Software-Defined Networks

Lab 9

Python Load Balancer Application

University of Colorado Boulder

Department of Computer Science

Professor Levi Perigo, Ph.D.

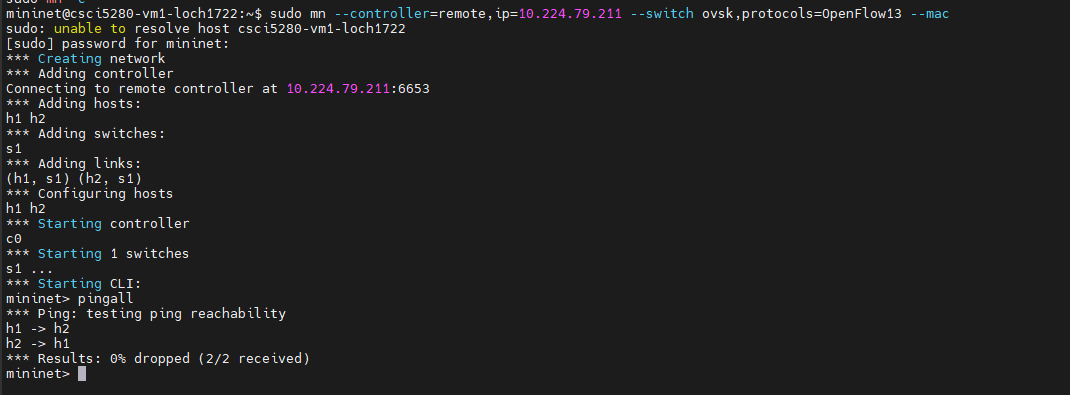
# Lab Summary

Server load-balancers (SLBs) are complex and expensive devices that perform load-balancing across servers based on several factors such as server capability, incoming requests, or round-robin fashion. SDN is a networking concept that aims to centralize networks and make network flows programmable, and NFV focusses on virtualized network functions. SDN/NFV can be used to manage networks better and reduce CapEx/OpEx. SDN-based load-balancers use SDNFV functions and applications to create flexible, programmable, and virtual load-balancing that can be deployed, managed and manipulated with ease in the industry. In this lab, students will deploy, study, and evaluate simple load-balancers, to increase understanding, and will develop their own round-robin load-balancer to application to show how a programming application can be used to control network behavior.

The objectives of this lab are to be used as guidelines, and additional exploration by the student is strongly encouraged.

# Objective 1 – Implement a Stateless Round-Robin Load-balancer

1. Initialize Ryu controller with the simple\_switch\_13.py application. Paste the screenshot of the command used to initialize the ryu application and paste a successful ping across two hosts. **[5 points]**

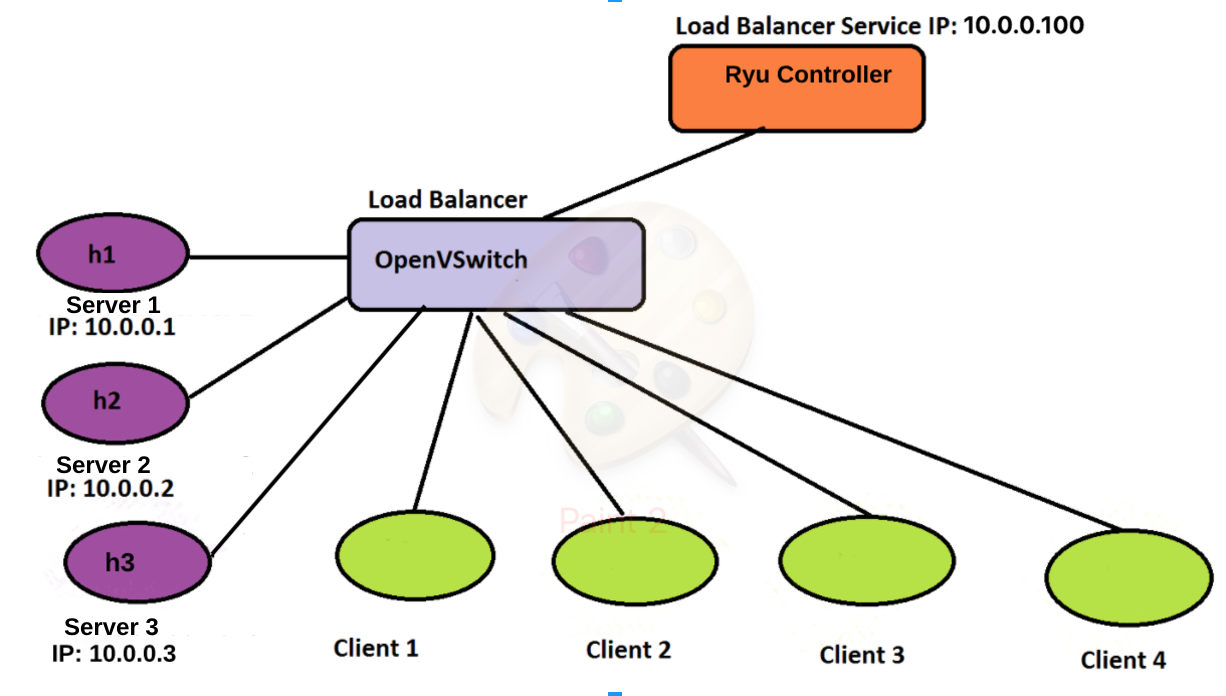


You can terminate your learning switch app now.

1. Use this link to get started with writing applications in Ryu:

<https://ryu.readthedocs.io/en/latest/writing_ryu_app.html>

1. Your task for this lab would be to develop a load balancer Ryu application along with the following specifications:
   * Refer to the network diagram below. That is the topology which you will have to use for this lab using Mininet.



* + The Ryu controller application that you are writing can be a new Python file (a separate one) or it can be run alongside with the existing learning switch app or be inbuilt with the learning switch code. This is left to your decision. Please mention your approach in your submission.
  + The objective of the load balancer application is to ensure that four clients (Client 1 to 4) would be able to talk to different servers (Server 1 to 3) in a round-robin fashion.
  + The main task to note is that the clients know only the IP address of the load balancer (10.0.0.100) and not the IP addresses of the individual servers. The server IPs are known to the Ryu controller app though.
  + Use the service IP address of the load-balancer as 10.0.0.100 and the IP addresses of the servers as 10.0.0.1 (h1), 10.0.0.2 (h2), and 10.0.0.3 (h3). Client IPs are your choice.
  + The servers can be hosting any application of your choice.
  + Your application code should have print statements notifying the details about the in and out packet (for example about details on, which client sent a request, which server is the packet being transmitted to, the server from which the reply is being received, how it is being sent back to the client). Not having print statements in your controller app will result in a loss of 20 points.

Ryu works on an event basis. When an event is received by the controller, you can specify a custom behavior for that event. This is done by identifying the event and writing a function containing the custom behavior code. Whenever that event hits the controller, it will automatically execute the function that you have defined in your code. This is why you just need to write the function definition and not call it in anywhere within your code. These special functions can be identified easily. Just before the “def function\_name(...):” line, you will see:

**@set\_ev\_cls(ofp\_event.EventOFPPacketIn, MAIN\_DISPATCHER)**

This is an example for the Packet-In event to the controller.

Below the @ line, your function definition can start. Example: \_packet\_in\_handler(self, ev).

To create a flow entry, you will have to create three entities: match, actions, mod.

Then you are going to have to send the mod using datapath.send\_msg() function. Reusing the variables from above:

match = parser.OFPMatch(param1=value, param2=value ...)

actions = [parser.OFPActionOutput(OUTPUT\_PORT)]

mod = parser.OFPFlowMod(datapath=datapath, match=match, actions=actions, priority=INTEGER, idle\_timeout=INTEGER, hard\_timeout=INTEGER, cookie=INTEGER, command=TYPE\_OF\_FLOW\_MOD)

datapath.send\_msg(mod)

where,

* param1, param2,.. could be in\_port, ip\_src, ip\_dst, eth\_src, eth\_dst and many more parameters.
* OUTPUT\_PORT is either be the integer output Port number or special values such as ofproto.OFPP\_NORMAL, ofproto.OFPP\_FLOOD etc.
* TYPE\_OF\_FLOW\_MOD defines the type of flow mod. Values such as ofproto.OFPFC\_ADD, ofproto.OFPFC\_MODIFY, ofproto.OFPFC\_DELETE are valid.
* INTEGER - 0 (or 1, 2 etc.)

To find more about OFPMatch, OFPActionOutput and OFPFlowMod, you will just have to visit this source page and search for “class <NAME>” to get an awesome idea of what to include, and examples on how to use the functions. If you are using OF v1.0 use this link.

The sequence:

* **match** helps you match the incoming packet with conditions specified in the arguments.
* **actions** sets the action for the switch (forwarding etc.)
* OpenFlow v1.3 supports something called as **instructions**, which you could look up online to perform complex action tasks.
* **mod** prepares the flow message based on the match and actions variables, and also sets the parameters like priority and timeout. command argument is essential for identifying the type of Flow message to be sent to the switches.
* **send\_msg(mod**) sends out the message.

1. Think of two easy server applications that you should use to test your code. List the applications over here and provide a simple way of starting/testing them. **[10 points]**

I can use OVS commands to see what flows are added and how the switches are behaving from my code. Also Wireshark is another easy application to use to test what traffic is actually being sent and whether it has the expected behavior.

[The definition of Round-robin load balancing is left to your interpretation. As long as consistent round-robin behavior is achieved, and you explain it well in your lab report, points would be awarded.]

1. What are the different types of Round-robin load balancing methods that you can come up with? How would their behavior vary with respect to client <-> server access? Write about two such implementations in your answer. **[10 points]**

The implementation I used was tracking what server was currently being used last and kept that as a global variable. For each server access, the server would change in a round robin sequence. This is seen in the round\_robin() function in my code. This way is handled on the Server side, with no input/state needed from the client. Another way could be a queue system, with the round robin function handling what client is accessing what server. This could be in a state based off their mac-address and we can assign which client is using a specific server.

1. Use xterm to pull up individual server terminals (h1, h2, h3). What command did you use for starting your choice of application on the server? Paste the screenshots of a server running one such application. **[5 points]**

I just used a simple HTTP server and I actually issued the commands on the mininet:

“h1 python -m SimpleHTTPServer 80 &

h2 python -m SimpleHTTPServer 80 &

h3 python -m SimpleHTTPServer 80 &”

A screenshot of a computer program

Description automatically generated

1. Using appropriate commands, make your clients talk to the load-balancer IP in a round-robin fashion and ultimately reach the server application.Paste the screenshot of the command and its corresponding output on all four hosts. **[10 points]**

A computer screen shot of a program

Description automatically generated

1. Paste screenshots of the terminal of corresponding servers that served the client request from the four hosts. **[10 points]**

A screenshot of a computer screen

Description automatically generated

1. Paste the screenshot of the debug messages and the print statements seen on the Ryu controller’s console identifying the traffic that is being directed to the different servers. **[10 points]**

A screenshot of a computer program

Description automatically generated

1. Were the requests being served in a round-robin fashion? Why or why not? **[5 points]**

They were because my round\_robin function was handling what server to send the clients to.

1. Submit the program along with the lab document. Mention the command to execute the program. Ensure your program follows the specification listed above in box points. Each specification carries a weightage while grading this deliverable. **[90 points]**

Command to run: “ryu run loadBalancer.py”

1. Give a fancy name to your Load Balancer application and use that as your filename. **[1 point]**

loadBalancer.py

1. How does SDN help improve/simplify load-balancing implementation? [**5 points**]

SDN improves and simplifies load-balancing implementation because being able to custom why, how and where your clients access the information is endless. SDN can fit whatever network structure you have and can match for any thing you want in terms of how to load balance.

1. To earn full credit please show the functioning of the code to your TA’s.

# Objective 3 – Stateful Load-balancer Application

This is the part where you can leverage SDN capabilities to make your network smarter. Modify your existing load-balancer in such a way that it remembers the clients based on any one state that you define. This could be done on an event basis as well. The statefulness of the load-balancer essentially means that it can force individual clients to go to specific servers based on their event/state. You can remove the learning switch component for this exercise. You will have to leverage the Layer 4 - 7 capabilities of SDN while building this stateful Load-balancer application.

Provide the code for your stateful load-balancer (this file should be different from the stateless load-balancer). Submit screenshots of working application and demonstrate it to one of your TAs for earning full credits. **[35 points]**

I added two new functions but kept the main functionality the same, add\_ip and find\_ip. These two functions track the actual IP of the client and assign it to a server, saving the state of the client and allowing for continued access to remain on the same server.

# Total Score = \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/ 196