NTNU
Kunnskap for en bedre verden

Department of Information Security and Communication Technology

TTM4180 - Applied Networking

Lab 6: Load balancer

Author:
S. Xiao Fernández Marín

April, 2022

Table of Contents

List of Figures

List of Tables

1 Introduction

In this lab, we are using our learned skills from Lab 1 to Lab 5. Our main goal is to implement a load balancer that uses the Round-Robin algorithm to redirect requests from four clients (h1, h2, h3 and h4) to three backend servers (h5, h6 and h7).

A load balancer is a way of handling extreme loads, distributing traffic to reduce congestion in the network and handling a large amount of traffic. The load balancer is placed in between the web servers and clients. The requests of these clients are distributed by the load balancer into the different servers. The load balancer in this lab is placed between the clients h1, h2, h3 and h4 and the servers h5, h6 and h7, as the Figure 1 shows.

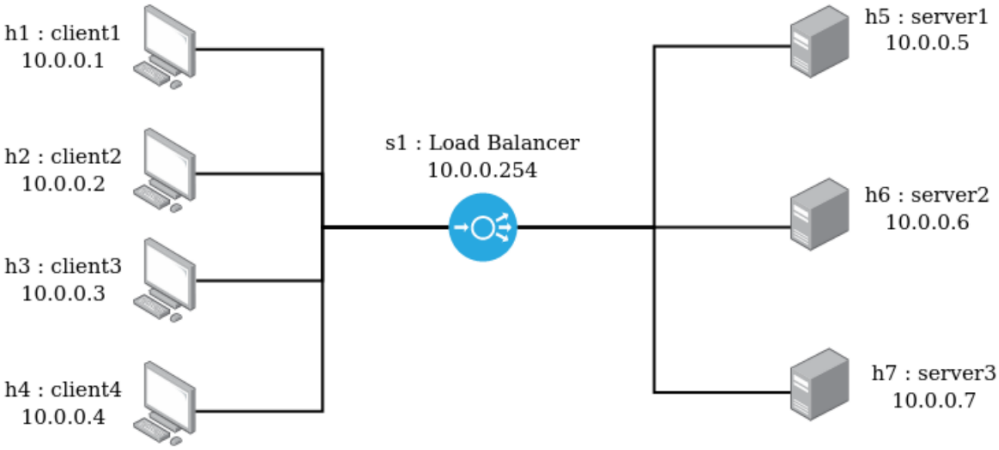

Figure 1: Diagram of the topology of the lab 6

For knowing which server to choose, between the three available, the algorithm used is Round-Robin, a schedule algorithm that distributes equal portions of a unit in a circular order, with no priority. The algorithm can be seen in Figure 2.

Figure 2: Round-Robin algorithm

2 Walkthrough

As I had the second approach implemented, I started by running a container, with the given command in Lab 1:

```
sudo docker run -it --rm --privileged -e DISPLAY \
-v /tmp/.X11-unix:/tmp/.X11-unix \
-v /lib/modules:/lib/modules \
aliesdocker6/mininet_ttm_4180
```

I also enabled the *man* command inside the new created container while I also added the root user to the local access control list of xhost and moved the folder *pox* to the directory *ubuntu*:

```
unminimize
xhost +si:localuser:root
mkdir /home/ubuntu
mv /root/pox /home/ubuntu
cd /home/ubuntu
```

As the file *SimpleLoadBalancer.py* has to be in the directory */home/ubuntu/pox/ext*, I went to that directory and downloaded the template usign the wget command (installing it first).

```
cd /home/ubuntu/pox/ext
sudo apt install wget
wget
↪  https://raw.githubusercontent.com/simehag/TTM4180/master/lab_6/SimpleLoadBala
↪  ncer.py
```

As I needed more than one terminal for the container I executed the next command for having more than one terminal pointing to the same container.

```
$ sudo docker ps -a
CONTAINER ID   IMAGE                              COMMAND            CREATED
↪  STATUS                   PORTS
↪  NAMES
14321e75797a   aliesdocker6/mininet_ttm_4180   "/ENTRYPOINT.sh"   28 seconds ago
↪  Up 26 seconds            6633/tcp, 6640/tcp, 6653/tcp, 8080/tcp, 8888/tcp
↪  optimistic_wing
0fe3b713f588   hello-world                        "/hello"           7 weeks ago
↪  Exited (0) 7 weeks ago
↪  romantic_hofstadter
$ sudo docker exec -it 14321e75797a bash
```

3 Load Balancer Functionality

Attached in Appendix A and a different file in the submission of Lab 6.

1. The switch should preemptively ask for the MAC addresses of all the servers with ARP requests, in order to associate these MAC addresses and the corresponding switch ports with the real IP addresses of the servers. This query should be performed upon the connection establishment of the controller to the switch in order to avoid having client flows waiting to be forwarded to the correct server.

   Implemented in function: *_handle_ConnectionUp()*.

2. Answer to ARP requests from the clients searching the MAC addresses of the service. The switch should proxy ARP replies that answer to the clients' requests with a fake MAC that is associated with the load balancer. It is useful to store the information contained in each ARP request (source MAC address of client, input port of ARP request packet). In this way, when the load balancer later needs to direct flows towards the clients, it will know their MAC addresses and ports to output the packets.

   Implemented in functions: *_handle_PacketIn()* and *send_arp_reply()*.

3. Answer to ARP requests from the servers searching the MAC addresses of clients. The switch should proxy ARP replies that answer with the fake MAC that is associated with the load balancer. At this point you should already know the MAC of the client, since it has previously requested the MAC address of the service (see previous step).

   Implemented in functions: *_handle_PacketIn()* and *send_arp_reply()*.

4. Redirect flows from the clients towards the servers using the Round-Robin Scheduling mechanism. Of course, the server should see packets with their MAC address changed to the MAC of the load balancer, but with the source client IP intact. The destination IP address should also be rewritten to the one of the destination server. Be careful: the redirection should only happen for flows that stem from client IPs (i.e., non-server IPs) and which are directed to the service IP.

   Implemented in functions: *_handle_PacketIn()*, *update_lb_mapping()*, *install_flow_rule_server_to_client()*, *install_flow_rule _client_to_server()* and *round_robin()*.

5. Direct flows from the servers to the clients. This should occur after rewriting the source IP address to the one of the service and the source MAC address to the load balancer fake MAC. In this way, the clients do not see any redirection happening, and they believe that all their communication takes place between their machines and the service IP (the load balancing mechanism is transparent).

   Implemented in functions: *_handle_PacketIn()* and *install_flow_rule_server_to_client()*.

4 Testing the code

1. Open Wireshark and begin listening on all interfaces;

   I have opened three terminals. In one of them, I run Wireshark with the command:

   ```
   sudo wireshark &
   ```

   In loopback mode and the filter *openflow_v1 and !(openflow_1_0.type==2 or openflow_1_0.type ==3)*

2. Launch Mininet and the Controller with run.sh;

   As I had the second approach, in the second terminal I started the controller with the command, while first configuring the openflow version for the switch.

   ```
   sudo ovs-vsctl set bridge s1 protocols=OpenFlow10
   cd /home/ubuntu/pox && ./pox.py log.level --DEBUG SimpleLoadBalancer
   ↪  --loadbalancer=10.0.0.254 --servers=10.0.0.5,10.0.0.6,10.0.0.7
   ```

   In the third terminal I ran the command for launch mininet.

   ```
   sudo mn --topo single,7 --mac --controller remote --switch ovsk
   ```

3. Ping a few times from each host to 10.0.0.254;

   ```
   mininet> h1 ping -c 2 10.0.0.254
   mininet> h2 ping -c 2 10.0.0.254
   mininet> h3 ping -c 2 10.0.0.254
   mininet> h4 ping -c 2 10.0.0.254
   mininet> h1 ping -c 2 10.0.0.254
   mininet> h2 ping -c 2 10.0.0.254
   mininet> h3 ping -c 2 10.0.0.254
   mininet> h4 ping -c 2 10.0.0.254
   ```

```
[SimpleLoadBalancer     ] FUNCTION: _handle_PacketIn
[SimpleLoadBalancer     ] Received ARP Packet
[SimpleLoadBalancer     ] ARP REQUEST Received
[SimpleLoadBalancer     ] Client 10.0.0.1 sent ARP req to LB 10.0.0.254
[SimpleLoadBalancer     ] FUNCTION: send_arp_reply
[SimpleLoadBalancer     ] FUNCTION: _handle_PacketIn
[SimpleLoadBalancer     ] Received IP Packet from 10.0.0.1
[SimpleLoadBalancer     ] FUNCTION: update_lb_mapping
[SimpleLoadBalancer     ] FUNCTION: round_robin
[SimpleLoadBalancer     ] Round robin selected: 10.0.0.5
[SimpleLoadBalancer     ] FUNCTION: install_flow_rule_client_to_server
[SimpleLoadBalancer     ] FUNCTION: install_flow_rule_server_to_client
[SimpleLoadBalancer     ] Installed flow rule: 10.0.0.5 -> 10.0.0.1
[SimpleLoadBalancer     ] Installed flow rule: 10.0.0.1 -> 10.0.0.5
[SimpleLoadBalancer     ] FUNCTION: _handle_PacketIn
[SimpleLoadBalancer     ] Received ARP Packet
[SimpleLoadBalancer     ] ARP REQUEST Received
[SimpleLoadBalancer     ] Server 10.0.0.5 sent ARP req to client
[SimpleLoadBalancer     ] FUNCTION: send_arp_reply
[openflow.of_01         ] 1 connection aborted
[SimpleLoadBalancer     ] FUNCTION: _handle_PacketIn
[SimpleLoadBalancer     ] Received ARP Packet
[SimpleLoadBalancer     ] ARP REQUEST Received
[SimpleLoadBalancer     ] Client 10.0.0.2 sent ARP req to LB 10.0.0.254
[SimpleLoadBalancer     ] FUNCTION: send_arp_reply
[SimpleLoadBalancer     ] FUNCTION: _handle_PacketIn
[SimpleLoadBalancer     ] Received IP Packet from 10.0.0.2
[SimpleLoadBalancer     ] FUNCTION: update_lb_mapping
[SimpleLoadBalancer     ] FUNCTION: round_robin
[SimpleLoadBalancer     ] Round robin selected: 10.0.0.6
[SimpleLoadBalancer     ] FUNCTION: install_flow_rule_client_to_server
[SimpleLoadBalancer     ] FUNCTION: install_flow_rule_server_to_client
[SimpleLoadBalancer     ] Installed flow rule: 10.0.0.6 -> 10.0.0.2
[SimpleLoadBalancer     ] Installed flow rule: 10.0.0.2 -> 10.0.0.6
[SimpleLoadBalancer     ] FUNCTION: _handle_PacketIn
[SimpleLoadBalancer     ] Received ARP Packet
[SimpleLoadBalancer     ] ARP REQUEST Received
[SimpleLoadBalancer     ] Server 10.0.0.6 sent ARP req to client
[SimpleLoadBalancer     ] FUNCTION: send_arp_reply
```

Figure 3: h1, h2 ping -c 2 10.0.0.254 first time

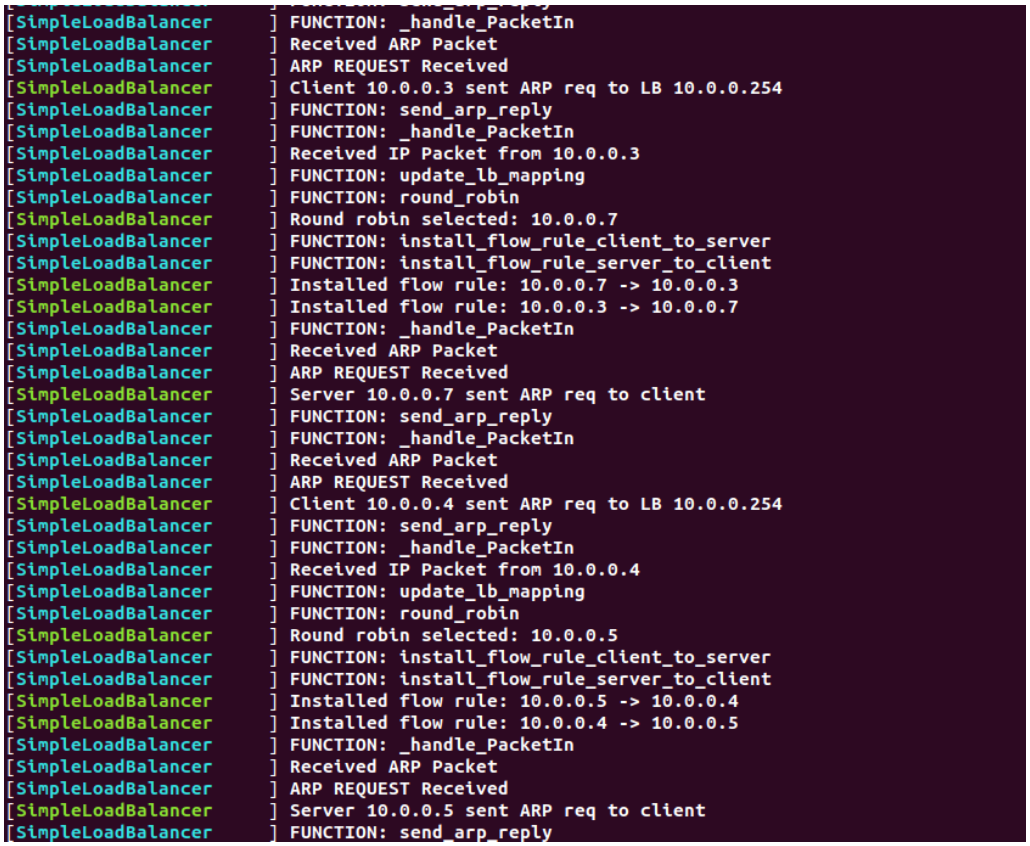

Figure 4: h3, h4 ping -c 2 10.0.0.254 first time

The first time, it uses the following functions:

- _handle_PacketIn(): It detects an ARP request package that has been sent from one client and trying to connect to the LB (10.0.0.254) and sends an ARP reply to the client. Further explained in Section 5.3.
- send_arp_reply(): The LB sends an ARP reply with the MAC address of the LB.
- _handle_PacketIn(): Sends an IP package to the LB, after get its MAC address.
- update_lb_mapping(): Map between LB and client (LOADBALANCER_MAP).
- round_robin(): Chooses which server to choose.
- install_flow_rule_client_to_server(): Install a flow rule between server and client. From now on, all the packets from h* will go to server their designated server.
- install_flow_rule_server_to_client(): The same as the function above but from server to client.
- _handle_PacketIn(): The server sends an ARP request to the LB, to connect to the client.
- send_arp_reply(): Sends the ARP reply message to the server.

```
[SimpleLoadBalancer     ] FUNCTION: _handle_PacketIn
[SimpleLoadBalancer     ] Received IP Packet from 10.0.0.1
[SimpleLoadBalancer     ] FUNCTION: update_lb_mapping
[SimpleLoadBalancer     ] FUNCTION: install_flow_rule_client_to_server
[SimpleLoadBalancer     ] FUNCTION: install_flow_rule_server_to_client
[SimpleLoadBalancer     ] Installed flow rule: 10.0.0.5 -> 10.0.0.1
[SimpleLoadBalancer     ] Installed flow rule: 10.0.0.1 -> 10.0.0.5
[SimpleLoadBalancer     ] FUNCTION: _handle_PacketIn
[SimpleLoadBalancer     ] Received ARP Packet
[SimpleLoadBalancer     ] ARP REQUEST Received
[SimpleLoadBalancer     ] Client 10.0.0.1 sent ARP req to LB 10.0.0.254
[SimpleLoadBalancer     ] FUNCTION: send_arp_reply
[SimpleLoadBalancer     ] FUNCTION: _handle_PacketIn
[SimpleLoadBalancer     ] Received IP Packet from 10.0.0.2
[SimpleLoadBalancer     ] FUNCTION: update_lb_mapping
[SimpleLoadBalancer     ] FUNCTION: install_flow_rule_client_to_server
[SimpleLoadBalancer     ] FUNCTION: install_flow_rule_server_to_client
[SimpleLoadBalancer     ] Installed flow rule: 10.0.0.6 -> 10.0.0.2
[SimpleLoadBalancer     ] Installed flow rule: 10.0.0.2 -> 10.0.0.6
[SimpleLoadBalancer     ] FUNCTION: _handle_PacketIn
[SimpleLoadBalancer     ] Received ARP Packet
[SimpleLoadBalancer     ] ARP REQUEST Received
[SimpleLoadBalancer     ] Client 10.0.0.2 sent ARP req to LB 10.0.0.254
[SimpleLoadBalancer     ] FUNCTION: send_arp_reply
[SimpleLoadBalancer     ] FUNCTION: _handle_PacketIn
[SimpleLoadBalancer     ] Received ARP Packet
[SimpleLoadBalancer     ] ARP REQUEST Received
[SimpleLoadBalancer     ] Server 10.0.0.6 sent ARP req to client
[SimpleLoadBalancer     ] FUNCTION: send_arp_reply
```

Figure 5: h1, h2 ping -c 2 10.0.0.254 second time

```
[SimpleLoadBalancer     ] FUNCTION: _handle_PacketIn
[SimpleLoadBalancer     ] Received ARP Packet
[SimpleLoadBalancer     ] ARP REQUEST Received
[SimpleLoadBalancer     ] Client 10.0.0.3 sent ARP req to LB 10.0.0.254
[SimpleLoadBalancer     ] FUNCTION: send_arp_reply
[SimpleLoadBalancer     ] FUNCTION: _handle_PacketIn
[SimpleLoadBalancer     ] Received IP Packet from 10.0.0.3
[SimpleLoadBalancer     ] FUNCTION: update_lb_mapping
[SimpleLoadBalancer     ] FUNCTION: round_robin
[SimpleLoadBalancer     ] Round robin selected: 10.0.0.7
[SimpleLoadBalancer     ] FUNCTION: install_flow_rule_client_to_server
[SimpleLoadBalancer     ] FUNCTION: install_flow_rule_server_to_client
[SimpleLoadBalancer     ] Installed flow rule: 10.0.0.7 -> 10.0.0.3
[SimpleLoadBalancer     ] Installed flow rule: 10.0.0.3 -> 10.0.0.7
[SimpleLoadBalancer     ] FUNCTION: _handle_PacketIn
[SimpleLoadBalancer     ] Received ARP Packet
[SimpleLoadBalancer     ] ARP REQUEST Received
[SimpleLoadBalancer     ] Server 10.0.0.7 sent ARP req to client
[SimpleLoadBalancer     ] FUNCTION: send_arp_reply
[SimpleLoadBalancer     ] FUNCTION: _handle_PacketIn
[SimpleLoadBalancer     ] Received ARP Packet
[SimpleLoadBalancer     ] ARP REQUEST Received
[SimpleLoadBalancer     ] Client 10.0.0.4 sent ARP req to LB 10.0.0.254
[SimpleLoadBalancer     ] FUNCTION: send_arp_reply
[SimpleLoadBalancer     ] FUNCTION: _handle_PacketIn
[SimpleLoadBalancer     ] Received IP Packet from 10.0.0.4
[SimpleLoadBalancer     ] FUNCTION: update_lb_mapping
[SimpleLoadBalancer     ] FUNCTION: round_robin
[SimpleLoadBalancer     ] Round robin selected: 10.0.0.5
[SimpleLoadBalancer     ] FUNCTION: install_flow_rule_client_to_server
[SimpleLoadBalancer     ] FUNCTION: install_flow_rule_server_to_client
[SimpleLoadBalancer     ] Installed flow rule: 10.0.0.5 -> 10.0.0.4
[SimpleLoadBalancer     ] Installed flow rule: 10.0.0.4 -> 10.0.0.5
[SimpleLoadBalancer     ] FUNCTION: _handle_PacketIn
[SimpleLoadBalancer     ] Received ARP Packet
[SimpleLoadBalancer     ] ARP REQUEST Received
[SimpleLoadBalancer     ] Server 10.0.0.5 sent ARP req to client
[SimpleLoadBalancer     ] FUNCTION: send_arp_reply
```

Figure 6: h3, h4 ping -c 2 10.0.0.254 second time

The second time, it uses the following functions:

- _handle_PacketIn(): It receives an IP package from the client
- update_lb_mapping(): Map between LB and client (LOADBALANCER_MAP) in the case it has changed.

- install_flow_rule_client_to_server(): Install a flow rule between server and client. Install a flow rule between server and client.
- install_flow_rule_server_to_client(): The same as the function above but from server to client.
- _handle_PacketIn(): LB intercepts ARP from Client -¿ Server
- send_arp_reply(): As the LB has its MAC address, it sends the ARP reply message to the client.

This is due to the fact that the controller already has the previous server the client had connected to.

4. Save the pcapng and submit it with your report.

   Attached in Appendix B and in a different file in the submission of the Lab 6.

5 Questions

5.1 What type of packets does the load balancer need to manage, in order to behave as explained in Section 3?

It needs ARP and IP packages. This can be seen in the function *_handle_PacketIn()*, as it is implemented for handling ARP packages and IP packages (lines 185 and 215). It can also be seen in the functions implemented as *send_arp_reply()* or send_arp_request() (lines 96 and 63 respectively).

5.2 Which messages are sent on the "internal" network (between s1 and h5-h7) when s1 connects to the controller?

```
root@14321e75797a:/home/ubuntu/pox# cd /home/ubuntu/pox && ./pox.py log.level --DEBUG SimpleLoadBalan
cer --loadbalancer=10.0.0.254 --servers=10.0.0.5,10.0.0.6,10.0.0.7
POX 0.3.0 (dart) / Copyright 2011-2014 James McCauley, et al.
[SimpleLoadBalancer     ] Loading Simple Load Balancer module:

---------------------------------CONFIG---------------------------------

[SimpleLoadBalancer     ] Loadbalancer IP: 10.0.0.254
[SimpleLoadBalancer     ] Backend Server IPs: 10.0.0.5, 10.0.0.6, 10.0.0.7

------------------------------------------------------------------------


[core                   ] POX 0.3.0 (dart) going up...
[core                   ] Running on CPython (2.7.17/Mar 18 2022 13:21:42)
[core                   ] Platform is Linux-5.4.0-107-generic-x86_64-with-Ubuntu-18.04-bionic
[core                   ] POX 0.3.0 (dart) is up.
[openflow.of_01         ] Listening on 0.0.0.0:6633
[openflow.of_01         ] [00-00-00-00-00-01 2] connected
[SimpleLoadBalancer     ] FUNCTION: _handle_ConnectionUp
[SimpleLoadBalancer     ] FUNCTION: send_arp_request
[SimpleLoadBalancer     ] FUNCTION: send_arp_request
[SimpleLoadBalancer     ] FUNCTION: send_arp_request
[SimpleLoadBalancer     ] Sent ARP Requests to all servers
[SimpleLoadBalancer     ] FUNCTION: _handle_PacketIn
[SimpleLoadBalancer     ] Received ARP Packet
[SimpleLoadBalancer     ] ARP REPLY Received
[SimpleLoadBalancer     ] FUNCTION: _handle_PacketIn
[SimpleLoadBalancer     ] Received ARP Packet
[SimpleLoadBalancer     ] ARP REPLY Received
[SimpleLoadBalancer     ] FUNCTION: _handle_PacketIn
[SimpleLoadBalancer     ] Received ARP Packet
[SimpleLoadBalancer     ] ARP REPLY Received
```

Figure 7: Messages sent from s1 to h5-h7

- _handle_ConnectionUp(): When s1 connects to the controller, this function is the one in charge of sending the ARP request of each server, to learn the MAC addresses of each one
- send_arp_request(): Each one for each server, in this case 3, because there are 3 servers.

- _handle_PacketIn(): Each one for each server. Handles the ARP requests to send a reply.

In Wireshark, we can search for it as Figure 8. I have only added the screenshot and not in the Wireshark trace given, as in the given trace I just kept the TCP because if not it was too messy.

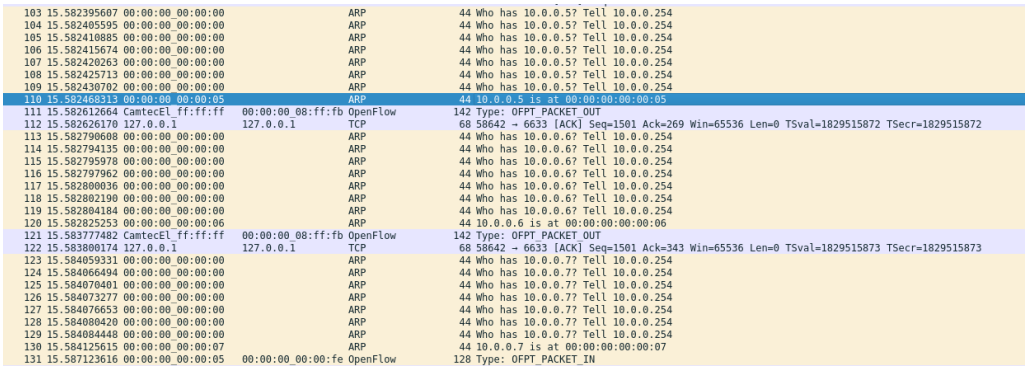

Figure 8: ARP messages sent from s1 to h5-h7

5.3 Which messages are sent when h1 pings the service at 10.0.0.254? Include traffic between h1 and switch, switch and the server, and switch and controller, in your answer.

Figure 9: Messages sent from h1 to h5-h7

The introduction to the explanation of Figure 9 is given in Section 4, when the first time the client tries to reach the server.

- Traffic between h1 and switch (LB): h1 sends an ARP request to find the MAC address of the IP 10.0.0.254.
- Traffic between the switch (LB) and controller: LB forward the message to the controller, as it does not have yet the information needed in the forwarding table. It will ask for instructions (OFPT_PACKAGE_IN) to the controller and it will answer with an OFPT_PACKAGE_OUT saying to send a packet to h1.
- Traffic between the switch (LB) and server: h1 gets an ARP reply with the MAC address of the LB.
- Traffic between h1 and switch (LB): Sends an echo request.

- Traffic between the switch (LB) and controller: LB gets an IP package do it asks the controller what to do with it and it tells to install a flow rule (OFPT_FLOW_MOD).
- Traffic between the switch (LB) and server: The echo request from h1 is forwarded to the server. It will reply with an echo answer, but it needs the MAC address of h1 so it asks for it.
- Traffic between the switch (LB) and controller: It will ask for instructions (OFPT_PACKAGE_IN) to the controller and it will answer with an OFPT_PACKAGE_OUT saying to send the package to 10.0.0.5 with an ARP reply.
- Traffic between the switch (LB) and server: The server gets an ARP reply with the MAC address of h1 and it sends an echo reply to the LB for it to forward the packet to h1.

5.4 Show that the load balancer distributes the traffic to the backend servers according to the RR scheduling algorithm. Support your answer with your pcapng file, and output from the Controller.

Client	Server
10.0.0.1	10.0.0.5
10.0.0.2	10.0.0.6
10.0.0.3	10.0.0.7
10.0.0.4	10.0.0.5

Table 1: Clients connections to servers from Controller output

In Figure 3 and Figure 4 we can see that the Round Robin function (*[SimpleLoadBalancer] Round Robin selected: 10.0.0.**) is choosing these servers which we see reflected in Table 1.

In the pcapng file we see it reflected in the following packages: Table 2 shows how a package is

No.	Src	Dst	Protocol	Length	Info
29	**00:00:00_05**:00:00	Cisco_10:00:00	OpenFlow	364	OFPT_PACKET_OUT
48	**00:00:00_06**:00:00	Cisco_10:00:00	OpenFlow	236	OFPT_PACKET_OUT
75	**00:00:00_07**:00:00	Cisco_10:00:00	OpenFlow	236	OFPT_PACKET_OUT
98	**00:00:00_05**:00:00	Cisco_10:00:00	OpenFlow	364	OFPT_PACKET_OUT
113	**00:00:00_05**:00:00	Cisco_10:00:00	OpenFlow	236	OFPT_PACKET_OUT
125	**00:00:00_06**:00:00	Cisco_10:00:00	OpenFlow	236	OFPT_PACKET_OUT
141	**00:00:00_07**:00:00	Cisco_10:00:00	OpenFlow	236	OFPT_PACKET_OUT
165	**00:00:00_05**:00:00	Cisco_10:00:00	OpenFlow	236	OFPT_PACKET_OUT

Table 2: Servers connection to load balancer from pcapng file

sent from the servers to the clients. We know that each package correspond to each client because it is what the Table 3 is showing.

No.	Src	Dst	Protocol	Info
26	10.0.0.1	10.0.0.254	OpenFlow	Type: OFPT_PACKET_IN
42	10.0.0.2	10.0.0.254	OpenFlow	Type: OFPT_PACKET_IN
70	10.0.0.3	10.0.0.254	OpenFlow	Type: OFPT_PACKET_IN
94	10.0.0.4	10.0.0.254	OpenFlow	Type: OFPT_PACKET_IN
107	10.0.0.1	10.0.0.254	OpenFlow	Type: OFPT_PACKET_IN
119	10.0.0.2	10.0.0.254	OpenFlow	Type: OFPT_PACKET_IN
136	10.0.0.3	10.0.0.254	OpenFlow	Type: OFPT_PACKET_IN
159	10.0.0.4	10.0.0.254	OpenFlow	Type: OFPT_PACKET_IN

Table 3: Clients connection to load balancer

Appendix

A SimpleLoadBalancer.py

```
from pox.core import core
from pox.openflow import *
import pox.openflow.libopenflow_01 as of
from pox.lib.packet.arp import arp
from pox.lib.packet.ipv4 import ipv4
from pox.lib.addresses import EthAddr, IPAddr
log = core.getLogger()
import time
import random
import pox.log.color


IDLE_TIMEOUT = 10
LOADBALANCER_MAC = EthAddr("00:00:00:00:00:FE") # <requested mac address>
ETHERNET_BROADCAST_ADDRESS=EthAddr("ff:ff:ff:ff:ff:ff")
N_SERVERS = 3

class SimpleLoadBalancer(object):

  def __init__(self, service_ip, server_ips = []):
    core.openflow.addListeners(self)
    self.SERVERS = {} #
    ↪  IPAddr[SERVER_IP]={'server_mac':EthAddr(SERVER_MAC),'port':
    ↪  PORT_TO_SERVER}
    self.CLIENTS = {} #
    ↪  IPAddr[CLIENT_IP]={'client_mac':EthAddr(CLIENT_MAC),'port':
    ↪  PORT_TO_CLIENT}
    self.LOADBALANCER_MAP = {} # Mapping between clients and servers
    self.LOADBALANCER_IP = service_ip
    self.SERVER_IPS = server_ips
    self.ROBIN_COUNT = 0


  def _handle_ConnectionUp(self, event):
    self.connection = event.connection
    log.debug("FUNCTION: _handle_ConnectionUp")
    # Send ARP Requests to learn the MAC address of all Backend Servers.
    for ip in self.SERVER_IPS:
      self.send_arp_request(self.connection, ip)
    log.debug("Sent ARP Requests to all servers")


  def round_robin(self):
    log.debug("FUNCTION: round_robin")
    # Implement logic to choose the next server according to the Round Robin
    ↪  scheduling algorithm
    # 4 Clients and 3 servers
    server_names = self.SERVER_IPS
    if self.ROBIN_COUNT < len(self.SERVER_IPS):
      server = server_names[self.ROBIN_COUNT]
      self.ROBIN_COUNT += 1
    else:
```

```
48        self.ROBIN_COUNT = 0
49        server = server_names[self.ROBIN_COUNT]
50        self.ROBIN_COUNT += 1
51      log.info("Round robin selected: %s" % server)
52      return server
53
54
55    def update_lb_mapping(self, client_ip):
56      log.debug("FUNCTION: update_lb_mapping")
57      if client_ip in self.CLIENTS.keys():
58        if client_ip not in self.LOADBALANCER_MAP.keys():
59          selected_server = self.round_robin() # select the server which will
            ↪  handle the request
60          self.LOADBALANCER_MAP[client_ip]=selected_server
61
62
63    def send_arp_reply(self, packet, connection, outport):
64      log.debug("FUNCTION: send_arp_reply")
65
66      arp_rep = arp() # Create an ARP reply; Help from
        ↪  https://noxrepo.github.io/pox-doc/html/#example-arp-messages
67      arp_rep.hwtype = arp_rep.HW_TYPE_ETHERNET
68      arp_rep.prototype = arp_rep.PROTO_TYPE_IP
69      arp_rep.hwlen = 6
70      arp_rep.protolen = arp_rep.protolen
71      arp_rep.opcode = arp.REPLY # Set the ARP TYPE to REPLY
72
73      arp_rep.hwdst = packet.src # Set MAC destination
74      arp_rep.hwsrc = LOADBALANCER_MAC # Set MAC source <requested mac address>
75
76      # Reverse the src, dest to have an answer; Help from l3_learning.py
77      arp_rep.protosrc =  packet.payload.protodst # Set IP source <IP of requested
        ↪  mac-associated machine>
78      arp_rep.protodst = packet.payload.protosrc # Set IP destination
79
80      eth = ethernet() # Create an ethernet frame and set the arp_rep as it's
        ↪  payload.
81      # eth = ethernet(type=packet.type, src=LOADBALANCER_MAC, dst=packet.src)
82
83      eth.type = packet.ARP_TYPE # Set packet Typee
84      eth.dst = packet.src # Set destination of the Ethernet Frame ######
85      eth.src = LOADBALANCER_MAC # Set source of the Ethernet Frame <requested mac
        ↪  address>
86      eth.set_payload(arp_rep)
87
88      msg = of.ofp_packet_out() # create the necessary Openflow Message to make the
        ↪  switch send the ARP Reply
89      msg.data = eth.pack()
90
91      msg.actions.append(of.ofp_action_output(port = of.OFPP_IN_PORT)) # Append the
        ↪  output port which the packet should be forwarded to; Help from Lab5 T4
92      msg.in_port = outport
93      connection.send(msg)
94
95
96    def send_arp_request(self, connection, ip):
97      # Help from https://en.wikipedia.org/wiki/Address_Resolution_Protocol
        ↪  #Example
```

```
98
99      log.debug("FUNCTION: send_arp_request")
100
101     arp_req = arp() # Create an instance of an ARP REQUEST PACKET
102     arp_req.hwtype = arp_req.HW_TYPE_ETHERNET
103     arp_req.prototype = arp_req.PROTO_TYPE_IP
104     arp_req.hwlen = 6
105     arp_req.protolen = arp_req.protolen
106     arp_req.opcode = arp_req.REQUEST # Set the opcode
107     arp_req.protodst = ip # IP the load balancer is looking for; From the
        ↪ example: 192.168.0.55
108     arp_req.hwsrc = LOADBALANCER_MAC # Set the MAC source of the ARP REQUEST
109     arp_req.hwdst = ETHERNET_BROADCAST_ADDRESS # Set the MAC address in such a
        ↪ way that the packet is marked as a Broadcast; FF:FF:FF:FF:FF:FF
110     arp_req.protosrc = self.LOADBALANCER_IP # Set the IP source of the ARP
        ↪ REQUEST
111
112     eth = ethernet() # Create an ethernet frame and set the arp_req as it's
        ↪ payload.
113     eth.type = eth.ARP_TYPE # Set packet Type
114     eth.dst = ETHERNET_BROADCAST_ADDRESS # Set the MAC address in such a way that
        ↪ the packet is marked as a Broadcast; FF:FF:FF:FF:FF:FF
115     eth.set_payload(arp_req)
116
117     msg = of.ofp_packet_out() # create the necessary Openflow Message to make the
        ↪ switch send the ARP Request
118     msg.data = eth.pack()
119     msg.actions.append(of.ofp_action_nw_addr(of.OFPAT_SET_NW_DST,ip))
120
121     msg.actions.append(of.ofp_action_output(port = of.OFPP_FLOOD)) # append an
        ↪ action to the message which makes the switch flood the packet out; Help
        ↪ from Lab5 T4
122     msg.in_port = of.OFPP_NONE
123     connection.send(msg)
124
125
126   def install_flow_rule_client_to_server(self,event, connection, outport,
      ↪ client_ip, server_ip):
127     log.debug("FUNCTION: install_flow_rule_client_to_server")
128     self.install_flow_rule_server_to_client(connection, event.port,
        ↪ server_ip,client_ip)
129
130     msg = of.ofp_flow_mod() # Create an instance of the type of Openflow packet
        ↪ you need to install flow table entries
131     msg.idle_timeout = IDLE_TIMEOUT
132
133     msg.match.dl_type = ethernet.IP_TYPE
134
135      # MATCH on destination and source IP; Help from
        ↪ https://noxrepo.github.io/pox-doc/html/#match-structure
136     msg.match.nw_dst = self.LOADBALANCER_IP
137     msg.match.nw_src = client_ip
138
139     # SET dl_addr source and destination addresses; Help from
        ↪ https://noxrepo.github.io/pox-doc/html/#openflow-actions
140     msg.actions.append(of.ofp_action_dl_addr.set_src (LOADBALANCER_MAC))
141     msg.actions.append(of.ofp_action_dl_addr.set_dst
        ↪ (self.SERVERS[server_ip].get("server_mac")))
```

```
142
143       # SET nw_addr source and destination addresses
144       msg.actions.append(of.ofp_action_nw_addr.set_src(client_ip))
145       msg.actions.append(of.ofp_action_nw_addr.set_dst(server_ip))
146
147       msg.actions.append(of.ofp_action_output(port=outport)) # Set Port to send
          ↪ matching packets out; Help from Lab5 T4
148       self.connection.send(msg)
149       log.info("Installed flow rule: %s -> %s" % (client_ip,server_ip))
150
151
152     def install_flow_rule_server_to_client(self, connection, outport, server_ip,
        ↪ client_ip):
153       log.debug("FUNCTION: install_flow_rule_server_to_client")
154
155       msg = of.ofp_flow_mod() # Create an instance of the type of Openflow packet
          ↪ you need to install flow table entries; Help from
          ↪ https://noxrepo.github.io/pox-doc/html/#match-structure
156       msg.idle_timeout = IDLE_TIMEOUT
157
158       msg.match.dl_type=ethernet.IP_TYPE
159
160       # MATCH on destination and source IP
161       msg.match.nw_src = server_ip
162       msg.match.nw_dst = client_ip
163
164       # SET dl_addr source and destination addresses; Help from
          ↪ https://noxrepo.github.io/pox-doc/html/#openflow-actions
165       msg.actions.append(of.ofp_action_dl_addr.set_dst
          ↪ (self.CLIENTS[client_ip].get("client_mac")))
166       msg.actions.append(of.ofp_action_dl_addr.set_src (LOADBALANCER_MAC))
167
168       # SET nw_addr source and destination addresses
169       msg.actions.append(of.ofp_action_nw_addr.set_src (self.LOADBALANCER_IP))
170       msg.actions.append(of.ofp_action_nw_addr.set_dst(client_ip))
171
172       msg.actions.append(of.ofp_action_output(port=outport)) # Set Port to send
          ↪ matching packets out; Help from Lab5 T4
173       self.connection.send(msg)
174       log.info("Installed flow rule: %s -> %s" % (server_ip,client_ip))
175
176
177     def _handle_PacketIn(self, event):
178       log.debug("FUNCTION: _handle_PacketIn")
179       packet = event.parsed
180       connection = event.connection
181       inport = event.port
182       if packet.type == packet.LLDP_TYPE or packet.type == packet.IPV6_TYPE:
183         log.info("Received LLDP or IPv6 Packet...")
184
185       elif packet.type == packet.ARP_TYPE: # Handle ARP Packets; Help from
          ↪ https://noxrepo.github.io/pox-doc/html/#example-arp-messages
186         log.debug("Received ARP Packet")
187         response = packet.payload
188         if packet.payload.opcode == packet.payload.REPLY: # Handle ARP replies
189           log.debug("ARP REPLY Received")
190           if response.protosrc not in self.SERVERS.keys():
```

```
191             # Add Servers MAC and port to SERVERS dict if there is no ip in the
                ↪  server dict
192             # self.send_arp_request(packet, connection, packet.next.dstip)
193             self.SERVERS[IPAddr(response.protosrc)]={
                ↪  'server_mac':EthAddr(packet.payload.hwsrc),'port': inport}
194
195         elif packet.payload.opcode == packet.payload.REQUEST: # Handle ARP
            ↪  requests
196           log.debug("ARP REQUEST Received")
197           if (response.protosrc not in self.SERVERS.keys() and response.protosrc
              ↪  not in self.CLIENTS.keys()): # if the IP is not in the clienet disct,
              ↪  it adds it
198             self.CLIENTS[response.protosrc]={
                ↪  'client_mac':EthAddr(packet.payload.hwsrc),'port':inport} #insert
                ↪  client's ip  mac and port to a forwarding table
199
200           if (response.protosrc in self.CLIENTS.keys() and response.protodst ==
              ↪  self.LOADBALANCER_IP):
201             log.info("Client %s sent ARP req to LB %s"%(response.protosrc,response.
                ↪  protodst))
202             # Load Balancer intercepts ARP Client -> Server
203             self.send_arp_reply(packet, connection, inport) # Send ARP Reply to the
                ↪  client, include the event.connection object
204
205           elif (response.protosrc in self.SERVERS.keys() and response.protodst in
              ↪  self.CLIENTS.keys()):
206             log.info("Server %s sent ARP req to client"%response.protosrc)
207             # Load Balancer intercepts ARP from Client <- Server
208             self.send_arp_reply(packet, connection, inport) # Send ARP Reply to the
                ↪  Server, include the event.connection object
209
210
211           else:
212             log.info("Invalid ARP request")
213           ######################################### MIRAR
214
215       elif packet.type == packet.IP_TYPE: # Handle IP Packets
216         log.debug("Received IP Packet from %s" % packet.next.srcip)
217         # Handle Requests from Clients to Servers
218         # Install flow rule Client -> Server
219         if (packet.next.dstip == self.LOADBALANCER_IP and packet.next.srcip not in
            ↪  self.SERVERS.keys()): # Check if the packet is destined for the LB and
            ↪  the source is not a server
220
221           self.update_lb_mapping(packet.next.srcip)
222           client_ip =  packet.payload.srcip # Get client IP from the packet
223           server_ip = self.LOADBALANCER_MAP.get(packet.next.srcip)
224           outport = self.SERVERS[server_ip].get('port') # Get Port of Server
225           self.install_flow_rule_client_to_server(event,con nection, outport,
              ↪  client_ip,server_ip)
226
227           # Either use the code below to create a new Ethernet packet, or use
              ↪  Buffer_Id
228           # Used the code with help of https://noxrepo.github.io/pox-doc/html/#ethe
              ↪  rnet-ethernet
229           eth = ethernet()
230           eth.type = ethernet.IP_TYPE # Set the correct Ethernet TYPE, to send an
              ↪  IP Packet
```

```
231         eth.dst = self.SERVERS[server_ip].get('server_mac') # Set the MAC
            ↪ destination
232         eth.src = LOADBALANCER_MAC # Set the MAC source
233         eth.set_payload(packet.next)
234
235         # Send the first packet (which was sent to the controller from the
            ↪ switch)
236         # to the chosen server, so there is no packetloss
237         msg = of.ofp_packet_out() # Create an instance of a message which can be
            ↪ used to instruct the switch to send a packet
238         msg.data = eth.pack()
239         msg.in_port = inport # Set the correct in_port
240
241         msg.actions.append(of.ofp_action_dl_addr.set_src (LOADBALANCER_MAC)) #
            ↪ Add an action which sets the MAC source to the LB's MAC
242
243         msg.actions.append(of.ofp_action_dl_addr.set_dst
            ↪ (self.SERVERS[server_ip].get('server_mac'))) # Add an action which
            ↪ sets the MAC destination to the intended destination...
244
245         msg.actions.append(of.ofp_action_nw_addr.set_src (client_ip)) # Add an
            ↪ action which sets the IP source
246         msg.actions.append(of.ofp_action_nw_addr.set_dst (server_ip)) # Add an
            ↪ action which sets the IP destination
247         msg.actions.append(of.ofp_action_output(port= outport)) # Add an action
            ↪ which sets the Outport
248
249         connection.send(msg)
250
251       # Handle traffic from Server to Client
252       # Install flow rule Client <- Server
253       elif packet.next.dstip in self.CLIENTS.keys(): #server to client
254         log.info("Installing flow rule from Server -> Client")
255         if packet.next.srcip in self.SERVERS.keys():
256
257           server_ip =  packet.payload.srcip # Get the source IP from the IP
              ↪ Packet
258           client_ip = self.LOADBALANCER_MAP.keys()[list(sel
              ↪ f.LOADBALANCER_MAP.values()).index(packet.next.srcip)]
259         ########################################
260           outport=int(self.CLIENTS[client_ip].get(' port'))
261           self.install_flow_rule_server_to_client(c onnection, outport,
              ↪ server_ip,client_ip)
262
263           # Either use the code below to create a new Ethernet packet, or use
              ↪ Buffer_Id
264           # As did before, I have used the code below
265           eth = ethernet()
266           eth.type = ethernet.IP_TYPE # Set the correct Ethernet TYPE, to send an
              ↪ IP Packet
267           eth.dst = self.CLIENTS[client_ip].get('client _mac') # Set the MAC
              ↪ destination
268           eth.src = LOADBALANCER_MAC # Set the MAC source
269           eth.set_payload(packet.next)
270
271           # Send the first packet (which was sent to the controller from the
              ↪ switch)
272           # to the chosen server, so there is no packetloss
```

```
273             msg =  of.ofp_packet_out() # Create an instance of a message which can
                ↪  be used to instruct the switch to send a packet
274             msg.data = eth.pack()
275             msg.in_port = inport # Set the correct in_port
276
277             msg.action.append(of.ofp_action_dl_addr. set_src(self.LOADBALANCER_IP))
                ↪  # Add an action which sets the MAC source to the LB's MAC
278             msg.action.append(of.ofp_action_dl_addr.
                ↪  set_dst(self.CLIENTS[client_ip].get('client_mac'))) # Add an action
                ↪  which sets the MAC destination to the intended destination...
279
280             msg.action.append(of.ofp_action_nw_addr. set_src(self.LOADBALANCER_IP))
                ↪  # Add an action which sets the IP source
281             msg.action.append(of.ofp_action_nw_addr. set_dst(client_ip)) # Add an
                ↪  action which sets the IP destination
282             msg.actions.append(of.ofp_action_output( port=outport)) # Add an action
                ↪  which sets the Outport
283
284             self.connection.send(msg)
285
286       else:
287         log.info("Unknown Packet type: %s" % packet.type)
288         return
289       return
290
291
292   def launch(loadbalancer, servers):
293     # Color-coding and pretty-printing the log output
294     pox.log.color.launch()
295     pox.log.launch(format="[@@@bold@@@level%(name)-23s@@@reset] " +
296                    "@@@bold%(message)s@@@norm al")
297     log.info("Loading Simple Load Balancer
        ↪  module:\n\n-----------------------------------CONFIG-----------
        ↪  ----------------------\n")
298     server_ips = servers.replace(","," ").split()
299     server_ips = [IPAddr(x) for x in server_ips]
300     loadbalancer_ip = IPAddr(loadbalancer)
301     log.info("Loadbalancer IP: %s" % loadbalancer_ip)
302     log.info("Backend Server IPs:
        ↪  %s\n\n-----------------------------------------------------------------
        ↪  -----------\n\n" % ', '.join(str(ip) for ip in server_ips))
303     core.registerNew(SimpleLoadBalancer, loadbalancer_ip, server_ips)
```

B trace.pcapng

```
"No.", "Time", "Source", "Destination", "Protocol", "Length", "Info"
"1", "0.000000000", "127.0.0.1", "127.0.0.1", "OpenFlow", "74", "Type:
↪  OFPT_ECHO_REQUEST"
"2", "0.000022933", "127.0.0.1", "127.0.0.1", "TCP", "66", "6633  >  57536 [ACK]
↪  Seq=1 Ack=9 Win=512 Len=0 TSval=1828293290 TSecr=1828293289"
"3", "0.020597623", "127.0.0.1", "127.0.0.1", "OpenFlow", "74", "Type:
↪  OFPT_ECHO_REPLY"
"4", "0.020616178", "127.0.0.1", "127.0.0.1", "TCP", "66", "57536  >  6633 [ACK]
↪  Seq=9 Ack=9 Win=512 Len=0 TSval=1828293310 TSecr=1828293310"
"5", "2.278492762", "fe80::200:ff:fe00:3", "ff02::2", "OpenFlow", "154", "Type:
↪  OFPT_PACKET_IN"
```

```
"6", "2.278545461", "127.0.0.1", "127.0.0.1", "TCP", "66", "6633  >  57536 [ACK]
↪  Seq=9 Ack=97 Win=512 Len=0 TSval=1828295568 TSecr=1828295568"
"7", "3.302603868", "fe80::200:ff:fe00:1", "ff02::2", "OpenFlow", "154", "Type:
↪  OFPT_PACKET_IN"
"8", "3.302624527", "127.0.0.1", "127.0.0.1", "TCP", "66", "6633  >  57536 [ACK]
↪  Seq=9 Ack=185 Win=512 Len=0 TSval=1828296592 TSecr=1828296592"
"9", "3.302775641", "fe80::200:ff:fe00:7", "ff02::2", "OpenFlow", "154", "Type:
↪  OFPT_PACKET_IN"
"10", "3.302786010", "127.0.0.1", "127.0.0.1", "TCP", "66", "6633  >  57536 [ACK]
↪  Seq=9 Ack=273 Win=512 Len=0 TSval=1828296592 TSecr=1828296592"
"11", "4.068869306", "fe80::200:ff:fe00:5", "ff02::2", "OpenFlow", "154", "Type:
↪  OFPT_PACKET_IN"
"12", "4.068890205", "127.0.0.1", "127.0.0.1", "TCP", "66", "6633  >  57536 [ACK]
↪  Seq=9 Ack=361 Win=512 Len=0 TSval=1828297358 TSecr=1828297358"
"13", "4.326751029", "fe80::200:ff:fe00:2", "ff02::2", "OpenFlow", "154", "Type:
↪  OFPT_PACKET_IN"
"14", "4.326770306", "127.0.0.1", "127.0.0.1", "TCP", "66", "6633  >  57536 [ACK]
↪  Seq=9 Ack=449 Win=512 Len=0 TSval=1828297616 TSecr=1828297616"
"15", "4.326842080", "fe80::200:ff:fe00:6", "ff02::2", "OpenFlow", "154", "Type:
↪  OFPT_PACKET_IN"
"16", "4.326846829", "127.0.0.1", "127.0.0.1", "TCP", "66", "6633  >  57536 [ACK]
↪  Seq=9 Ack=537 Win=512 Len=0 TSval=1828297616 TSecr=1828297616"
"17", "4.326855426", "fe80::200:ff:fe00:4", "ff02::2", "OpenFlow", "154", "Type:
↪  OFPT_PACKET_IN"
"18", "4.326858591", "127.0.0.1", "127.0.0.1", "TCP", "66", "6633  >  57536 [ACK]
↪  Seq=9 Ack=625 Win=512 Len=0 TSval=1828297616 TSecr=1828297616"
"19", "5.001546426", "127.0.0.1", "127.0.0.1", "OpenFlow", "74", "Type:
↪  OFPT_ECHO_REQUEST"
"20", "5.001574579", "127.0.0.1", "127.0.0.1", "TCP", "66", "6633  >  57536 [ACK]
↪  Seq=9 Ack=633 Win=512 Len=0 TSval=1828298291 TSecr=1828298291"
"21", "5.005622626", "127.0.0.1", "127.0.0.1", "OpenFlow", "74", "Type:
↪  OFPT_ECHO_REPLY"
"22", "5.005643055", "127.0.0.1", "127.0.0.1", "TCP", "66", "57536  >  6633 [ACK]
↪  Seq=633 Ack=17 Win=512 Len=0 TSval=1828298295 TSecr=1828298295"
"23", "5.172720150", "00:00:00_00:00:01", "Broadcast", "OpenFlow", "126", "Type:
↪  OFPT_PACKET_IN"
"24", "5.186696623", "00:00:00_00:00:fe", "00:00:00_00:00:01", "OpenFlow", "132",
↪  "Type: OFPT_PACKET_OUT"
"25", "5.186722883", "127.0.0.1", "127.0.0.1", "TCP", "66", "57536  >  6633 [ACK]
↪  Seq=693 Ack=83 Win=512 Len=0 TSval=1828298476 TSecr=1828298476"
"26", "5.187515036", "10.0.0.1", "10.0.0.254", "OpenFlow", "182", "Type:
↪  OFPT_PACKET_IN"
"27", "5.197578546", "127.0.0.1", "127.0.0.1", "OpenFlow", "194", "Type:
↪  OFPT_FLOW_MOD"
"28", "5.241371933", "127.0.0.1", "127.0.0.1", "TCP", "66", "57536  >  6633 [ACK]
↪  Seq=809 Ack=211 Win=512 Len=0 TSval=1828298531 TSecr=1828298487"
"29", "5.241407119", "00:00:00_05:00:00", "Cisco_10:00:00", "OpenFlow", "364",
↪  "Type: OFPT_PACKET_OUT"
"30", "5.241417168", "127.0.0.1", "127.0.0.1", "TCP", "66", "57536  >  6633 [ACK]
↪  Seq=809 Ack=509 Win=510 Len=0 TSval=1828298531 TSecr=1828298531"
"31", "5.243033003", "00:00:00_00:00:05", "Broadcast", "OpenFlow", "126", "Type:
↪  OFPT_PACKET_IN"
"32", "5.278204929", "00:00:00_00:00:fe", "00:00:00_00:00:05", "OpenFlow", "132",
↪  "Type: OFPT_PACKET_OUT"
"33", "5.278265773", "127.0.0.1", "127.0.0.1", "TCP", "66", "57536  >  6633 [ACK]
↪  Seq=869 Ack=575 Win=512 Len=0 TSval=1828298568 TSecr=1828298568"
"34", "9.999045195", "127.0.0.1", "127.0.0.1", "OpenFlow", "74", "Type:
↪  OFPT_ECHO_REQUEST"
```

```
"35", "10.045665748", "127.0.0.1", "127.0.0.1", "TCP", "66", "6633  >  57536
↪  [ACK] Seq=575 Ack=877 Win=512 Len=0 TSval=1828303335 TSecr=1828303289"
"36", "10.049848043", "127.0.0.1", "127.0.0.1", "OpenFlow", "74", "Type:
↪  OFPT_ECHO_REPLY"
"37", "10.049869623", "127.0.0.1", "127.0.0.1", "TCP", "66", "57536  >  6633
↪  [ACK] Seq=877 Ack=583 Win=512 Len=0 TSval=1828303339 TSecr=1828303339"
"38", "14.786259417", "00:00:00_00:00:02", "Broadcast", "OpenFlow", "126", "Type:
↪  OFPT_PACKET_IN"
"39", "14.786317587", "127.0.0.1", "127.0.0.1", "TCP", "66", "6633  >  57536
↪  [ACK] Seq=583 Ack=937 Win=512 Len=0 TSval=1828308076 TSecr=1828308076"
"40", "14.812785621", "00:00:00_00:00:fe", "00:00:00_00:00:02", "OpenFlow",
↪  "132", "Type: OFPT_PACKET_OUT"
"41", "14.812804336", "127.0.0.1", "127.0.0.1", "TCP", "66", "57536  >  6633
↪  [ACK] Seq=937 Ack=649 Win=512 Len=0 TSval=1828308102 TSecr=1828308102"
"42", "14.813562986", "10.0.0.2", "10.0.0.254", "OpenFlow", "182", "Type:
↪  OFPT_PACKET_IN"
"43", "14.813577132", "127.0.0.1", "127.0.0.1", "TCP", "66", "6633  >  57536
↪  [ACK] Seq=649 Ack=1053 Win=512 Len=0 TSval=1828308103 TSecr=1828308103"
"44", "14.827087812", "127.0.0.1", "127.0.0.1", "OpenFlow", "194", "Type:
↪  OFPT_FLOW_MOD"
"45", "14.827107829", "127.0.0.1", "127.0.0.1", "TCP", "66", "57536  >  6633
↪  [ACK] Seq=1053 Ack=777 Win=511 Len=0 TSval=1828308117 TSecr=1828308117"
"46", "14.828533447", "127.0.0.1", "127.0.0.1", "OpenFlow", "194", "Type:
↪  OFPT_FLOW_MOD"
"47", "14.828547393", "127.0.0.1", "127.0.0.1", "TCP", "66", "57536  >  6633
↪  [ACK] Seq=1053 Ack=905 Win=510 Len=0 TSval=1828308118 TSecr=1828308118"
"48", "14.829893351", "00:00:00_06:00:00", "Cisco_10:00:00", "OpenFlow", "236",
↪  "Type: OFPT_PACKET_OUT"
"49", "14.829912968", "127.0.0.1", "127.0.0.1", "TCP", "66", "57536  >  6633
↪  [ACK] Seq=1053 Ack=1075 Win=509 Len=0 TSval=1828308119 TSecr=1828308119"
"50", "14.830788151", "00:00:00_00:00:06", "Broadcast", "OpenFlow", "126", "Type:
↪  OFPT_PACKET_IN"
"51", "14.830892447", "127.0.0.1", "127.0.0.1", "TCP", "66", "6633  >  57536
↪  [ACK] Seq=1075 Ack=1113 Win=512 Len=0 TSval=1828308120 TSecr=1828308120"
"52", "14.838505605", "00:00:00_00:00:fe", "00:00:00_00:00:06", "OpenFlow",
↪  "132", "Type: OFPT_PACKET_OUT"
"53", "14.880319736", "127.0.0.1", "127.0.0.1", "TCP", "66", "57536  >  6633
↪  [ACK] Seq=1113 Ack=1141 Win=512 Len=0 TSval=1828308170 TSecr=1828308128"
"54", "15.846217854", "fe80::200:ff:fe00:3", "ff02::2", "OpenFlow", "154", "Type:
↪  OFPT_PACKET_IN"
"55", "15.846279991", "127.0.0.1", "127.0.0.1", "TCP", "66", "6633  >  57536
↪  [ACK] Seq=1141 Ack=1201 Win=512 Len=0 TSval=1828309136 TSecr=1828309136"
"56", "18.659962327", "fe80::200:ff:fe00:7", "ff02::2", "OpenFlow", "154", "Type:
↪  OFPT_PACKET_IN"
"57", "18.659979550", "127.0.0.1", "127.0.0.1", "TCP", "66", "6633  >  57536
↪  [ACK] Seq=1141 Ack=1289 Win=512 Len=0 TSval=1828311950 TSecr=1828311949"
"58", "18.660041326", "fe80::200:ff:fe00:1", "ff02::2", "OpenFlow", "154", "Type:
↪  OFPT_PACKET_IN"
"59", "18.660046625", "127.0.0.1", "127.0.0.1", "TCP", "66", "6633  >  57536
↪  [ACK] Seq=1141 Ack=1377 Win=512 Len=0 TSval=1828311950 TSecr=1828311950"
"60", "18.999630382", "127.0.0.1", "127.0.0.1", "OpenFlow", "74", "Type:
↪  OFPT_ECHO_REQUEST"
"61", "18.999650390", "127.0.0.1", "127.0.0.1", "TCP", "66", "6633  >  57536
↪  [ACK] Seq=1141 Ack=1385 Win=512 Len=0 TSval=1828312289 TSecr=1828312289"
"62", "19.003770177", "127.0.0.1", "127.0.0.1", "OpenFlow", "74", "Type:
↪  OFPT_ECHO_REPLY"
"63", "19.003791407", "127.0.0.1", "127.0.0.1", "TCP", "66", "57536  >  6633
↪  [ACK] Seq=1385 Ack=1149 Win=512 Len=0 TSval=1828312293 TSecr=1828312293"
```

```
"64", "20.451777262", "fe80::200:ff:fe00:6", "ff02::2", "OpenFlow", "154", "Type:
↪  OFPT_PACKET_IN"
"65", "20.451838036", "127.0.0.1", "127.0.0.1", "TCP", "66", "6633  >  57536
↪  [ACK] Seq=1149 Ack=1473 Win=512 Len=0 TSval=1828313741 TSecr=1828313741"
"66", "20.614618201", "00:00:00_00:00:03", "Broadcast", "OpenFlow", "126", "Type:
↪  OFPT_PACKET_IN"
"67", "20.614636275", "127.0.0.1", "127.0.0.1", "TCP", "66", "6633  >  57536
↪  [ACK] Seq=1149 Ack=1533 Win=512 Len=0 TSval=1828313904 TSecr=1828313904"
"68", "20.631735789", "00:00:00_00:00:fe", "00:00:00_00:00:03", "OpenFlow",
↪  "132", "Type: OFPT_PACKET_OUT"
"69", "20.631753413", "127.0.0.1", "127.0.0.1", "TCP", "66", "57536  >  6633
↪  [ACK] Seq=1533 Ack=1215 Win=512 Len=0 TSval=1828313921 TSecr=1828313921"
"70", "20.632489425", "10.0.0.3", "10.0.0.254", "OpenFlow", "182", "Type:
↪  OFPT_PACKET_IN"
"71", "20.641239738", "127.0.0.1", "127.0.0.1", "OpenFlow", "194", "Type:
↪  OFPT_FLOW_MOD"
"72", "20.641262441", "127.0.0.1", "127.0.0.1", "TCP", "66", "57536  >  6633
↪  [ACK] Seq=1649 Ack=1343 Win=511 Len=0 TSval=1828313931 TSecr=1828313931"
"73", "20.643288426", "127.0.0.1", "127.0.0.1", "OpenFlow", "194", "Type:
↪  OFPT_FLOW_MOD"
"74", "20.643307121", "127.0.0.1", "127.0.0.1", "TCP", "66", "57536  >  6633
↪  [ACK] Seq=1649 Ack=1471 Win=510 Len=0 TSval=1828313933 TSecr=1828313933"
"75", "20.644714294", "00:00:00_07:00:00", "Cisco_10:00:00", "OpenFlow", "236",
↪  "Type: OFPT_PACKET_OUT"
"76", "20.644731797", "127.0.0.1", "127.0.0.1", "TCP", "66", "57536  >  6633
↪  [ACK] Seq=1649 Ack=1641 Win=509 Len=0 TSval=1828313934 TSecr=1828313934"
"77", "20.645802407", "00:00:00_00:00:07", "Broadcast", "OpenFlow", "126", "Type:
↪  OFPT_PACKET_IN"
"78", "20.653769565", "00:00:00_00:00:fe", "00:00:00_00:00:07", "OpenFlow",
↪  "132", "Type: OFPT_PACKET_OUT"
"79", "20.653795203", "127.0.0.1", "127.0.0.1", "TCP", "66", "57536  >  6633
↪  [ACK] Seq=1709 Ack=1707 Win=512 Len=0 TSval=1828313943 TSecr=1828313943"
"80", "20.707454155", "fe80::200:ff:fe00:5", "ff02::2", "OpenFlow", "154", "Type:
↪  OFPT_PACKET_IN"
"81", "20.750807095", "127.0.0.1", "127.0.0.1", "TCP", "66", "6633  >  57536
↪  [ACK] Seq=1707 Ack=1797 Win=512 Len=0 TSval=1828314040 TSecr=1828313997"
"82", "22.757700485", "fe80::200:ff:fe00:2", "ff02::2", "OpenFlow", "154", "Type:
↪  OFPT_PACKET_IN"
"83", "22.757728778", "127.0.0.1", "127.0.0.1", "TCP", "66", "6633  >  57536
↪  [ACK] Seq=1707 Ack=1885 Win=512 Len=0 TSval=1828316047 TSecr=1828316047"
"84", "22.757748425", "fe80::200:ff:fe00:4", "ff02::2", "OpenFlow", "154", "Type:
↪  OFPT_PACKET_IN"
"85", "22.757753625", "127.0.0.1", "127.0.0.1", "TCP", "66", "6633  >  57536
↪  [ACK] Seq=1707 Ack=1973 Win=512 Len=0 TSval=1828316047 TSecr=1828316047"
"86", "25.000912291", "127.0.0.1", "127.0.0.1", "OpenFlow", "74", "Type:
↪  OFPT_ECHO_REQUEST"
"87", "25.000935786", "127.0.0.1", "127.0.0.1", "TCP", "66", "6633  >  57536
↪  [ACK] Seq=1707 Ack=1981 Win=512 Len=0 TSval=1828318290 TSecr=1828318290"
"88", "25.012701864", "127.0.0.1", "127.0.0.1", "OpenFlow", "74", "Type:
↪  OFPT_ECHO_REPLY"
"89", "25.012723174", "127.0.0.1", "127.0.0.1", "TCP", "66", "57536  >  6633
↪  [ACK] Seq=1981 Ack=1715 Win=512 Len=0 TSval=1828318302 TSecr=1828318302"
"90", "25.902891619", "00:00:00_00:00:04", "Broadcast", "OpenFlow", "126", "Type:
↪  OFPT_PACKET_IN"
"91", "25.902955028", "127.0.0.1", "127.0.0.1", "TCP", "66", "6633  >  57536
↪  [ACK] Seq=1715 Ack=2041 Win=512 Len=0 TSval=1828319192 TSecr=1828319192"
"92", "25.933800069", "00:00:00_00:00:fe", "00:00:00_00:00:04", "OpenFlow",
↪  "132", "Type: OFPT_PACKET_OUT"
```

```
"93", "25.933822100", "127.0.0.1", "127.0.0.1", "TCP", "66", "57536  >  6633
↪  [ACK] Seq=2041 Ack=1781 Win=512 Len=0 TSval=1828319223 TSecr=1828319223"
"94", "25.935006044", "10.0.0.4", "10.0.0.254", "OpenFlow", "182", "Type:
↪  OFPT_PACKET_IN"
"95", "25.935124516", "127.0.0.1", "127.0.0.1", "TCP", "66", "6633  >  57536
↪  [ACK] Seq=1781 Ack=2157 Win=512 Len=0 TSval=1828319225 TSecr=1828319225"
"96", "25.942981653", "127.0.0.1", "127.0.0.1", "OpenFlow", "194", "Type:
↪  OFPT_FLOW_MOD"
"97", "25.985369179", "127.0.0.1", "127.0.0.1", "TCP", "66", "57536  >  6633
↪  [ACK] Seq=2157 Ack=1909 Win=512 Len=0 TSval=1828319275 TSecr=1828319232"
"98", "25.985412701", "00:00:00_05:00:00", "Cisco_10:00:00", "OpenFlow", "364",
↪  "Type: OFPT_PACKET_OUT"
"99", "25.985421989", "127.0.0.1", "127.0.0.1", "TCP", "66", "57536  >  6633
↪  [ACK] Seq=2157 Ack=2207 Win=510 Len=0 TSval=1828319275 TSecr=1828319275"
"100", "25.993832074", "00:00:00_00:00:05", "Broadcast", "OpenFlow", "126",
↪  "Type: OFPT_PACKET_IN"
"101", "26.023598467", "00:00:00_00:00:fe", "00:00:00_00:00:05", "OpenFlow",
↪  "132", "Type: OFPT_PACKET_OUT"
"102", "26.023622412", "127.0.0.1", "127.0.0.1", "TCP", "66", "57536  >  6633
↪  [ACK] Seq=2217 Ack=2273 Win=512 Len=0 TSval=1828319313 TSecr=1828319313"
"103", "31.000875356", "127.0.0.1", "127.0.0.1", "OpenFlow", "74", "Type:
↪  OFPT_ECHO_REQUEST"
"104", "31.046035642", "127.0.0.1", "127.0.0.1", "TCP", "66", "6633  >  57536
↪  [ACK] Seq=2273 Ack=2225 Win=512 Len=0 TSval=1828324336 TSecr=1828324290"
"105", "31.048373021", "127.0.0.1", "127.0.0.1", "OpenFlow", "74", "Type:
↪  OFPT_ECHO_REPLY"
"106", "31.048393670", "127.0.0.1", "127.0.0.1", "TCP", "66", "57536  >  6633
↪  [ACK] Seq=2225 Ack=2281 Win=512 Len=0 TSval=1828324338 TSecr=1828324338"
"107", "33.531378698", "10.0.0.1", "10.0.0.254", "OpenFlow", "182", "Type:
↪  OFPT_PACKET_IN"
"108", "33.531401421", "127.0.0.1", "127.0.0.1", "TCP", "66", "6633  >  57536
↪  [ACK] Seq=2281 Ack=2341 Win=512 Len=0 TSval=1828326821 TSecr=1828326821"
"109", "33.544133011", "127.0.0.1", "127.0.0.1", "OpenFlow", "194", "Type:
↪  OFPT_FLOW_MOD"
"110", "33.544157207", "127.0.0.1", "127.0.0.1", "TCP", "66", "57536  >  6633
↪  [ACK] Seq=2341 Ack=2409 Win=511 Len=0 TSval=1828326834 TSecr=1828326834"
"111", "33.546289071", "127.0.0.1", "127.0.0.1", "OpenFlow", "194", "Type:
↪  OFPT_FLOW_MOD"
"112", "33.546338474", "127.0.0.1", "127.0.0.1", "TCP", "66", "57536  >  6633
↪  [ACK] Seq=2341 Ack=2537 Win=510 Len=0 TSval=1828326836 TSecr=1828326836"
"113", "33.547502400", "00:00:00_05:00:00", "Cisco_10:00:00", "OpenFlow", "236",
↪  "Type: OFPT_PACKET_OUT"
"114", "33.547518279", "127.0.0.1", "127.0.0.1", "TCP", "66", "57536  >  6633
↪  [ACK] Seq=2341 Ack=2707 Win=509 Len=0 TSval=1828326837 TSecr=1828326837"
"115", "38.002193760", "127.0.0.1", "127.0.0.1", "OpenFlow", "74", "Type:
↪  OFPT_ECHO_REQUEST"
"116", "38.002258522", "127.0.0.1", "127.0.0.1", "TCP", "66", "6633  >  57536
↪  [ACK] Seq=2707 Ack=2349 Win=512 Len=0 TSval=1828331292 TSecr=1828331292"
"117", "38.041485894", "127.0.0.1", "127.0.0.1", "OpenFlow", "74", "Type:
↪  OFPT_ECHO_REPLY"
"118", "38.041509028", "127.0.0.1", "127.0.0.1", "TCP", "66", "57536  >  6633
↪  [ACK] Seq=2349 Ack=2715 Win=512 Len=0 TSval=1828331331 TSecr=1828331331"
"119", "42.436354646", "10.0.0.2", "10.0.0.254", "OpenFlow", "182", "Type:
↪  OFPT_PACKET_IN"
"120", "42.436413256", "127.0.0.1", "127.0.0.1", "TCP", "66", "6633  >  57536
↪  [ACK] Seq=2715 Ack=2465 Win=512 Len=0 TSval=1828335726 TSecr=1828335726"
"121", "42.450779116", "127.0.0.1", "127.0.0.1", "OpenFlow", "194", "Type:
↪  OFPT_FLOW_MOD"
```

```
"122", "42.450800987", "127.0.0.1", "127.0.0.1", "TCP", "66", "57536  >  6633
↪  [ACK] Seq=2465 Ack=2843 Win=511 Len=0 TSval=1828335740 TSecr=1828335740"
"123", "42.452107741", "127.0.0.1", "127.0.0.1", "OpenFlow", "194", "Type:
↪  OFPT_FLOW_MOD"
"124", "42.452130294", "127.0.0.1", "127.0.0.1", "TCP", "66", "57536  >  6633
↪  [ACK] Seq=2465 Ack=2971 Win=510 Len=0 TSval=1828335742 TSecr=1828335742"
"125", "42.453156331", "00:00:00_06:00:00", "Cisco_10:00:00", "OpenFlow", "236",
↪  "Type: OFPT_PACKET_OUT"
"126", "42.453174395", "127.0.0.1", "127.0.0.1", "TCP", "66", "57536  >  6633
↪  [ACK] Seq=2465 Ack=3141 Win=509 Len=0 TSval=1828335743 TSecr=1828335743"
"127", "43.238934307", "fe80::200:ff:fe00:3", "ff02::2", "OpenFlow", "154",
↪  "Type: OFPT_PACKET_IN"
"128", "43.239086793", "127.0.0.1", "127.0.0.1", "TCP", "66", "6633  >  57536
↪  [ACK] Seq=3141 Ack=2553 Win=512 Len=0 TSval=1828336528 TSecr=1828336528"
"129", "46.999605983", "127.0.0.1", "127.0.0.1", "OpenFlow", "74", "Type:
↪  OFPT_ECHO_REQUEST"
"130", "46.999627052", "127.0.0.1", "127.0.0.1", "TCP", "66", "6633  >  57536
↪  [ACK] Seq=3141 Ack=2561 Win=512 Len=0 TSval=1828340289 TSecr=1828340289"
"131", "47.015202414", "127.0.0.1", "127.0.0.1", "OpenFlow", "74", "Type:
↪  OFPT_ECHO_REPLY"
"132", "47.015225427", "127.0.0.1", "127.0.0.1", "TCP", "66", "57536  >  6633
↪  [ACK] Seq=2561 Ack=3149 Win=512 Len=0 TSval=1828340305 TSecr=1828340305"
"133", "49.382652667", "fe80::200:ff:fe00:1", "ff02::2", "OpenFlow", "154",
↪  "Type: OFPT_PACKET_IN"
"134", "49.382751453", "fe80::200:ff:fe00:7", "ff02::2", "OpenFlow", "154",
↪  "Type: OFPT_PACKET_IN"
"135", "49.405229289", "127.0.0.1", "127.0.0.1", "TCP", "66", "6633  >  57536
↪  [ACK] Seq=3149 Ack=2737 Win=512 Len=0 TSval=1828342695 TSecr=1828342672"
"136", "49.882737629", "10.0.0.3", "10.0.0.254", "OpenFlow", "182", "Type:
↪  OFPT_PACKET_IN"
"137", "49.904319020", "127.0.0.1", "127.0.0.1", "OpenFlow", "194", "Type:
↪  OFPT_FLOW_MOD"
"138", "49.904344007", "127.0.0.1", "127.0.0.1", "TCP", "66", "57536  >  6633
↪  [ACK] Seq=2853 Ack=3277 Win=511 Len=0 TSval=1828343194 TSecr=1828343194"
"139", "49.906552936", "127.0.0.1", "127.0.0.1", "OpenFlow", "194", "Type:
↪  OFPT_FLOW_MOD"
"140", "49.906571942", "127.0.0.1", "127.0.0.1", "TCP", "66", "57536  >  6633
↪  [ACK] Seq=2853 Ack=3405 Win=510 Len=0 TSval=1828343196 TSecr=1828343196"
"141", "49.907894906", "00:00:00_07:00:00", "Cisco_10:00:00", "OpenFlow", "236",
↪  "Type: OFPT_PACKET_OUT"
"142", "49.907911267", "127.0.0.1", "127.0.0.1", "TCP", "66", "57536  >  6633
↪  [ACK] Seq=2853 Ack=3575 Win=509 Len=0 TSval=1828343197 TSecr=1828343197"
"143", "51.428836533", "fe80::200:ff:fe00:6", "ff02::2", "OpenFlow", "154",
↪  "Type: OFPT_PACKET_IN"
"144", "51.471152686", "127.0.0.1", "127.0.0.1", "TCP", "66", "6633  >  57536
↪  [ACK] Seq=3575 Ack=2941 Win=512 Len=0 TSval=1828344761 TSecr=1828344718"
"145", "53.476780603", "fe80::200:ff:fe00:5", "ff02::2", "OpenFlow", "154",
↪  "Type: OFPT_PACKET_IN"
"146", "53.476799569", "127.0.0.1", "127.0.0.1", "TCP", "66", "6633  >  57536
↪  [ACK] Seq=3575 Ack=3029 Win=512 Len=0 TSval=1828346766 TSecr=1828346766"
"147", "54.002990134", "127.0.0.1", "127.0.0.1", "OpenFlow", "74", "Type:
↪  OFPT_ECHO_REQUEST"
"148", "54.003014439", "127.0.0.1", "127.0.0.1", "TCP", "66", "6633  >  57536
↪  [ACK] Seq=3575 Ack=3037 Win=512 Len=0 TSval=1828347293 TSecr=1828347292"
"149", "54.015275141", "127.0.0.1", "127.0.0.1", "OpenFlow", "74", "Type:
↪  OFPT_ECHO_REPLY"
"150", "54.015298425", "127.0.0.1", "127.0.0.1", "TCP", "66", "57536  >  6633
↪  [ACK] Seq=3037 Ack=3583 Win=512 Len=0 TSval=1828347305 TSecr=1828347305"
```

```
"151", "57.571568257", "fe80::200:ff:fe00:2", "ff02::2", "OpenFlow", "154",
↪  "Type: OFPT_PACKET_IN"
"152", "57.571646955", "127.0.0.1", "127.0.0.1", "TCP", "66", "6633  >  57536
↪  [ACK] Seq=3583 Ack=3125 Win=512 Len=0 TSval=1828350861 TSecr=1828350861"
"153", "59.002072891", "127.0.0.1", "127.0.0.1", "OpenFlow", "74", "Type:
↪  OFPT_ECHO_REQUEST"
"154", "59.002095373", "127.0.0.1", "127.0.0.1", "TCP", "66", "6633  >  57536
↪  [ACK] Seq=3583 Ack=3133 Win=512 Len=0 TSval=1828352292 TSecr=1828352292"
"155", "59.041548659", "127.0.0.1", "127.0.0.1", "OpenFlow", "74", "Type:
↪  OFPT_ECHO_REPLY"
"156", "59.041572414", "127.0.0.1", "127.0.0.1", "TCP", "66", "57536  >  6633
↪  [ACK] Seq=3133 Ack=3591 Win=512 Len=0 TSval=1828352331 TSecr=1828352331"
"157", "59.625133581", "fe80::200:ff:fe00:4", "ff02::2", "OpenFlow", "154",
↪  "Type: OFPT_PACKET_IN"
"158", "59.625206337", "127.0.0.1", "127.0.0.1", "TCP", "66", "6633  >  57536
↪  [ACK] Seq=3591 Ack=3221 Win=512 Len=0 TSval=1828352915 TSecr=1828352915"
"159", "60.534095112", "10.0.0.4", "10.0.0.254", "OpenFlow", "182", "Type:
↪  OFPT_PACKET_IN"
"160", "60.534114518", "127.0.0.1", "127.0.0.1", "TCP", "66", "6633  >  57536
↪  [ACK] Seq=3591 Ack=3337 Win=512 Len=0 TSval=1828353824 TSecr=1828353824"
"161", "60.588621237", "127.0.0.1", "127.0.0.1", "OpenFlow", "194", "Type:
↪  OFPT_FLOW_MOD"
"162", "60.588634662", "127.0.0.1", "127.0.0.1", "TCP", "66", "57536  >  6633
↪  [ACK] Seq=3337 Ack=3719 Win=511 Len=0 TSval=1828353878 TSecr=1828353878"
"163", "60.590036655", "127.0.0.1", "127.0.0.1", "OpenFlow", "194", "Type:
↪  OFPT_FLOW_MOD"
"164", "60.590050361", "127.0.0.1", "127.0.0.1", "TCP", "66", "57536  >  6633
↪  [ACK] Seq=3337 Ack=3847 Win=510 Len=0 TSval=1828353880 TSecr=1828353880"
"165", "60.590926286", "00:00:00_05:00:00", "Cisco_10:00:00", "OpenFlow", "236",
↪  "Type: OFPT_PACKET_OUT"
"166", "60.590935874", "127.0.0.1", "127.0.0.1", "TCP", "66", "57536  >  6633
↪  [ACK] Seq=3337 Ack=4017 Win=509 Len=0 TSval=1828353880 TSecr=1828353880"
"167", "65.000773611", "127.0.0.1", "127.0.0.1", "OpenFlow", "74", "Type:
↪  OFPT_ECHO_REQUEST"
"168", "65.000833624", "127.0.0.1", "127.0.0.1", "TCP", "66", "6633  >  57536
↪  [ACK] Seq=4017 Ack=3345 Win=512 Len=0 TSval=1828358290 TSecr=1828358290"
"169", "65.063790311", "127.0.0.1", "127.0.0.1", "OpenFlow", "74", "Type:
↪  OFPT_ECHO_REPLY"
"170", "65.063809637", "127.0.0.1", "127.0.0.1", "TCP", "66", "57536  >  6633
↪  [ACK] Seq=3345 Ack=4025 Win=512 Len=0 TSval=1828358353 TSecr=1828358353"
"171", "65.771814720", "00:00:00_00:00:04", "00:00:00_00:00:fe", "OpenFlow",
↪  "126", "Type: OFPT_PACKET_IN"
"172", "65.771876917", "127.0.0.1", "127.0.0.1", "TCP", "66", "6633  >  57536
↪  [ACK] Seq=4025 Ack=3405 Win=512 Len=0 TSval=1828359061 TSecr=1828359061"
"173", "65.788973596", "10.0.0.1", "10.0.0.254", "OpenFlow", "182", "Type:
↪  OFPT_PACKET_IN"
"174", "65.788996008", "127.0.0.1", "127.0.0.1", "TCP", "66", "6633  >  57536
↪  [ACK] Seq=4025 Ack=3521 Win=512 Len=0 TSval=1828359079 TSecr=1828359078"
"175", "65.804822101", "00:00:00_00:00:fe", "00:00:00_00:00:04", "OpenFlow",
↪  "132", "Type: OFPT_PACKET_OUT"
"176", "65.804836769", "127.0.0.1", "127.0.0.1", "TCP", "66", "57536  >  6633
↪  [ACK] Seq=3521 Ack=4091 Win=512 Len=0 TSval=1828359094 TSecr=1828359094"
"177", "65.807262564", "127.0.0.1", "127.0.0.1", "OpenFlow", "194", "Type:
↪  OFPT_FLOW_MOD"
"178", "65.807274186", "127.0.0.1", "127.0.0.1", "TCP", "66", "57536  >  6633
↪  [ACK] Seq=3521 Ack=4219 Win=511 Len=0 TSval=1828359097 TSecr=1828359097"
"179", "65.808059331", "127.0.0.1", "127.0.0.1", "OpenFlow", "194", "Type:
↪  OFPT_FLOW_MOD"
```

```
"180", "65.808065853", "127.0.0.1", "127.0.0.1", "TCP", "66", "57536  >  6633
↪  [ACK] Seq=3521 Ack=4347 Win=510 Len=0 TSval=1828359098 TSecr=1828359098"
"181", "65.808554902", "00:00:00_05:00:00", "Cisco_10:00:00", "OpenFlow", "236",
↪  "Type: OFPT_PACKET_OUT"
"182", "65.808559791", "127.0.0.1", "127.0.0.1", "TCP", "66", "57536  >  6633
↪  [ACK] Seq=3521 Ack=4517 Win=509 Len=0 TSval=1828359098 TSecr=1828359098"
"183", "70.000539764", "127.0.0.1", "127.0.0.1", "OpenFlow", "74", "Type:
↪  OFPT_ECHO_REQUEST"
"184", "70.000565072", "127.0.0.1", "127.0.0.1", "TCP", "66", "6633  >  57536
↪  [ACK] Seq=4517 Ack=3529 Win=512 Len=0 TSval=1828363290 TSecr=1828363290"
"185", "70.044153309", "127.0.0.1", "127.0.0.1", "OpenFlow", "74", "Type:
↪  OFPT_ECHO_REPLY"
"186", "70.044197742", "127.0.0.1", "127.0.0.1", "TCP", "66", "57536  >  6633
↪  [ACK] Seq=3529 Ack=4525 Win=512 Len=0 TSval=1828363334 TSecr=1828363334"
"187", "70.884216874", "00:00:00_00:00:05", "00:00:00_00:00:fe", "OpenFlow",
↪  "126", "Type: OFPT_PACKET_IN"
"188", "70.884280223", "127.0.0.1", "127.0.0.1", "TCP", "66", "6633  >  57536
↪  [ACK] Seq=4525 Ack=3589 Win=512 Len=0 TSval=1828364174 TSecr=1828364174"
"189", "70.884308236", "00:00:00_00:00:01", "00:00:00_00:00:fe", "OpenFlow",
↪  "126", "Type: OFPT_PACKET_IN"
"190", "70.884313415", "127.0.0.1", "127.0.0.1", "TCP", "66", "6633  >  57536
↪  [ACK] Seq=4525 Ack=3649 Win=512 Len=0 TSval=1828364174 TSecr=1828364174"
"191", "70.937715579", "00:00:00_00:00:fe", "00:00:00_00:00:05", "OpenFlow",
↪  "132", "Type: OFPT_PACKET_OUT"
"192", "70.937741227", "127.0.0.1", "127.0.0.1", "TCP", "66", "57536  >  6633
↪  [ACK] Seq=3649 Ack=4591 Win=512 Len=0 TSval=1828364227 TSecr=1828364227"
"193", "70.939617261", "00:00:00_00:00:fe", "00:00:00_00:00:01", "OpenFlow",
↪  "132", "Type: OFPT_PACKET_OUT"
"194", "70.939636246", "127.0.0.1", "127.0.0.1", "TCP", "66", "57536  >  6633
↪  [ACK] Seq=3649 Ack=4657 Win=512 Len=0 TSval=1828364229 TSecr=1828364229"
```