min/avg/max/mdev = 0.087/5.178/19.332/8.176 ms
```

```
mininet> cli_6 ping -c 4 10.0.0.101
PING 10.0.0.101 (10.0.0.101) 56(84) bytes of data.
64 bytes from 10.0.0.101: icmp_seq=1 ttl=64 time=38.9 ms
64 bytes from 10.0.0.101: icmp_seq=2 ttl=64 time=0.448 ms
64 bytes from 10.0.0.101: icmp_seq=3 ttl=64 time=0.597 ms
64 bytes from 10.0.0.101: icmp_seq=4 ttl=64 time=0.352 ms

--- 10.0.0.101 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3006ms
rtt min/avg/max/mdev = 0.352/10.082/38.933/16.657 ms
mininet> pingall
*** Ping: testing ping reachability
cli_1 -> cli_2 cli_3 cli_4 cli_5 cli_6 srv_1 srv_2 srv_3 srv_4
cli_2 -> cli_1 cli_3 cli_4 cli_5 cli_6 srv_1 srv_2 srv_3 srv_4
cli_3 -> cli_1 cli_2 cli_4 cli_5 cli_6 srv_1 srv_2 srv_3 srv_4
cli_4 -> cli_1 cli_2 cli_3 cli_5 cli_6 srv_1 srv_2 srv_3 srv_4
cli_5 -> cli_1 cli_2 cli_3 cli_4 cli_6 srv_1 srv_2 srv_3 srv_4
cli_6 -> cli_1 cli_2 cli_3 cli_4 cli_5 srv_1 srv_2 srv_3 srv_4
srv_1 -> cli_1 X X X cli_5 X srv_2 srv_3 srv_4
srv_2 -> X cli_2 X X X cli_6 srv_1 srv_3 srv_4
srv_3 -> X X cli_3 X X X srv_1 srv_2 srv_4
srv_4 -> X X X cli_4 X X srv_1 srv_2 srv_3
*** Results: 20% dropped (72/90 received)
mininet> dpctl dump-flows
*** s1 ------------------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
*** s2 ------------------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=390.693s, table=0, n_packets=24, n_bytes=1512, idle_age=309,
  dl_src=fe:1c:e8:bb:47:2b actions=output:4
 cookie=0x0, duration=397.979s, table=0, n_packets=25, n_bytes=1554, idle_age=312,
  dl_src=5a:fb:3a:24:c3:d0 actions=output:3
 cookie=0x0, duration=368.291s, table=0, n_packets=21, n_bytes=1386, idle_age=303,
  dl_src=26:1a:ac:72:e4:51 actions=output:3
 cookie=0x0, duration=383.206s, table=0, n_packets=23, n_bytes=1470, idle_age=303,
  dl_src=ba:d3:b1:08:9d:20 actions=output:5
 cookie=0x0, duration=406.713s, table=0, n_packets=26, n_bytes=1596, idle_age=315,
  dl_src=86:bd:bc:16:52:e3 actions=output:2
 cookie=0x0, duration=376.218s, table=0, n_packets=21, n_bytes=1386, idle_age=306,
  dl_src=ca:6d:f2:8c:c1:a9 actions=output:2
mininet> cli_1 ifconfig
```

```
cli_1-eth0 Link encap:Ethernet  HWaddr 86:bd:bc:16:52:e3
          inet addr:10.0.0.1  Bcast:10.255.255.255  Mask:255.0.0.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:101 errors:0 dropped:0 overruns:0 frame:0
          TX packets:41 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:5306 (5.3 KB)  TX bytes:2786 (2.7 KB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

mininet> cli_2 ifconfig
cli_2-eth0 Link encap:Ethernet  HWaddr 5a:fb:3a:24:c3:d0
          inet addr:10.0.0.2  Bcast:10.255.255.255  Mask:255.0.0.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:101 errors:0 dropped:0 overruns:0 frame:0
          TX packets:41 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:5306 (5.3 KB)  TX bytes:2786 (2.7 KB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

mininet> cli_3 ifconfig
cli_3-eth0 Link encap:Ethernet  HWaddr fe:1c:e8:bb:47:2b
          inet addr:10.0.0.3  Bcast:10.255.255.255  Mask:255.0.0.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:101 errors:0 dropped:0 overruns:0 frame:0
          TX packets:41 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:5306 (5.3 KB)  TX bytes:2786 (2.7 KB)
```

```
lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

mininet> cli_4 ifconfig
cli_4-eth0 Link encap:Ethernet  HWaddr ba:d3:b1:08:9d:20
          inet addr:10.0.0.4  Bcast:10.255.255.255  Mask:255.0.0.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:101 errors:0 dropped:0 overruns:0 frame:0
          TX packets:41 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:5306 (5.3 KB)  TX bytes:2786 (2.7 KB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

mininet> cli_5 ifconfig
cli_5-eth0 Link encap:Ethernet  HWaddr ca:6d:f2:8c:c1:a9
          inet addr:10.0.0.5  Bcast:10.255.255.255  Mask:255.0.0.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:100 errors:0 dropped:0 overruns:0 frame:0
          TX packets:40 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:5264 (5.2 KB)  TX bytes:2744 (2.7 KB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
```

```
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

mininet> cli_6 ifconfig
cli_6-eth0 Link encap:Ethernet  HWaddr 26:1a:ac:72:e4:51
          inet addr:10.0.0.6  Bcast:10.255.255.255  Mask:255.0.0.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:101 errors:0 dropped:0 overruns:0 frame:0
          TX packets:41 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:5306 (5.3 KB)  TX bytes:2786 (2.7 KB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

mininet>
```