Software-Defined Networking

Lab 6

REST – Routing, VLANs, Firewall & QoS

University of Colorado Boulder

Department of Computer Science

Professor Levi Perigo, Ph.D.

# Lab Summary

In the new world of software-defined networking, the crucial component of an SDN network is the communication between network elements - whether it’s between a forwarding plane element and a controller, or between the controller and the applications or business logic.

The northbound API, which provides a list of vendor-agnostic base network functions, is used to interact with the controller and instruct it how to configure the forwarding plane.

A REST API, or an API that is RESTful (adheres to the constraints of REST) is not a protocol, language, or an established standard. It is essentially a set of constraints that an API must follow to be RESTful. The point of these constraints is to maximize the scalability and independence/interoperability of software interactions. REST architectural style gains increasing popularity in the networking protocol design, and it has become a prevalent choice for northbound API of software-defined networking.

The purpose of this lab is to get you familiar with the REST API’s of controllers and the use of cURL to interact with the controller. The objectives of this lab are to be used as guidelines, and additional exploration by the student is strongly encouraged.

You will be using cURL to complete the objectives in the lab. You can use Postman which is a great tool to test the HTTP API. [<https://www.getpostman.com/>]

# Objective 1 – REST Router in Ryu

## Part – A: Static routing via cURL

1. Login to the Controllers VM.
2. How will you check if *ryu-manager* is installed on your VM? Paste the screenshot of the command. [**2 points**]

A close up of a black background

Description automatically generated

1. What is ryu-manager? [**3 points**]

Ryu-manager is a component based SDN framework.

1. Refer the Lab 0 document to initialize the *rest\_router.py* app using-

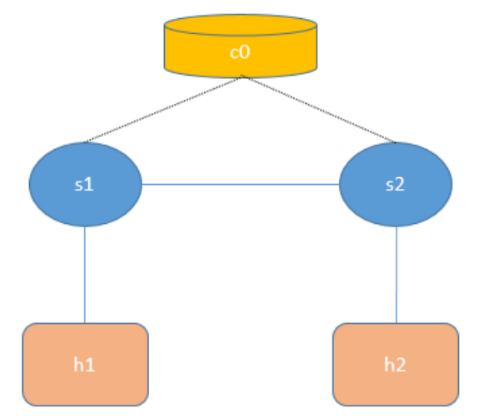
sudo ryu-manager --verbose <path-to-ryu>/rest\_router.py

Provide a screenshot that shows the controller debug output. [**2 points**]

A screenshot of a computer program

Description automatically generated

1. Login to the Mininet VM.
2. Initialize the following topology in Mininet and assign Ryu as the remote controller. Paste a screenshot of the command used. [**3 points**]



A screenshot of a computer

Description automatically generated

1. Check the controller debug output and the flow entries on the two switches. What does it indicate? [**3 points**]

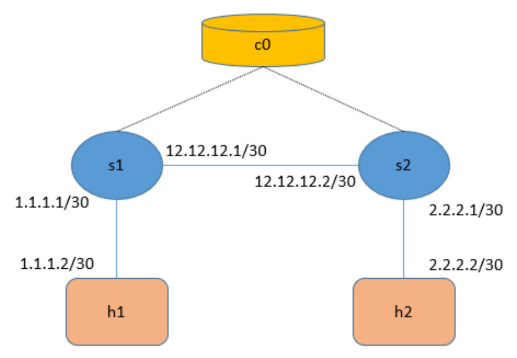
It indicates that the switches are set up as routes, have ARP handling, L2 switching, and a default route set. It also shows a flow that drops IP packets. So it indicates that these switches work as routers.

1. From the Mininet CLI, open xterm windows for s1 and s2, and change the OpenFlow versions of the switches from 1.0 to 1.3. Paste screenshots of the commands used. [**2 points**]





1. From the Mininet CLI, change the IP addresses on h1 and h2 according to the below figure. Paste screenshots of the commands used. [**4 points**]




1. Try to ping h2 from h1. Will it work? Why, why not? [**1 point**]

It will not work because they are not in the same network and there are no routes to each other.

1. Open a new xterm for controller (c0) and use cURL commands to assign IP addresses to both interfaces of each switch s1 and s2. Paste screenshots of the commands used. [**10 points**]

curl -X POST -d '{"address": "1.1.1.1/30"}' http://10.224.78.252:8080/router/000000

0000000001

curl -X POST -d '{"address": "12.12.12.1/30"}' http://10.224.78.252:8080/router/000000

0000000001

curl -X POST -d '{"address": "2.2.2.2/30"}' http://10.224.78.252:8080/router/000000

0000000002

curl -X POST -d '{"address": "12.12.12.2/30"}' http://10.224.78.252:8080/router/000000

0000000002

A screenshot of a computer code

Description automatically generated

1. Explain how the IP addresses were assigned to the switch, whether it was specific to a port or not? Which traditional networking technology can you relate this to? [**10 points**]

It was not a to a specific port, rather they have an address\_id. This could act like a SVI or a loopback that has an IP that is not dedicated to a certain port

1. Paste screenshots of the cURL command to check the IP addresses assigned on each switch s1 and s2. [**5 points**]

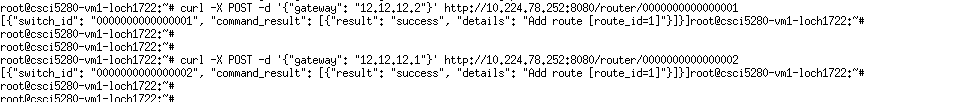
A screenshot of a computer

Description automatically generated

1. Try to ping from h2 from h1. Will it work? Why, why not? [**1 point**]

It is unreachable because there are still no routes to the network.

1. Use cURL commands to set switches s1 and s2 as default routes for each other. Provide screenshots of commands used. [**10 points**]



1. Set s1 as default-gateway for h1 and s2 as default-gateway for h2. Provide screenshots of commands used. [**4 points**]

A black text on a white background

Description automatically generated

A black text on a white background

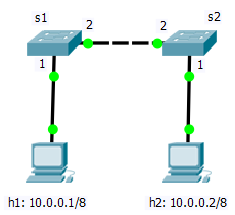
Description automatically generated

1. Try to ping h2 from h1? Will it work? Why, why not? [**1 point**]

It worked because now there are interface Ips and a default route to the respective networks.

## Part B: VLAN configuration via cURL

1. Initialize Ryu using the command: sudo ryu run <path-to-ryu>/ofctl\_rest.py
2. Create the below topology in Mininet which uses easy to read MAC addresses, connects to the remote Ryu controller and uses OpenFlow v1.3:



1. Try to ping from h1 to h2. Will it work? Why/why not? Check the flow entries on the switches. What does it imply? [**5 points**]

It will not work because there are no flow entries linking the hosts together. The controller also doesn’t set up that functionality on start.

1. Using cURL commands, add the below flow entries:
   1. A flow entry on both switches s1 and s2 with priority 20000 to flood ARP packets. To check if ARP is working, try to ping from h1 to h2 and then check the ARP table on h1. Paste the screenshots of the commands used. [**20 points**]

A black text on a white background

Description automatically generated

A computer screen shot of a computer

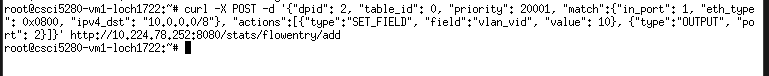
Description automatically generated

A screenshot of a computer

Description automatically generated

* 1. A flow entry on both switches s1 and s2 with priority 20001 which matches any IP packet ingress on port 1 destined for 10.0.0.0/8 with the action to set the VLAN\_ID as 10 and output to port 2. Paste the screenshots of the commands used. [**10 points**]

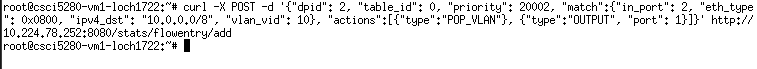




* 1. A flow entry on both switches s1 and s2 with priority 20002 which matches any IP packet ingress on port 2 destined for 10.0.0.0/8 with the action to remove the VLAN\_ID and output to port 1. Paste the screenshots of the commands used. [**10 points**]

A black text on a white background

Description automatically generated



1. Open Wireshark in another terminal. Take captures on interfaces s1-eth2 and s2-eth1. Try to ping from h1 to h2. Will it work? Why/why not? Do you see tagged and untagged packets in the captures? [**10 points**]

It should be able to work because there are flow entries to enable pinging now when receiving a certain type of packet on a VLAN. There would be an untagged packet on s1-eth2 and a tagged packet on s2-eth1.

1. Can you explain the significance of this objective? [**5 points**]

The significance of this objective is to understand the function of flows in a flow table relating to tagging VLANs and showing that any type of traffic can flow given the flow entries.

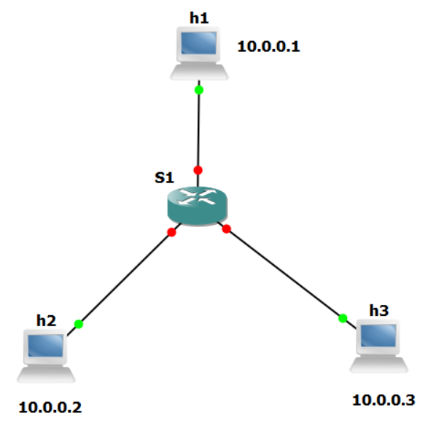
# Objective 2 REST Firewall in Ryu

1. Login to the Controllers VM.
2. Refer the Lab 0 document to initialize the Ryu *rest\_firewall.py* app using-

sudo ryu-manager --verbose <path-to-ryu>/ rest\_firewall.py

Provide a screenshot that shows the controller debug output.

1. Initialize the following Mininet topology. Ensure that the command also opens up xterm windows, and all switches are running OpenFlow v1.3. Provide the screenshot of the command used. [**2 points**]

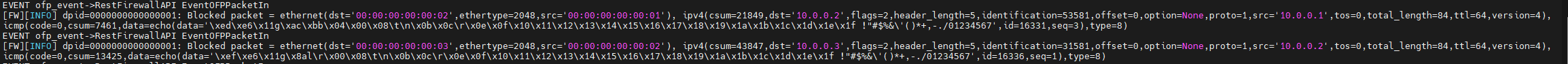




1. From the controller xterm window, enable the Firewall module using cURL command. Provide screenshot of the success message returned. [**5 points**]



1. Try pinging from h1-h2 and h2-h3. Was it successful? What messages do you see on the controller debug? [**2 points**]



1. Modify firewall rules using cURL commands such that ICMP traffic is allowed between h1 and h2 while denied between h2 and h3. How did you achieve this? Provide screenshots of the commands used, message returned and successful/unsuccessful pings. [**20 points**]

A close up of text

Description automatically generated

A black background with white text

Description automatically generated

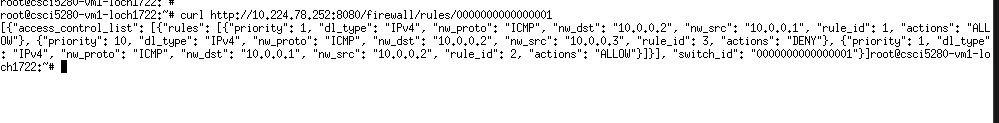


A black screen with white text

Description automatically generated

What I did is do an allow rule for both way IP traffic for h1 and h2, but denied h2 to h3 traffic with a higher priority.

1. How can you check the flow entries installed on the switch using cURL commands? [**5 points**]



1. Using cURL command, delete the rule between h1-h2. How do the results change? Paste relevant screenshots. [**10 points**]

A close-up of a white background

Description automatically generated



When I deleted the rule, they were not able to ping anymore. Because there were no rules matched, it did an implicit deny.

# Objective 3 – REST QoS in Ryu

1. Login to your Controllers VM.
2. Refer this document - <https://osrg.github.io/ryu-book/en/html/rest_qos.html>

* Create a QoS queue with max rate = 500 Kbps.
* Provide a screenshot of the queue created. [**10 points**]

A close up of a computer screen

Description automatically generated

1. Create the below QoS rule:

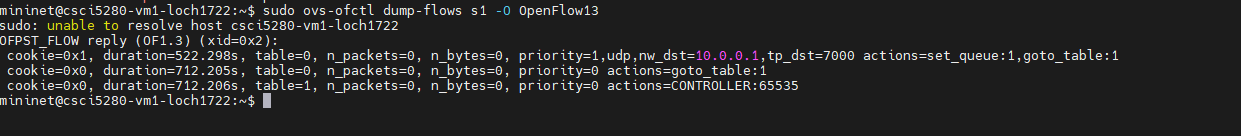
Destination address: Host h1

Destination port: 7000

Protocol: UDP

Provide screenshots of the rule created (both from curl command and OVS flow-table). [**15 points**]

****

****

1. Execute the below commands on the Mininet VM (not on the Mininet CLI), provide screenshots, and explain the outputs. [**15 points**]

sudo ovs-vsctl list port

A screenshot of a computer program

Description automatically generated

This lists all the ports with in depth information about them, one namely being the qos policy field.

sudo ovs-vsctl list qos

A computer screen with white text

Description automatically generated

This gives an in depth description of the qos policy shown above including max rate and queues.

sudo ovs-vsctl list queue

A computer screen with white text

Description automatically generated

And this shows the queues of the device going into depth what I configured before.

1. Test the QoS rule by running iperf on hosts h1 and h2 and provide relevant screenshots. Explain the results. [**20 points**]

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

Here we see 2 IPerf tests. One is sending data to h1 on a random port, and the other is sending data to the port we put the QoS rule on. It seems that our rule allows for full bandwidth while other ports must go through the queue and limit bandwidth.

# Objective 4 – REST ACL in Floodlight

1. Login to your Controllers VM.
2. Refer the Lab 0 document to initialize Floodlight.
3. Create a Mininet topology with a single switch and 5 hosts, using easy-to-read MAC addresses, remote controller as Floodlight and OpenFlow v1.3.
4. Use xterm to open a window for the controller (c0). Using cURL commands, check the existing ACL rules on the controller. Paste screenshot of the command used. [**10 points**]

A close-up of a computer screen

Description automatically generated

1. On the Mininet CLI, run an HTTP server on h5 and perform a wget from h1 to h5. What commands did you use? [**5 points**]

A computer screen with white text

Description automatically generated

h5 python -m SimpleHTTPServer 80 &

h1 wget 10.0.0.5

1. Using cURL commands, add a new ACL rule with these properties:

Protocol: TCP

Source IP: h1

Dest IP: h5

Dest port: 80

Action: Deny

Paste a screenshot of the command used. [**10 points**]

A black text with numbers

Description automatically generated

1. Now perform a wget from h1 to h5, a wget from h2 to h5, and a ping from h1 to h5. Explain the results and paste relevant screenshots. [**20 points**]

A screen shot of a computer

Description automatically generated

A computer screen with white text

Description automatically generated

A screen shot of a computer

Description automatically generated

What is happening here is I just simply created a ACL rule that blocks data destined for port 80. I basically just blocked that website for h1 but it can still ping and h2 can access the website still.

# Objective 5 – REST in ODL

1. Login to your Controllers VM and initialize ODL.
2. Create a Mininet topology with a single switch and 3 hosts, using easy-to-read MAC addresses, remote controller as ODL and OpenFlow v1.3.
3. Refer the URL http://{VM-IP}:8181/apidoc/explorer/index.html for the REST API resources supported by ODL. Paste a screenshot of the webpage. [**2 points**]

A screenshot of a computer

Description automatically generated

1. Using cURL commands, change the following parameters in ODL–
   1. In the l2switch-config, change the is-learning-only-mode and is-install-dropall-flow parameters from false to true. Paste screenshots of the commands used. [**5 points**]
   2. In the arphandler-config, change the is-proactive-flood-mode from true to false. Paste screenshot of the command used. [**5 points**]

Explain the changes in ODL behavior with the above configurations. [**5 points**]

1. Delete any flows on the switch s1. Paste screenshot of the command used. [**2 points**]
2. Try pinging between the 3 hosts. Does the ping work? Why/why not? [**2 points**]
3. Using cURL commands, add flows on the switch s1 to make the ping between the three hosts work. Paste screenshots of the commands used and of the successful pings between the hosts. [**6 points**]

# Total Points \_\_\_\_\_\_\_ / 297