Assignment 2:
Build a simple load balancer

Deadline:30/04/2018

1 Introduction

The final objective of the exercise is to write yourself a load balancer application which uses an OpenFlow switch to balance load stemming from a set of clients towards a set of servers (4 clients-4 servers in particular). More details on creating and submitting the code will be provided later on in the instructions. So, make sure that you follow each step carefully.

The network you'll use in this exercise includes 8 hosts and a switch with an OpenFlow controller (POX). The first 4 hosts (h1 to h4) act as clients for a service offered by the hosts h5 to h8, which act as servers. The switch acts as a load balancer, and balances the flows from the clients towards the servers. The clients are addressing the public IP of the service, and the switch acts as a transparent proxy between the clients and the actual server. The setup is depicted in Fig. 1.

1.1 What the load balancer should do?

- Answer to ARP requests from the clients searching the MAC of the service IP address. The switch should proxy ARP replies that answer to the clients' requests with a fake MAC that is associated with the load balancer (you can use "00:00:00:00:00:1F" for simplicity). 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 needs later to direct flows towards the clients, it will know their MAC addresses and ports to output the packets.

- 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).

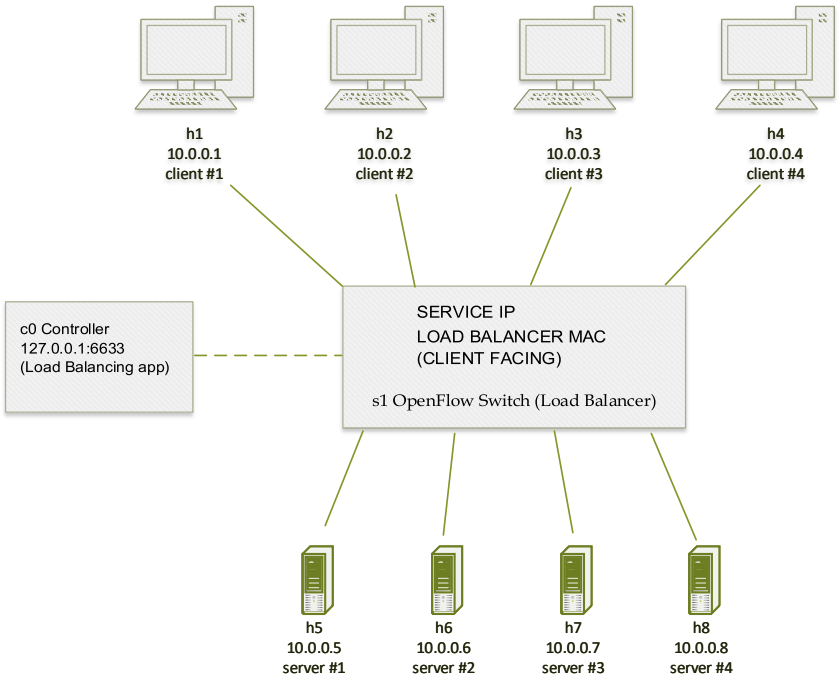

Figure 1: Load Balancer Setup

- Redirect flows from the clients towards the servers using the following load balancing mechanism: **for each new IP flow from a client, select a server at random and direct the flow to this server**. 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 servers. 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.

- 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).

- There is no need to handle forwarding between the servers themselves; in this exercise we are interested in the load-balancing behavior and the traffic that flows between clients and servers.

All the flows should expire after a 10 seconds idle timeout. Default hard timeouts should not be altered. Packet-ins (corresponding to new flows) should be properly

handled by your application. You should only consider ARP and IP protocol packet types.

2 Running Your Application

Write your application as a .py file under ∼/pox/ext (name it SimpleLoadBalancer.py). Your code should run with the following arguments:

ip - the public service ip (could be anything you want, except for the already assigned addresses 10.0.0.1-10.0.0.8)

servers - a list of the server ip addresses (here these are the 10.0.0.5-10.0.0.8)

Therefore, your application should be invoked as follows:

```
$ ./pox.py SimpleLoadBalancer --ip=10.1.2.3
                  --servers=10.0.0.5,10.0.0.6,10.0.0.7,10.0.0.8
```

Try to ping from the clients (hosts $h1$-$h4$) to the service IP address. Show that you achieve load balancing as expected.

The recommended way for showing that load balancing is working, is to send n (n is at least $10,000$) new requests from clients. The controller will log the server that was randomly selected for each request. At the end, plot the number of requests handled by each server. You can use *matplotlib* [1] for plotting.

3 Submission Instructions

Please submit your assignment **BEFORE 23:50 April 30th, 2018**. Submissions after the deadline will not be graded.

Attach a submission comment file which includes short description of how to run your code and full names with IDs of your team members. Specify, who from the team is a corresponding member if a correspondence is required.

ZIP all your code, files, and documents (if any) into a single file named:

```
assignment2_sdn_%MEMBER_ID%.zip
```

where %MEMBER_ID% is the ID of the corresponding member.

Submissions in any other name formats will be penalized. You are strongly encouraged to submit your first draft of your assignment as soon as possible. Later (but before the deadline) you can submit the final version of your assignment. Only the last submission will be graded.

This script should work exactly as expected to get full grade. Points will be subtracted for missing or incorrect functionality. The penalty for late submission is 10% per day. The submitted code will be tested for plagiarism. Any attempt to plagiarize will be accordingly punished.

Before contact, please make sure that you have formulated your question clearly and that you have already studied the POX wiki, the Mininet documentation, and the OpenFlow tutorial thoroughly.

GOOD LUCK!

References

[1] https://matplotlib.org/